

RATS Handbook for Vector Autoregressions

Thomas A. Doan
Estima

2nd Edition
January 18, 2026

Copyright © 2026 by Thomas A. Doan

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

Contents

Preface	v
1 Introduction	1
1.1 Vector Autoregressions	1
1.2 Log Likelihood Function	2
1.3 Choosing Lag Length	3
1.4 SYSTEM definition and ESTIMATE	6
1.5 Variables and Residuals.	8
1.6 Alternative Estimation Methods	9
1.1 Lag Selection by AIC	11
1.2 Estimation Techniques	12
1.3 Long Lag VAR	13
2 Impulse Response Functions	14
2.1 Moving Average Representation	14
2.2 Computing Impulse Responses	16
2.3 Orthogonalization	21
2.4 Variance Decomposition.	23
2.5 RATS Tips and Tricks	26
2.1 IRF with input shocks	28
2.2 IRF with Cholesky shocks	29
3 Error Bands	30
3.1 Delta method	30
3.2 Bootstrapping	33
3.2.1 Kilian bootstrap-after-bootstrap	34
3.3 Monte Carlo Integration	35
3.3.1 Creating %%RESPONSES	38
3.3.2 The @MCPROCESSIRF procedure	39

3.4	RATS Tips and Tricks	42
3.4.1	The %MODEL function family	42
3.4.2	Block Sampling	43
3.1	Error Bands by Delta Method	45
3.2	Error Bands by Bootstrapping	46
3.3	Error Bands by Kilian bootstrap	47
3.4	Error Bands by Monte Carlo	48
3.5	Error Bands by Bootstrapping with Random Initial Values	49
4	Historical Decomposition and Counterfactual Simulations	51
4.1	Historical Decomposition	51
4.2	Counterfactual Simulations	53
4.3	Error Bands	54
4.1	Historical Decomposition	54
5	Structural VAR's	56
5.1	Eigen Factorizations	56
5.2	Generalized Impulse Responses	57
5.3	Parametric Structural VAR's	58
5.4	Identification.	61
5.5	Estimation.	62
5.6	Structural Residuals	67
5.7	Error Bands	67
5.7.1	Bootstrapping	68
5.7.2	Monte Carlo: Deriving the Posterior Distribution	68
5.7.3	Monte Carlo with Importance Sampling	71
5.7.4	Monte Carlo with Random Walk Metropolis	76
5.7.5	Monte Carlo with the Waggoner-Zha Sampler	81
5.8	Tips and Tricks	84
5.8.1	Using PARMSETS	84
5.8.2	Waggoner-Zha sampler	85
5.1	Eigen factorization	87
5.2	SVAR: A-style Model	88

5.3	SVAR: A-B style model	90
5.4	SVAR: Importance Sampling for Error Bands	92
5.5	SVAR: Random Walk Metropolis for Error Bands	96
5.6	SVAR: Waggoner-Zha Sampler	99
6	Semi-Structural VAR's	101
6.1	ForcedFactor Procedure	102
6.2	Short- and Long-Run Restrictions	103
6.3	Multi-step Restrictions	105
6.4	Error Bands	107
6.1	Blanchard-Quah Decomposition	110
6.2	Multi-Step Restrictions	112
6.3	Short- and Long-Run Restrictions with Error Bands	114
7	Sign Restrictions	116
7.1	Generating Impulse Vectors	116
7.2	Penalty functions.	122
7.3	Multiple Shocks	124
7.3.1	FORCEDFACTOR and the QR Decomposition	125
7.4	Zero Constraints	126
7.5	Fry-Pagan Critique	127
7.6	Historical Decomposition	130
7.7	Convenience functions for sign restrictions	131
7.1	Sign Restrictions: Part I	134
7.2	Sign Restrictions: Part II	136
7.3	Sign Restrictions: Part III	139
7.4	Sign Restrictions: Median Target Method	141
A	Probability Distributions	145
A.1	Multivariate Normal	145
A.2	Chi-Squared Distribution	147
A.3	(Scaled) Inverse Chi-Squared Distribution	148
A.4	Gamma Distribution	149
A.5	Inverse Gamma Distribution	150
A.6	Wishart Distribution	151
A.7	Inverse Wishart Distribution	152

Contents	iv
B VAR Likelihood Function	154
C Properties of Multivariate Normals	157
D Deriving the Schwarz criterion	160
E Delta method	162
F Gibbs Sampling and Markov Chain Monte Carlo	163
G GNU Free Documentation License	167
1. APPLICABILITY AND DEFINITIONS	167
2. VERBATIM COPYING	169
3. COPYING IN QUANTITY.	169
4. MODIFICATIONS	170
5. COMBINING DOCUMENTS	172
6. COLLECTIONS OF DOCUMENTS	173
7. AGGREGATION WITH INDEPENDENT WORKS	173
8. TRANSLATION	173
9. TERMINATION	174
10. FUTURE REVISIONS OF THIS LICENSE	174
11. RELICENSING	175
Bibliography	176
Index	178

Preface

The Vector Autoregression (VAR) was introduced to the economics literature in the famous paper “Macroeconomics and Reality” (Sims (1980*b*)). Since then it, and its close relatives, have become the standard for analyzing multiple time series. Even when more complicated and tightly parameterized models are used, it’s the stylized facts gleaned from VAR analysis that they are expected to explain. In this course, we will be examining techniques that use “flat priors”; that is, the techniques designed to elicit information from the data without the use of informative Bayesian priors. Strongly informative priors (such as the so-called Minnesota prior) are widely used for building forecasting models, but they tend to improve forecasts by shutting down much of the cross-variable interaction. The techniques we will examine are designed primarily to analyze precisely that type of interaction.

This was originally written for the VAR e-course, offered in the fall of 2009. The second edition expands and updates the original offering. In particular,

- Chapter 3 on “Error Bands” adds a discussion of Kilian (1998)’s bias-corrected bootstrap, and the newer `@MCPROCESSIRF` procedure, and updates the descriptions to use features added to RATS with version 9
- Chapter 5 on “Structural VARs” has been greatly expanded to include detailed discussion of methods of computing error bands for impulse responses in structural models
- Chapter 6 on “Semi-Structural VARs” describes procedures for incorporating “medium-run” constraints in addition to impact (short-run) and long-run constraints.
- Chapter 7 on “Sign Restrictions” has been expanded to include coverage of zero constraints (in addition to the sign constraints) and calculation of the Fry-Pagan median target responses.

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

Introduction

1.1 Vector Autoregressions

The Vector Autoregression was not the first significant attempt to extend the ideas of parametric time series modeling from one series to many. Before it came the vector ARMA model. This is a direct extension of the univariate ARMA (Box-Jenkins) model. This takes the form $\Phi(L)\mathbf{Y}_t = \Theta(L)\varepsilon_t$, where Φ and Θ are $m \times m$ finite matrix polynomials in the lag operator. With $\Phi_0 = \Theta_0 = \mathbf{I}$ and $\varepsilon_t|\Omega_{t-1} \sim N(0, \Sigma)$, the free parameters can be estimated by maximum likelihood. However, these proved, in practice, to work poorly: there were no rules of thumb for picking models that transferred over from univariate ARMA models; they tended to be fairly expensive to estimate for the time (1970's); it was quite possible for even low order models to have unidentified parameters.¹ Since the simple global choices for the lag lengths may fail, the models require selecting lag lengths separately for each of the $m \times m$ component polynomials. This is very clumsy when m is more than 2 or 3.

There was one other problem which wasn't fully appreciated until a bit later. Just as with the standard Box-Jenkins procedures for univariate modeling, one step is to decide whether to difference the data or not. However, if a vector process is cointegrated (a linear combination of non-stationary series is stationary), a model specified in first differences loses that long-run connection.

The vector autoregression (VAR) is just a special case of the VARMA. However, it avoids what were seen as the two main pitfalls of the VARMA: there is no potential for unidentified parameters since you can't have polynomial cancellation, and they can be estimated by least squares, rather than maximum likelihood. An additional benefit was discovered a few years later when Engle and Granger introduced the concept of cointegration. Because least squares could handle non-stationary data, the VAR's were able to preserve the long-run relationships among variables.

The main drawback of the VAR is that it has too many free parameters. Considering that ARMA models had largely supplanted simple AR's for univariate modeling precisely because they reduced the number of parameters to be estimated, switching back to the autoregressive formulation when the number of

¹Suppose that you look at a VARMA(1,1) model. If one of the series is white noise, its "own" polynomials in the two parts will cancel regardless of the values of the parameters.

parameters now goes up with the square of the number of variables seemed foolish. And, in fact, the simple VAR's proposed by Sims forecast quite 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 loss function 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. Bayesian VAR's (BVAR's) are an alternative methodology for forecasting with multiple time series. However, for analyzing the joint dynamics of a set of series, the use of either an uninformative prior (OLS) or possibly a weakly informative reference prior makes more sense.

The basic VAR equation takes the form

$$\mathbf{Y}_t = c + \Phi_1 \mathbf{Y}_{t-1} + \dots + \Phi_p \mathbf{Y}_{t-p} + \varepsilon_t \quad (1.1)$$

We'll use m to represent the number of variables. Your variable set should be rich enough to allow for the dynamics of greatest interest. In preparing your data, you generally do the same preliminary transformations that you would likely choose for univariate models (such as taking logs of variables like GDP), though you need to think about maintaining possibly linear relationships among the variables. Interest rates, for instance, are almost always kept in levels, even though one might subject them to some stabilizing transformation if modeled alone. By doing this, if $\log(\text{price level})$ is also included, the real interest rate will be recoverable linearly. In general, you *don't* want to difference. Because the model includes lags, again, the first differences are going to be recoverable linearly if they turn out to be important. And potential cointegrating relationships will be lost if you difference.²

1.2 Log Likelihood Function

A more complete derivation of the log likelihood for a VAR is in appendix B. With identical explanatory variables and serially independent Normal errors, the model can be written in the form:

$$y_t = (\mathbf{I}_m \otimes x_t) \beta + \varepsilon_t, \varepsilon_t \sim N(0, \Sigma), i.i.d.$$

where x_t is the k vector of explanatory variables, \otimes is the Kroneker product, β is the km vector of coefficients for the full system. The likelihood element for data point t is:

$$p(y_t | x_t, \beta, \Sigma) \propto |\Sigma|^{-1/2} \exp \left(-\frac{1}{2} (y_t - (\mathbf{I}_m \otimes x_t) \beta)' \Sigma^{-1} (y_t - (\mathbf{I}_m \otimes x_t) \beta) \right) \quad (1.2)$$

²There *are* situations where a VAR needs to be estimated in first difference form, though that's usually as part of a more involved analysis.

The key part of the exponent:

$$(y_t - (\mathbf{I}_m \otimes x_t) \beta)' \Sigma^{-1} (y_t - (\mathbf{I}_m \otimes x_t) \beta)$$

can usefully be rewritten as:

$$\text{trace} (\Sigma^{-1} \varepsilon_t(\beta) \varepsilon_t'(\beta))$$

where $\varepsilon_t(\beta) = y_t - (\mathbf{I}_m \otimes x_t) \beta$. This uses the property that $\text{trace}(\mathbf{AB}) = \text{trace}(\mathbf{BA})$. If we take logs of (1.2) and sum across t , we get:

$$\log L(\mathbf{Y}|\mathbf{X}, \beta, \Sigma) = \text{const} - \frac{T}{2} \log |\Sigma| - \frac{T}{2} \text{trace} \Sigma^{-1} \hat{\Sigma}(\beta) \quad (1.3)$$

where

$$\hat{\Sigma}(\beta) = \frac{1}{T} \sum_{t=1}^T \varepsilon_t(\beta) \varepsilon_t'(\beta)$$

We can concentrate out the Σ , which has a maximum likelihood estimate of $\hat{\Sigma}(\beta)$. The final term in (1.3) can then be simplified to:

$$-\frac{T}{2} \text{trace} \hat{\Sigma}(\beta)^{-1} \hat{\Sigma}(\beta) = -\frac{T}{2} \text{trace} I_m = -\frac{mT}{2}$$

which doesn't depend upon β . Thus, the likelihood reduces to:

$$\log L(\mathbf{Y}|\mathbf{X}, \beta) = \text{const} - \frac{T}{2} \log |\hat{\Sigma}(\beta)|$$

The constant is $-\frac{mT}{2} \log 2\pi - \frac{mT}{2}$ which has no effect on any properties of the estimates. It is included in all calculations done by RATS, but sometimes omitted by other software. Thus, the reported log likelihoods can be quite different, depending upon choice of software.

As shown in the appendix, the coefficients themselves can be estimated by OLS equation by equation—a well-known result for systems of equations with identical explanatory variables. The overall covariance matrix of the coefficients takes the convenient form

$$\hat{\Sigma} \otimes \left(\sum x_t' x_t \right)^{-1}$$

Even though this could be a huge matrix ($mk \times mk$), it breaks down into the Kroneker product of two much smaller matrices: an $m \times m$ with a $k \times k$.

1.3 Choosing Lag Length

How do we choose p ? That will depend upon what you want to do with the VAR. Aside from being a standard way to examine the joint dynamics of a group of series, VAR's are also a convenient way to deal with joint serial correlation that is basically "noise." What's an appropriate lag length for one use may not be appropriate for the other.

One approach is to use one of the *information criteria* (IC). The basic form for these is $-2\log L + \text{penalty}(p, T)$. However, there are alternatives that give the same rank order of models. For instance, you can use the same formula divided through by the number of observations T . And for the VAR, the constant terms in the likelihood are the same regardless of p (depending only upon m and T), so it's also possible to use $T \log |\hat{\Sigma}| + \text{penalty}(p, T)$ (or the same thing divided by T).

It's important to remember when using any of the IC that all your regressions need to be run over the same range. If you run each VAR over its maximum range, you'll tend to bias the decision one way or the other depending upon the sign of the log likelihood elements of the early data points. Since the log likelihood will change if you do an otherwise inconsequential rescaling of data,³ trivial changes to the data could result in very different decisions. The procedure followed is to pick a p_{\max} larger than you think you might need, then run all VAR's starting at entry $p_{\max} + 1$.

The three main information criteria are the Akaike Information criterion (AIC), the Schwarz Bayesian criterion (variously SBC, BIC or SIC) and the Hannan-Quinn criterion (HQ). The penalty functions for these take the form:

$$\begin{array}{ll} \text{AIC} & mk \times 2 \\ \text{SBC} & mk \times \log T \\ \text{HQ} & mk \times 2 \log(\log T) \end{array}$$

where mk is the number of coefficients in the full β vector for the VAR. Since the number of coefficients in a p lag VAR with d deterministic terms per equation is $m(mp + d)$ and md is the same for all models being examined, we can just use $k = m^2 p$ and get the same decision.⁴

Note that the AIC often is used in a slightly different form which has a minor adjustment to the "2" factor to correct for small-sample bias. AIC picks the model which is "closest" in an approximate calculation of the Kullback-Leibler distance between the test model and the true model. SBC chooses the model which has the highest (approximate) marginal likelihood. HQ uses a penalty function which is specific to choosing the order of an autoregression. All of these are asymptotic approximations, which discard pieces of information that are asymptotically negligible, but which may be quite important in samples of common size.

When models are nested (which will be the case here), the difference between the $-2\log L$ terms in the information criteria will be the likelihood ratio test statistic. This will be asymptotically χ^2 with degrees of freedom equal to the

³If you multiply all the dependent variables by 1000, $|\Sigma|$ goes up by a factor of 1000^m so the reported likelihood goes down.

⁴Technically, the number of estimated parameters should include the $m(m + 1)/2$ free parameters in Σ . However, since that's the same for all models, it has no effect on the decision.

difference in the number of parameters. How do the IC's compare with doing standard hypothesis tests? Suppose $m = 4$ and we have $T = 400$. Adding one lag adds 16 parameters. The penalty for adding one lag is 32.0 for AIC, 95.9 for SBC, and 57.3 for HQ; we will prefer the larger model only if the likelihood ratio statistic is greater than the chosen one of those. If, instead, we were to use a .05 significance level on an LR test, the critical value would be 26.3. The .01 significance level is 32.0. So at degrees of freedom of around 16, AIC and an LR test procedure will give a similar result.⁵ As the degrees of freedom goes up, however, AIC becomes more stringent. This is due to the shape of the chi-squared distribution. The mean of a χ^2_ν is ν , but the standard deviation is $\sqrt{2\nu}$. The ratio of likelihood ratio critical value to ν goes to 1.0 with increasing ν for any fixed significance level, while it's equal to 2.0 for the AIC.

Except for very small numbers of observations, the AIC penalty will always be less than HQ which will always be less than SBC. Since the likelihood function is the same for each, because of the higher penalty SBC cannot pick a model larger than HQ, and HQ cannot pick a model larger than AIC. In practice, when applied to VAR's, SBC and HQ will usually pick models which are fairly similar in size, while AIC will usually pick one quite a bit larger. SBC and HQ both have a property known as *consistency*, which means that as the number of data points increases the probability of their selecting the correct model goes to 1. AIC does not have that property; it will always have a positive probability of picking a model which is too large. Note, however, that this requires that there *is* a correct value of p . If the data aren't representable by a finite VAR, then AIC has a property known as *efficiency*; it comes closer to the other two in approximating the true model. Various writers tend to choose a favorite criterion based upon their feelings about the relative merits of each of those two properties.

An alternative to these is general-to-specific pruning. Like the IC's, this starts with a large number of lags, but rather than minimizing across all choices for p , it does a sequence of tests for p vs $p - 1$. Lags are dropped as long as they test insignificant. This is more commonly applied to univariate autoregressions, such as picking lag lengths in a unit root test. As we can see from the relationship above between AIC and LR testing, this is likely to pick lag lengths even longer than AIC. When this is used for VAR's, it's more commonly used to test larger blocks of lags like $p = 4$ vs $p = 8$. When the data series are highly correlated (as they typically are in an applied VAR), test statistics on a single lag tend to give poor indication of the added explanatory value, particularly on the final lag, which tends to pick up any effect from the already excluded lags.⁶

If you're using the VAR to study joint dynamics, it's usually a good idea to rely partly on one of the IC's combined with judgment about how long a VAR might

⁵The HQ and SBC limits are way out in the tails of the χ^2_{16} .

⁶By constant, in a unit root test, the lags being tested are on the *differenced* data which generally has fairly low serial correlation.

be adequate under the circumstances. For instance, it's usually a good idea to include a year's worth of lags (for macro data) if the data set is long enough to permit it. If you're using a larger model (say $m > 5$), that likely will be infeasible: once you start fitting equations where the number of coefficients per equation is more than 25% of the number of data points, you will be very likely to see signs of overfitting.

If, on the other hand, you're using the VAR to clean up nuisance serial correlation (for instance, when your main interest is in cointegration), then there's nothing wrong with using an automatic method to find the smallest VAR which roughly whitens the residuals.

The procedure `@VARLagSelect` can be used if you want to employ either one of the IC's or the general to specific selection method. An example is:

```
@VARLagSelect (lags=12,crit=aic)
# g3year g3month
```

This uses AIC to select among lags 0 to 12 on a two variable model. The `CRIT` option takes the value `AIC`, `SBC` (or equivalently `BIC`), `HQ` or `GTOS` to select one of the other methods. If you have a situation where you just want to go with the lags chosen by the criterion, you can just use the variable `%%AUTOP` defined by the procedure.

1.4 SYSTEM definition and ESTIMATE

As we've seen, an unconstrained VAR can be estimated using OLS equation by equation. However, that's not the most convenient way to do the calculation. In practice, a VAR is more than just a set of equations; it's a complete unit, with the coefficients, covariance matrix and residuals being needed to fully summarize the information taken from the data. RATS offers a data type called a `MODEL` which is designed precisely to put together equations into this type of unit.

The easiest way to define a `MODEL` for a VAR is with **SYSTEM** and its subcommands. A simple example is:

```
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
```

This defines a model called `VARMODEL`. This has dependent variables (named on the **VARIABLES** line) `DINV`, `DINC` and `DCONS`. Each has lags 1 and 2 on each dependent variable in each equation (**LAGS** line). The only "deterministic" regressor is the `CONSTANT` (**DETERMINISTIC** line). A couple of things to note:

1. The **LAGS** will usually take the form `1 TO p`, whatever number you choose for p . However, you can skip lags if you think that's appropriate. For instance `LAGS 1 2 3 6 9 12`.
2. The **DETERMINISTIC** line (generally shortened to **DET**) includes any regressors that aren't lagged dependent variables. Usually, these are, indeed deterministic variables like `CONSTANT`, seasonals and shift dummies. However, they might also be current and lags of some other variables which are being looked at as exogenous to this set of variables.

The VAR (*Setup/Estimate*) wizard on the *Time Series* menu can be helpful in setting up this set of commands. For a very small model like this, you might very well find that it's easier to just put the commands in directly. However, if you have a much larger collection of variables, and perhaps longer variable names, you might find it handy, even if you know exactly what commands you need.

The standard instruction for estimating the model is **ESTIMATE**. By default, this estimates the model most recently defined. However, you can also use `ESTIMATE (MODEL=model name)` if you've defined more than one model. The VAR (*Setup/Estimate*) wizard both defines and estimates the model, since that's the usual sequence.

The basic form of the instruction is

```
ESTIMATE( options ) start end
```

where *start* and *end* are the limits of the estimation range. In most cases, you can skip those and let **ESTIMATE** figure out the maximum range. However, there are many situations in which you need to control the range, either to compare dynamics across different ranges or to standardize a range to compare models with (for instance) different lags. An example is:

```
estimate * 1978:4
```

which starts as early as possible at the front end (* means default) but ends at 1978:4.

As written, these will show the full regression output. Other than in textbooks, it's generally considered to be bad form and a waste of paper to report the regression coefficients from a VAR. This is because there is almost nothing useful that can be read from these. The dynamics of the data are a very complicated interaction of the coefficients from all the equations, and it's almost impossible to track that through beyond the case of two variables and one lag. If you do look carefully at an output you will generally see something similar to this. Table 1.1 is from a two variable, four lag VAR, where the dependent variable is the second one (R):

A few things to note. First, you can see that there are quite a few sign changes from one lag to the next. That's quite common. Also, notice how the standard

Table 1.1: Output from **ESTIMATE**

	Variable	Coeff	Std Error	T-Stat	Signif
1.	M1{1}	0.001281845	0.000674363	1.90082	0.06992916
2.	M1{2}	-0.002139858	0.001051099	-2.03583	0.05344489
3.	M1{3}	0.002176402	0.001159526	1.87698	0.07326454
4.	M1{4}	-0.001479179	0.000694926	-2.12854	0.04422351
5.	R{1}	1.139310363	0.191265308	5.95670	0.00000450
6.	R{2}	-0.309055947	0.318884410	-0.96918	0.34253821
7.	R{3}	0.052365157	0.323974966	0.16163	0.87300608
8.	R{4}	0.001073235	0.216463351	0.00496	0.99608682
9.	Constant	4.919031297	5.424156442	0.90687	0.37387664

errors are low for the first and last coefficients but higher in the middle. The same thing underlies both of these. When the series themselves have fairly high serial correlation, it's hard to sort out the contribution of each lag individually, since its neighbors have much of the same information. The end lags have fewer neighbors, and thus aren't subject to as much uncertainty. Positive correlation between regressors generally leads to negative correlation among the coefficients, hence the tendency for sign flips. Also part of the standard VAR output are the block F-tests. These are more useful than the coefficients themselves, as they summarize entire blocks—the block behavior is better determined than the individual coefficients. In the case of a two variable system such as is shown below, these function as exogeneity tests. They don't have quite as clear an interpretation when you get into a larger system. It's possible, for instance, for y_2 to have zero coefficients in the equation for y_1 and yet it could be the most important driving force in a 3 variable system if it comes into the y_3 equation and y_3 is in the y_1 equation.

F-Tests, Dependent Variable R		
Variable	F-Statistic	Signif
M1	3.0673	0.0366292
R	15.7588	0.0000024

If you don't need any of the output, just add a `NOPRINT` option to the **ESTIMATE**. This suppresses both the regression report and the block F -tests.

1.5 Variables and Residuals

When you estimate, the **MODEL** keeps track of coefficients and residuals of each equation and the estimated Σ . For many uses, all you'll need is a **MODEL** option on a later instruction and that will take whatever it needs. However, you will also sometimes need more information than the **MODEL** keeps, or need it in a different form.

ESTIMATE defines a few fairly standard variables. For instance, `%NOBS` is the number of observations and `%LOGL` is the log likelihood (assuming Normal errors). However, most of the variables are specific to the VAR. For instance, it defines both `%NREG`, which is the number of regressors in an individual equation and `%NREGSYSTEM`, which is the number of regressors in the full system.

The latter is what you need for computing an IC. `%NVAR` is the number of dependent variables in the system. By using variables like that, you can write code to do additional calculations without having to keep track of the size of the model yourself.

`%SIGMA` is the covariance matrix of residuals and `%XX` is the $(X'X)^{-1}$ matrix from the OLS regression for any of the equations. `%BETASYS` is the stacked coefficient vector: it's a km vector blocked by equation, so the first k elements are the coefficients in the first equation, the next k are from the second equation. The full system covariance for `%BETASYS` is the Kroneker product of `%SIGMA` and `%XX`.⁷

Although the `MODEL` keeps track of the residuals for each equation, it will often be necessary to work with these as a group. You can organize these by using the option `RESIDUALS` on the **ESTIMATE**.

```
estimate(resids=vresids)
```

creates a `VECTOR[SERIES]` called `VRESIDS`. `VRESIDS` will be an m vector, each element of which is a complete series. `VRESIDS(I)` is the series of residuals for the I th equation in your model. What makes this particularly useful is the `%XT` function, which extracts the vector of residuals across equations at a given time period. We'll use that many times.

1.6 Alternative Estimation Methods

There are two alternatives to using **ESTIMATE** which can be used when appropriate. The first uses the `%SWEEP` functions applied to a cross product matrix. `%SWEEP`, `%SWEEPTOP` and `%SWEEPLIST` are discussed in the RATS *Additional Topics* PDF. To use these, start with a cross product matrix (computed using the instruction **CMOM**) set up with the dependent variables in the top rows and the explanatory variables below them. The cross product matrix will thus have the form:

$$\begin{bmatrix} Y'Y & X'Y \\ Y'X & X'X \end{bmatrix}$$

where Y and X are matrices formed by stacking the dependent variables and the regressors across time. If you sweep on the explanatory variables, the resulting matrix will have the form:

$$\begin{bmatrix} T\hat{\Sigma} & B \\ -B' & (X'X)^{-1} \end{bmatrix}$$

where B is the coefficient vector reshaped into a $m \times k$ matrix, with row i being the coefficients for equation i . Note that this one matrix has all the summary information about the estimated VAR other than residuals.

⁷The RATS function for the Kroneker product $A \otimes B$ is `%KRONEKER(A,B)`.

If this is all we do, we haven't really accomplished much, since all of that can be calculated fairly easily by standard regressions. Where this comes in handy is when you sweep out the explanatory variables a few at a time. An example is in the procedure `@VARLagSelect` (described in Section 1.3). In choosing lag length by IC, we first need to ensure that all regressions are run over the same range. By computing a single cross product matrix with range determined by p_{max} lags, we do that. The procedure then starts by sweeping out just the `CONSTANT`. The top $m \times m$ corner can be used to get the log likelihood for a model with $p = 0$. Now sweep out the first lags. Sweeping on one set of variables, then another (non-overlapping) set is the same as sweeping on all of those at once. So we now have the regression on `CONSTANT` and one lag. Again, the top $m \times m$ corner gives us the information to form the log likelihood for $p = 1$. Continue adding a lag at a time to generate the full table of IC for everything up to p_{max} .

OLS and sequential sweeping give exactly the same results. Solving Yule-Walker equations is a different calculation with different results. Yule-Walker equations should only be applied when the model is known to be stationary (no unit roots). This solves out equations using the sample autocovariances:

$$c_{YY}(h) = \frac{1}{T} \sum_{t=h+1}^T (\mathbf{Y}_t - \bar{\mathbf{Y}}) (\mathbf{Y}_{t-h} - \bar{\mathbf{Y}})'$$

This has some (minor) advantages in computational efficiency for situations when you need to do a sequence of these (as in picking lag lengths), but that's not enough to overcome their inability to handle properly non-stationary data. If, for some reason, you need this type of estimate, you can use the RATS procedure `@YuleVAR`. It's quite similar in form to `@VARLagSelect`. For example:

```
@yulevar(model=varmodel,lags=2) 1960:4 1978:4
# dinv dinc dcons
```


Example 1.1 Lag Selection by AIC

This is the preliminary code from Tsay (1998). This is an ideal situation for automatic lag selection since the main interest isn't this specific VAR, but an examination of whether there is a break in the form of the VAR. (The paper demonstrates a test for threshold VAR's). Thus, we really just want to get an adequate number of lags to "whiten" the residuals.

```
open data usrates.xls
calendar(m) 1959
data(format=xls,org=columns) 1959:1 1993:2 fcm3 ftb3
*
* Compute returns and moving average of the spread
*
set g3year = log(fcm3/fcm3{1})
set g3month = log(ftb3/ftb3{1})
set spread = log(ftb3)-log(fcm3)
set sspreload = (spread+spread{1}+spread{2})/3
compute sspreload(1959:1)=spread(1959:1)
compute sspreload(1959:2)=(spread(1959:1)+spread(1959:2))/2
*
spgraph(vfields=3,footer=$
  "Figure 3. Time Plots of Growth Series of U.S. Monthly Interest Rates")
graph(vlabel="3-month")
# g3month
graph(vlabel="3-year")
# g3year
graph(vlabel="Spread")
# sspreload
spgraph(done)
*
* Pick the lags by minimum AIC
*
@VARLagSelect(lags=12,crit=aic)
# g3year g3month
```

Example 1.2 Estimation Techniques

This combines several examples out of Luetkepohl (2006). This actually violates one of the recommendations by unnecessarily converting the data into growth rates. However, it's being used early in the book and the author didn't want to deal with stationarity issues. This does all the lag selection criteria, which give 0 for both HQ and SBC and 2 for AIC and GTOS. The estimates are actually done with $p = 2$.

```
open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
@varlagselect(lags=4,det=constant,crit=aic) * 1978:4
# dinc dcons dinv
@varlagselect(lags=4,det=constant,crit=sbc) * 1978:4
# dinc dcons dinv
@varlagselect(lags=4,det=constant,crit=hq) * 1978:4
# dinc dcons dinv
@varlagselect(lags=4,det=constant,crit=gtos) * 1978:4
# dinc dcons dinv
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate * 1978:4
*
* The Yule-Walker estimates are slightly different from those in the
* text because @yulevar uses a common divisor on the autocovariances
* rather than one which corrects for the number of observations.
*
@yulevar(model=varmodel,lags=2) 1960:4 1978:4
# dinv dinc dcons
```

Example 1.3 Long Lag VAR

This is the setup code for the analysis in Uhlig (2005). This would be a good example to try the wizard for setting up the model, since it has six variables with very long names. If you try one of the lag selection criteria, you'll find that it picks a lag length much smaller than 12. There's enough data here to get up to a full 12 lags. Since the study of the dynamics is the whole point, we don't want to shut that down early.

```
open data uhligdata.xls
calendar(m) 1965:1
data(format=xls,org=columns) 1965:1 2003:12 $
    gdpc1 gdpdef cprindex totresns bognonbr fedfunds
*
set gdpc1      = log(gdpc1)*100.0
set gdpdef     = log(gdpdef)*100.0
set cprindex   = log(cprindex)*100.0
set totresns   = log(totresns)*100.0
set bognonbr   = log(bognonbr)*100.0
*
system(model=sixvar)
variables gdpc1 gdpdef cprindex fedfunds bognonbr totresns
lags 1 to 12
end(system)
estimate(noprint)
```

Impulse Response Functions

As mentioned in the previous chapter, the coefficients on the estimated VAR model have almost no direct usefulness. Individually, they tend to be estimated imprecisely, and collectively, they interact in a very complicated way. So how can we interpret them? The most important summary of the information in the VAR is the Impulse Response Function (IRF) which shows how the system reacts to specific isolated shocks. The complete set of IRF's provides all the information available in the estimated VAR, but in a form which is much easier to understand.

2.1 Moving Average Representation

The basis for the IRF is the multivariate Moving Average Representation (MAR). If a vector process is covariance stationary:

$$\mathbf{Y}_t = \sum_{s=0}^{\infty} \Psi_s \varepsilon_{t-s}, \varepsilon_t = \mathbf{Y}_t - E(\mathbf{Y}_t | \mathbf{Y}_{t-1}, \mathbf{Y}_{t-2}, \dots) \quad (2.1)$$

ε_t forms a vector white noise process: $E\varepsilon_t \varepsilon_{t-k}' = 0$ if $k \neq 0$.¹ Note, however, that (2.1) is not the only possible MAR. Given any non-singular \mathbf{G} , $\eta_t = \mathbf{G}\varepsilon_t$ is also a vector white noise process and

$$\mathbf{Y}_t = \sum_{s=0}^{\infty} (\Psi_s \mathbf{G}^{-1}) (\mathbf{G}\varepsilon_{t-s})$$

is an equivalent MAR. It turns out that the (2.1) form of the MAR, written in terms of the forecast errors, while the easiest to compute, is one of the least useful of the infinite number of MAR's available. The forecast error process might be highly correlated across variables at t , that is, $\Sigma = E\varepsilon_t \varepsilon_t'$ might be far from diagonal; this is referred to as *contemporaneous correlation*. The collection of $\Psi_s(i, j)$ across s (the i, j elements of MAR matrices) show the response of the i th component of \mathbf{Y} over time to a shock in the j th forecast error component, with all other components being held at zero. If the forecast errors are highly correlated, this will be an “atypical” situation. Instead, the standard technique is to choose a matrix \mathbf{G} such that $E(\mathbf{G}\varepsilon_t)(\mathbf{G}\varepsilon_t)'$ is diagonal (or more specifically, the

¹The MAR can be defined more generally for linear projection rather than conditional expectation. The two are the same for a Gaussian process.

identity). The $\mathbf{G}\varepsilon_t$ are known as *orthogonalized innovations*, while the forecast error process is known as the *non-orthogonalized innovations*. Looking at the responses to orthogonalized shocks in isolation is more reasonable since the shocks aren't correlated. Also, because the components of $\mathbf{G}\varepsilon_t$ are uncorrelated both across time and contemporaneously, the variance of

$$\mathbf{Y}_t = \sum_{s=0}^{\infty} \Psi_s \varepsilon_{t-s} = \sum_{s=0}^{\infty} (\Psi_s \mathbf{G}^{-1}) \eta_{t-s}$$

can be computed as

$$\sum_{s=0}^{\infty} \Psi_s^* \Psi_s^{*'} \quad , \text{ where } \quad \Psi_s^* = \Psi_s \mathbf{G}^{-1}$$

rather than the

$$\sum_{s=0}^{\infty} \Psi_s \Sigma \Psi_s'$$

which would be used with the original MAR. Because computing Ψ_s^* turns out to be only slightly more difficult than computing the Ψ_s themselves, this does save calculation time. The real advantage, however, comes from fact that the i th diagonal element of $\Psi_s^* \Psi_s^{*'}$ (the contribution to the variance of the i th component of \mathbf{Y} from orthogonalized shocks s periods back) can be written

$$\sum_j \Psi_s^*(i, j)^2$$

The overall variance is thus

$$\sum_{s=0}^{\infty} \sum_j \Psi_s^*(i, j)^2 = \sum_j \left(\sum_{s=0}^{\infty} \Psi_s^*(i, j)^2 \right)$$

This is the *decomposition of variance*: the overall variance is written as the sum of variance contributions of the components of the orthogonalized innovations. In practice, because the MAR is often computed for a non-stationary set of data, the sums are truncated to

$$\sum_j \left(\sum_{s=0}^{H-1} \Psi_s^*(i, j)^2 \right)$$

which is the variance in an H -step-ahead forecast. This is done because the infinite sum won't converge for the non-stationary variables. For macro data, the horizon of interest is usually somewhere around 2 to 5 years.

2.2 Computing Impulse Responses

An IRF maps out $\partial Y_{t+h}/\partial e_t$ —the response of the system over time to a specific shock pattern. Since Y has m components, a single IRF generates m series of values, one for each dependent variable. A complete moving average representation has an IRF for each of m shocks, thus the full MAR is an $m \times m$ collection of series. Because of the linearity of the process, the IRF's don't depend upon the values of Y_{t-k} or any of the other shocks. Whatever is going on with the process, the IRF tells what the additive effect is to current and future Y if the e_t shock is added in.

Also because of linearity, the response of a rescaled shock is just the rescaling of the response. If you want the response to a .01 shock rather than a 1.0 shock, you can multiply the shock by .01 and compute the IRF, or you can compute the IRF to the 1.0 shock and multiply the response by .01. In general, if you've generated an MAR with respect to a particular set of shocks, you can get the IRF with respect to any specific weighted sum of those shocks by taking the weighted sum of the MAR components. That will be used in doing IRF's with sign restrictions (Chapter 7).

The calculation for an IRF is quite simple. Because of linearity, we can just set all lagged values of Y and future values of ε_t to zero. The only thing driving the process will be shock that we input. Given our choice for e_t , solve:

$$Y_t = \Phi_1 Y_{t-1} + \dots + \Phi_p Y_{t-p} + e_t \quad (2.2)$$

for Y_t . This is the VAR with any deterministic variables (such as the constant) removed. The solution is particularly easy, since the other lags are zero: $Y_t = e_t$. These are often called the *impact responses*. Now solve:

$$Y_{t+1} = \Phi_1 Y_t + \dots + \Phi_p Y_{t-p+1} \quad (2.3)$$

which will be the VAR equation shifted forward in time, with the future shock removed (since it will be zero). This will give us the responses one step out, which will be $\Phi_1 e_t$. If we shift this one step further forward, we'll get two terms, since now we have both Y_{t+1} and Y_t on the right side. The second step response function will be $\Phi_1 \Phi_1 e_t + \Phi_2 e_t$. In general the h step response will be a polynomial of degree h in the Φ matrices.

The analytical expressions for the IRF are only important for computing standard errors (for the IRF itself) by linearization. In practice, given a specific set of Φ matrices, we just need to do the calculations above numerically. With RATS, that can be done using the instruction **IMPULSE**, though it's also included in the computations done by other instructions such as **ERRORS**, **HISTORY** and **FORECAST**.

The two options *required* for **IMPULSE** are **MODEL** and **STEPS**, which tell it what the model is and how many steps of responses you want.² If you just do

²There are other ways to get the model information into **IMPULSE** besides the **MODEL** option. These are described in this chapter's *Tips and Tricks* (Section 2.5).

```
impulse(model=varmodel, steps=8)
```

you'll get three separate tables, with three columns in each. Each table represents a shock; each column in a table shows the response of one of the dependent variables to that shock. Tables, however, aren't generally used to present IRF's; they tend to be much more easily understood as graphs. To create the graphs, we need to save the responses—that's done with the `RESULTS` option. We also need to decide what shock(s) we want. The default is a full set of orthogonalized shocks, as described in Section 2.3. We'll show here how to create IRF's one at a time.

The data set is the investment, income, consumption data set from Lutkepohl, used in example 1.2. These have all been converted to (quarterly) growth rates by log differencing. We first need to specify and estimate the system. We use 2 lags as selected by AIC.

```
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma) * 1978:4
```

We've added the `SIGMA` option to `ESTIMATE`, which shows the covariance/correlation matrix of the residuals:

Covariance\Correlation Matrix of Residuals			
	DINV	DINC	DCONS
DINV	0.001925417927	0.1324241544	0.2827548249
DINC	0.000064749315	0.000124168356	0.5552610759
DCONS	0.000111422795	0.000055565371	0.000080649752

This shows the variances on the diagonal, the covariances below the diagonal and the correlations above the diagonal. We can see from this that the innovations in `DCONS` and `DINC` are highly correlated (.555).

To start, we want to see the response of the variables to a unit shock to income. Income is the second variable in the system. To input an initial shock with specific form, use the option `SHOCKS=shock vector`. In this case, it will be `SHOCKS=||0.0,1.0,0.0||`.³ `RESULTS=TOINCOME` creates a `RECT[SERIES]` with the IRF. `TOINCOME` has 3 rows (one for each variable) and one column (for the one shock that this is computing). The complete instruction to create 8 steps of responses is:

```
impulse(model=varmodel, steps=8, shock=||0.0,1.0,0.0||, $
noprnt, results=toincome)
```

The response of investment (first series in the `VAR` list in the model definition) will be in `TOINCOME(1,1)`, the response of income is `TOINCOME(2,1)` and the response of consumption is `TOINCOME(3,1)`.

³This could also be written as `%UNITV(3,2)`, which generalizes more easily.

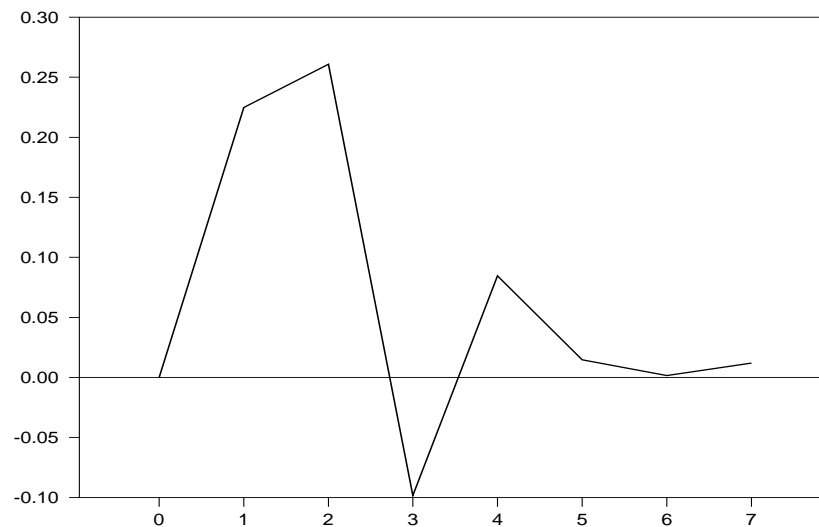


Figure 2.1: Estimated responses of consumption growth to a shock in income

We need a couple of extra options on the **GRAPH** instruction to make IRF graphs look correct. **GRAPH** is designed to graph time series. The IRF is really a zero-based sequence, with no real direct connection to the values observed in a given data set. The options **NODATES** and **NUMBER=0** change the axis labeling so it uses only sequence numbers (**NODATES**) and starts that sequence at 0 (**NUMBER** option). This generates Figure 2.1:

```
graph(nodates,number=0,$
      footer="Estimated responses of ...")
# toincome(3,1)
```

The interpretation of this is that a 1% shock to income growth creates a roughly .25% response of consumption growth in the next two periods, followed by very small changes thereafter. Now, the “shock” here is an unexpected positive change in income which produces no immediate (within quarter) effect on consumption. Given the very high correlation between the residuals for income and consumption, this is a very unrealistic. More interesting would be a shock of the same size to income and consumption. That would be done with:

```
impulse(model=varmodel,steps=8,$
        shock=||0.0,1.0,1.0||,noprint,results=tojoint)
graph(nodates,number=0,footer=$
      "Responses of consumption growth to a common shock in Y and C")
# tojoint(3,1)
```

The response of consumption to this shock is quite different: Figure 2.2.

Note that this is a common shock in the *growth rates* of Y and C, not in the levels. A common shock to the levels (so consumption takes all the initial change in income) is a non-linear function of the variables. As such, it can't be done in isolation from the values of Y and C.

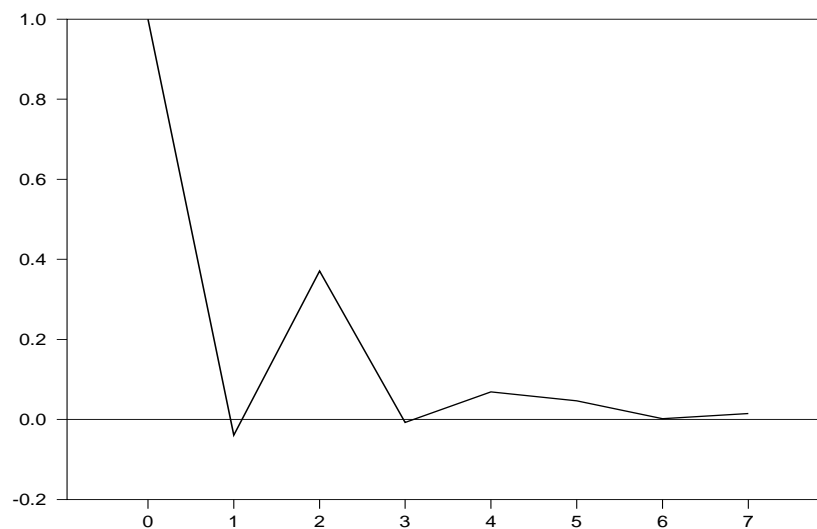


Figure 2.2: Responses of consumption growth to a common shock in Y and C

If we compare the two graphs, we see that in Figure 2.1 the next two periods' consumption apparently needs to “catch up” with income. Since in Figure 2.2, consumption goes up right away, the dynamic is that, at least for the next period, it's the other components that need to catch up and consumption growth goes to zero, though there's a slight rebound in two quarters out.

Note that because all variables are in growth rates, we can choose whatever scale we want. Given the scale of the data (in fractional growth rates: numbers like .01 and .02), a shock of 1.0 is truly enormous—it means 100%. However, because of linearity, the only difference between the responses to a .01 shock and a 1.0 shock is a scale factor of 100. A 1.0 shock can be interpreted as 1% (not 100%) and generates responses with the same interpretation. Suppose, however, that we had interest rates in the common units of percentages per annum in the model. Because the interest rates are in a completely different scale, the 1.0 shock to one of the income variables will likely produce responses in interest rates on the order of 100's of points; now the fact that it is unnaturally high given the data becomes apparent. While you can interpret the interest rate response as basis points, it still is going to look wrong. When you have a mix of data like that, it's generally a good idea to scale the data so that responses can have reasonable interpretations. In this case, for instance, using something like

```
set dinc = 100.0*log(income/income{1})
```

to convert to percentage rather than fractional growth rates will give you graphs that are easier to read.

Converting all variables to comparable scales isn't always enough if the variables are sufficiently different. An alternative is to use *standardized shocks*, which take into account the historical scale, though not the correlation. These

can be computed using the option `SHOCKS=%SQRT(%DIAG(%SIGMA))`. The interpretation is that these are shocks which are of comparable size historically. In particular, if you look at the *responses* of each variable, those responses will be comparable in magnitude and independent of choice of scale of the other variables. If you don't standardize by the historical values, you might, for instance, get a very misleading impression if you included shocks to short rates and shocks to long rates in a single table of graphs since the short rates are noisier and have a much higher historical forecast error standard deviation. Note, however, that if you take this step, it generally will be better to go the whole way to orthogonalization (Section 2.3).

As mentioned earlier, this particular data set probably shouldn't be run in growth rates, since it's quite possible that that will be hiding a long-run relationship among the variables. However, given that we have responses in growth rates, how do we get the responses of the underlying variables? It's possible to add identities to the model which say, for example, $\log Y_t \equiv \log Y_{t-1} + DY_t$. If **IMPULSE** is applied to such an augmented model, it will generate six responses to each shock. However, a simpler way to handle this is to just cumulate up the responses. This again relies on the linearity of the model. Cumulating the growth rate responses gives the responses of the log variable. This is most easily done with the **ACCUMULATE** instruction, which here could just look like:

```
acc tojoint(2,1)
acc tojoint(3,1)
```

This replaces `TOJOINT(2,1)` and `TOJOINT(3,1)` by their running sums. The first element doesn't change, the second is the sum of the first two in the original series, the third the sum of the first three, etc. The response to the joint shock is now Figure 2.3.

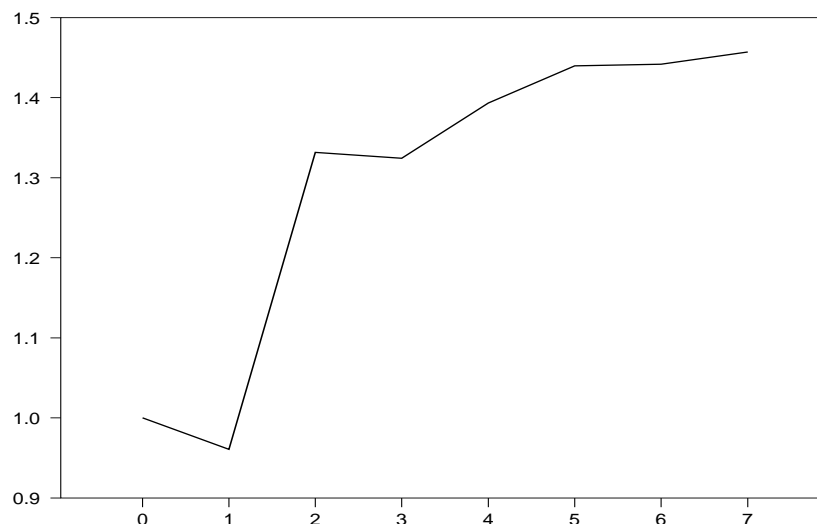


Figure 2.3: Responses of consumption to a common shock in Y and C

One thing to note is that just slapping “results” series into a **GRAPH** instruction is unlikely to produce graphs that convey the information the way you really want. For instance, in this last graph, the minimum value on the scale is .9. It will be more natural (given what the graph means) to set a minimum value on the scale of 0.0. If you look at our first two graphs, even though they show the responses to the same variable to shocks which are comparable in size, a quick glance at the graphs would give a misleading impression—2.1 seems to show a very large delayed response, but only because it’s graphed on a different scale; the period two bump in 2.2 is actually larger. When organizing graphs, you should take this into account and graph comparable responses on comparable scales.

2.3 Orthogonalization

In the last section, we saw that a non-orthogonalized shock was unrealistic when the residuals are correlated. In order to compensate, we replaced it with a joint shock hitting the two most highly correlated components. In some cases (such as that one), we might have a reasonable alternative suggested by theory (like growth theory), but generally we won’t. In most situations, it’s better to let the data guide us to what combinations of shocks are historically “typical.” This is done by orthogonalizing the shocks.

If $E(G\varepsilon_t)(G\varepsilon_t)' = I$, then if $F \equiv G^{-1}$, $FF' = \Sigma$. This is known as a *factorization* of Σ . You can approach the orthogonalization question from either direction. Because Σ is symmetric, it has $m(m+1)/2$ free elements. For a just identified model, either the F matrix or the G matrix will also have $m(m+1)/2$ free elements, and thus must be a restriction from a general $m \times m$ matrix. If you specify the orthogonalization in terms of a G matrix, you’re indicating which linear combinations of the forecast errors are orthogonal. If you work with an F matrix, you’re indicating how the orthogonalized shocks load onto the forecast errors: $\varepsilon_t = F\eta_t$.

The original method of factorization used by Sims in *Macroeconomics and Reality* was the Cholesky factorization. For any positive semi-definite matrix Σ , there is a unique lower triangular matrix L with non-negative elements on the diagonal such that $LL' = \Sigma$.⁴

Because lower triangularity is preserved under inversion, if $F = L$, then $G = F^{-1}$ is also lower triangular. So the first orthogonalized innovation is just a rescale of the first nonorthogonal innovation. Since F will, in general, have all elements in its first column nonzero, this first orthogonalized innovation will hit all variables contemporaneously. The second orthogonalized shock will hit all variables but the first, etc. What is happening here is that all the correlation between the first variable and all others is being attributed to that first

⁴See, e.g. Hamilton (1994), pp 87-92 or Luetkepohl (2006), pp 658-9.

orthogonalized shock. The second orthogonalized shock is the part of the second forecast error that's uncorrelated with the first; the third is the part of the third forecast error uncorrelated with the first two, etc. A VAR which has been analyzed extensively for quite a few countries is the “rmpy” (interest rates, money, price and income). The Cholesky factorization in effect defines a Wold causal chain running from r to m to p to y .⁵

The Cholesky factorization has several advantages: its calculation is straightforward and non-iterative, involving nothing more complicated than a square root, and it's well-defined. Matrix “square roots” in general, aren't. For instance, if, in a 3×3 case, the G matrix takes the form

$$G = \begin{bmatrix} g_{11} & g_{12} & 0 \\ 0 & g_{22} & g_{23} \\ g_{31} & 0 & g_{33} \end{bmatrix} \quad (2.4)$$

in general, there are two non-trivially different solutions, that is solutions which differ by more than sign flips.

By default, the **IMPULSE** instruction generates a full set of responses to each of the Cholesky factor shocks in turn. The results will thus be an $m \times m$ `RECT[SERIES]`. Each column in this represents a shock; each row represents a variable.

How do we present this graphically? Because these are naturally on a common scale (and there are only three variables), one possibility is to have three graphs, each focusing on one shock, showing the response of all three variables to it. However, beyond three variables, graphs like that tend to be rather confusing, and they aren't really appropriate except when all variables reasonably can be represented on a common scale. An alternative is to have three graphs, each focusing on one variable, showing its responses to all shocks. Since the sizes of the shocks are determined empirically, all the responses on a graph will be “comparable” in the sense that they will be scaled based upon the historical size of each shock. While this is a more useful arrangement than the first, it still runs into problems with crowding when $m > 3$.

By far the most common presentation is an $m \times m$ “matrix” of graphs, with each row showing variables and each column showing a shock. In order to make this give an honest view of the responses, you need to make sure you use a common scale on all graphs in a row (that is, all responses of a single variable). The easiest way to produce this is with the **VARIRF** procedure. The basic version of this generates Figure 2.4:

```
@varirf(model=varmodel, steps=8)
```

⁵Wold argued for recursive models rather than simultaneous ones based upon the idea that the “simultaneity” was in many cases only an artifact of the coarseness of the sampling period of the data.

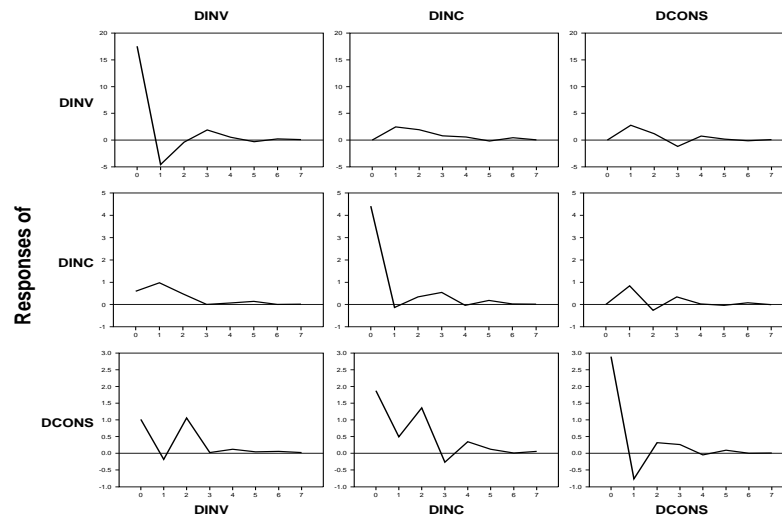


Figure 2.4: Responses to Cholesky factor shocks

With the Cholesky factorization in the order: investment, income, consumption; the impact response of investment to the income and consumption shocks is zero by construction, and the impact response of income to consumption is similarly zero. If we compare scales, it's rather clear that investment is, by far, the noisiest of the series, consumption the smoothest, with income in between. (This is the same data set, but scaled into annual percentage growth). The single greatest response for each series is its own impact response. The only series which shows much else besides that is consumption; whether that is solely because of it coming last in the factorization isn't clear from this. But given a choice, having investment first and consumption last is probably a reasonable timing.

For publication, you would want to make that graph a bit fancier. `@VARIRF` has options for adding a footer, relabeling the variables and shocks, splitting the columns onto separate pages (individual graphs can get too small as m gets large). It also has an option to allow the responses to be accumulated. A fancier version is:

```
@varirf(model=varmodel, steps=8, accum=||1,2,3||, $
  varlabels=||"Investment", "Income", "Consumption"||, $
  footer="Responses to Cholesky factor shocks")
```

2.4 Variance Decomposition

If you do any form of orthogonalization, you can use that to create the decomposition of variance or, more properly, the Forecast Error Variance Decomposition (FEVD).⁶ The IRF in this simple case of a 3 variable model, 8 steps, has 72 com-

⁶The “forecast error” part is because that’s all we can analyze with a non-stationary model; the series variances themselves are infinite.

Table 2.1: Decomposition of Variance for Consumption Growth

Step	Std Error	DINV	DINC	DCONS
1	3.5922083	7.995	27.292	64.713
2	3.7101659	7.725	27.385	64.890
3	4.1028690	12.973	33.364	53.663
4	4.1197619	12.870	33.499	53.631
5	4.1362328	12.859	33.924	53.217
6	4.1394164	12.852	33.963	53.185
7	4.1398539	12.870	33.956	53.174
8	4.1403472	12.870	33.968	53.161

ponents. With 6 variables, 48 steps, it goes up to 1728. The FEVD helps to put those into context—we can see which shocks actually seem to have *economic* significance.

The simplest way to generate the FEVD is with the **ERRORS** instruction:

```
errors(model=varmodel, steps=8)
```

This generates m tables, one for each variable. Each column in a table represents a shock, and each row is for a forecast horizon. As you read across a row, you get the percentage of the forecast error variance at that horizon that is explained by that shock in that and the preceding periods. If we take diagonal elements of:

$$\sum_{s=0}^{H-1} \sum_j \Psi_s^*(i, j)^2 = \sum_j \left(\sum_{s=0}^{H-1} \Psi_s^*(i, j)^2 \right)$$

which is the full covariance matrix of the H -step-ahead forecasts, the percentage of that corresponding to each value for j is that shock's contribution. Table 2.1 shows the decomposition for consumption in the 3 variable model.

The FEVD is usually presented in a table. Because the values tend to be fairly smooth, it's common to skip steps, to do, for instance, 1, 2, 3, 6, 12, 24, 36, 48, to get a feel for the effects at different horizons without wasting space with a set of duplicated values. For a stationary process like this one, the first column (which has the forecast error variance itself) will generally converge fairly quickly, and, as you can see in Table 2.1, the percentages also converge.

For a non-stationary series, the forecast error variance will keep increasing (theoretically, it will eventually go up linearly) and the percentages don't necessarily settle down. However, for non-stationary data, the very long term decomposition tends to be dominated by just one very slightly unstable root. For instance, if one root is (in sample) 1.005, and all others are 1.0 and smaller, at 100 steps, that one root will generate a component sized 1.005^{100} or about 1.65, which will tend to dominate the others. The 1.005, however, is almost certainly just 1 with sampling error. To summarize, don't write a paper based upon what the FEVD tells you about the 30 year horizon. The high end in most published work is around 4-5 years.

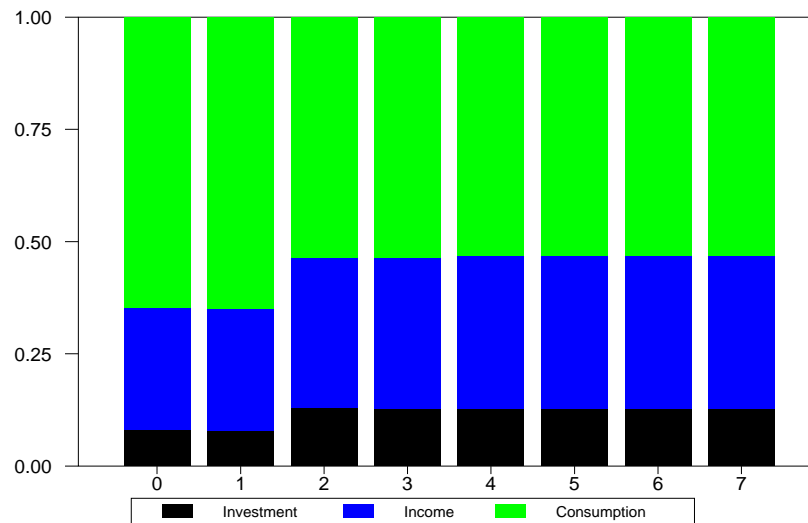


Figure 2.5: Decomposition of Variance for Consumption

Why aren't graphs used much for the FEVD? First, the FEVD's are so smooth (in most cases), that the graphs are rather dull. A table edited to show a limited selection of steps provides more information in a concise fashion. Also, the natural way to display an FEVD is with a stacked bar graph, since it partitions the value 1 (or 100) among the shocks. However, a stacked bar graph with 3 or more series has the middle series with "hanging" limits, making it hard to track them visually. Only the top and bottom, which have one end anchored, are easy to read. If you really want to do this, use the `RESULTS` option on **ERRORS**, which saves the decomposition (as fractions of 1) in a `RECT[SERIES]`. The breakdown for variable i is in row i of this. The following produces the decomposition of variance using Cholesky shocks, with the decomposition of variance for consumption shown in Figure 2.5.

```
dec vect[strings] varlabels(%nvar)
compute varlabels=||"Investment","Income","Consumption"||
errors(model=varmodel,steps=8,results=fevd)
do i=1,%nvar
    graph(footer="Decomposition of "+varlabels(i),style=stacked,$
        number=0,series=%srow(fevd,i),key=below,klabels=varlabels)
end do i
```

The `SERIES` option was added to **GRAPH** with version 9 of RATS which makes it easier to handle graphs of impulse responses and related statistics with greater flexibility. `%SROW` returns a `VECTOR[INTEGER]` with the series handles of a row out of a `RECT[SERIES]`. The related `%SCOL` returns the handles for a column.

2.5 RATS Tips and Tricks

The `MODEL` option as the base for the `VAR` instructions was introduced with RATS version 5. If you used or have run across a program from earlier versions of RATS, you will probably see something like:

```
system 1 to 5
variables hrt1 trt2 srt3 krt4 brt5
lags 1 to 4
det constant
end(system)
estimate(noprint,outsigma=v)
list ieqn 1 to 5
errors(impulses) nvar nstep
cards ieqn ieqn
```

We'll show how to translate this into a more modern coding.

```
system 1 to 5
```

This creates equations with numbers 1 to 5. They are defined at the time that the **END (SYSTEM)** is executed. Together, they make up the `VAR`, but they aren't considered to be a unified whole except when you do an **ESTIMATE** instruction.

```
list ieqn = 1 to 5
errors(impulses) nvar nstep
cards ieqn ieqn
```

The older way to provide the information to instructions like the forecasting instructions **ERRORS**, **FORECAST**, **HISTORY**, **IMPULSE** and **SIMULATE** was a collection of supplementary cards, with one for each equation. In order to simplify this and make it easier to adjust the number of equations, it's possible to use the combination of **LIST**, which gives a controlling list of values, and **CARDS**, which gives a standard form into which the **LIST** index is substituted. This is equivalent to

```
error(impulses) nvar nstep
# 1 1
# 2 2
# 3 3
# 4 4
# 5 5
```

In most cases, these older programs rely heavily on the use of “numbered” series. In this case, the second parameter in each supplementary card gives the series into which the computed forecast standard errors are placed. In more modern coding, these “target” series are more easily handled as a `VECT[SERIES]`. These are much easier to handle since each has its own “namespace”. A better way to write this is:


```

system(model=varmodel)
variables hrt1 trt2 srt3 krt4 brt5
lags 1 to 4
det constant
end(system)
estimate(noprint, outsigma=v)
errors(impulses, model=varmodel, steps=nstep, stderrs=stderrs)

```

In later processing, use `stderrs(1)` to `stderrs(5)` to replace series 1 to 5.

In another example (without the definition of the VAR equations):

```

list ieqn = 1 to nvar
forecast nvar 1 1949:3
cards ieqn ieqn+dumbase 1949:3
do iv=dumbase+1, dumbase+nvar
    set iv 1948:1 1949:3 = ([series]iv)*dumwght
end do iv

```

The forecasts go into series `ieqn+1` to `ieqn+nvar` (based upon the **CARDS** formula). A newer version of this is:

```

forecast(model=varmodel, steps=1, from=1949:3, results=ivf)
do i=1, nvar
    set ivf(i) 1948:1 1949:3 = ivf(i)*dumwght
end do i

```

Note, first, that this uses the `STEPS` and `FROM` options to replace parameters on `FORECAST`. That isn't necessary, but it makes the program easier to read. Instead of the forecasts going into numbered series `DUMBASE+1` to `DUMBASE+NVAR` (which requires the user to create those series and figure out what the base number `DUMBASE` needs to be), they now go into the `VECT[SERIES]` named `IVF`, where the forecasts are in `IVF(1)` to `IVF(NVAR)`.

Example 2.1 IRF with input shocks

This shows how to generate and graph impulse response functions with user input shocks.

```
open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma) * 1978:4
*
impulse(model=varmodel, steps=8, $
    shock=||0.0,1.0,0.0||, noprint, results=toincome)
graph(nodates, number=0, footer=$
    "Estimated responses of consumption growth to a shock in income")
# toincome(3,1)
*
impulse(model=varmodel, steps=8, $
    shock=||0.0,1.0,1.0||, noprint, results=tojoint)
graph(nodates, number=0, footer=$
    "Responses of consumption growth to a common shock in Y and C")
# tojoint(3,1)
graph(nodates, number=0, footer=$
    "Responses of income growth to a common shock in Y and C")
# tojoint(2,1)
*
* Cumulate to get responses of (log) income and consumption.
*
acc tojoint(2,1)
acc tojoint(3,1)
graph(nodates, number=0, footer=$
    "Responses of consumption to a common shock in Y and C")
# tojoint(3,1)
graph(nodates, number=0, footer=$
    "Responses of income to a common shock in Y and C")
# tojoint(2,1)
```

Example 2.2 IRF with Cholesky shocks

This shows how to graph impulse response functions and compute the decomposition of variance using Cholesky factor shocks. Note that while the data set is the same as the previous example, the data are scaled to annual percent growth rates. This makes the scale on the graphs more in keeping with the way information about these series is reported.

```
open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = 400.0*log(income/income{1})
set dcons = 400.0*log(cons/cons{1})
set dinv = 400.0*log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma) * 1978:4
*
* Basic impulse response graphs
*
@varirf(model=varmodel,steps=8)
*
* Fancier output with accumulated responses
*
@varirf(model=varmodel,steps=8,accum=||1,2,3||,$
    varlabels=||"Investment","Income","Consumption"||,$
    footer="Responses to Cholesky factor shocks",errors)
*
* Do decomposition of variance
*
errors(model=varmodel,steps=8)
```

Error Bands

The IRF is a function of the VAR coefficients. Since the coefficient estimates are subject to sampling variation, the IRF is also subject to that. While the error decomposition can tell you something about the economic significance of responses (whether they actually produce any significant movement in other variables), that doesn't tell you whether a response is *statistically* significant.

Three methods have been proposed to compute some form of error bands for IRF's:

- Monte Carlo integration
- Bootstrapping
- Delta method

Macroeconomics and Reality didn't include error bands. The first paper to present them was Sims (1980a) which used Monte Carlo integration (though there really aren't technical details in that paper). The overwhelming majority of subsequent papers which have presented error bands have used Monte Carlo methods, and it's the one we recommend for many reasons. However, we'll present the alternatives as well.

3.1 Delta method

Luetkepohl (1990) derives the asymptotic distribution for impulse response functions and variance decompositions for estimated VAR's. This is a special case of the more general "delta method". The underlying result is that if

$$\sqrt{T}(\hat{\theta} - \theta) \xrightarrow{d} N(0, \Sigma_{\theta})$$

and if $f(\theta)$ is continuously differentiable, then, by using a first order Taylor expansion and some standard results, we get:

$$\sqrt{T}(f(\hat{\theta}) - f(\theta)) \xrightarrow{d} N(0, f'(\hat{\theta})\Sigma_{\theta}f'(\hat{\theta})')$$

The delta method (see Appendix E for more) is fairly commonly used in conventional maximum likelihood estimation, primarily when a particular parameterization is used for computational purposes. For instance, it's sometimes conve-

nient to estimate a variance in log form, $\kappa = \log \sigma^2$.¹ The delta method in that case gives you exactly the same result (subject to minor roundoff) estimating σ^2 indirectly as it would if you could estimate it directly.

In this case, however, the IRF isn't a different parameterization of the VAR. The finite lag VAR corresponds to an infinite lag IRF, and there might not even be a convergent moving average representation if there are unit roots. When employed in this case, the delta method has two main problems:

- The f functions of interest are rather complicated functions of the underlying VAR parameters. While there are convenient recursive methods to calculate them, they are still quite different for IRF's compared with forecasts compared with error decompositions. By contrast, Monte Carlo integration and bootstrapping require just one sampling technique for all applications.
- The f functions are highly non-linear at longer horizons and, in practice, the VAR coefficients themselves aren't all that precisely estimated. As a result, the linear expansion on which these are based is increasingly inaccurate. This is discussed on page 1125 of Sims & Zha (1999).

Although we don't recommend using this, it still is somewhat instructive to see how the calculation can be organized. If you try to write out directly the h step response, and linearize it, you'll face a daunting task. Even as low as $h = 6$, you will have 15 terms with products of up to six of the Φ matrices of lag coefficients. Instead, it's more convenient to write the VAR in the one lag (state space) form:

$$\begin{bmatrix} Y_t \\ Y_{t-1} \\ \vdots \\ Y_{t-p+1} \end{bmatrix} = \begin{bmatrix} \Phi_1 & \Phi_2 & \dots & \Phi_p \\ I & & & \\ & \ddots & & \\ & & I & 0 \end{bmatrix} \begin{bmatrix} Y_{t-1} \\ Y_{t-2} \\ \vdots \\ Y_{t-p} \end{bmatrix} + \begin{bmatrix} \varepsilon_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where

$$A \equiv \begin{bmatrix} \Phi_1 & \Phi_2 & \dots & \Phi_p \\ I & & & \\ & \ddots & & \\ & & I & 0 \end{bmatrix}$$

is zero everywhere except the first m rows, and the block of identity matrices trailing below the diagonal. With this notation, we can compute responses to

¹Nonlinear parameters with values very close to zero can cause problems for numerical differentiation and for convergence checks in standard optimization software. Taking logs changes the possibly troublesome value of 10^{-6} to roughly -13.

ε_t at any horizon h by:

$$\begin{bmatrix} Y_{t+h} \\ Y_{t+h-1} \\ \vdots \\ Y_{t+h-p+1} \end{bmatrix} = \mathbf{A}^{h-1} \begin{bmatrix} \varepsilon_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The derivative of \mathbf{A} with respect to any VAR coefficient is quite simple, since it's made up of copies of them in the first m rows. The derivative of \mathbf{A}^k with respect to any variable θ can be built recursively from

$$\frac{\partial \mathbf{A}^k}{\partial \theta} = \frac{\partial \mathbf{A}^{k-1}}{\partial \theta} \mathbf{A} + \mathbf{A}^{k-1} \frac{\partial \mathbf{A}}{\partial \theta}$$

The response at step h to shock ε_t will be the first m rows of \mathbf{A}^{h-1} times ε_t . That's the function to which we apply the delta method.

The \mathbf{A} matrix for an (already estimated) VAR can be constructed easily using the function `%MODELCOMPANION(model)`. Note also that this doesn't include any deterministic regressors like the constant. It's useful for analyzing the dynamics of the endogenous variables, but doesn't have enough information for actually working with data. The companion matrix has the coefficients in a different order from the one in which they are put into the equations,² so the procedure for computing the asymptotic variance has to allow for that.

This adds the calculation of the delta method error bands to Example 2.1 to create Example 3.1. The procedure `@VARIRFDelta` computes the 3×3 covariance matrix for the responses at each horizon examined. Since consumption is the third variable in the system, we want the 3,3 element of that for the variance of the response.

```
impulse(model=varmodel, steps=8, shock=||0.0,1.0,0.0||, $
  noprint, results=toincome)
*
dec series upper lower
do h=1,7
  @VARIRFDelta(model=varmodel, h=h, shock=||0.0,1.0,0.0||) covxx
  set upper h+1 h+1 = toincome(3,1)+sqrt(covxx(3,3))*2.0
  set lower h+1 h+1 = toincome(3,1)-sqrt(covxx(3,3))*2.0
end do h
graph(nodates, number=0, footer=$
  "Responses of consumption growth to a shock in income") 3
# toincome(3,1)
# upper
# lower
```

²The coefficients in the estimated VAR are blocked by variable, not by lag, so all lags of a single variable are in consecutive locations.

3.2 Bootstrapping

By bootstrapping, we mean a simulation technique which involves resampling the actual data, or something derived from it (such as residuals). There are a number of special issues that arise in bootstrapping with (correlated) time series data. You can't just resample the data since that would break the time sequencing between the dependent variables and their lags. The *block bootstrap* samples the data in time blocks, so there are only occasional data points (at the boundaries between blocks) which are subject to sequencing issues. However, choosing a proper block size can be a bit tricky with highly persistent data. The most common form of bootstrapping used with VAR's is known as the *parametric bootstrap*. This shuffles the residuals, and rebuilds the data using the estimated VAR model.

The “shuffling” part is done with the help of the RATS instruction **BOOT**. This does not, itself, do the resampling—it merely builds the index used in resampling. In order to maintain the contemporaneous relationship among the residuals, they need to be sampled together.

The two procedures **@VARBootSetup** and **@VARBootDraw** can be used to do most of the work.³ **@VARBootSetup** creates a second parallel system for the resampled data; there's not much in it that will be of great interest unless you want to learn how to manipulate **MODELS** and **EQUATIONS** as “symbols”. However, it's useful to see the workings in **@VARBootDraw**. These are the commands that create the resampled residuals:

```
boot entries rstart rend
do i=1,nvar
    set udraws(i) rstart rend = resids(i) (entries(t))
end do i
```

As mentioned above, **BOOT** merely generates the index on entries needed for doing the resampling. What you do with that will depend upon the situation. This particular use of it will generate a **SERIES** of **INTEGERS** drawn with replacement from the range **RSTART** to **REND**. Here, we need to create a rearranged set of shocks (called **UDRAWS**) over that range, keeping residuals from a given time period together. **RESIDS** is the **VECT[SERIES]** of residuals created by **ESTIMATE** on the original model. **UDRAWS** is a **VECT[SERIES]** being created. The **SET** instruction starts with entry **RSTART**. It looks up the number in the **ENTRIES** series corresponding to **RSTART** and takes the data out of component *i* of **RESIDS** for that entry. It will use the same entry number of each of the series. This is then repeated for each entry up to **REND**.

```
forecast (paths,model=model,from=rstart,to=rend,$
    results=%%VARResample)
# udraws
```

³Both are in the file named **VARBootSetup.src**, so when you use it, you automatically pull in the **@VARBootDraw** procedure.

This rebuilds the data with the rearranged shocks. This uses the original model, and the original pre-sample data.⁴ The `PATHS` option allows the input of a complete sequence of shocks into the set of forecasts. If we used `RESIDS` instead of `UDRAWS` on the supplementary card, we would rebuild (exactly) the original data. The generated data goes into the `VECT[SERIES] %%VARResample`, which is used as the dependent variables in the parallel system.

Because the remainder of the code is almost the same as Monte Carlo integration, we'll cover it later.

3.2.1 Kilian bootstrap-after-bootstrap

Kilian (1998) points out that the original OLS estimates are biased away from unit roots (a well-known property of all forms of autoregressions). As a result, a parametric bootstrap based upon them alone will tend to produce incorrect results for the IRF's. He proposes an alternative which uses an initial bootstrap to estimate the bias in the estimates (average difference between the estimates from bootstrapped data to the original OLS estimates that generate the bootstrapped data).

Kilian's bootstrap can be done using the procedures `@KilianBootSetup` and `@KilianBootDraw` in place of the `@VARBootSetup` and `@VARBootDraw` used above. `@KilianBootSetup` does a set of bootstraps of the OLS model, treating the OLS estimates as the true DGP to compute the bias. The subsequent bootstrapping draws are done using the OLS estimates adjusted by the bias.⁵ `@KilianBootDraw` then draws a bootstrap sample using the adjusted model. Since that, too, would have biased coefficients when the VAR is estimated, the IRF's are computed using a bias-corrected set of coefficients (again, with precaution taken against explosive roots).

The use of these procedures is demonstrated in Example 3.3. Kilian's result shows that "naive" parametric bootstrapping produces poor results when applied to VAR's. However, Kilian's procedure, in practice, seems to produce almost identical results to the better-understood Monte Carlo integration. It probably has greater value in models which aren't full VAR's, as for those, Monte Carlo integration isn't as simple as it is when all equations have the same set of explanatory variables.

⁴It's also possible to randomly select a p entry block from the data set to use as the pre-sample values, in case the original data aren't representative. That rarely seems to be done in practice. However, we show how to add this to the bootstrap in the *Tips and Tricks* section (page 43).

⁵There is an added step to check that the adjustment process doesn't produce a model with an explosive root. If the full adjustment does create that, then the largest partial adjustment which leaves the model stationary is taken.

3.3 Monte Carlo Integration

Monte Carlo integration is by far the most common method used to assess the statistical significance of the results generated indirectly from the VAR, such as IRF's and FEVD's. This assumes a Normal likelihood for the residuals. The posterior distribution for Σ and the VAR lag coefficients under the standard “flat” prior for a multivariate regression model is derived in Appendix B. The distribution of any function of those can be generated by simulation. We can approximate the mean, variance, percentiles, or whatever, from any function of interest by computing those across a large number of simulations from the posterior.

The **ESTIMATE** instruction defines the following things that we will need in order to produce draws

- 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 Σ from its unconditional distribution. Its inverse (the *precision matrix* of the residuals) has a Wishart distribution, which is a matrix generalization of a gamma. See Appendix A.6 for more on this. To draw Σ , we need to use %RANWISHARTI which draws an inverse Wishart. It takes two inputs: one is a factor of the target covariance matrix, the other the degrees of freedom. The scale matrix for the Wishart is $(T \times \Sigma(\hat{B}))^{-1}$. The quickest way to factor a matrix is the Cholesky factor, which is done with the %DECOMP function. So we can get the needed factor and the degrees of freedom using the VAR statistics with:

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

The draw for Σ is then done by

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

Conditional on Σ , the (stacked) coefficient vector for the VAR has mean equal to the OLS estimates and covariance matrix:

$$\Sigma \otimes \left(\sum x_t'x_t \right)^{-1} \quad (3.1)$$

While this is potentially a huge matrix (6 variables, 12 lags + constant is 438×438), it has a very nice structure. Drawing from a multivariate Normal requires

a factor of the covariance matrix, which would be very time-consuming at that size.⁶ Fortunately, a factor of a Kroneker product is a Kroneker product of factors. And since the second of the two factors depends only on the data, we can calculate it just once.

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

The random part of the draw for β can be done with:

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

`%RANMVKRON` is a specialized function which draws a multivariate Normal from a covariance matrix given by the Kroneker product of two factors. We need to add this to the OLS estimates to get a draw for the coefficient vector. There are several ways to get this, but since it gives us what we need directly, we'll use `%MODELGETCOEFFS`. 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 3.4.1).

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

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

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

For most applications, you can use the `@MCVARDODraws` procedure. If you take a look at it, you'll see that it has a minor refinement—it uses what's known as *antithetic acceleration*. This is something that can't hurt and might help with the convergence of the estimates. On the odd draws, a value of `BETAU` is chosen as described above. On the even draws, instead of taking another independent draw, the sign flip of `BETAU` is used instead. Of course, the draws are no longer independent, but they are independent by pairs, so the laws of large numbers and central limit theorems apply to those. However, if a function of interest is close to being linear, the `BETAU` and `-BETAU` will nearly cancel between the odd and even draws, leaving a very small sampling variance between pairs. For IRF components, the efficiency gain generally runs about 80%.

Once we have the draw for the coefficients, we reset them in the model with:

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

One major advantage Monte Carlo integration has over the delta method is that once we have a draw for the coefficients, we can evaluate *any* function of

⁶The biggest calculation in estimating that size model is inverting a 73×73 matrix. Inversion and factoring both have number of calculations of order $size^3$, so factoring the 438×438 matrix would take over 200 times as long as factoring the 73×73 .

those using the standard instructions, the same as we would for the OLS estimates. However, we *do* need to keep quite a bit of information organized. A full set of IRF's out to horizon H will have m^2H values for each draw. Since various procedures can produce simulated sets of responses for many draws, we've set these up to produce them in a consistent manner, so they can be processed by the same graphing procedure. This is done with a global `VECT[RECT]` called `%%RESPONSES`. Each element of the `VECTOR` corresponds to a draw. The impulse responses are then stuffed into a `RECT`. Ideally that would be three-dimensional (variable \times shock \times horizon), but RATS doesn't support that, so they're saved as (variable \times shock) \times horizon.

In `@MCVARDoDraws`, the impulse responses are calculated with⁷ :

```
if %defined(ffunction)
    compute factor=ffunction(%outerxx(fsigmad),model)
else
    compute factor=fsigmad
*
impulse(noprint,model=model,results=impulses,steps=steps,$
    factor=factor)
```

By default, this does the Cholesky factor orthogonalization. Note that because the covariance matrix isn't fixed, the shocks themselves will change, so even the impact responses will have some uncertainty (other than the ones forced to be zero). If you want to use any shocks other than Cholesky shocks, you can override the default by passing an `FFUNCTION` option to the procedure. The `FFUNCTION` needs to point to a `FUNCTION` which takes as inputs the draw for the covariance matrix and a copy of the model and returns a `RECTANGULAR` "factor"—we'll see in Chapter 6 why the model (might be) needed. Note that this doesn't need to be an actual factor; just something that defines the shocks that you want.

The options for `@MCVARDoDraws` are:

- `MODEL=`*model to analyze* [required]
- `STEP=`*number of response steps*[48]
- `DRAWS=`*number of Monte Carlo draws*[1000]
- `ACCUMULATE=`*list of variables (by position) to accumulate* [none]
- `FFUNCTION=``FUNCTION(SYMMETRIC,MODEL)` returning `RECTANGULAR`

In the example from this section (Example 3.4, we want to do unit shocks, so we define the following `FUNCTION` for overriding the Cholesky factor:

⁷The descriptions in this section are for a copy of the procedure which requires version 9 because of the use of a `FUNCTION` as an option—a feature added with version 9.

```

function FUnitShock sigma model
type rect          FUnitShock
type symmetric sigma
type model          model
*
compute FUnitShock=%identity(%nvar)
end

```

The draws are then done with

```

@MCVARDoDraws(model=varmodel, ffunction=FUnitShock, $
draws=2000, steps=8)

```

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. This has quite a few options for arranging and labeling the graphs. In our case, we're doing the following:

```

@mcgraphirf(model=varmodel, $
shocks=|| "Investment", "Income", "Consumption" ||, $
center=median, percent=|| .025, .975 ||, $
footer="95% Monte Carlo bands")

```

This renames the shocks (otherwise, they would have the less descriptive names of the variables in the model), uses the median response as the representative and puts bands based upon the 2.5% and 97.5% percentiles of each. You can also do upper and lower bounds based upon a certain number of standard deviations off the central value—that's done with the `STDERRS` option. Although one or two standard error bands around the IRF have been the norm for much of the past thirty years, percentile bands give a better picture in practice.

3.3.1 Creating %%RESPONSES

The three examples which do bootstrapping use basically the same `@MCGraphIRF` to do the graphs, but, since they're not doing Monte Carlo integration, they can't use `@MCVARDoDraws`. To fill in the `%%RESPONSES` values for use in `@MCGraphIRF`, you first have to declare and dimension it properly:

```

declare vect[rect] %%responses
dim %%responses(ndraws)

```

where `NDRAWS` is the number of draws that you are making for the IRF functions. The draw loop in Example 3.2 takes the form:

```

do draw=1,ndraws
  @VARBootDraw(model=varmodel,resids=resids) rstart rend
  *
  * Estimate the model with resampled data
  *
  estimate(noprint,noftests)
  impulse(noprint,model=bootvar,factor=%identity(nvar),$
    flatten=%responses(draw),steps=nsteps)
  infobox(current=draw)
end do draw

```

and the other bootstrap examples are similar. The two key options on the **IMPULSE** are the **FACTOR** option, which defines the shocks that you want to analyze (here, as with the other examples from this chapter, unit shocks), and the **FLATTEN** option, which takes the responses and “flattens” them into the form that **%%RESPONSES** needs, from the natural three dimensions to just two.⁸ The alternative way to pack the information into an element of **%%RESPONSES** would be something like the following:

```

impulse(noprint,model=bootvar,factor=%identity(nvar),$
  results=impulses,steps=nsteps)

dim %%responses(draw) (nvar*nvar,nstep)
ewise %%responses(draw) (i,j)=ix=%vec(%xt(impulses,j)),ix(i)

```

This saves the impulse responses into a **RECT[SERIES]** named **IMPULSES**, dimensions element **DRAW** of **%%RESPONSES** (its dimension should be the number of shocks times the number of variables for the first dimension and the number of steps for the second), then packs the information into **%%RESPONSES** using **EWISE**. The **EWISE** calculation takes the “cross-section” of the response matrix at step **J** (pulled out with the **%XT** function), and converts it into a **VECTOR** (the **%VEC** function). Because of the way that the **RESULTS** option works on **IMPULSE**, that **VECTOR** will be blocked by the originating shock, so the first *m* elements will be the responses to the first shock, the next *m* responses to the second shock, etc. That **VECTOR** is put into the temporary **IX**, and element **I** is pulled out of that to give the required value at subscripts **I** and **J**.

Obviously, it’s much easier to use the **FLATTEN** option if it will work. If the responses can’t be done directly with a single **IMPULSE** instruction, though, you have to use the **EWISE** method.

3.3.2 The @MCPROCESSIRF procedure

While the procedure **@MCGRAPHIRF** is useful for organizing the graphs for the IRF’s in the typical application, there are many situations where you need to graphs set up differently. In those situations, you can use **@MCPROCESSIRF**

⁸**FLATTEN** was added with version 9.

instead. This takes the same information stored into the `%%RESPONSES` array, but rather than producing graphs, just produces `SERIES` with the bounds (and center). Those series can be graphed using a different layout than is permitted by `@MCGRAPHIRF`, or can be put into a table or exported to a file.

An example is the `MACROMODEL.RPF` program from the Kilian (1998) replication files. The main point of the paper is to compare the properties of a “naive” bootstrap with the bias-corrected bootstrap described in the paper. Since the two are done as separate bootstrapping operations, to show a comparison on a single graph requires that the bounds be computed and stored after each set is done.

The bias-corrected bootstrap is done using:

```
@KilianBootSetup(model=bgmodel,resids=resids) rstart rend
declare vect[rect] %%responses(ndraws)
*
infobox(action=define,lower=1,upper=ndraws,progress) $
    "Second stage bootstrap"
do draw=1,ndraws
    @KilianBootDraw(resids=resids) rstart rend
    impulse(model=%%VARReModel,shocks=%unitv(nvar,1),steps=nsteps,$
        flatten=%%responses(draw),noprint)
    infobox(current=draw)
end do draw
infobox(action=remove)
```

Note that this (as written) is doing only the responses to a unit shock in the first variable (Federal Funds rate). These are processed to produce the .025 lower bounds, the .975 upper bounds (thus a 95% limit) and the medians using:

```
@MCProcessIRF(percentiles=||.025,.975||,lower=klower,upper=kupper,$
    irf=kmedian,model=%%VARReModel)
```

`KLOWER` and `KUPPER` (and `KMEDIAN`) are `RECT[SERIES]` which have dimensions 4×1 where 4 is the number of dependent variables in the model and 1 is the number of shocks (the latter just 1 by construction).

The corresponding conventional bootstrap is done with:

```

@VARBootSetup(model=bgmodel) %%VARReModel
*
infobox(action=define,lower=1,upper=ndraws,progress) $
    "Conventional bootstrap"
do draw=1,ndraws
    @VARBootDraw(model=bgmodel,resids=resids) rstart rend
    estimate(noprint,model=%%VARReModel)
    impulse(model=%%VARReModel,shocks=%unitv(nvar,1),steps=nsteps,$
        flatten=%%responses(draw),noprint)
    infobox(current=draw)
end do draws
infobox(action=remove)

```

with the output processed with

```

@MCProcessIRF(percentiles=||.025,.975||,lower=clower,upper=cupper,$
    irf=cmedian,model=%%VARReModel)

```

The program also computes and saves the point estimates of the IRF from the OLS estimates into `BASEIRF` (which is used as the “mean” in the graph), and the Monte Carlo bounds into `MCLOWER` and `MCUPPER`. The following graphs the bounds computed all three ways (Figure 3.1):

```

spgraph(footer="Confidence intervals to shock in Federal Funds Rate",$
    hfields=3,key=below,style=line,symbols=||1,2,3,4||,$
    klabels=||"OLS Response","Bootstrap-after-bootstrap",$
        "Conventional bootstrap","MC Integration"||)
do for target = 2 3 1
    compute header="Response of "+%if(target==2,"IP",$
        %if(target==1,"FF","CPI"))+" to FF"
    graph(number=0,header=header) 7
    # baseirf(target,1)
    # klower(target,1) / 2
    # kupper(target,1) / 2
    # clower(target,1) / 3
    # cupper(target,1) / 3
    # mclower(target,1) / 4
    # mcupper(target,1) / 4
end do for target
spgraph(done)

```

This uses a feature added with version 9 which allows a single key to be placed outside of an `SPGRAPH` array. Since there is no content at that point, you have to input the intended style, key labels, and collection of corresponding “symbols” that you want for the key. This uses the base style (1, solid black by default) for the OLS response and different colors (dash styles if rendered in black and white) for the upper and lower bound pairs for each of the three methods of generating bounds. Note that there are actually four variables in the VAR, but

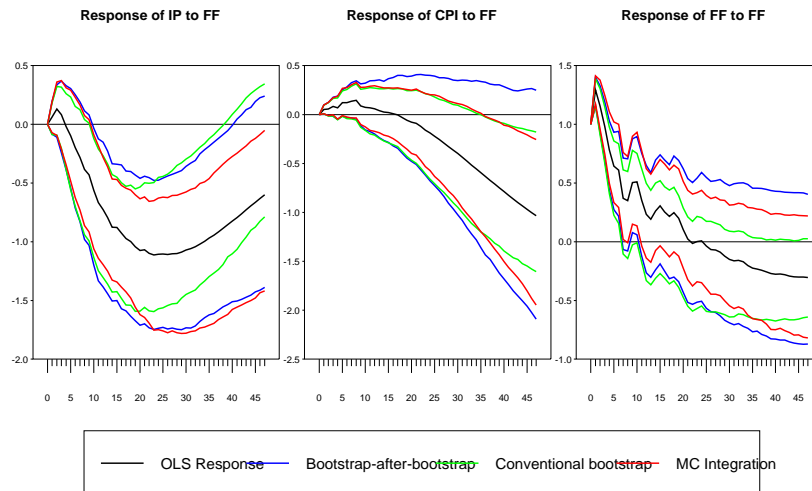


Figure 3.1: Comparison of error band computations

only three were included in the graph and they are done in the specific left-to-right order that was used in the paper.

3.4 RATS Tips and Tricks

3.4.1 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.

%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.

3.4.2 Block Sampling

BOOT with the `BLOCK=blocksize` option can be used to draw entries to sample contiguous blocks of a specific size. The method for handling the blocks is controlled by the `METHOD` option. Of these, `METHOD=OVERLAP` (the default) is likely to be the most useful—it allows the same probability for each full block of size *blocksize* from the available range.

Block bootstrapping isn't recommended for VAR's as an alternative to the parametric bootstrapping of section 3.2—it only works well when the data have zero means (and obviously no drift) and are only lightly serially correlated. Where block selection *can* be helpful is in randomly choosing the initial conditions for the parametric bootstrap. If a set of series are relatively highly serially correlated, the initial conditions can have a fairly persistent effect on the generated data, so randomizing across those can reduce that effect.

You can still largely rely upon the `@VARBootSetup` procedures for doing the enhanced bootstrap as we see in Example 3.5.

There are two source ranges used in this type of bootstrapping: the defined range of the residuals as before, and the range of possible *p* entry blocks in the data range where *p* is the number of lags in the VAR. The latter is the same as the estimation range except that the prior *p* periods are also available. You include the full source range (through the end of the sample) on the **BOOT** instruction when block bootstrapping and **BOOT** will make sure it handles that correctly. With the simple (default) block bootstrap, it won't choose a block which would run off the end of the data. However, there are some other choices (the stationary and circular bootstraps which aren't useful in this setting) which can start up to the final entry in the range.

```
*
* Range of residuals
*
compute rstart=%regstart()
compute rend   =%regend()
*
* Range of blocks for pre-sample data
*
compute dstart=%regstart()-p
compute dend   =%regend()
```

It's necessary to make a copy of the original data since we need to re-sample those for forming the initial conditions. (The simpler bootstrap requires only the residuals and the fixed pre-sample data).

```
dec vect[series] %%VAROrig(nvar)
do i=1,nvar
  set %%VAROrig(i) = %%VARResample(i)
end do i
```

The adjustment inside the draw loop is to first do:

```
boot(block=p) dentries dstart dstart+p-1 dstart dend
do i=1,nvar
  set %%VARResample(i) dstart dstart+p-1 = $
  %%VAROrig(i) (dentries(t))
end do i
```

This draws a set of p consecutive values for entries `DSTART` to `DSTART+P-1` (which will be the pre-sample range for a p lag model) from the range from `DSTART` to `DEND`. The pre-sample range of each of the `%%VARResample` series is then patched over from the original data using the source entry numbers just drawn. The rest of the loop is the same as before, using `@VARBootDraw` to draw a bootstrapped data set.

For this example, there is relatively little difference between the two bootstrap procedures because the model (done in log differences) has relatively low serial correlation, so the initial conditions wear off rather quickly.

Example 3.1 Error Bands by Delta Method

```

open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma) * 1978:4
*
impulse(model=varmodel,steps=8,shock=||0.0,1.0,0.0||,$
        noprint,results=toincome)
*
dec series upper lower
do h=1,7
    @VARIRFDelta(model=varmodel,h=h,shock=||0.0,1.0,0.0||) covxx
    set upper h+1 h+1 = toincome(3,1)+sqrt(covxx(3,3))*2.0
    set lower h+1 h+1 = toincome(3,1)-sqrt(covxx(3,3))*2.0
end do h
graph(nodates,number=0,footer=$
    "Responses of consumption growth to a shock in income") 3
# toincome(3,1)
# upper
# lower

```

Example 3.2 Error Bands by Bootstrapping

```

open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma,resids=resids) * 1978:4
*
@VARBootSetup(model=varmodel) bootvar
*
compute rstart=%regstart()
compute rend =%regend()
*
compute ndraws=2000
compute nvar =3
compute nsteps=8
declare vect[rect] %%responses
dim %%responses(ndraws)
*
infobox(action=define,progress,lower=1,upper=ndraws) $
  "Bootstrap Simulations"
do draw=1,ndraws
  @VARBootDraw(model=varmodel,resids=resids) rstart rend
  *
  * Estimate the model with resampled data
  *
  estimate(noprint,noftests)
  impulse(noprint,model=bootvar,factor=%identity(nvar),$
    flatten=%%responses(draw),steps=nsteps)
  infobox(current=draw)
end do draw
infobox(action=remove)
*
@mcgraphirf(model=varmodel,$
  shocks=||"Investment","Income","Consumption"||,$
  center=median,percent=||.025,.975||,$
  footer="95% other-percentile bootstrap bands")

```

Example 3.3 Error Bands by Kilian bootstrap

```

open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma,resids=resids) * 1978:4
*
compute nsteps=8
compute nvar =%nvar
compute ndraws=2000
*
compute rstart=%regstart()
compute rend =%regend()
*
@KilianBootSetup(model=varmodel,resids=resids) rstart rend
declare vect[rect] %%responses(ndraws)
*
infobox(action=define,lower=1,upper=ndraws,progress) "Second stage bootstrap"
*
do draw=1,ndraws
  @KilianBootDraw(resids=resids) rstart rend
  impulse(model=%VARReModel,factor=%identity(nvar),steps=nsteps,$
    flatten=%%responses(draw),noprint)
  infobox(current=draw)
end do draw
infobox(action=remove)
*
@mcgraphirf(model=varmodel,$
  shocks=||"Investment","Income","Consumption"||,$
  center=median,percent=||.025,.975||,$
  footer="95% bootstrap-after-bootstrap bands")

```

Example 3.4 Error Bands by Monte Carlo

```

open data e1.dat
calendar(q) 1960
data(format=prn,org=columns, skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 2
det constant
end(system)
estimate(sigma,resids=resids) * 1978:4
*
function FUnitShock sigma model
type rect      FUnitShock
type symmetric sigma
type model     model
*
compute FUnitShock=%identity(%nvar)
end
*
@MCVARDoDraws(model=varmodel, ffunction=FUnitShock, $
    draws=2000, steps=8)
@mcgraphirf(model=varmodel, $
    shocks=| | "Investment", "Income", "Consumption" | |, $
    center=median, percent=| | .025, .975 | |, $
    footer="95% Monte Carlo bands")

```

Example 3.5 Error Bands by Bootstrapping with Random Initial Values

```

open data e1.dat
calendar(q) 1960
data(format=prn,org=columns,skips=6) 1960:01 1982:04 invest income cons
*
set dinc = log(income/income{1})
set dcons = log(cons/cons{1})
set dinv = log(invest/invest{1})
*
* Number of lags
*
compute p=2
*
system(model=varmodel)
variables dinv dinc dcons
lags 1 to p
det constant
end(system)
estimate(sigma,resids=resids) * 1978:4
*
@VARBootSetup(model=varmodel) bootvar
*
compute nvar=%modelsz(bootvar)
*
* Range of residuals
*
compute rstart=%regstart()
compute rend =%regend()
*
* Range of blocks for pre-sample data
*
compute dstart=%regstart()-p
compute dend =%regend()
*
* Make copy of the original data (since we will need it for
* bootstrapping the initial values)
*
dec vect[series] %%VAROrig(nvar)
do i=1,nvar
    set %%VAROrig(i) = %%VARResample(i)
end do i
*
compute ndraws=2000
compute nsteps=8
*
declare vect[rect] %%responses
dim %%responses(ndraws)
*
infobox(action=define,progress,lower=1,upper=ndraws) $
    "Bootstrap Simulations"
do draw=1,ndraws

```

```

boot(block=p) dentries dstart dstart+p-1 dstart dend
do i=1,nvar
    set %%VARResample(i) dstart dstart+p-1 = $
        %%VAROrig(i) (dentries(t))
end do i
@VARBootDraw(model=varmodel,resids=resids) rstart rend
*
* Estimate the model with resampled data
*
estimate(noprint,nofstests)
impulse(noprint,model=bootvar,factor=%identity(nvar),$
    flatten=%%responses(draw),steps=nsteps)
infobox(current=draw)
end do draw
infobox(action=remove)
*
@mcgraphirf(model=varmodel,$
    shocks=| | "Investment", "Income", "Consumption" | |,$
    center=median,percent=| | .025, .975 | |,$
    footer="95% other-percentile bootstrap bands")

```


Historical Decomposition and Counterfactual Simulations

If you forecast a VAR, adding at each stage the residuals that are generated using the estimated model, you will, *per force*, re-construct the observed data. In this chapter, we will look at several related techniques which examine what would happen to the data if the sequence of shocks were somehow different from those corresponding to the estimated model.

4.1 Historical Decomposition

The historical decomposition has been a part of the VAR methodology since the early 1980's. It was first developed during the work on Sims (1980a), though the first paper based upon it would seem to be Burbidge & Harrison (1985). It is based upon the following reorganization of the moving average representation for a vector time series \mathbf{Y} :

$$\mathbf{Y}_{T+j} = \sum_{s=0}^{j-1} \Psi_s \varepsilon_{T+j-s} + \left[\hat{\mathbf{Y}}_{T+j} + \sum_{s=j}^{\infty} \Psi_s \varepsilon_{T+j-s} \right] \quad (4.1)$$

The first sum represents that part of \mathbf{Y}_{T+j} due to innovations in periods $T+1$ to $T+j$ (the j -step forecast error), while the term in brackets is the forecast of \mathbf{Y}_{T+j} given data through T .¹ If the $\hat{\mathbf{Y}}_{T+j}$, ε_t and Ψ_s are all from the same estimated VAR, (4.1) is an identity.

While there are some similarities between this and the decomposition of variance, there are also some key differences. The most important is that this is *not* a partitioning of any quantity. If $\varepsilon = \mathbf{F}\eta$ for any (full rank) matrix \mathbf{F} , then we can rewrite the forecast error as

$$\sum_{s=0}^{j-1} \Psi_s \varepsilon_{T+j-s} = \sum_{s=0}^{j-1} \Psi_s \mathbf{F} \mathbf{F}^{-1} \varepsilon_{T+j-s} = \sum_{i=1}^m \left(\sum_{s=0}^{j-1} \Psi_s \mathbf{F} \mathbf{e}(i) \mathbf{e}(i)' \mathbf{F}^{-1} \varepsilon_{T+j-s} \right) \quad (4.2)$$

($\mathbf{e}(i)$ is the i th unit vector).

In computing the FEVD, we choose an \mathbf{F} matrix so $\eta = \mathbf{F}^{-1} \varepsilon$ has a diagonal covariance matrix. This eliminates the covariances between terms in the final

¹We're denoting the deterministic part as $\hat{\mathbf{Y}}_{T+j}$

sum that have non-matching values of i . The variance of the forecast error is thus a sum of (non-negative) variances of the orthogonalized components.

When we're looking at the observed data, however, the sum can include both positive and negative quantities. Thus, it's quite possible to have two or more components that seem to explain all (or even more than all) of the observed forecast error. Since this is true regardless of whether or not the shocks are correlated, the historical decomposition can be done with any mapping matrix F , not just orthogonalizing ones.

If we break down the final sum in (4.2), we get the following procedure for computing the historical decomposition:

Step 1. For each time period $t = T + 1, \dots, T + j$, transform the VAR residuals ε_t to the structural residuals η_t by $\eta_t = F^{-1}\varepsilon_t$.

Step 2. For each $i = 1, \dots, m$, and each time period $t = T + 1, \dots, T + j$, multiply column i of F (that is, $Fe(i)$) by η_{it} (that is, $e(i)'\eta_t$) to get the contribution of η_{it} to ε_t . Call that $\varepsilon_t^{(i)}$. Note that $\sum_{i=1}^m \varepsilon_t^{(i)} = \varepsilon_t$.

Step 3. For each $i = 1, \dots, m$, forecast the model from $T + 1$ to $T + j$, adding the $\varepsilon_t^{(i)}$ as shocks. Subtracting the base forecast from that will give the cumulative contributions of structural component i .

With RATS, this can all be done easily with the instruction **HISTORY**. It takes as input the estimated VAR model (which includes its residuals), and the F matrix itself (with the **FACTOR** option), or information required to compute it, such as the covariance matrix of residuals (with the **CV** option). The output is an $(m + 1) \times m$ matrix of series, where component $1, i$ is the base forecast for dependent variable i and component $j + 1, i$ is the cumulated effect of structural shock j to dependent variable i . With the option **ADD**, the base forecast isn't subtracted off in step 3. That can be convenient for presenting the results graphically.

This is part of Example 4.1. It computes an historical decomposition, and builds an estimate of the "potential" GDP by adding the base forecast and the cumulative results of two "supply" shocks.² It computes an estimate of the output gap by subtracting the potential GDP series from the actual.

²GDP is the second series in the model; the supply shocks are the first two shocks in the factorization. Thus `history(1, 2)` is the base forecast for GDP, `history(2, 2)` is the historical effect of the first shock and `history(3, 2)` is the historical effect of the second.

```

history(results=history,model=canmodel,$
        factor=bfactor,from=hstart,to=hend)
set potgdp = history(1,2)+history(2,2)+history(3,2)
graph(footer="Potential and Actual GDP") 2
# potgdp
# logcangdp
set gap = potgdp-logcangdp
graph(footer="Output Gap")
# gap

```

4.2 Counterfactual Simulations

The historical decomposition is a special case of a *counterfactual simulation*. The actual data can be recreated by adding the base forecast to the sum of contributions of *all* the components. If we omit some of those components, we're seeing what data would have been generated if some (linear combinations) of the residuals had been zero rather than what was actually observed.

The tool for doing more general counterfactual simulations is **FORECAST** with the option **PATHS**. With the **PATHS** option, **FORECAST** takes as input an additional **VECT[SERIES]** which has the paths of shocks to add in during the forecast period. Note that these have to be non-orthogonal shocks. If the restrictions on the residuals are in a transformed space (like orthogonalized shocks), these have to be transformed back into the space for non-orthogonal shocks. If ε are the VAR residuals and η are the structural residuals, then $\varepsilon_t = F\eta_t$ and $\eta_t = F^{-1}\varepsilon_t$. The general procedure is:

1. Transform the VAR residuals into structural residuals.
2. Make the desired changes (zeroing, for instance) to those.
3. Transform back into the space for equation shocks.
4. Use **FORECAST(PATHS)** inputting the **VECT[SERIES]** of shocks computed in step 3.

It's possible to combine several of these steps into a single matrix calculation. For instance, the code using this technique for computing potential GDP with the model is:

```

compute supplymask=||1.0,1.0,0.0,0.0,0.0,0.0||
compute supplyonly=bfactor*%diag(supplymask)*inv(bfactor)
dec vect[series] supplyshocks(6)
do t=hstart,hend
    compute %pt(supplyshocks,t,supplyonly*%xt(varresids,t))
end do t
forecast(paths,from=hstart,to=hend,model=canmodel,$
        results=withshocks)
# supplyshocks
set potgdp = withshocks(2)
set gap    = potgdp-logcangdp

```

Reading its factors from right to left, the `SUPPLYONLY` matrix transforms the VAR residuals into orthogonalized space, zeros out all but the two supply shocks, then transforms back into standard residual space. This uses `%PT` to break this up into six separate shock series for input into **FORECAST**.

4.3 Error Bands

The same type of Monte Carlo integration used for impulse responses can be used for error bands for the historical decomposition. However, (4.1) only holds when the residual vectors ε_t match with the VAR coefficients. When we draw a new set of coefficients, we also need to recompute residuals. That can be done most easily with the **FORECAST** instruction with the `STATIC` and `ERRORS` options. The following computes the residuals (first instruction) and the base forecasts (second instruction). Because **HISTORY** uses the internal copy of the residuals saved with the model when it's estimated, the historical decomposition needs to be computed using **FORECAST** with `PATHS`. This same basic idea can also be used to do counterfactual simulations where residuals generated using one set of estimated equations is combined with a different set of equations.

```
dec vect[series] v(6)
forecast(model=canmodel, static, from=hstart, to=hend, errors=u)
forecast(model=canmodel, from=hstart, to=hend, results=base)
compute ff=inv(%decomp(sigmad))
do t=hstart, hend
    compute %pt(v, t, ff*%xt(u, t))
end do t
forecast(model=canmodel, from=hstart, to=hend, results=baseplus, paths)
# v
```

Example 4.1 Historical Decomposition

This computes an estimate of potential GDP using the instruction **HISTORY** and a second more direct calculation of the counterfactual simulation, zeroing out all but the “supply shocks”. This uses a structural model (from Chapter 5).

```
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 logcandefl logcanm1
lags 1 to 4
det constant
end(system)
*
estimate(noprint,resids=varresids)
compute hstart=%regstart(),hend=%regend()
*
dec frml[rect] afrml bfrml
nonlin 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=0.0
*
cvmodel(b=bfrml,method=bfgs,pmethod=genetic,piters=50,$
        factor=bfactor) %sigma
*
* Method 1 - Using HISTORY
*
history(results=history,model=canmodel,$
        factor=bfactor,from=hstart,to=hend)
set potgdp = history(1,2)+history(2,2)+history(3,2)
graph(footer="Potential and Actual GDP") 2
# potgdp
# logcangdp
set gap = potgdp-logcangdp
graph(footer="Output Gap")
# gap
*
* Method 2 - Using FORECAST with PATHS
*
compute supplymask=||1.0,1.0,0.0,0.0,0.0,0.0||
compute supplyonly=bfactor*%diag(supplymask)*inv(bfactor)
dec vect[series] supplyshocks(6)
do t=hstart,hend
    compute %pt(supplyshocks,t,supplyonly*%xt(varresids,t))
end do t
*
* FORECAST with PATHS will now give the variables forecast with the
* addition of the supply shocks. withshocks(2) will be potential
* Canadian GDP.
*
forecast(paths,from=hstart,to=hend,model=canmodel,results=withshocks)
# supplyshocks
set potgdp = withshocks(2)
set gap    = potgdp-logcangdp

```

Structural VAR's

To this point, we've looked at two types of shocks: unit shocks and shocks orthogonalized by Cholesky factorization. Unit shocks have the problem of not being "typical" in size given the data and of ignoring the often rather strong correlation among the residuals. The main objection to the Cholesky factorization is that it identifies (in the interpretive sense, not statistical) the first shock as being "the" shock to the first variable. While this might not be unreasonable in a case like the "rmpy" model, where a Wold ordering is somewhat defensible, it clearly is dubious if you have several interest rates or asset prices, or variables which are involved in accounting identities (such as consumption and income). In this chapter, we'll look at alternative ways of interpreting the contemporaneous correlations.

5.1 Eigen Factorizations

Besides the Cholesky factorization, there are other "mechanical" procedures for producing a factorization. The most important of these is the eigen decomposition. If $\Sigma = P\Lambda P'$ is an eigen decomposition of Σ , then $F = P\Lambda^{1/2}$ makes a factorization. The η_t end up being the principal components of the ε_t process. This is most likely to work well when the set of variables are fairly similar: all asset prices or all interest rates, for instance. On the other hand, if you have a collection of major macro variables, you run into the standard problem with applying principal components to such variable sets: interpreting linear combinations like $3r - 5m + 2y$.

The factor for the eigenvalue decomposition can be computed with:

```
eigen(scale) %sigma * eigfact
```

EIGFACT is the matrix of (column) eigenvectors with each column scaled to have squared norm equal to its eigenvalue, which is the F matrix described above. The columns are sorted in decreasing order, so the largest eigenvalue (first principal component) is in column one. The following is taken from Example 5.1. Note that this uses the options for relabeling the shocks as PC1, PC2 and PC3. By default, the shocks are assumed to correspond to the variables, so the shocks would be labeled (in this case), GS1, GS3 and GS10, even though they're not constructed based upon any ordering for the endogenous variables.

Table 5.1: Pct Variance Explained by First PC

Step	1 Year	3 Year	10 Year
1	93.843	97.051	80.490
2	96.395	97.677	81.172
3	97.011	97.891	83.172
6	97.424	98.097	84.209
12	97.223	98.447	85.251
24	93.715	95.738	84.128
48	88.390	88.845	78.478

```
@varirf(model=ratevar, factor=eigfact, page=byshock, steps=48, $
  shocks=||"PC1", "PC2", "PC3"||, $
  variables=||"1 Year", "3 Year", "10 Year"||)
errors(model=ratevar, steps=48, factor=eigfact, $
  labels=||"PC1", "PC2", "PC3"||, results=fevd)
```

That example also includes code for generating a report with the error decomposition at selected horizons. In this case, the only interesting shock is really the first PC, which produces Table 5.1.

Principal components analysis is the simplest case of more general factor analysis. Factor analysis for the contemporaneous modeling in the VAR can be done as a special case of structural VAR's.

5.2 Generalized Impulse Responses

Generalized impulse responses (Pesaran & Shin (1998)) are an attempt to avoid the difficulties of identifying orthogonal shocks in VAR models. This is not a particularly complicated idea; in fact, all these do is compute the shocks with each variable in turn being first in a Cholesky order. These can be done quite easily using RATS: the following will give you the GIR, where the “factor” matrix is computed by dividing each column in %SIGMA by the square root of its diagonal element.

```
compute f=%ddivide(%sigma,%sqrt(%xdiag(%sigma)))
impulse(factor=f,other options) etc.
```

However, while these can, quite easily, be done in RATS, we do not recommend them. While coming up with an orthogonalizing model can, in practice, be somewhat difficult, it is a necessary step. A set of responses to highly correlated shocks are almost impossible to interpret sensibly. For instance, you can't run an **ERRORS** instruction to assess the economic significance of the responses, since that requires an orthogonal decomposition of the covariance matrix.

If you have a referee insisting that you do GIR, and you need error bands, you can use a **FUNCTION** to calculate the generalized factor for @MCVARDODRAWS:

```

function GIRFactor sigma model
type rect GIRFactor
type symm sigma
type model model
*
compute GIRFactor=%ddivide(%sigma,%sqrt(%xdiag(%sigma)))
end
*
@MCVARDoDraws(ffunction=GIRFactor,other options)

```

A related calculation is the Generalized FEVD—an example of which is the generalized spillover index from Diebold & Yilmaz (2012). Because the GIR “factor” isn’t actually a factor, putting it into an **ERRORS** instruction produces a decomposition of something other than the forecast error variances of the series. Note that this is a somewhat contrived measure, which may be useful in some situations but not in others.¹ The **RESULTS** option for **ERRORS** saves the decomposition (in decimals) in a `RECT[SERIES].GFEVD(i,j)` is a series running from entries 1 to `NSTEPS` with, at each step, the fraction of series i explained by shock j . The “cross-section” of that at entry `NSTEPS` is pulled with `%XT(GFEVD,NSTEPS)`, returning an $m \times m$ array with variables in the rows and shocks in the columns. The various spillover measures will be sums of non-diagonal elements in this matrix. The rows represent the target variable, and the columns represent the shock variable. So the following, which sum either a row or column and subtract the diagonal, will give the spillover to and spillover from each variable.

```

dec vect tovar(%nvar) fromvar(%nvar)
ewise tovar(i)=%sum(%xrow(gfevdx,i))-gfevdx(i,i)
ewise fromvar(i)=%sum(%xcol(gfevdx,i))-gfevdx(i,i)

```

5.3 Parametric Structural VAR's

There are two main alternatives to the mechanical procedures that we’ve seen so far. One, proposed independently by Sims (1986) and Bernanke (1986), is known as structural VAR modeling (SVAR for short). In this, a full parametric model of the factorization is proposed:

$$A(\theta)\varepsilon_t = B(\theta)\eta_t \quad (5.1)$$

with $E\eta_t\eta_t' = I$. In many cases, either A or B is absent (or, more correctly, the identity). Depending upon what’s present, these can be called, for short,

¹It partitions a value which has no direct interest. For instance, if you have a situation where a pair of variables have the property that the *second one* in an ordering explains more of the variance than the first, then a calculation where no variable is ever “second” will completely miss that. This should really only be applied to situations with relatively similar values for (for instance) different countries or firms.

A, B and A-B models. “A” models describe the construction of the shocks, and correspond roughly to a standard simultaneous equations model. “B” models describe how the shocks load onto the variables in the system. “A-B” models tend to be primarily “A” models with some free parameters in the “B” part to deal with constructed shocks that can’t be assumed to be uncorrelated.

Another way to write these models is as

$$\Sigma = \mathbf{G}(\theta)$$

where $\Sigma = E\varepsilon_t'\varepsilon_t$. For the general model:

$$\mathbf{G}(\theta) = \mathbf{A}(\theta)^{-1}\mathbf{B}(\theta)\mathbf{B}(\theta)'\mathbf{A}(\theta)^{\prime-1}$$

An $m \times m$ covariance matrix like Σ has $m(m+1)/2$ distinct elements, so a well-constructed SVAR can have no more than that number of elements in θ . If there are *exactly* that many, the model will (by the order condition) be just identified; if there are fewer, it will be overidentified. The latter means that the model has testable implications.

This is the model used by Sims in a number of papers (it adds unemployment and investment to rmpy):

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & 0 & a_{36} \\ a_{41} & 0 & a_{43} & a_{44} & 0 & a_{46} \\ a_{51} & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & a_{66} \end{bmatrix} \begin{bmatrix} u_R \\ u_M \\ u_Y \\ u_P \\ u_U \\ u_I \end{bmatrix} = \begin{bmatrix} v_{MS} \\ v_{MD} \\ v_Y \\ v_P \\ v_U \\ v_I \end{bmatrix} \quad (5.2)$$

Notice that the first two orthogonal shocks aren’t identified as R and M shocks: they’re money supply and money demand. And note that there is nothing in the model which will force these to behave the way that money supply and money demand shocks might be expected to behave, and, in fact, the statistical identification of those first two equations turns out to be tenuous. The other four are less problematic in practice, though even there, “identifying” the shocks as Y , P , U (unemployment) and I (investment) rather than 3, 4, 5 and 6 is a leap of faith until you see their behavior.

This has 19 free parameters, while the covariance matrix has 21, so it’s overidentified by 2.

As long as there is no interaction between the θ and the lag coefficients in the VAR, the result in formula (B.3) shows that the likelihood maximizer for the lag coefficients is just OLS equation by equation, regardless of the model used for the precision (inverse variance) $\mathbf{H} = \mathbf{G}(\theta)^{-1}$. This is true even if the model is overidentified, and so can’t hit all possible covariance matrices. The maximum likelihood estimates for θ (assuming Normal residuals) can be done

by computing the OLS estimates $\hat{\beta}$ and then maximizing:

$$|\mathbf{G}(\theta)|^{-T/2} \exp \left(-\frac{1}{2} \text{trace} [\mathbf{G}(\theta)]^{-1} (T \times \Sigma(\hat{\mathbf{B}})) \right) \quad (5.3)$$

This has, as sufficient statistics, just T and $\Sigma(\hat{\mathbf{B}})$, that is, the number of observations and covariance matrix of the VAR residuals.

In practice, the structural model typically normalizes on one coefficient per row in an A model or one per column in a B model (both in an A-B model). This leaves the scale of the η_t process free while still constraining its components to be contemporaneously uncorrelated (just not variance one). With this, the model becomes

$$\begin{aligned} \mathbf{A}(\theta)\varepsilon_t &= \mathbf{B}(\theta)\eta_t \\ E\eta_t\eta_t' &= \mathbf{\Lambda} \end{aligned} \quad (5.4)$$

where $\mathbf{\Lambda}$ is diagonal. If we define

$$\mathbf{C}(\theta) = \mathbf{B}(\theta)^{-1} \mathbf{A}(\theta)$$

then \mathbf{C} is designed to diagonalize the covariance matrix Σ . Since

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

then

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

so the log likelihood is

$$|\mathbf{\Lambda}|^{-T/2} |\mathbf{C}(\theta)|^T \text{trace} \left\{ \mathbf{\Lambda}^{-1} \times \mathbf{C}(\theta)' (T \times \Sigma(\hat{\mathbf{B}})) \mathbf{C}(\theta) \right\} \quad (5.5)$$

Since $\mathbf{\Lambda}$ is diagonal,

$$|\mathbf{\Lambda}| = \prod_{i=1}^m \lambda_{ii}$$

and

$$\text{trace} \left\{ \mathbf{\Lambda}^{-1} \times \mathbf{C}(\theta)' (T \times \Sigma(\hat{\mathbf{B}})) \mathbf{C}(\theta) \right\} = \sum_{i=1}^m \lambda_{ii}^{-1} \left(\mathbf{C}(\theta)' (T \times \Sigma(\hat{\mathbf{B}})) \mathbf{C}(\theta) \right)_{ii}$$

The λ_{ii} variances can be concentrated out easily since their maximizers are just

$$\lambda_{ii} = \left(\mathbf{C}(\theta)' (\Sigma(\hat{\mathbf{B}})) \mathbf{C}(\theta) \right)_{ii}$$

as the $T/2$ factors cancel out. The estimators for the shock variances are what we get by applying the model's diagonalizing matrix \mathbf{C} to the least squares covariance matrix $\Sigma(\hat{\mathbf{B}})$.

The two methods (normalizing the model and concentrating out the variance rather than freely estimating the non-zero parameters and using an identity covariance matrix) give identical log likelihoods. Using normalized equations fixes the sign of the η_t components and so is generally easier to interpret. With a full parameterization, the sign of any given row for an A model or column for a B model can be switched without affecting the likelihood. Thus a “price” shock could, for instance, be estimated as having an impact of either sign on price. Since the likelihood is unaffected by changing the signs, you could always correct that after the estimation is complete so you get exactly the same “factor matrix” and thus shocks. However, there is a subtle difference between the two ways of handling the diagonal when it comes to computing error bands (Section 5.7).

5.4 Identification

Statistical identification of SVAR's is a tricky business. Other than the crude counting rule for whether the model as a whole is identified, there is no “order condition” that can be applied to individual equations. Identification comes from the restriction that the resulting η_t process is orthonormal. As we can see from the 3×3 example (2.4), it's quite possible to have global identification fail in a way that would be highly unlikely in standard simultaneous equations. If a *simultaneous equations* model is unidentified, it is generally the case that the model is also *locally* unidentified, that is, its covariance matrix isn't full rank. If it's full rank only because of sampling error (for instance, because the order condition is fulfilled due to an irrelevant variable), you'll usually notice that there are some quite large standard errors. However, when you have multiple modes in an SVAR, you don't usually have one of those obvious signs of a problem: the second derivatives around each mode will clearly be full rank.

This major hole in the theory was finally plugged in Rubio-Ramirez et al. (2010). While this is a very technical paper designed to apply to a wide range of situations, it provides as special cases a couple of easy-to-verify rules:

- In an “A” model identified with zero restrictions, the SVAR is identified if and only if exactly one *row* has j zeros for each j from 0 to $m - 1$. Equivalently, if and only if exactly one row has j non-zero elements for each j from 1 to m .
- In a “B” model identified with zero restrictions, the SVAR is identified if and only if exactly one *column* has j zeros for each j from 0 to $m - 1$ with the analogous condition for non-zeros.

For example, the following pattern of zeros and non-zeros:

$$\begin{bmatrix} \bullet & 0 & 0 & \bullet \\ \bullet & \bullet & \bullet & 0 \\ \bullet & \bullet & 0 & \bullet \\ \bullet & \bullet & 0 & 0 \end{bmatrix}$$

will not properly identify an “A” model (rows have 2-1-1-2 zeros, not 0-1-2-3 in some order), but will identify a “B” model (columns have 0-1-3-2). There is nothing “obviously” wrong with the A model (like two rows being identical); you’ll just have multiple modes with identical values for the likelihood.

5.5 Estimation

Parametric structural VAR's are estimated using the instruction **CVMODEL**. This takes as input the covariance matrix of residuals and formulas which give A, B or both. The syntax of this has changed a bit over the years; the easiest way to input A and/or B now is to use the **A** and **B** options.²

The first example of **CVMODEL** is the Sims model (Example 5.2 and equation (5.2)). With normalizations, this can be set up as follows:

```
nonlin(parmset=svar) a12 a21 a23 a24 a31 a36 a41 a43 a46 $
      a51 a53 a54 a56
*
dec frml[rect] afrml
frml afrml = ||1.0,a12,0.0,0.0,0.0,0.0|$
              a21,1.0,a23,a24,0.0,0.0|$
              a31,0.0,1.0,0.0,0.0,a36|$
              a41,0.0,a43,1.0,0.0,a46|$
              a51,0.0,a53,a54,1.0,a56|$
              0.0,0.0,0.0,0.0,0.0,1.0||
```

The **A** and **B** functions need to be defined as **FRML[RECT]**, and written in terms of the free parameters. Note that you need to do the **NONLIN** instruction first to make sure the free parameters are defined before you can use them in the **FRML** instruction. If you make changes to the **FRML** (adding or removing free parameters), make sure you make the required changes to the **NONLIN** list as well.

When this model is analyzed in Sims & Zha (1999), they do not use the normalized form shown here. Instead, **a11**, **a22**, etc. variables are put on the diagonal. The model can be estimated in that form by adding the extra six parameters to the parameter set, and including the option **DMATRIX=IDENTITY** to the **CVMODEL** instruction—we’ll show the adjustment for that a bit later. The default is **DMATRIX=CONCENTRATED** which does the concentrated likelihood with rows normalized. Note that while the concentrated form needs one

²There is also a **V** option if it’s simpler to provide a formula for the **G** matrix itself.

element per row set to one (in an A model like this), it doesn't *have* to be the diagonal. In fact, if the first two are supposed to be money supply and money demand, the more standard way of writing them would have a normalization with the M coefficients being 1 in both. However, that turns out to work poorly here. The one problem with pegging a coefficient is that you want it to be a coefficient which is not (in reality) zero. In this case, the first equation is really more of an R shock than anything involving money.

The estimation is done with:

```
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0
cvmodel (a=afrml, parmset=svar, method=bfgs, factor=fsvar) %sigma
```

This initializes the free parameters and estimates the model. For a normalized model, starting with the off-diagonal elements at zero usually works fairly well. The `A=AFRML` option and `%SIGMA` parameter are the inputs. This produces the estimation output and allows you to retrieve the factor matrix for the model with the `FACTOR` option.³ Because the model is overidentified, this is not an exact factor of `%SIGMA`, but should be close if the model passed the test for the overidentifying restrictions. The output from the `CVMODEL` is Table 5.2. The test of the overidentifying restrictions passes fairly easily. The signs of `A12` and `A21` are correct for the desired interpretation of those two equations in the SVAR. However, one thing to note is that `A12` in particular isn't strongly signed as negative, given its estimated standard error. This will prove to be a problem when trying to do error bands.

The following does the impulse responses and error decomposition at the point estimates for the model.

```
compute shocklabels=||"MS","MD","Y","P","U","I"||
compute varlabels=||"R","M","Y","P","U","I"||
*
@varirf(model=varmodel, steps=24, page=byvariable, $
    shocks=shocklabels, var=varlabels, factor=fsvar)
errors(model=varmodel, steps=24, labels=shocklabels, factor=fsvar)
```

Part of the decomposition for GNP is shown in Table 5.3. Note that because of the structure of the model, all shocks except `U` have an immediate impact. If this were a Cholesky factor, the `P`, `U` and `I` shocks (variables after `Y` in the ordering) would explain zero at step 1.

Next, we show how to modify the estimation to handle an unnormalized model. Again, this should give the same likelihood as the normalized model. We have the same `PARMSET` as above for the off-diagonal, and, for ease of switching between the two forms, create a separate `PARMSET` for the diagonals.⁴ We also use a slightly different name for the `FRML`:

³You can also fetch the estimates of the shock variances with the `DVECTOR` option.

⁴See Section 5.8.1 for more on the use of `PARMSET`s.

Table 5.2: Output from CVMODEL

Covariance Model-Concentrated Likelihood - Estimation by BFGS					
Convergence in 44 Iterations. Final criterion was 0.0000007 <= 0.0000100					
Observations		163			
Log Likelihood		3633.4674			
Log Likelihood Unrestricted		3633.8726			
Chi-Squared(2)		0.8104			
Significance Level		0.6668560			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	A12	-3.291	1.418	-2.320	0.020
2.	A21	1.527	0.509	2.999	0.003
3.	A23	-0.795	0.318	-2.503	0.012
4.	A24	-1.141	0.690	-1.654	0.098
5.	A31	0.576	0.233	2.471	0.013
6.	A36	-0.249	0.039	-6.353	0.000
7.	A41	0.252	0.170	1.478	0.139
8.	A43	-0.093	0.097	-0.957	0.338
9.	A46	-0.031	0.024	-1.301	0.193
10.	A51	0.091	0.026	3.481	0.000
11.	A53	0.168	0.021	8.127	0.000
12.	A54	0.065	0.034	1.921	0.055
13.	A56	0.016	0.010	1.620	0.105

Table 5.3: Decomposition of Variance for GNP

Step	Std Error	MS	MD	Y	P	U	I
1	0.00840391	1.929	9.727	64.897	2.283	0.000	21.164
2	0.01287890	2.831	9.519	63.967	1.192	0.411	22.080
3	0.01694349	7.247	12.535	59.560	0.996	0.274	19.388
4	0.01973139	10.080	14.876	55.992	1.894	0.516	16.642
5	0.02185868	12.259	16.595	51.026	3.694	1.653	14.774
6	0.02369966	13.322	19.061	45.097	6.774	3.008	12.738
7	0.02550188	13.214	21.349	39.193	10.632	4.610	11.001
8	0.02716060	12.342	22.840	34.553	14.620	5.893	9.751
9	0.02870734	11.233	23.708	30.955	18.317	6.991	8.796
10	0.03009602	10.233	24.147	28.204	21.442	7.915	8.059
11	0.03129063	9.475	24.186	26.113	23.944	8.798	7.484
12	0.03230403	8.931	23.971	24.505	25.909	9.649	7.035

```

nonlin(parmset=diags) a11 a22 a33 a44 a55 a66
*
dec frml[rect] afrmld
frml afrmld = ||a11,a12,0.0,0.0,0.0,0.0|$
               a21,a22,a23,a24,0.0,0.0|$
               a31,0.0,a33,0.0,0.0,a36|$
               a41,0.0,a43,a44,0.0,a46|$
               a51,0.0,a53,a54,a55,a56|$
               0.0,0.0,0.0,0.0,0.0,a66||

```

The off-diagonal elements can be initialized to zero as before, but that would be a bad (in fact, invalid) choice for the diagonals, since it would create an undefined likelihood. Instead, we start with square roots of the diagonal elements of %SIGMA:

```

compute a11=sqrt(%sigma(1,1)),a22=sqrt(%sigma(2,2)), $
        a33=sqrt(%sigma(3,3)),a44=sqrt(%sigma(4,4)), $
        a55=sqrt(%sigma(5,5)),a66=sqrt(%sigma(6,6))
*
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0

```

The **CVMODEL** is adjusted by changing the **A** option to point to the new **FRML**, the **PARMSET** to include both parts of the overall **PARMSET**, and the **DMATRIX** option to now take the value **IDENTITY**. We also (in this case) increase the number of iterations: this has more free parameters and so would be expected to (and does) take more iterations to converge.

```

cvmmodel(a=afrmld,parmset=diags+svar,dmatrix=identity,itors=500,$
        method=bfgs,factor=fsvar) %sigma

```

The output is in Table 5.4. As expected, the log likelihood matches the normalized value. As noted earlier, the signs of any single row aren't identified in this model; however, here the diagonals are all positive so the factor matrix will match exactly with the normalized form. In Table 5.2, we saw that A_{12} was negative (as would be expected if the design of the model was correct). Now, A_{12} is more strongly negative, but what's happened here is that the "sign problem" has been moved over to A_{11} .

Example 5.3 shows an A-B model from Bernanke & Mihov (1998). This is a simplified version of the analysis in that paper, looking only at a single time period rather than multiple ranges and lag lengths.

$$\begin{bmatrix} 1 & 0 & \alpha \\ 1 & -1 & -\beta \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_{TR} \\ u_{NBR} \\ u_{FF} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \varphi_d & \varphi_b & 1 \end{bmatrix} \begin{bmatrix} v^d \\ v^b \\ v^s \end{bmatrix}$$

The three observables are, in order, total reserves, non-borrowed reserves and the Federal funds rate. The three orthogonal shocks are interpreted as disturbances to money demand, borrowing and money supply. As written, the model

Table 5.4: Output from CVMODEL for Unnormalized Model

Covariance Model-Likelihood - Estimation by BFGS					
Convergence in 152 Iterations. Final criterion was 0.0000090 <= 0.0000100					
Observations		163			
Log Likelihood		3633.467			
Log Likelihood Unrestricted		3633.873			
Chi-Squared(2)		0.810			
Significance Level		0.667			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	A11	56.843	22.802	2.493	0.013
2.	A22	83.309	26.928	3.094	0.002
3.	A33	118.635	12.821	9.253	0.000
4.	A44	196.243	21.381	9.178	0.000
5.	A55	501.472	26.338	19.040	0.000
6.	A66	49.796	2.478	20.098	0.000
7.	A12	-187.075	14.556	-12.852	0.000
8.	A21	127.238	13.539	9.398	0.000
9.	A23	-66.225	15.833	-4.183	0.000
10.	A24	-95.075	41.356	-2.299	0.022
11.	A31	68.327	24.221	2.821	0.005
12.	A36	-29.587	3.935	-7.519	0.000
13.	A41	49.405	31.061	1.591	0.112
14.	A43	-18.315	18.096	-1.012	0.312
15.	A46	-6.130	4.600	-1.332	0.183
16.	A51	45.560	12.843	3.548	0.000
17.	A53	84.362	10.755	7.844	0.000
18.	A54	32.842	14.942	2.198	0.028
19.	A56	7.827	4.235	1.848	0.065

is underidentified by one. The authors examine a number of different restrictions to identify the model (such as $\alpha = 0$ and $\varphi_b = -\varphi_d$). Note that this model doesn't have the unit diagonal on the left which allows the usual "identification" of shocks with their corresponding variables. The non-trivial matrix on the right means that while the model is assumed to produce orthogonal v^d and v^b , barring a restriction on the φ the u_{NBR} from the third equation can have *any* correlation with v^d and v^b as v^s will be whatever part of u_{NBR} is uncorrelated with the other v 's. In some sense, this is combining both the SVAR approach and the Cholesky: a tightly restricted structural model which is not assumed to produce fully orthogonal residuals, with some relatively unstructured model to deal with the correlation that's allowed.

The general setup is:


```

dec frml[rect] ab phi
nonlin alpha beta phid phib
frml ab = ||1.0 , 0.0 , alpha|$
          1.0 ,-1.0 ,-beta|$
          0.0 , 1.0 , 0.0||
frml phi = ||1.0 , 0.0 , 0.0|$
            0.0 , 1.0, 0.0|$
            phid,phib, 1.0||

```

This defines the full set of parameters, and the general formulas for the matrices. The individual models are estimated by adding restrictions to the parameter sets. Thus:

```

compute alpha=0.00,beta=.05
*
nonlin alpha beta phid=1 phib=-1
cvmodel(a=ab,b=phi) moneyvcv

```

which estimates with a constraint of the φ loadings (this overidentifies the model, since it put on two constraints) and

```

nonlin alpha=0 beta phid phib
compute phid=0.8,phib=-0.8,beta=0.05
cvmodel(a=ab,b=phi) moneyvcv

```

which estimates the just-identified model.

5.6 Structural Residuals

The *structural residuals* are the observed values of η_t . If the model produces a factor F of the covariance matrix, then $F\eta_t = \varepsilon_t$ or $\eta_t = F^{-1}\varepsilon_t$. In general, each component of η_t will be a linear combination of the full corresponding vector of VAR residuals ε_t . The easiest way to generate this is with the procedure `@StructResids`. This takes as input the factor matrix and the `VECT[SERIES]` of VAR residuals, and produces a `VECT[SERIES]` of the structural residuals:

```
@StructResids(factor=factor) resids / sresids
```

The `/` is taking the place of an entry range in case you want to control that.

5.7 Error Bands

The calculation of error bands for structural models for *just-identified* models is similar to the techniques from Chapter 3. However, while the calculations may be nominally similar, there are some very special problems that can arise. As a simple example, take the case of the eigen factorizations from Section 5.1. This is a “just-identified” model where the factor can be calculated using the **EIGEN** instruction. Since there are no restrictions being put on the covariance matrix,

this can (theoretically) be handled by drawing the covariance matrix from the unconditional inverse Wishart distribution, computing the eigen factorization.⁵ The problem with this is that the first (or any other) principal component may not, in practice, represent the same “shock” from one draw to the next (whether done using bootstrapping or simulation). In the specific case of Example 5.1, the first PC is probably so dominant that there won't be any problem with it, but in other applications, you could have two principal components which have relatively similar eigenvalues in the original sample, and minor changes to the covariance matrix could cause them to switch positions.

Something similar can (and does) happen with the Sims model (5.2), at least with the benchmark dataset. The problem there is with the money supply and money demand shocks. As pointed out above, in Table 5.2 the A_{12} parameter isn't *that* strongly negative. If the approximation to the standard error is accurate, there is roughly a 1 in 50 chance that it could be positive. If you do 10000 draws, that means somewhere around 200 of them will have the response of money to a “money supply” shock that's negative.

5.7.1 Bootstrapping

Bootstrapping with SVAR's is relatively simple. Because the residuals incorporate the contemporaneous relationships, those will be included in the bootstrapped data as well. It doesn't matter whether the model is just-identified or over-identified. Just generate bootstrapped data using the techniques of Section 3.2 and, inside the draw loop, put whatever you need to estimate the SVAR on those data rather than the original data. In addition to possibility described above of shocks “switching positions”, you also have to be careful about convergence of estimates if you are using **CVMODEL**, as you won't be able to put the same level of attention into estimating the model for the bootstrapped data that you would with just the original data.

5.7.2 Monte Carlo: Deriving the Posterior Distribution

If a structural model is just-identified, it can (theoretically) be handled by drawing the covariance matrix unconditionally as shown in Section 3.3 and estimating the structural model from that. A number of early papers attempted to handle *overidentified* structural models in the same way, which was to just adapt the standard VAR Monte Carlo, drawing unrestricted Σ , then maximizing (5.3) at the drawn Σ to get θ . 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.

⁵The lag coefficients can be drawn straightforwardly once the draw for the covariance matrix has been done.

If we track through all the calculations isolating β in Appendix B, we see that conditioned on θ , the posterior for β will still be Normal:

$$\beta|y, X, \theta \sim N\left(\hat{\beta}, [\mathbf{G}(\theta)]^{-1} \otimes \sum x'_t x_t\right) \quad (5.6)$$

Continuing, we see that, this gives us:

$$p(y|X, \mathbf{B}, \theta) \propto |\mathbf{G}(\theta)|^{-(T-k)/2} \exp\left(-\frac{1}{2}\text{trace}[\mathbf{G}(\theta)]^{-1}(T \times \Sigma(\hat{\mathbf{B}}))\right) \times p(\beta|y, \theta) \quad (5.7)$$

The k in $|\mathbf{G}(\theta)|^{-(T-k)/2}$ in (5.7) doesn't show up in (5.3) because (5.3) is the result of maximizing (concentrating) β out, while (5.7) results from *integrating* β out.

If we look at (5.7), we see that we can evaluate the likelihood for any value of θ , 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 (5.7) rather than the concentrated log likelihood in (5.3).

It's probably not unreasonable to assume that the m diagonal elements of $\Lambda(\theta)$ in (5.5) 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 Λ (understanding that the parameters are the diagonal elements) and just refer to the parameters in the \mathbf{A} and \mathbf{B} matrices. 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 Λ is diagonal,

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

for any matrix \mathbf{V} . Let the prior be the rather uninformative

$$p(\Lambda, \theta) = |\Lambda|^{-\delta}$$

then defining

$$\mathbf{V}(\theta) = \mathbf{C}(\theta)(T \times \Sigma(\hat{\mathbf{B}}))\mathbf{C}(\theta)' \quad (5.8)$$

the posterior for Λ conditional on θ is

$$p(\Lambda|y, \theta) \propto \prod_{i=1}^m \left(\lambda_{ii}^{-(T-k)/2-\delta} \exp\left(-\frac{1}{2}\lambda_{ii}^{-1}\mathbf{V}(\theta)_{ii}\right) \right) \quad (5.9)$$

By inspection, each λ_{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 $1/2V_{ii}$. Aside from the somewhat odd degrees of freedom, this makes quite a bit of sense. Under the model, $C(\theta)$ is supposed to orthogonalize Σ , so $C(\theta)\Sigma(\hat{B})C(\theta)'$ should be nearly a diagonal matrix. The diagonal elements of that are used to “center” the distribution for the draws for the λ_{ii} .

We can now integrate out the Λ to get the posterior for θ . The λ_{ii} themselves are inverse gammas, so the integral is the reciprocal of the integrating constant from Appendix A.5. We can ignore the $\Gamma(a)$, since it doesn't depend upon θ and we need to re-introduce the $|C(\theta)|^{T-k}$, since it *does*, to get

$$p(\theta|y) \propto |C(\theta)|^{T-k} \exp \left(-\frac{T-k+2\delta+2}{2} \sum_{i=1}^m \log(V(\theta)_{ii}) \right) \quad (5.10)$$

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) (5.10) with **CVMODEL**, you have to

1. Create the parameter set θ
2. Define the **A** and/or **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 $\Sigma(\hat{B})$
4. Choose δ for the prior. The standard value here is $.5(m+1)$ which guarantees that the posterior will be integrable.

CVMODEL has three ways for handling the scale factors (the Λ). We've already discussed `DMATRIX=CONCENTRATED` and `DMATRIX=IDENTITY`. The one described above is `DMATRIX=MARGINALIZED`. The δ is input using the `PDF` option, and the k with the `DFC` option.

The procedure for doing error bands will be to somehow draw values of θ , then, given that, to draw the Λ using (5.9), and then to draw the lag coefficients using (5.6). The latter two steps are from well-known distributions. However, θ has a non-standard distribution, so we can't sample from it directly. Instead, there are three methods for doing inference on it:

1. Random walk M-H
2. Independence chain M-H
3. Importance sampling

We have a brief description of the first two in Appendix F. We'll examine the use of random walk Metropolis-Hastings in Section 5.7.4. We'll first look at importance sampling.

5.7.3 Monte Carlo with Importance Sampling

Importance sampling is discussed in the *RATS User's Guide*. It's very similar to independence chain M-H. The difference is that independence chain M-H overweights "desirable" θ (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 θ 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.⁶ If you can do importance sampling, you should prefer it over independence chain M-H in this situation. 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 θ). 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 data set for Example 5.4 is the one used in the `CVMODEL.RPF` example from the *RATS User's Guide* with a very slight modification to the model:

$$\begin{bmatrix} u_u \\ u_c \\ u_r \\ u_x \\ u_p \\ u_m \end{bmatrix} = \begin{bmatrix} 1 & 0 & \beta_{uf1} & 0 & 0 & 0 \\ \beta_{cr1} & 1 & \beta_{cf1} & 0 & 0 & 0 \\ 0 & 0 & 1 & \beta_{rf2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \beta_{pn2} \\ 0 & \beta_{mr2} & \beta_{mf1} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{r1} \\ v_{r2} \\ v_{f1} \\ v_{f2} \\ v_{n1} \\ v_{n2} \end{bmatrix}$$

⁶If we were using a Bayesian VAR, we would also no longer be able to do importance sampling, since the distribution for θ would depend upon a draw for the lag coefficients and thus would have to use M-H rather than importance sampling for θ . With a flat-prior VAR, the distribution of θ depends only upon the OLS covariance matrix of residuals.

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.

After reading the data, setting up and estimating the OLS VAR, we do the following, which get the factor of the system $(X'X)^{-1}$ matrix, the OLS coefficient matrix and the counts of coefficients per equation and number of variables:

```
compute fxx      =%decomp(%xx)
compute betaols=%modelgetcoeffs(canmodel)

compute ncoef=%nreg
compute nvar  =%nvar
```

The FRML for the B matrix is set up with:

```
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 pn2 mr2 mf1
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|$
           0.0,0.0,0.0,0.0,1.0,pn2|$
           0.0,mr2,mf1,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=pn2=mr2=mf1=0.0
```

Note again, that you need to define the free parameters first (typically on the **NONLIN** instruction) before you can use them in the FRML. We have defined the parameters descriptively as the dependent variable followed by the shock (for instance, UF1 as the loading of F1 on U). That's a convenient setup as it makes it easier to add or remove parameters.

With the marginalized diagonal matrix (with the recommended $\delta = (6 + 1)/2$), the model can be estimated with:

```
compute delta=3.5
cvmodel(b=bfrml,parmset=svar,dfc=ncoef,pdf=delta,$
        dmatrix=marginal,pmethod=genetic,piters=50,method=bfgs) %sigma
compute nfree=%nreg
```

The output from that is Table 5.5. The coefficients would be expected to be fairly similar to those done with maximum likelihood since the prior is fairly diffuse on the diagonal. Note that there is no test for overidentification, since this is not being estimated by maximum likelihood.

Table 5.5: CVMODEL output with marginalized diagonal

Covariance Model-Marginal Posterior - Estimation by BFGS					
Convergence in 15 Iterations. Final criterion was 0.0000042 <= 0.0000100					
Observations		100			
Function Value		-625.8364			
Prior DF		3.5000			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	UF1	0.1052	0.0749	1.4041	0.1603
2.	CR1	0.5006	0.1075	4.6568	0.0000
3.	CF1	-0.0750	0.0837	-0.8968	0.3698
4.	RF2	-0.0897	0.0397	-2.2560	0.0241
5.	PN2	0.1246	0.0492	2.5310	0.0114
6.	MR2	0.4266	0.2656	1.6062	0.1082
7.	MF1	-0.3322	0.1783	-1.8637	0.0624

For a model with a well-behaved likelihood, the following, or at least some minor adjustment to it, should generate a reasonable “importance” function:

```
compute [vector] thetamean=%parmspeek(svar)
compute fthetaxx=%decomp(%xx)
compute nu=10.0
```

THETAMEAN is the mean of the proposal, and is the maximizer for the posterior. FTHETAXX is the Cholesky factor of the covariance matrix of the estimates and will be used as the underlying covariance matrix for the draws. NU will be the degrees of freedom for a multivariate t draw. You can scale FTHETAXX (typically up) and adjust NU in either direction, though generally a low yield requires fatter tails so NU would typically be adjusted downwards. Importance sampling works best when the importance function is thicker in the tails than the density it's trying to match, and works very poorly if it is thinner—see the *User's Guide* for a more complete discussion of this.

Importance sample can be affected by the same “overflow” problem in the calculations that is described on page 165. The following saves the (log) maximum posterior density for fixing scale problems in the calculations.

```
compute scladjust=%funcval
```

After a few other preliminary instructions, we have the following, which is specific to importance sampling:

```
*
* This will hold the (relative) weights
*
set weights 1 ndraws = 0.0
```

Each draw will have its relative weight computed and saved into the series `WEIGHTS`. At the end, the actual weights will be computed by dividing this by the sum over all draws.

Inside the draw loop, we do the following:

```
compute thetadraw=thetamean+%ranmvt(fthetaxx,nu)
compute logqdensity=%ranlogkernel()
```

This draws a new θ from the multivariate t that we described earlier. We save the (kernel of the) density at that draw into `LOGQDENSITY`—we don't need all the integrating constants because we only need relative weights and the integrating constants are fixed across draws. The model is then evaluated at the draw using the same options as before (`DMATRIX=MARGINALIZED`, same `DFC` and `PDF`), but with `METHOD=EVALUATE` rather than the other estimation options:

```
compute %parmspoke(svar,thetadraw)
cvmmodel(parmset=svar,dfc=ncoef,pdf=delta,dmatrix=marginalized,$
method=evaluate,b=bfrml) %sigma
compute logpdensity=%funcval
```

The log posterior density is saved into `LOGPDENSITY`. A “safe-guarded” relative weight is computed with

```
compute weight =exp(logpdensity-scladjust-logqdensity)
compute weights(draw)=weight
```

The scales of the two densities (the kernel for the draw and the posterior) are completely different, so doing this without subtracting off `SCLADJUST` before taking the `exp` risks an over- or underflow. The point of subtracting `SCLADJUST` is to replace the posterior density (which could be a very large or very small number) with its ratio to the maximum posterior density. (`LOGQDENSITY` already has a standardized scale since it's the log of the kernel rather than the log of the actual density.)

For a B form model, the following computes the diagonals of (5.8) given the current values for θ and uses those to draw the Λ values from the gamma distribution in (5.9).⁷

```
compute b=bfrml(1)
compute vdiag=%mqformdiag(%sigma,tr(inv(b)))
ewise lambdadraw(i)=$
(%nobs/2.0)*vdiag(i)/%rangamma(.5*(%nobs-ncoef)+delta+1)
```

⁷`BFRML(1)` is needed in evaluating the B matrix because any type of `FRML` has to be evaluated at *some* entry (1 is as good as any) even though there is nothing about this one that's time-varying. For an A form model, you would use just A (the value of the A matrix formula) in place of `INV(B)` in computing `VDIAG`.

and this (again, specifically for a B model—an A model would need `INV(A)` in place of B) computes a factor for the implied covariance matrix:

```
compute fsigmad=b*%diag(%sqrt(lambdadraw))
```

The lag coefficients can be drawn using (5.6) with the rather familiar-looking:

```
compute betau=%ranmvkron(fsigmad,fxx)
compute betadraw=betaols+betau
compute %modelsetcoeffs(canmodel,betadraw)
```

and the impulse responses for the draw are computed as with simpler types of models:

```
impulse(model=canmodel,noprint,factor=fsigmad,steps=nsteps,$
    flatten=%responses(draw))
```

Outside the loop, we can compute the performance statistic on the importance sampler with:

```
sstats 1 ndraws weights>>sumwt weights^2>>sumwt2
compute effsize=100.0*sumwt^2/(ndraws*sumwt2)
disp "Effective sample size (percent)" *.## effsize
```

This converts the statistic into a percentage “yield”. With this model, and this importance function (the multivariate t with 10 degrees of freedom), the yield is fairly consistent in the 80-90% range, which is very good. If the t were a spot-on match for the posterior, we would get exactly 100% (the weight series would be constant), so this means the weights aren’t varying too much. By contrast, if we had a situation where 1 in 100 draws had a weight 100 times the typical draw (which is possible if the posterior has a direction in which it is quite a bit fatter than is predicted by the asymptotic distribution, even with the t tails for the importance function), then the yield will be around 4%. A number below 10%, especially if different runs have it dropping down closer to 1%, is a sign that your importance function isn’t working. A consistent low yield value usually can be fixed with fatter tails; a highly variable one usually is a sign that the asymptotic distribution has the wrong basic shape.

The graphs can be done with `@MCGRAPHIRF` (or you can use `@MCPROCESSIRF` to generate graphs with your own layout). With either, however, you need to include the `WEIGHTS` option to get the draws properly weighted. You may notice that it takes a bit longer to do the number-crunching in `@MCGRAPHIRF` than for a model of similar size and number of draws without the weights. In fact, for this model, it takes longer to “post-process” the draws than to do them in the first place. This is because calculating weighted quantiles is a time-consuming process which goes up more than linearly in the number of draws.

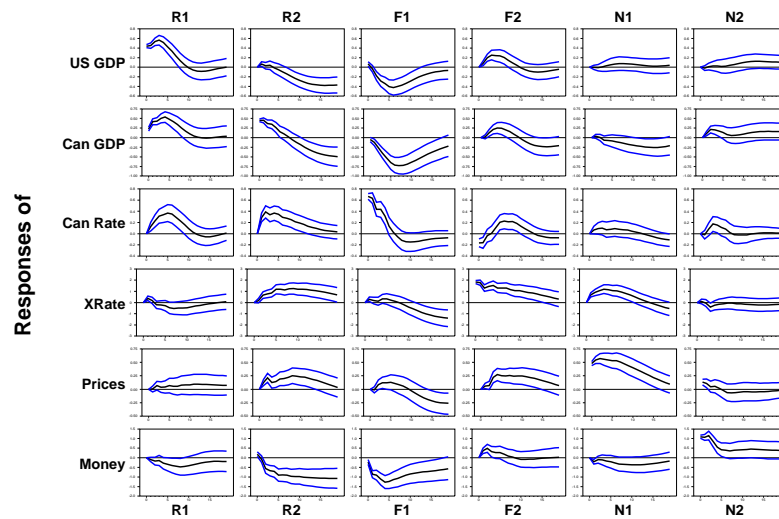


Figure 5.1: IRF's from Structural VAR, B Model

```
set weights 1 ndraws = weights/sumwt
*
@mcgraphirf(model=canmodel,weight=weights,$
  shocks=||"R1","R2","F1","F2","N1","N2"||,$
  variables=||"US GDP","Can GDP","Can Rate","XRate","Prices","Money"||)
```

The graph from one run can be seen in Figure 5.1. It appears that shocks can reasonably be interpreted as US and Canadian GDP for the two real shocks, interest and exchange rates (in that order) for the two financial and price and money (in that order) for the two nominal shocks. There's relatively little *contemporaneous* interaction (R1 hitting Canadian GDP being the most obvious exception) but quite a bit of dynamic interaction.

5.7.4 Monte Carlo with Random Walk Metropolis

The model above worked well with importance sampling with a fairly straightforward importance function. For many models, particularly those of modest size, that's likely to be the case. However, in some models, the asymptotic distribution from the estimation isn't a good match for the posterior. An example of this is the Sims model from Example 5.2 and (5.2). As we noted when we estimated that, the sign of the A_{11} in the unnormalized model isn't strongly positive. That suggests that normalizing on A_{11} may not be a good idea—there is some chance that it's close to zero. A possible fix to that would be to normalize on A_{12} instead, or to switch the order of the first two equations. However, A_{22} also only has a t statistic of around 3, so while somewhat better, it's still not the very strongly-signed value that we see in the diagonal elements for the other four shocks. If we look more carefully at the model, we can see where we could run into problems:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & 0 & a_{36} \\ a_{41} & 0 & a_{43} & a_{44} & 0 & a_{46} \\ a_{51} & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & a_{66} \end{bmatrix} \begin{bmatrix} u_R \\ u_M \\ u_Y \\ u_P \\ u_U \\ u_I \end{bmatrix} = \begin{bmatrix} v_{MS} \\ v_{MD} \\ v_Y \\ v_P \\ v_U \\ v_I \end{bmatrix} \quad (5.11)$$

The sixth equation defines v_I by construction. If we look at the fifth equation, it's the only one with u_U , so *it* should be well-defined since it's the only equation available to explain the residual variance of U . The first four equations, however, model quite a bit of contemporaneous interplay among the variables. The third and fourth equations differ only in the presence of u_P in the fourth. If the u_P didn't appear in the second equation as well, then the fourth equation would likely be well-defined as the only one to catch the remaining variance in P .⁸ However, u_P *does* appear in the second equation as well, and the second equation has quite a bit of similarity to both the fourth and the first equations. In this data set, it turns out that the third and fourth equations *do* manage to achieve the intended behavior of defining the Y and P shocks. If the same model were applied to a different country, or perhaps a different time period, that might not be so clear.

Where there *is* a problem with this model with this data set is with the first two equations. The only difference between the two is that equation two also includes both Y and P . However, in a given data set, there may be only minimal correlation between the residuals for Y and P , and those for R and M .⁹ Since those correlations are being relied upon to separate the two equations, it's quite possible to end up with problems even more striking than in the benchmark data set. As it is, the contours of the posteriors for the parameters in the top corner are quite far from being the ellipses that the asymptotic distribution predicts and the importance function above uses. Importance sampling (or independence chain Metropolis) relies upon having a good approximation for the *entire* parameter set. An alternative which allows the sampler to explore an oddly-shaped surface is random walk Metropolis. This requires quite a bit more "tuning" to get right, so again, try importance sampling first.

Example 5.5 applies random walk Metropolis to the (unnormalized) form of the Sims model. The set up and initial estimation is largely the same as in Example 5.2. One early addition is:

```
compute nburn =10000
compute ndraws=25000
```

⁸It shouldn't be a major concern that u_P also appears in the fifth equation, since that one will be determined largely by how it handles u_U .

⁹Even if series are strongly *dynamically* linked, there may be relatively little contemporaneous correlation to their residuals.

which sets the number of “burn-in” and saved draws. The Markov chain has to start somewhere, and the obvious place is at the maximum likelihood estimates:

```
compute fullparms=diags+svar
cvmmodel(a=afrmld,parmset=fullparms,dmatrix=identity,itors=500,$
  method=bfgs,factor=fsvar) %sigma
*
* Start the chain at the maximizer
*
compute [vector] thetadraw=%parmspeek(fullparms)
compute logplast=%funcval
compute thetalast=thetadraw
```

The “last” variables keep track of the most recently chosen set of parameters (`thetalast`) and its log likelihood (`logplast`). With random walk Metropolis, we will get a proposal by adding an increment to that and will move to the proposal or stay put depending upon the relative likelihoods. The increment that we will use is a multivariate t , again based upon the covariance matrix from the maximum likelihood estimation. However, it is a scaled-down version of that:

```
compute fthetaxx=%decomp(%xx)*0.50
compute nu=30
```

The advantage of using a scale of the covariance matrix is that it's readily available, and it adapts to the relative scales of the parameters. Whether the .5 is a good choice can only be determined once we try it—we may have to adjust it in order to get good sampling properties. In practice, as you vary the multiplier from near zero to one or more, the probability of moving will go from near 1 for the small increments to near zero for the large ones. Neither extreme is what you want.¹⁰

We have one extra task that we're doing in this example which is specific to this problem: we will keep track of the draws for the `A11` and `A12` parameters to demonstrate how the model has a difficult time with that equation.

```
set a11mh 1 ndraws = 0.0
set a12mh 1 ndraws = 0.0
```

Inside the draw loop, the sampler is executed by first coming up with a new proposal by adding the t increment and computing the log likelihood there:

¹⁰If you have limited experience with MCMC techniques, you might want to get the *Bayesian Econometrics* e-course.

```

compute thetatest=thetalast+%ranmvt(fthetaxx,nu)
compute %parmspoke(fullparms,thetatest)
cvmmodel(parmset=fullparms,dmatrix=identity,method=evaluate,$
    a=afrmlid) %sigma
compute logptest=%funcval

```

While the “last” variables are for the previously selected values, the “test” variables are for the proposal. What follows is the standard acceptance test; again note that, to avoid over- or underflow problems, the calculation subtracts the variables in log form before exp'ing . If the proposal is accepted, we replace the “last” variables with the “test” variables and increment the value of `accept`; if the proposal *isn't* accepted, we will keep the previous values for “last”.

```

compute alpha =exp(logptest-logplast)
if alpha>1.0.or.%uniform(0.0,1.0)<alpha
    compute thetalast=thetatest,logplast=logptest,accept=accept+1

```

The next instruction is extremely useful:

```

infobox(current=draw) $
    %strval(100.0*accept/(2*(draw+nburn+1)), "##.##")

```

This displays the percentage of moves which have been accepted. If this is running too high or too low, you can cancel the calculation and adjust the size of the random walk increment.¹¹

The draw loop is set up so the burn-in draws have a negative value for the `DRAW` index. During the burn-in, we don't need to do any further calculations (drawing the VAR coefficients and computing the impulse responses) so we just skip the rest of the loop with

```

if draw<=0
    next

```

Once we're into the draws which we want to save, the first step is to put the current values of the parameter vector back into the `PARMSET`:

```

compute %parmspoke(fullparms,thetalast)

```

We have to do this because the values poked into the `PARMSET` variables right now will be the “test” values, which we might have rejected.

The next two lines, which are specific to this example, save the current values of `A11` and `A12`.

¹¹Unlike importance sampling, where a higher percentage is always better, in random walk sampling, too high an acceptance percentage means that your moves aren't large enough to let you know that you're exploring the parameter space adequately. Too low a percentage means that your chain will be highly autocorrelated and it will be difficult to do inference from the values it generates.

```
compute a11mh(draw)=a11
compute a12mh(draw)=a12
```

The next lines evaluate the A matrix and the implied factor of the covariance matrix. Since this is an unnormalized model, the factor is just the inverse, since the target diagonal is the identity.

```
compute a=afirmld(1)
compute fsigmad=inv(a)
```

Given the factor of the covariance matrix, this is the standard calculation for the draws of the VAR coefficients conditional on that factor:

```
compute betau=%ranmvkron(fsigmad, fxx)
compute betadraw=betaols+betau
compute %modelsetcoeffs(varmodel, betadraw)
```

and this is the familiar code for computing and saving the responses:

```
impulse(model=varmodel, noprint, factor=fsigmad, steps=nsteps, $
  flatten=%responses(draw))
```

When this is run with the settings above, the acceptance rate runs around 13%. That's a bit lower than one would like, but in this case, it helps to overcome the slightly odd shape of the likelihood contours. The graph (Figure 5.2) can be done with the standard-looking:

```
@mcgraphirf(model=varmodel, shocks=shocklabels, $
  variables=varlabels)
```

Note that there is no need for weights, since the “weights” are incorporated by staying on a set of parameters for multiple sweeps.

The scatter plot of the A11 and A12 parameters is shown in Figure 5.3. This is generated with:

```
set a11c 1 628 = thetadraw(1)+6.0*$
  (fthetaxx(1,1)*cos(.01*t)+fthetaxx(1,7)*sin(.01*t))
set a12c 1 628 = thetadraw(7)+6.0*$
  (fthetaxx(7,1)*cos(.01*t)+fthetaxx(7,7)*sin(.01*t))
scatter(footer="Scatter of A11 vs A12 from Random Walk MH", $
  hlabel="A11", vlabel="A12", overlay=line, ovsame) 2
# a11mh a12mh
# a11c a12c / 6
```

which includes a typical contour from the asymptotic distribution for the parameters (which should enclose about 99% of the draws if the asymptotic distribution were accurate).¹² Importance sampling might fail with this because

¹²In this parameter set, A11 is the first parameter and A12 is the seventh, hence the subscripts.

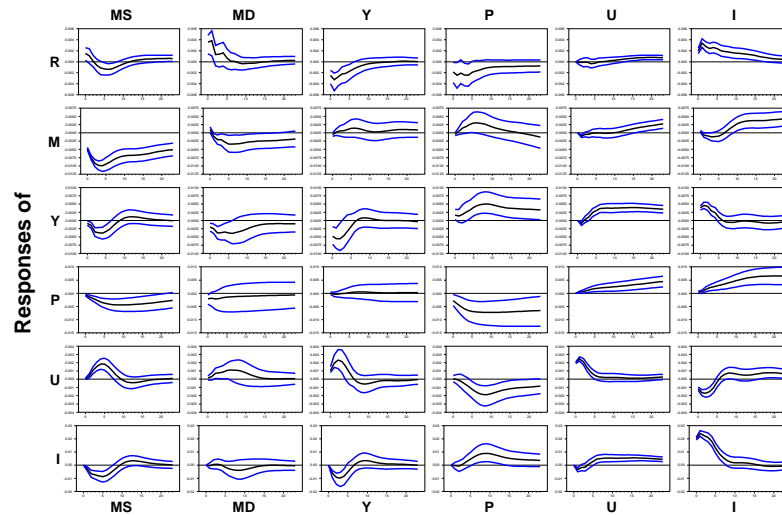


Figure 5.2: IRF with Error Bands with Random Walk Metropolis

both the “northeast” and “northwest” have many more draws than would be expected. Some of the draws northwest of the main part of the distribution would be very unlikely to be selected by either a Normal or even a t approximation.

Note also, that if the intention was to normalize the equation on R (that is, dividing through by A_{11}) for interpretive purposes, it would work quite poorly, since a non-trivial part of the mass is on the “wrong” side of zero and there are some draws quite close to zero. Instead, it would make more sense to normalize on M (dividing through by A_{12}), since it never even comes close to zero.

5.7.5 Monte Carlo with the Waggoner-Zha Sampler

The difficulty in coming up with a reasonable *importance sampler* for a model such as the one from Example 5.5 led Waggoner & Zha (2003) to propose a sampling method which does exact draws for the parameters in each row of an unnormalized A form model conditional on all the other parameters in the SVAR. This is a very technical algorithm, described in Section 5.8.2 if you’re interested. It’s important to note that this has a randomized “sign” choice for each row buried deep in the algorithm, so it will typically produce a distribution which has mirror images. It’s necessary to come up with a rule to pick one of those and that would generally have to wait until the sampler is done and you can analyze the “clouds” of parameters.

Note that Example 5.5 did random walk sampling rather than the more problematic importance sampling. The advantage of the W-Z sampler is that there is no “tuning”—the sampling is done from exact distributions. The advantage of the random walk sampler is that, while it’s possible for the sign to flip in the course of sampling, it’s unlikely, so the original interpretation is likely to hold. It may not be a bad idea to look at both if it’s an appropriate model. The

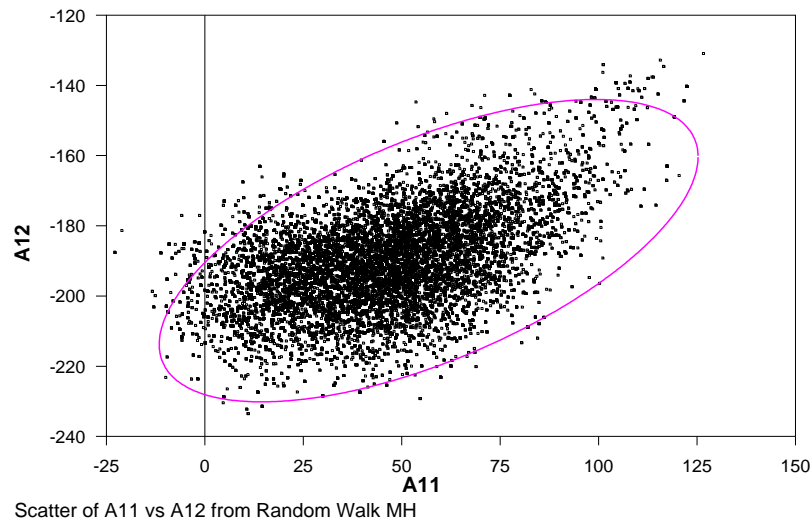


Figure 5.3: A11-A12 in Sims Model with Random Walk Metropolis

W-Z sampler *only* applies to unnormalized A form models, while random walk sampling can be applied to B or A-B models.

Example 5.6 employs the W-Z sampler to the same model as Example 5.5. The basic setup of the program is the same until after the model is estimated by **CVMODEL**. Then

```
@WZSetup AFrm1D(1)
```

takes the estimated A matrix and sets up the W-Z sampler from that. Note that, while the W-Z sampler can handle equality constraints or other restrictions other than just zero restrictions within a row, the procedures described here don't support that. The following is to save the full set of parameters at each draw—**ALLWZ** is a **PARMSET** defined by **@WZSetup** and has the parameters organized by row.

```
dec series[vect] wzbdraws
gset wzbdraws 1 ndraws = %parmspeek(allwz)
```

The draw loop is relatively straightforward because most of the work is done by the procedures and functions that are pulled in when you do **@WZSetup**. **@WZDrawA** draws the set of parameters (again, these are done by row-at-a-time Gibbs sampling) and puts them into the **ALLWZ** **PARMSET**. You might ask where the calculation of the coefficients and impulse responses are. It really doesn't make much sense to do those at this point because it isn't until you've drawn a large number of parameter sets that you can determine an appropriate sign convention. If you just take the draws as computed and do IRF's, you'll get a meaningless set of graphs, since each will have half the draws on one side of the axis and half on the other. (Thus the mean or median will be near zero with very wide error bands).


```

infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Waggoner-Zha"
do draw=-nburn,ndraws
  @WZDrawA
  infobox(current=draw)
  if draw<=0
    next
  compute wzbdraws(draw)=%parmspeek(allwz)
end do draw
infobox(action=remove)

```

The scatter graph of the A11 and A12 draws analogous to the one from Example 5.5 can be generated using:

```

set allwz 1 ndraws = wzbdraws(t) (1)
set a12wz 1 ndraws = wzbdraws(t) (2)
*
scatter(footer="Scatter of A11 vs A12 from WZ Sampler",$
  hlabel="A11",vlabel="A12",smp1=a12wz<0.0)
# allwz a12wz

```

Based upon the experience from the previous example, this is picking the sub-sample where A12 is negative—without the `SMPL`, you'll get two almost identical clouds with signs of both components flipped. The graph (not shown here) is virtually indistinguishable from the one generated by the random walk sampler, which is clearly a good sign.

It looks like picking a normalization on a row that has the smallest (most negative) kurtosis tends to work well—that's the parameter that is most “bimodal”. For the first row, the kurtosis for A11 is around -1.17 while for A12, it's -1.97 so A12 would be the choice by that criterion.

5.8 Tips and Tricks

5.8.1 Using PARMSETS

A `PARMSET` is an object which organizes a set of parameters for non-linear estimation. `CVMODEL` doesn'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.

In this chapter we use the named `PARMSET`s in two rather different ways. Example 5.2 splits the parameters in the structural model into the off-diagonal and the diagonals to make it simpler to switch between the two ways of handling the diagonals. This idea is used heavily in more complicated forms of non-linear estimation, particularly GARCH models which can't be done directly using the `GARCH` instruction.

The named `PARMSET` is also useful when you need to be able to work with the parameters as a `VECTOR`, as is done in the section on error bands. The functions for this are `%PARMSPEEK(parmset)` and `%PARMSPOKE(parmset,vector)`. The first of these returns the current settings as a `VECTOR`; the second resets the parameters based upon the values in the `VECTOR`. In the examples in this chapter, the vector of SVAR parameters is sampled as a group using matrix operations and then `%PARMSPOKE` is used to put that back into the parameter set so `CVMODEL` can evaluate the function, for instance

```
compute thetadraw=thetamean+%ranmvt(fthetaxx,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
```

draws a proposal for the coefficients in the structural model from a multivariate t , pokes them into the `PARMSET` `SVAR` and evaluates the log posterior density there using `CVMODEL`.

5.8.2 Waggoner-Zha sampler

Waggoner & Zha (2003) proposes a (very technical) Gibbs sampling procedure which applies specifically to the 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 β 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:¹³

$$\mathbf{A}(\theta)' = \mathbf{Q}\mathbf{R}$$

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.

¹³Note that this is transposed so we can use the standard QR .

Example 5.1 Eigen factorization

This is an example of using a “principal components” breakdown of the covariance matrix using an eigenvalue-based factorization. This is discussed in Section 5.1.

```
open data ustreasuries.rat
calendar(m) 1953:4
data(format=rats) 1953:04 2009:10 gs1 gs3 gs10
*
* VARLagSelect picks 3 lags, but 7 gives almost the same HQ value,
* so with this amount of data, we'll use the higher one.
*
@varlagselect(lags=12,crit=hq)
# gs1 gs3 gs10
*
system(model=ratevar)
variables gs1 gs3 gs10
lags 1 to 7
det constant
end(system)
estimate(sigma)
*
* Compute eigen factorization.
*
eigen(scale) %sigma * eigfact
*
* Do impulse responses and error decomposition, labeling the shocks as
* PC1, PC2 and PC3.
*
@varirf(model=ratevar, factor=eigfact, page=byshock, steps=48, $
  shocks=| "PC1", "PC2", "PC3" |, $
  variables=| "1 Year", "3 Year", "10 Year" |)
errors(model=ratevar, steps=48, factor=eigfact, $
  labels=| "PC1", "PC2", "PC3" |, results=fevd)
*
* Construct a report with the fractions explained by the first
* principal component for a selected group of forecast horizons.
*
report(action=define)
report(atrow=1, atcol=1, span) $
  "Pct Variance Explained by First Principal Component"
report(row=new, atcol=1) "Step" "1 Year" "3 Year" "10 Year"
report(row=new, atcol=1) 1 100.0*tr(%xcol(%xt(fevd,1),1))
report(row=new, atcol=1) 2 100.0*tr(%xcol(%xt(fevd,2),1))
report(row=new, atcol=1) 3 100.0*tr(%xcol(%xt(fevd,3),1))
report(row=new, atcol=1) 6 100.0*tr(%xcol(%xt(fevd,6),1))
report(row=new, atcol=1) 12 100.0*tr(%xcol(%xt(fevd,12),1))
report(row=new, atcol=1) 24 100.0*tr(%xcol(%xt(fevd,24),1))
report(row=new, atcol=1) 48 100.0*tr(%xcol(%xt(fevd,48),1))
report(action=format, picture="###.###")
report(action=show)
```

Example 5.2 SVAR: A-style Model

This demonstrates maximum likelihood estimation of an “A” form structural model for a VAR. This is described in Section 5.5.

```

open data simszha.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:01 1989:03 gin82 gnp82 gnp gd $
    lhur fygn3 m1
*
compute nlags=4
*
set gin82 = log(gin82)
set gnp82 = log(gnp82)
set gd    = log(gd)
set m1    = log(m1)
set fygn3 = fygn3*.01
set lhur  = lhur*.01
*
system(model=varmodel)
variables fygn3 m1 gnp82 gd lhur gin82
lags 1 to nlags
det constant
end(system)
*
estimate(noprint)
*
nonlin(parmset=svar) a12 a21 a23 a24 a31 a36 a41 a43 a46 $
    a51 a53 a54 a56
*
dec frml[rect] afrml
frml afrml = ||1.0,a12,0.0,0.0,0.0,0.0|$
              a21,1.0,a23,a24,0.0,0.0|$
              a31,0.0,1.0,0.0,0.0,a36|$
              a41,0.0,a43,1.0,0.0,a46|$
              a51,0.0,a53,a54,1.0,a56|$
              0.0,0.0,0.0,0.0,0.0,1.0||
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0
*
cvmodel(a=afrml,parmset=svar,method=bfgs,factor=fsvar) %sigma
*
compute shocklabels=||"MS","MD","Y","P","U","I"||
compute varlabels=||"R","M","Y","P","U","I"||
*
@varirf(model=varmodel,steps=24,page=byvariable,$
    shocks=shocklabels,var=varlabels,factor=fsvar)
errors(model=varmodel,steps=24,labels=shocklabels,factor=fsvar)
*
* Estimate same model without normalization
*
nonlin(parmset=diags) a11 a22 a33 a44 a55 a66
*
dec frml[rect] afrmld

```

```

frml afrmld = ||a11,a12,0.0,0.0,0.0,0.0|$
               a21,a22,a23,a24,0.0,0.0|$
               a31,0.0,a33,0.0,0.0,a36|$
               a41,0.0,a43,a44,0.0,a46|$
               a51,0.0,a53,a54,a55,a56|$
               0.0,0.0,0.0,0.0,0.0,a66||

*
* Initialize parameters with the square roots of the diagonals of %sigma
* for the diagonal elements and zeros elsewhere.
*
compute a11=sqrt(%sigma(1,1)),a22=sqrt(%sigma(2,2)), $
        a33=sqrt(%sigma(3,3)),a44=sqrt(%sigma(4,4)), $
        a55=sqrt(%sigma(5,5)),a66=sqrt(%sigma(6,6))

*
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0
*
cvmodel(a=afrmld,parmset=diags+svar,dmatrix=identity,itors=500,$
        method=bfgs,factor=fsvar) %sigma

```

Example 5.3 SVAR: A-B style model

This demonstrates maximum likelihood estimation of an “A” form structural model for a VAR. This is also described in Section 5.5.

```

cal(m) 1959
all 1996:12
*****
*               DEFINITIONS
*****
*
compute NSTEP      = 48
compute NPOLICY    = 3
compute NNONPOLICY = 3
*
compute NVAR       = NPOLICY+NNONPOLICY
DEC vect[labels] implab(nvar)
input IMPLAB
    GDP PGDP PCOM TR NBR FFR

*****
*   DATA Transformation
*****
*
*   Reserves
*
open data bmdata.rat
data(format=rats) 1959:1 1996:12 TRES NBREC FYFF PSCCOM
*
*   Interpolated monthly variables: GDP and PGDP
*
data(format=rats) 1965:1 1996:12 gdpm pgdpm
*
*   Take logs
*
set psccom = 100.0*log(psccom)
set gdpm   = 100.0*log(gdpm)
set pgdpm  = 100.0*log(pgdpm)

***** 36-month MA normalization

mvstats(means=trma,span=36) tres 1961:12 1996:12
set TR1 1962:1 1996:12 = tres/trma{1}
set NBREC1 1962:1 1996:12 = nbrec/trma{1}

***** Set up the sample range

DEC vector[integer] date1(5) date2(5) lags(5)
compute date1 = ||1965:01,1965:01,1979:10,1984:02,1988:09||
compute date2 = ||1996:12,1979:09,1996:12,1996:12,1996:12||
compute lags = || 13 , 11 , 12 , 7 , 11 ||

dec frml[rect] ab phi

```



```

nonlin alpha beta phid phib
frml ab = ||1.0 , 0.0 , alpha|$
          1.0 ,-1.0 ,-beta|$
          0.0 , 1.0 , 0.0||
frml phi = ||1.0 , 0.0 , 0.0|$
            0.0 , 1.0, 0.0|$
            phid,phib, 1.0||

*****
*                               ESTIMATION
*****

system(model=bmmmodel)
variables gdpm pgdpm psccom tr1 nbrec1 fyff
lags 1 to 13
det constant
end(system)

*
* Estimate the full VAR over the current range
*
estimate(resids=u0,noprint) (1965:1)+13 *
*
* Bernanke-Mihov model
* Get 3x3 matrix of monetary sector shocks with the three other
* variables swept out.
*
compute sweepvcv=%sweeptop(%sigma,NNONPOLICY)
compute [symm] moneyvcv=$
      %xsubmat (sweepvcv,NNONPOLICY+1,NVAR,NNONPOLICY+1,NVAR)
*
* Estimate FFR model
*
compute alpha=0.00,beta=.05
*
nonlin alpha beta phid=1 phib=-1
cvmodel(a=ab,b=phi) moneyvcv
*
* Estimate NBR model
*
nonlin alpha beta phid=0 phib=0
compute alpha=.03,beta=.014
cvmodel(a=ab,b=phi) moneyvcv
*
* Estimate NBR/TR Model (Orthogonalized NBR)
*
nonlin alpha=0 beta phid phib=0
compute beta=.05,phid=.80
cvmodel(a=ab,b=phi) moneyvcv
*
* Estimate BR model
*
nonlin alpha beta phid=1 phib=alpha/beta
compute alpha=0.0,beta=.05
cvmodel(a=ab,b=phi) moneyvcv

```

```

*
* Estimate JI model
*
nonlin alpha=0 beta phid phib
compute phid=0.8,phib=-0.8,beta=0.05
cvmodel(a=ab,b=phi) moneyvcv

```

Example 5.4 SVAR: Importance Sampling for Error Bands

This example demonstrates estimation and calculation of IRF's with error bands for a "B" type structural model using importance sampling. The detailed are discussed in Section 5.7.3.

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpchs canexpgdpds $
    canmls canusxsr usaexpgdpch
*
set logcangdp = 100.0*log(canexpgdpchs)
set logcandefl = 100.0*log(canexpgdpds)
set logcanml = 100.0*log(canmls)
set logusagdp = 100.0*log(usaexpgdpch)
set logexrate = 100.0*log(canusxsr)
*
system(model=canmodel)
variables logusagdp logcangdp can3mthpcp logexrate logcandefl logcanml
lags 1 to 4
det constant
end(system)
*
estimate(noprint)
*
* Save the decomposition of the X'X inverse matrix from the VAR and the
* OLS estimates of the coefficients for use in drawing coefficients.
*
compute fxx =%decomp(%xx)
compute betaols=%modelgetcoeffs(canmodel)
*
compute ncoef=%nreg
compute nvar =%nvar
*
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 pn2 mr2 mf1
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|$
              0.0,0.0,0.0,0.0,1.0,pn2|$
              0.0,mr2,mf1,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=pn2=mr2=mf1=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=marginal,pmethod=genetic,piters=50,method=bfgs) %sigma
compute nfree=%nreg
*
compute nsteps=20
compute ndraws=10000
*
* thetamean is the maximizing vector of coefficients.
*
compute [vector] thetamean=%parmspeek(svar)
*
* fthetaxx is a factor of the (estimated) inverse Hessian at the final
* estimates for use in the proposal density.
*
compute fthetaxx=%decomp(%xx)
*
* nu is the degrees of freedom for the multivariate Student used in
* drawing A's
*
compute nu=10.0
*
* The combination of thetamean, fthetaxx and nu determine the proposal
* density. The two which can be "tuned" if you need to improve the
* performance of the sampler are fthetaxx and nu. You can scale fthetaxx
* (typically up) and adjust nu in either direction, though generally a
* low yield requires fatter tails so nu would typically be adjusted
* downwards.
*
* scladjust is used to prevent overflows when computing the weight
* function
*
compute scladjust=%funcval
*
* This will be used in drawing values for lambda.
*
declare vect lambdadraw(nvar) vdiag(nvar)
*
declare vect[rect] %%responses(ndraws)
*
* This will hold the (relative) weights
*
set weights 1 ndraws = 0.0
*
infobox(action=define,progress,lower=1,upper=ndraws) $
    "Monte Carlo Integration/Importance Sampling"
do draw=1,ndraws
    *
    * Draw a new theta centered at thetamean from a multivariate t-density

```

```

*
compute thetadraw=thetamean+%ranmvt(fthetaxx,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
*
* Conditioned on theta, make a draw for lambda.
*
compute b=bfrml(1)
compute vdiag=%mqformdiag(%sigma,tr(inv(b)))
ewise lambdadraw(i)=$
    (%nobs/2.0)*vdiag(i)/%rangamma(.5*(%nobs-ncoef)+delta+1)
*
* Compute the implied factor
*
compute fsigmatd=b*%diag(%sqrt(lambdadraw))
*
* Create a draw for the coefficients conditional on the factor and
* reset the coefficients of the model.
*
compute betau=%ranmvkron(fsigmad,fxx)
compute betadraw=betaols+betau
compute %modelsetcoeffs(canmodel,betadraw)
*
* Compute and save the impulse responses
*
impulse(model=canmodel,noprint,factor=fsigmatd,steps=nsteps,$
        flatten=%responses(draw))
*
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.
*
sstats 1 ndraws weights>>sumwt weights^2>>sumwt2
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
*
@mcgraphirf(model=canmodel,weight=weights,$
  shocks=||"R1","R2","F1","F2","N1","N2"||,$
  variables=||"US GDP","Can GDP","Can Rate","XRate","Prices","Money"||)
```

Example 5.5 SVAR: Random Walk Metropolis for Error Bands

This continues the analysis from Example 5.2 by computing IRF's with error bands using Random Walk Metropolis. This is discussed in Section 5.7.4.

```

compute nburn =10000
compute ndraws=25000
compute nsteps=24
*
open data simszha.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:01 1989:03 gin82 gnp82 gnp gd lhur fygn3 m1
*
compute nlags=4
*
set gin82 = log(gin82)
set gnp82 = log(gnp82)
set gd    = log(gd)
set m1    = log(m1)
set fygn3 = fygn3*.01
set lhur  = lhur*.01
*
system(model=varmodel)
variables fygn3 m1 gnp82 gd lhur gin82
lags 1 to nlags
det constant
end(system)
*
estimate(noprint)
*
compute fxx      =%decomp(%xx)
compute betaols=%modelgetcoeffs(varmodel)
*
compute ncoef=%nreg
compute nvar =%nvar
*****
*
compute shocklabels=||"MS","MD","Y","P","U","I"||
compute varlabels=||"R","M","Y","P","U","I"||
*
nonlin(parmset=svar) a12 a21 a23 a24 a31 a36 a41 a43 a46 $
                    a51 a53 a54 a56
nonlin(parmset=diags) a11 a22 a33 a44 a55 a66
*
* Estimate the model in unnormalized form
*
dec frml[rect] afrmld
frml afrmld = ||a11,a12,0.0,0.0,0.0,0.0|$
              a21,a22,a23,a24,0.0,0.0|$
              a31,0.0,a33,0.0,0.0,a36|$
              a41,0.0,a43,a44,0.0,a46|$
              a51,0.0,a53,a54,a55,a56|$
              0.0,0.0,0.0,0.0,0.0,a66||

```

```

*
compute a11=sqrt(%sigma(1,1)),a22=sqrt(%sigma(2,2)), $
      a33=sqrt(%sigma(3,3)),a44=sqrt(%sigma(4,4)), $
      a55=sqrt(%sigma(5,5)),a66=sqrt(%sigma(6,6))
*
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0
*
compute fullparms=diags+svar
cvmmodel(a=afrmld,parmset=fullparms,dmatrix=identity,itters=500,$
  method=bfgs,factor=fsvar) %sigma
*
* Start the chain at the maximizer
*
compute [vector] thetadraw=%parmspeek(fullparms)
compute logplast=%funcval
compute thetalast=thetadraw
*
* 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 fthetaxx=%decomp(%xx)*0.50
compute nu=30
*
declare vect[rect] %%responses(ndraws)
*
set a11mh 1 ndraws = 0.0
set a12mh 1 ndraws = 0.0
*
compute accept=0
infobox(action=define,progress,lower=-nburn,upper=ndraws) "Random Walk MH"
do draw=-nburn,ndraws
  compute thetatest=thetalast+%ranmvt(fthetaxx,nu)
  compute %parmspoke(fullparms,thetatest)
  cvmmodel(parmset=fullparms,dmatrix=identity,method=evaluate,$
    a=afrmld) %sigma
  compute logptest=%funcval
  *
  * Compute the acceptance probability
  *
  compute alpha =exp(logptest-logplast)
  if alpha>1.0.or.%uniform(0.0,1.0)<alpha
    compute thetalast=thetatest,logplast=logptest,accept=accept+1
  *
  infobox(current=draw) $
    %strval(100.0*accept/(2*(draw+nburn+1)),"##.##")
  if draw<=0
    next
  *
  compute %parmspoke(fullparms,thetalast)
  *
  compute a11mh(draw)=a11
  compute a12mh(draw)=a12
  *

```

```

compute a=afmld(1)
compute fsigmad=inv(a)
*
* Create a draw for the coefficients conditional on the factor and
* reset the coefficients of the model.
*
compute betau=%ranmvkron(fsigmad, fxx)
compute betadraw=betaols+betau
compute %modelsetcoeffs(varmodel, betadraw)
*
* Compute and save the impulse responses
*
impulse(model=varmodel, noprint, factor=fsigmad, steps=nsteps, $
    flatten=%responses(draw))
*
end do draws
infobox(action=remove)
*****
@mcgraphirf(model=varmodel, shocks=shocklabels, $
    variables=varlabels)
*
set a11c 1 628 = thetadraw(1)+6.0*$
    (fthetaxx(1,1)*cos(.01*t)+fthetaxx(1,7)*sin(.01*t))
set a12c 1 628 = thetadraw(7)+6.0*$
    (fthetaxx(7,1)*cos(.01*t)+fthetaxx(7,7)*sin(.01*t))
scatter(footer="Scatter of A11 vs A12 from Random Walk MH", $
    hlabel="A11", vlabel="A12", overlay=line, ovsame) 2
# a11mh a12mh
# a11c a12c / 6

```


Example 5.6 SVAR: Waggoner-Zha Sampler

This employs the Waggoner-Zha sampler to the same model as Example 5.5. The details are in Section 5.7.5.

```

compute nburn =10000
compute ndraws=25000
*
open data simszha.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:01 1989:03 gin82 gnp82 gnp gd lhur fygn3 m1
*
compute nlags=4
*
set gin82 = log(gin82)
set gnp82 = log(gnp82)
set gd    = log(gd)
set m1    = log(m1)
set fygn3 = fygn3*.01
set lhur  = lhur*.01
*
system(model=varmodel)
variables fygn3 m1 gnp82 gd lhur gin82
lags 1 to nlags
det constant
end(system)
*
estimate(noprint)
*
compute fxx      =%decomp(%xx)
compute betaols=%modelgetcoeffs(varmodel)
*
compute wztncobs=%ncobs
compute ncoef=%nreg
compute nvar =%nvar
*****
*
compute shocklabels=||"MS","MD","Y","P","U","I"||
compute varlabels=||"R","M","Y","P","U","I"||
*
nonlin(parmset=svar) a12 a21 a23 a24 a31 a36 a41 a43 a46 $
                    a51 a53 a54 a56
nonlin(parmset=diags) a11 a22 a33 a44 a55 a66
*
* Estimate the model in unnormalized form
*
dec frml[rect] afrmld
frml afrmld = ||a11,a12,0.0,0.0,0.0,0.0|$
              a21,a22,a23,a24,0.0,0.0|$
              a31,0.0,a33,0.0,0.0,a36|$
              a41,0.0,a43,a44,0.0,a46|$
              a51,0.0,a53,a54,a55,a56|$
              0.0,0.0,0.0,0.0,0.0,a66||

```

```

*
compute a11=sqrt(%sigma(1,1)),a22=sqrt(%sigma(2,2)), $
        a33=sqrt(%sigma(3,3)),a44=sqrt(%sigma(4,4)), $
        a55=sqrt(%sigma(5,5)),a66=sqrt(%sigma(6,6))
*
compute a12=a21=a23=a24=a31=a36=a41=a43=a46=a51=a53=a54=a56=0.0
*
compute fullparms=diags+svar
cvmmodel(a=afmld,parmset=fullparms,dmatrix=identity,itters=500,$
        method=bfgs,factor=fsvar) %sigma
*****
*
* Set up the sampler using the estimated SVAR
*
@WZSetup AFmld(1)
*
dec series[vect] wzbdraws
gset wzbdraws 1 ndraws = %parmspeek(allwz)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
        "Waggoner-Zha"
do draw=-nburn,ndraws
    @WZDrawA
    infobox(current=draw)
    if draw<=0
        next
    compute wzbdraws(draw)=%parmspeek(allwz)
end do draw
infobox(action=remove)
*
set allwz 1 ndraws = wzbdraws(t)(1)
set a12wz 1 ndraws = wzbdraws(t)(2)
*
scatter(footer="Scatter of A11 vs A12 from WZ Sampler",$
        hlabel="A11",vlabel="A12",smpl=a12wz<0.0)
# allwz a12wz
*
* Compute statistics to get the kurtosis
*
stats allwz
stats a12wz

```

Semi-Structural VAR's

Instead of a fully specified orthogonal factorization, it's also possible to examine the behavior of a single shock which has a specified set of properties. This is based upon the following:

Proposition If Σ is an $m \times m$ positive definite symmetric matrix, and \mathbf{x} is a non-zero $m \times 1$ vector, then

- a) there exists a factor of $\Sigma = \mathbf{G}^{-1}\mathbf{G}'^{-1}$ where the first *row* of \mathbf{G} is a scale multiple of \mathbf{x} .
- b) there exists a factor of $\Sigma = \mathbf{F}\mathbf{F}'$ where the first *column* of \mathbf{F} is a scale multiple of \mathbf{x} .

Proof. For (a), factor $\Sigma = \mathbf{P}\mathbf{P}'$ (any factor will do). Generate an orthonormal matrix \mathbf{V} with a scale of $\mathbf{P}'\mathbf{x}$ as the first column. $\mathbf{P}\mathbf{V}$ is the desired factor. For (b), generate an orthonormal matrix \mathbf{U} with a scale multiple of $\mathbf{P}^{-1}\mathbf{x}$ as the first column. Then $\mathbf{P}\mathbf{U}$ gives the desired factor. \square

Forcing a column fixes the impact responses. Forcing a row in the inverse fixes as one of the orthogonalized shocks a particular linear combination of the non-orthogonalized innovations. For instance, in a two variable system, $\mathbf{x} = \{1, 1\}$ used with (a) means that the innovation is the sum of the non-orthogonal innovations in the two variables. Used with (b) means that the innovation would impact both variables equally in the first period.

By far the most common of these is type (b), which sets the loading of the shock onto the variables. Uhlig (2005) defines the term *impulse vector* for the scale multiple $\lambda\mathbf{x}$ that makes up the column of the factor. His proposition 1 characterizes impulse vectors—the most important of these in practice is his statement 4 (paraphrased):

Statement 4 If $\mathbf{P}\mathbf{P}' = \Sigma$, then the space of impulse vectors is precisely the set of $\mathbf{P}\alpha$, where $\|\alpha\| = 1$.

A well-known example of a semi-structural VAR is the Blanchard-Quah factorization (Blanchard & Quah (1989)). While BQ name the two shocks as “demand” and “supply”, they’re actually “demand” and whatever shock is required to complete an orthogonal factorization. In the 2×2 case, the BQ restriction is that one of the shocks has a zero long-run effect on one of the two variables: in

their original case, it was that a demand shock has no long-run effect on real GNP. To implement this, the model is run with the first difference of real GNP as one of the variables. The long run effect on GNP of a particular shock will be the sum of the MAR coefficients for the first difference of GNP. If $\Psi(1)$ is the long-run response matrix to the non-orthogonal innovations, the BQ demand shock is defined to be a solution to

$$\Psi(1)\mathbf{x} = \begin{bmatrix} 0 \\ * \end{bmatrix}$$

assuming GNP is the first variable.

In King et al. (1991), a “balanced growth” shock is defined which loads equally in the long-run onto the three variables in a consumption, investment and income VAR. You can determine that by solving for \mathbf{x} in

$$\Psi(1)\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Note that the remaining two orthogonalized shocks in this last case can't really be interpreted sensibly: they'll be just one of many ways to complete the “model”. The responses of the system to the shock of interest don't depend upon them, and, in particular, neither does the fraction of the variance explained by it.

6.1 ForcedFactor Procedure

The procedure `@FORCEDFACTOR` computes a factorization based upon the proposition at the start of the chapter. With `FORCE=COLUMN`, it takes as input an m vector or an $m \times r$ matrix and the covariance matrix and produces a factor which has a multiple of the m vector in the first column, or the input $m \times r$ matrix postmultiplied by $r \times r$ (upper triangular) matrix Π in the first r columns of the factor. With `FORCE=ROW`, it takes an m vector or an $r \times m$ matrix and the covariance matrix and produces a factor whose *inverse* has a multiple of the vector in the first row or, for the matrix input, an $r \times r$ (lower triangular) matrix Π pre-multiplying the input matrix as the first r rows of the factor.

For the vector case, this is done exactly as described in the proof. For the matrix case, it's a generalization of that to higher dimensions. There are many general Π matrices which will produce the proper type of factorization—the triangular structures mean that, in the `FORCE=COLUMN` case, the first column in the factor is a multiple of the first column in the input; the second column in the factor is a linear combination of just the first two columns in the input, etc., with analogous behavior for the row case. If, in the column case, you input r columns which are already constructed to be impulse vectors, then the resulting factor will reproduce those in the first r columns. In that case, you're

using `@FORCEDFACTOR` to get the span of the space orthogonal to those, which will be columns $r + 1$ to m .

```
@ForcedFactor(force=column) sigma ||1.0,1.0,0.0,0.0|| f1
```

F1 will be a factorization where the first orthogonal component loads equally onto the first two series, and hits none of the other variables contemporaneously.

```
@forcedfactor(force=column) s i1~i2 f
compute f3=%xsubmat(f,1,4,3,4)
```

In a four variable VAR, the 4×2 matrix F3 will span the space of all impulse vectors that are orthogonal to the vectors \mathbf{i}_1 and \mathbf{i}_2 .¹

6.2 Short- and Long-Run Restrictions

The Blanchard-Quah model is a simple case of a model identified with short- and long-run restrictions; in this case, because of the size, just a long-run restriction. In practice, these have generally been a special case of the “B” variety of structural VAR, except the parametrization is an implicit rather than explicit function.

What we’re seeking is a matrix B where $BB' = \Sigma$. The short-run constraints are that specific elements of B are zero. The long-run constraints are that specific elements of $\Psi(1)B$ are zero. The combination of these forms the “model”.

In order for the model to be just identified (by the order condition), there need to be $m(m - 1)/2$ total restrictions. For the 2 variable case, that means 1, which the BQ model does by making the (1,2) long-run element equal to zero. Note that, because the target values are zero, the identification process doesn’t determine the sign of the shocks, so you might have to flip signs in order to produce shocks with the correct interpretation.

In practice, the $\Psi(1)$ (which is the sum of moving average coefficients) needs to be computed from the VAR coefficients. If the VAR has been written to be invertible (no non-stationary series), you can get the sum of lag coefficients for the autoregression with the `%VARLAGSUMS` matrix defined by `ESTIMATE`, or, if you alter the coefficients in the model by, for instance, a Monte Carlo draw, by using the function `%MODELLAGSUMS(model)`. The inverse of that matrix is the estimate of $\Psi(1)$.

The BQ factorization can be generated constructively using a Cholesky factorization of a transformed matrix. The special function `%BQFACTOR` function can be used for that:

```
compute bqfactor=%bqfactor(%sigma,%varlagsums)
```

¹~ does a horizontal concatenation (one next to the other) of a pair of matrices or vectors.

This generalizes to m variables by choosing a restriction that the long-run response matrix is lower triangular—that's what you'll get if you apply %BQFACTOR to a larger system. That, however, is unlikely to be a very interesting structure. Instead, models are generally identified using a combination of short-run and long-run restrictions. With that, there no longer is a simple constructive solution for the factor; instead, you get a system of (non-linear) equations that must be solved.

Unlike the models which are entirely contemporaneous, models with long-run restrictions cannot easily be handled if they're over-identified. Because the maximum likelihood estimates of the lag coefficients are just the OLS estimates regardless of the value of Σ , they can be concentrated out, allowing us to focus only on modeling the contemporaneous relationship. For a just-identified model of Σ , long-run restrictions don't restrict the lag coefficients (even though they're a function of them), since we have enough freedom in the coefficients matrix to make the restrictions work while achieving the overall maximum for the likelihood. (All just-identified structural VAR's have the same likelihood). If the model is over-identified, however, it's almost certain that you can achieve a higher likelihood by adjusting the lag coefficients. While this could be done using **NLSYSTEM**, it involves estimating all the coefficients subject to rather nasty non-linear constraints.

For a model just-identified by short- and long-run restrictions, the simplest way to estimate the model is with the procedure **@ShortAndLong**. This takes as input two "pattern" matrices for the short- and long-run restrictions, with zeros where you want zeros and non-zeros elsewhere. An example is:

```
dec rect lr(4,4)
dec rect sr(4,4)
input sr
. 0 0 .
. 0 . .
. . . .
. . . .
input lr
. 0 0 0
. . . .
. . . .
. . . .
@shortandlong(sr=sr,lr=lr,massums=inv(%varlagsums),factor=b) %sigma
```

While you could use any non-zero value to represent the non-zero slots, we've found that using the . (missing value) makes this easier to read and understand.

The result from Section 5.4 applies to the global identification (other than sign) for this model. If you count zeros in columns, you get 0-3-2-1, which is fine.

Change the long-run (1,3) restriction to a (3,1) and it no longer is globally identified.

The first paper to combine short- and long-run restrictions was Gali (1992). That, however, has the complication that it also includes a single restriction on the structure of the shocks (an element of B^{-1}) so it doesn't have a simple reparametrization. What **@ShortAndLong** can do there is to solve out as much as possible, by recasting the short- and long-run loading restrictions into a reduced set of parameters. To do this, add the options **NOESTIMATE** and **RPERP=RPerp Matrix**. The **RPERP** matrix multiplied by the (reduced) parameter vector creates a vec'ed form of the "B" matrix with the short- and long-run loading restrictions.

This, for instance, has two short-run, three long-run and one additional restriction that variable 3 not enter the construction of shock 3. That last one has to be added as a restriction on the inverse of the B matrix.

```
dec rect lr(4,4)
dec rect sr(4,4)
*
input sr
. 0 0 .
. . . .
. . . .
. . . .
input lr
. 0 0 0
. . . .
. . . .
. . . .
*
@shortandlong(sr=sr,lr=lr,massums=massums,$
noestimate,rperp=rperp) %sigma
*
dec frml[rect] bf af
frml bf = %vectorect(rperp*theta,4)
frml af = inv(bf(0))
*
nonlin(parmset=base) theta
nonlin(parmset=r8) af(0)(3,3)==0.0
cvmodel(parmset=base+r8,b=bf,itors=400) %sigma
```

6.3 Multi-step Restrictions

The same basic idea as **@ShortAndLong** can be applied to impose zero restrictions on responses at steps other than impact and long-run. If Ψ_s is the response matrix at step s to the non-orthogonal shocks, then $\Psi_s B$ is the response at step s to the orthogonalized shocks with factor matrix B . A step s restriction

on the structural model will be that some element of $\Psi_s B$ is zero. Since the Ψ_s values are known (from doing impulse responses to unit shocks), this creates a linear restriction on the $m \times m$ matrix B . A short-run (impact) restriction does this as well, though it's a very simple restriction that a particular B_{ij} is zero. And a long-run restriction works the same way but with the long-run matrix $\Psi(1)$ in place of Ψ_s . A combination of short-, long- and intermediate-run restrictions combined will create the restriction:

$$Rvec(B) = 0 \quad (6.1)$$

That, combined with $BB' = \Sigma$ is the (just-identified) model assuming that you have a total of $m(m-1)/2$ zero restrictions. As with the long-run restrictions, the estimation process is only simple for just-identified models, as the multi-step response restrictions would impose a restriction on the lag coefficients in an over-identified model.

The procedure `@IRFRestrict` can be used to build up the R matrix for a set of constraints. You do one call to `@IRFRestrict` for each restriction. Example 6.2 shows how to do this. It does impact constraints of:

$$\begin{array}{cccc} \cdot & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}$$

and one-step ahead constraints of

$$\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}$$

(This combination just-identifies a B-style model). First, we need the responses to unit shocks for at least two steps (this actually does three) and we extract the Ψ_0 (which is the identity) and Ψ_1 matrices by applying the `%XT` function to the `RECT[SERIES] IRFS`.

```
impulse(model=peersman, factor=%identity(%nvar), steps=3, results=irfs)
compute step0=%xt(irfs,1)
compute step1=%xt(irfs,2)
```

This builds up the constraint matrix by six calls to `@IRFRESTRIC`. After the first call, the RR matrix will be 1×16 , after the second 2×16 , etc. You have to use all three of the options: `IRF`, `VARIABLE` and `SHOCK` with each call to identify the desired zero position (which variable response to which shock) and the Ψ matrix that weights the impacts.


```

dec rect rr(0,0)
@IRFRestrict(irf=step0,variable=1,shock=2) rr
@IRFRestrict(irf=step0,variable=1,shock=3) rr
@IRFRestrict(irf=step0,variable=1,shock=4) rr
@IRFRestrict(irf=step0,variable=2,shock=4) rr
@IRFRestrict(irf=step1,variable=2,shock=3) rr
@IRFRestrict(irf=step1,variable=2,shock=4) rr

```

Similar to the above, this converts the constraints (6.1) into its “inverse”

$$\text{vec}(\mathbf{B}) = \mathbf{R}^\perp \theta \quad (6.2)$$

where θ is a VECTOR of free parameters and then converts the vectorized \mathbf{B} matrix back into a properly-sized impact matrix as part of the FRML.

```

compute rp=%perp(rr)
dec vect theta(%cols(rp))
dec frml[rect] lrfml
*
frml lrfml = %vectorect(rp*theta,%nvar)

```

The elements of `THETA` are usually almost impossible to interpret directly, so it's not clear what a good set of guess values will be. The following picks the set that come closest to giving a Cholesky factor:

```
compute [vect] theta=%ginv(rp)*%vec(%decomp(%sigma))
```

The following estimates the free parameters and checks that the solution has the desired constraints:

```

nonlin theta
cvmmodel(b=lrfml,f,dmatrix=identity) %sigma
disp "Impact responses with multi-step restrictions" ###.### f
disp "Step one responses" ###.### step1*f

```

6.4 Error Bands

As we've emphasized throughout this chapter, models with long (or multi-step) restrictions are only tractable when exactly identified. With an exactly identified model, it's possible to draw the covariance matrix from its unconditional distribution, then draw the lag coefficients conditional on that, and solve out for the structural model. However, it's important to remember that the structural model depends upon the new set of coefficients—an error that's been made in some empirical work is to form the structural constraints using the original OLS rather than the coefficients from the draw.

It's also important to remember that the combination of zero restrictions (at any horizon) and $\mathbf{B}\mathbf{B}' = \Sigma$ will still be satisfied if any column is multiplied

by -1. If you're just doing a one-off estimate, you can always change the signs to match what you want, but if it's part of a error band calculation, you need to force the proper signs with each draw. This is most easily done using the %DMULT function as shown below.

Example 6.3 uses the the FFUNCTION option added to @MCVARDODRAWS with version 9 to simplify the setup. This uses the same basic VAR as Example 6.2 but uses the original short- and long-run restrictions:

```
input lr
. . . .
. . 0 0
. . . .
. . . .
input sr
. 0 0 0
. . . 0
. . . .
. . . .
```

The four variables are oil price, output, price and interest rates, in order (the first three in growth rates) and the four shocks are identified (in order) as oil price, aggregate supply, aggregate demand and (contractionary) money. Thus, none of the other non-oil shocks has an impact on oil prices, money shocks have no impact on output, and aggregate demand and money shocks have no long-run effects on output. The obvious sign conventions are that the oil price shock is positive on oil prices, AS and AD are positive on output and the money shock is positive on interest rates.

The following is the “factor” function used for this model:

```
function FLongShort sigma model
type rect      FLongShort
type symmetric sigma
type model     model
*
local rect MASums
local rect f
*
compute MASums=inv(%ModelLagSums(model))
@shortandlong(factor=f,lr=lr,sr=sr,masum=MASums) sigma
*
compute FLongShort=%dmult(f,$
    ||%sign(f(1,1)),%sign(f(2,2)),%sign(f(2,3)),%sign(f(4,4))||)
end
```

Note that this recomputes the lag sums based upon the *current* coefficients in the model—it's applications like this why the model is passed to the function. The only line in this that is specific to this particular application is the second

line in the `compute FLongShort` which corrects the signs. `%DMULT` multiplies the elements of a matrix (here `F`) by a diagonal matrix whose diagonal elements are in the second parameter, which, for readability, we've put on its own line. `%SIGN(x)` is +1 if x is positive and -1 if x is negative. `f(i,j)` is the impact of shock j on variable i . So the `%sign(f(1,1))` in the first slot of the `VECTOR` multiplies column one of `F` by +1 if the 1,1 element is positive and by -1 if the 1,1 element is negative, thus ensuring that the 1,1 element of the output matrix is positive (thus oil shock is positive on oil price). Similarly, the `%sign(f(2,3))` in the third slot multiplies by third column (the AD shock) by +1 if the impact on variable 2 (output) is already positive and by -1 if it is negative. If we wanted to define the money shock as expansionary rather than contractionary, we would use `-%sign(f(4,4))` in the fourth slot. That would multiply by -1 if the impact on interest rates (variable 4) were positive and by +1 if it were negative, thus getting the desired sign on the impact.

With the `FUNCTION` set, the draws are generated with:

```
@MCVARDoDraws(model=peersman,ffunction=FLongShort,$
draws=2000,steps=40,accum=||1,2,3||)
```

This does 2000 draws over 40 steps, accumulating the responses for the first three series (the ones that are modeled in growth rates).

The graphs are generated with:

```
@MCGraphIRF(model=peersman,$
shocks=||"Oil Price","Supply Shock","Demand Shock","Money Shock"||,$
variables=||"Oil Price","Output","Price","Interest Rate"||)
```

Example 6.1 Blanchard-Quah Decomposition

This is a replication of the analysis from Blanchard & Quah (1989). There is a very slight difference in the data set, which shows up mainly in the display of the structural residuals.

Because the data are done with GNP in differences, and because the data are de-meanned in two separate ranges, the historical decomposition requires some adjustments in order to put get it back to the historical values. Also, for graphics purposes, the logged data are scaled up by 100; this again needs to be reversed when working with the historical decomposition.

```
cal(q) 1948
open data bqdata.xls
data(format=xls,org=cols) 1948:1 1987:4 gnp gd87 lhmur
*
set loggnp = log(gnp)
set loggd = log(gd87)
set logrgnp = loggnp-loggd+log(100)
set dlogrgnp = 100.0*(logrgnp-logrgnp{1})
*
* Extract separate means from the GNP growth series. Save the fitted
* values for rebuilding the data later.
*
set dummy1 = t<=1973:4
set dummy2 = t>1973:4
linreg dlogrgnp
# dummy1 dummy2
set gdpadjust = %resids
prj means_from_gnp
*
* Remove a linear trend from unemployment
*
filter(remove=trend) lhmur / uradjust
set trend_from_ur = lhmur-uradjust
*
system(model=bqmodel)
var gdpadjust uradjust
lags 1 to 8
det constant
end(system)
*
estimate(noprint,resids=resids)
*
* Do the standard B-Q factor. By construction, this will have a long
* run loading of zero from the second shock ("demand") to the first
* dependent variable.
*
compute factor=%bqfactor(%sigma,%varlagsums)
*
* The zero restriction only determines the shape, not the sign, of
* the demand shock. If we want it defined so that the contemporaneous
```

```

* response of output is positive (that will be the (1,2) element in
* the factor), we need to flip the sign of the second column if the
* response has the wrong sign.
*
{
if factor(1,2)<0.0
  compute factor=factor*%diag(||1.0,-1.0||)
}
*
* Figures 1 and 2. Responses of output (1st variable) are accumulated.
*
@varirf(model=bqmodel,decomp=factor,steps=40,page=byshocks,$
  accum=||1||,variables=||"Output","Unemployment"||,$
  shocks=||"Supply","Demand"||)
*
history(model=bqmodel,factor=factor,results=histdecomp)
*
* Put the removed means back into the GNP growth, and the removed trend
* back into unemployment.
*
set histdecomp(1,1) = histdecomp(1,1)+means_from_gnp
set histdecomp(1,2) = histdecomp(1,2)+trend_from_ur
*
* Accumulate the GNP components and scale them back by .01
*
do j=1,3
  acc histdecomp(j,1)
  set histdecomp(j,1) = histdecomp(j,1)*.01
end do j
*
* Add back into the forecast component the final value of log gnp before
* the forecast period.
*
set histdecomp(1,1) = histdecomp(1,1)+logrgnp(1950:1)
*
* Add the base + effects of the supply shock to get the output without
* the demand shock.
*
set lessdemand = histdecomp(1,1)+histdecomp(2,1)
graph(footer="Figure 7. Output Fluctuations Absent Demand")
# lessdemand
*
@NBERCycles(peaks=peaks,trough=troughs)
*
* This gets the scale correct for the spikes showing the peaks and
* troughs.
*
set peaks    = .10*peaks
set troughs = -.10*troughs
*
graph(footer="Figure 8. Output Fluctuations Due to Demand",$
  ovcount=2,overlay=spike,ovsame) 3
# histdecomp(3,1)
# peaks    1950:2 * 2

```

```

# troughs 1950:2 * 2
*
* Repeat for unemployment
*
set lessdemand = histdecomp(1,2)+histdecomp(2,2)
graph(min=0.0,$
  footer="Figure 9. Unemployment Fluctuations Absent Demand")
# lessdemand
*
@NBERCycles(peaks=peaks,trough=troughs)
set peaks = 4.0*peaks
set troughs = -4.0*troughs
*
graph(ovcount=2,overlay=spike,ovsame,$
  footer="Figure 10. Unemployment Fluctuations Due to Demand") 3
# histdecomp(3,2)
# peaks 1950:2 * 2
# troughs 1950:2 * 2
*
* Construction of Table 1
*
@StructResids(factor=factor) resids / sresids
*
* As noted above, this is the one place where the difference between
* this data set and the one used in producing the paper is apparent.
*
report(action=define)
report(atrow=1,atcol=1,align=center) "Quarter" $
  "Demand(Percent)" "Supply(Percent)"
do time=1973:3,1975:1
  report(row=new,atcol=1) %datelabel(time) $
    sresids(2)(time) sresids(1)(time)
end do time
report(row=new)
do time=1979:1,1980:2
  report(row=new,atcol=1) %datelabel(time) $
    sresids(2)(time) sresids(1)(time)
end do time
report(action=format,picture="*.#")
report(action=show)

```

Example 6.2 Multi-Step Restrictions

This is an example of a structural SVAR with multiple-step constraint. This is based Application 14.6.1 from Martin et al. (2012), but converts the long-run restrictions to second-step restrictions (for illustration).

```

open data peersman_data.dat
calendar(q) 1970:1
data(format=prn,nolabels,org=columns) 1970:01 2002:02 oilprice outputemu $
  cpiemu rateemu outputus cpius rateus

```

```

*
set groil = 100.0*(log(oilprice/oilprice{1}))
set grout = 100.0*(log(outputus/outputus{1}))
set grp   = 100.0*(log(cpius/cpius{1}))
set r     = rateus
*
set trend = t
*
system(model=peersman)
variables groil grout grp r
lags 1 2 3
det constant trend
end(system)
*
estimate(sigma) 1979:5 *
*
* These need to be responses to unit shocks, hence the FACTOR=%IDENTITY(%NVAR)
*
impulse(model=peersman,factor=%identity(%nvar),steps=3,results=irfs)
*
* This will do constraints of the form
*
* Impact:
* . 0 0 0
* . . . 0
* . . . .
* . . . .
*
* First step:
* . . . .
* . . 0 0
* . . . .
* . . . .
*
compute step0=%xt(irfs,1)
compute step1=%xt(irfs,2)
*****
dec rect rr(0,0)
@IRFRestrict(irf=step0,variable=1,shock=2) rr
@IRFRestrict(irf=step0,variable=1,shock=3) rr
@IRFRestrict(irf=step0,variable=1,shock=4) rr
@IRFRestrict(irf=step0,variable=2,shock=4) rr
@IRFRestrict(irf=step1,variable=2,shock=3) rr
@IRFRestrict(irf=step1,variable=2,shock=4) rr
*
compute rp=%perp(rr)
dec vect theta(%cols(rp))
dec frml[rect] lrfml
*
frml lrfml = %vectorect(rp*theta,%nvar)
*
* Get guess values as closest to a Cholesky factor
*
compute [vect] theta=%ginv(rp)*%vec(%decomp(%sigma))

```

```

*
nonlin theta
cvmodel(b=lrfrml,factor=f,dmatrix=identity) %sigma
disp "Impact responses with multi-step restrictions" ###.### f
disp "Step one responses" ###.### step1*f

```

Example 6.3 Short- and Long-Run Restrictions with Error Bands

Example of a structural VAR with short- and long-run constraints with error bands. This is based upon Application 14.6.1 from Martin et al. (2012), but adds the calculation of error bands.

```

open data peersman_data.dat
calendar(q) 1970:1
data(format=prn,nolabels,org=columns) 1970:01 2002:02 oilprice outputemu $
  cpiemu rateemu outputus cpius rateus
*
set groil = 100.0*(log(oilprice/oilprice{1}))
set grout = 100.0*(log(outputus/outputus{1}))
set grp   = 100.0*(log(cpius/cpius{1}))
set r     = rateus
*
set trend = t
*
system(model=peersman)
variables groil grout grp r
lags 1 2 3
det constant trend
end(system)
*
estimate(sigma) 1979:5 *
dec rect lr(4,4) sr(4,4)
*
* Shock order is oil, AS, AD, money
*
input lr
. . . .
. . 0 0
. . . .
. . . .
input sr
. 0 0 0
. . . 0
. . . .
. . . .
*****
*
* FLongShort is a function to compute a long-and-short-run factorization
* given the current draw for sigma and the coefficients of the model.
* Note that this recomputes the MA sums given the lag sums at the draw

```



```

* for the coefficients, not the lag sum at the OLS estimates.
*
function FLongShort sigma model
type rect      FLongShort
type symmetric sigma
type model      model
*
local rect MASums
local rect f
*
compute MASums=inv(%ModelLagSums(model))
@shortandlong(factor=f,lr=lr,sr=sr,massum=MASums) sigma
*
* The sign fixing will be specific to an application. Here this makes
* shock 1 positive on variable 1 (oil price) (when you read this, the
* row in F is the target variable and the column is the shock), shock 2
* positive on variable 2 (output), shock 3 positive on variable 2 and
* shock 4 positive on variable 4 (interest rates). Use a - in front of
* the %sign to force an impact to be negative.
*
compute FLongShort=%dmult(f,$
    ||%sign(f(1,1)),%sign(f(2,2)),%sign(f(2,3)),%sign(f(4,4))||)
end
*****
*
* All but the first interest rates (variable 4) are in growth rates so
* we accumulate the responses to get (log) level effects.
*
@MCVARDoDraws(model=peersman,ffunction=FLongShort,$
    draws=2000,steps=40,accum=||1,2,3||)
*
@MCGraphIRF(model=peersman,$
    shocks=||"Oil Price","Supply Shock","Demand Shock","Money Shock"||,$
    variables=||"Oil Price","Output","Price","Interest Rate"||)

```

Sign Restrictions

Parametric SVAR's have, too frequently, been unable to produce models where shocks have the desired properties. An example of this is the standard Sims 6-variable model (Example 5.2), where money supply and money demand are poorly identified. And those two (in particular) don't lend themselves easily to identification through the types of zero restrictions that can be imposed by the semi-structural approach of Chapter 6.

Uhlig (2005) and Canova & De Nicolo (2002) independently proposed an even less-structured approach, in which the shock is identified by sign restrictions, satisfying the prior economic understanding of how a particular shock should behave. Our focus here will be on Uhlig's methodology.

7.1 Generating Impulse Vectors

In the previous chapter, we defined the term *impulse vector* and stated the key result from Uhlig:

Statement 4 If $PP' = \Sigma$, then the space of impulse vectors is precisely the set of $P\alpha$, where $\|\alpha\| = 1$.

Uhlig's sign-restriction approach is to look at the distribution of the space of impulse vectors, conditional on the requirement that the responses to those impulse vectors meet a particular set of sign restrictions. These restrictions will be something like: positive response of interest rates in the first four periods, negative response of money over the same period. In most cases, this can be done fairly easily by generating a random impulse vector, checking whether it meets the sign restrictions; if so, keep it, if not, reject it. This is a very simple example of the *acceptance-rejection method*¹ for generating random draws from a distribution. The output from this can then be processed by a procedure such as `@MCGraphIRF`. So long as a reasonable percentage (which can be as low as 1%) of randomly drawn impulse vectors meet the restrictions, this will work.

Given a chosen set of constraints, the method is as follows:

1. Generate a draw for the VAR coefficients and covariance matrix using standard methods.

¹Also sometimes called the *acceptance method* or the *rejection method*.

2. Compute a Cholesky factor (the arbitrary P factor in Statement 4) and the responses to it.
3. Generate a random unit vector (α) in m -space. This is the start of a “sub-draw”.
4. Weight the impulse responses from step 2 by α to get the responses to the chosen impulse vector.
5. If the impulse responses meet the restrictions, save them.
6. Repeat steps 3-5 a certain number of times for each main draw.
7. Repeat steps 1-6 until the desired number of draws have been accepted.

Because all that needs to be done for each sub-draw is to compute a weighted average of the impulse responses from step 2 and check whether they meet the restrictions, those are individually quite quick. The only slowdown here would be if the percentage of draws that meet the restrictions is small, requiring many sub-draws just to get one acceptance.

The simplest way to get the random unit vector is to draw an m -vector of independent $N(0, 1)$ and divide it by its norm. The RATS function %RANSHERE (m) does exactly that—chooses a random point on the unit sphere in \mathbb{R}^m .

To demonstrate the sub-draw procedure, we’ll look first at a preliminary calculation done in the Uhlig paper, which is to find the range of the sign-restricted IRF’s using only the OLS estimates. This is example 7.1.

This paper is based upon a six variable system, with real GDP, the GDP deflator, commodity price index, Federal funds rate (FFR), non-borrowed reserves and total reserves. A contractionary monetary shock is defined as one which causes the FFR to go up, the two price indices to go down and non-borrowed reserves also to go down.

The following set various limits:

```
compute nvar  =6
compute nsteps=60
compute nkeep =1000
compute n2    =50000
compute KMAX  =5
```

NSTEPS is the number of steps of responses that we want to compute. This is *not* the number that are being constrained; we use KMAX for that. NKEEP is the number of accepted draws that we want, and N2 is the upper bound on the number of draws that we will do in trying to get NKEEP.² This is slightly different from what Uhlig did, as he took N2=10000 and used as many draws as were accepted.

We need the IRF’s to a factor (any factor) of the covariance matrix, so we use the Cholesky factor.

²We’re using N2 here because this will be the inner or second loop in a more general procedure that also draws VAR coefficients.

```

compute atilde=%decomp(%sigma)
impulse(model=sixvar, steps=nsteps, results=impulses, $
        factor=atilde, noprint)

```

In this example, we need the minimum and maximum achieved across all accepted draws, and (for a separate graph) the complete record of the initial (impact) responses. In order to do the minimum and maximum, we initialize series (for each variable) with values of +large number for the minimum and -large number for the maximum.

```

dec vect[series] mini(nvar) maxi(nvar)
dec vect[series] impacts(nvar)
*
do i=1,nvar
    set impacts(i) 1 nkeep = %na
end do i
*
do i=1,nvar
    set mini(i) 1 nsteps = 1.e+10
    set maxi(i) 1 nsteps = -1.e+10
end do i

```

Inside the sub-draw loop, we do the following:

```

compute a=%ransphere(nvar)
compute atest=atilde*a
if atest(4)<0
    compute a=-1.0*a

```

The first draws the random direction (called *A*). Postmultiplying the factor by that gives the impulse vector. The final two lines are a (minor) time saver—you can make one sign restriction correct by force without changing the distribution. This is due to the symmetry with regard to sign in the draws for the impulse vectors, and will double the yield of the procedure. In this case, we're testing the 4th variable (FFR) and making its impact response positive.

The acceptance code is the following loop:

```

do k=1,KMAX+1
    compute ik=%xt(impulses,k)*a
    if ik(4)<0.or.ik(3)>0.or.ik(2)>0.or.ik(5)>0
        goto reject
    end do k

```

In Uhlig's paper(s), the *KMAX* is actually one greater than the number of responses constrained, since he is using horizons based at 0 rather than 1. This explains the *KMAX+1* in the *DO* loop. `%xt(impulses,k)` pulls the $nvar \times nvar$ matrix of responses to the Cholesky shocks at step *k*. Post-multiplying that by the weights *A* gives the responses at step *k* to this draw for the impulse vector.

We then check whether that meets the restrictions. In this case, we do the “rejection” logic, so we reject if it has the wrong sign on any of the components, on any of the steps between 1 and $KMAX+1$. `REJECT` is a location in the loop after the following bookkeeping for accepted draws:

```
gset oneresp 1 nsteps = %xt(impulses,t)*a
do i=1,nvar
  set maxi(i) 1 nsteps = %max(oneresp{0}(i),maxi(i))
  set mini(i) 1 nsteps = %min(oneresp{0}(i),mini(i))
end do i
compute accept=accept+1
compute %pt(impacts,accept,atilde*a)
```

This first computes a `SERIES[VECTORS]` called `ONERESP` which are the responses to the impulse vector across the full range of steps that we want. The loop updates the maximum and minimum series, replacing the old maximum with the max of the response to the current draw and the old maximum; similarly for the minimum. The final line saves the impacts.

We’ve skipped over the main loop instructions, which we’ll now describe, with ...’s being used to replace the working code described above.³

```
compute accept=0
infobox(action=define,progress,lower=1,upper=nkeep) $
  "Monte Carlo Integration"
do draws=1,n2
  ...calculations...
  if ...reject draw...
    goto reject
  ...bookkeeping...
  compute accept=accept+1
  infobox(current=accept)
  if accept==nkeep
    break
:reject
end do draws
infobox(action=remove)
```

The `INFOBOX` instructions are very handy with any potentially time-consuming calculation like this. This puts up a progress bar with an indicator running from 1 to `NKEEP`—this is updated each time we get an acceptance, so it won’t (necessarily) fill in at a steady pace. Because we hope that we’re overestimating the number of draws needed, we need a way out of the loop when we fill our quota of accepted draws. This is done with the `IF` near the bottom. Once done with the loop, we remove the info box.

³What we’re doing below is known as “pseudo-coding”, which is a good way to organize the writing of a relatively complicated process like this. You get the overall loop structure figured out, then put in the detailed calculations, rather than trying to do them together.

We follow the loop with the following:

```
{
  if accept<nkeep
    messagebox(style=alert) $
      "Only "+accept+" draws met restrictions. You wanted "+nkeep
  else
    messagebox(style=alert) $
      ""+nkeep+" draws accepted in "+draws+" attempts"
}
```

This stops the execution of the program (until you OK the alert box) to report on how well the restrictions were met. The top alert box is triggered if the process fails to create the required number of draws. The second, more informational one, appears if the process *did* generate the requested count. When you run this example, you should get 1000 on about 16000 draws, so the yield is roughly 6%, which isn't bad. If, on the other hand, you keep everything else the same but change `KMAX` to 30, you'll probably get about 200 accepted out of 50000 (thus giving you the first warning). While this would suggest that you can get 1000 good draws by upping the draw limit to somewhere around 250000, it is strongly suggesting that you're restricting the IRF too much.

Note that you need to enclose the **IF...ELSE** in `{ }` because this is otherwise outside of any loop, procedure or function. If you find the second **MESSAGEBOX** to be too annoying (since it's telling you things are OK), you can replace the **MESSAGEBOX(STYLE=ALERT)** with **DISPLAY**. We would not recommend doing that with the first **MESSAGEBOX**—when earlier versions of this merely reported the number of accepted draws, we had too many users read right past the fact that their model had basically failed to work.

The complete procedure for this approach requires a double loop; the outer loop draws a new covariance matrix and set of coefficients, while the inner loop is as above. For efficiency, this allows more than one sub-draw per VAR draw. We'll now explain the code that is new in Example 7.2, which does the full Uhlig sampler for sign restrictions:

```
compute nvar=6
compute nstep=60
compute nkeep=1000
compute n1=200
compute n2=200
compute KMIN=1
compute KMAX=5
```

Much of this is the same, except we now are defining `N1` as the number of outer draws. With the combination of outer and inner draws, this will do up to 40000 attempts to get 1000 keepers. The next is the standard setup code for doing draws from a VAR (Section 3.3):

```

compute fxx      =%decomp(%xx)
compute fwish    =%decomp(inv(%nobs*%sigma))
compute betaols=%modelgetcoeffs(varmodel)
compute ncoef    =%rows(fxx)
compute wishdof=%nobs-ncoef

```

The outer loop (in isolation) takes the pseudo-code form:

```

infobox(action=define,progress,lower=1,upper=n1) $
  "Monte Carlo Integration"
do draws=1,n1
  ...draw VAR...
  ...do subdraws...
  if accept>=nkeep
    break
  infobox(current=draws)
end do draws
infobox(action=remove)

```

Note that this could easily cut out long before it finishes $N1$ draws if the subdraw procedure gets a high enough yield.

The subdraw procedure is simplified by the use of the function `UhligAccept`. `UhligAccept(q,first,last,constrained)` returns a 1 for accept and a 0 for reject for the draw Q on the unit sphere. This is applied to the Cholesky draws in the global `RECT[SERIES] IMPULSES` that we compute as part of the draw for the VAR. The constraints are coded into the `CONSTRAINED` parameter as `+slot` for a positive constraint and `-slot` for a negative one. Since we want 4 (FFR) to be positive and 3, 2 and 5 (price indices and non-borrowed reserves) to be negative, the logic now can be written:

```

compute a=%ransphere(nvar)
if UhligAccept(a,KMIN,KMAX+1,||+4,-3,-2,-5||)==0
  goto reject

```

The `FIRST` and `LAST` parameters on the `UhligAccept` function are the range of horizons being constrained. In this case, we want these from `KMIN=1` to `KMAX+1` (keeping with Uhlig's description of K). See section 7.7 for more detail on `UhligAccept`.

This example also includes a calculation of the FEVD, or at least the fraction of the variance explained by this particular shock. Because the overall forecast variance doesn't depend upon the factorization used, this isn't that hard to compute. In the code for the VAR draw is the following:

```

gset irfsquared 1 1      = %xt(impulses,t).^2
gset irfsquared 2 nstep = irfsquared{1}+%xt(impulses,t).^2

```

`.^` is the elementwise power operator, so this generates running sums of squares of the Cholesky factor responses. `IRFSQUARED(t)` is an $nvar \times nvar$

matrix which has in each row, the total variance at horizon t due to each of the shocks. The sum across row i is the forecast error variance for series i at that step. On a keeper draw, the following finishes the calculation:

```
ewise goodfevd(accept) (i, j) = $
    ik=(irfsquared(i)*(a.^2))./(irfsquared(i)*ones), ik(j)
```

ONES is a vector of all 1's (defined earlier in the program), so the denominator is a vector with row sums (total variances). The numerator is the weighted sum using the squares of the alpha weights on the Cholesky factor shocks. `./` is the elementwise division operator so this divides (for each variable) the amount of variance due to the alpha shock by the total variance.

7.2 Penalty functions

It's possible that there will be no (at least very few) impulse vectors which satisfy a full set of restrictions. As an alternative to the acceptance-rejection method described in Section 7.1, Uhlig also proposes a penalty function approach. This maps the unit sphere (the weights for the impulse vectors) to a function which is smaller for impulse vectors which come "close" to satisfying the constraints. The minimizer of the penalty function across the unit sphere is chosen as the impulse vector for this draw for the VAR coefficients and covariance matrix.

In order to treat all failures of the constraints similarly, the penalty function has to be adjusted for the scale of the variables. In different papers, Uhlig has used slightly different scale factors, but they are always some type of estimate of a standard error. The penalty function is the sum across the constrained shocks of

$$f(x) = \begin{cases} x & \text{if } x \leq 0 \\ 100x & \text{if } x \geq 0 \end{cases}$$

where, for a given constraint, x is the rescaled (and sign flipped, if the constraint requires positivity) response for that variable and horizon. The second line penalizes (strongly) responses that have the wrong sign, while the first rewards (weakly) those that have the correct sign. The "reward", while not strictly in keeping with the idea of a penalty for incorrect signs, apparently helps in the optimization.

So we need to do an optimization across the unit sphere. There are many ways to parameterize the unit sphere in m space using $m - 1$ parameters, all of which have at least some flaw. Uhlig uses a generalization of polar coordinates, which suffers from a lack of uniqueness—many sets of values are equivalent. In our RATS code, we use the stereographic projection, which has the problem that a one in the final component can't really be represented⁴ since it requires all

⁴The stereographic projection of \mathbb{R}^{n-1} to the unit sphere in \mathbb{R}^n maps $x = \{x_1, \dots, x_{n-1}\}$ to $\left\{ \frac{2x_1}{\|x\|^2+1}, \dots, \frac{2x_{n-1}}{\|x\|^2+1}, \frac{\|x\|^2-1}{\|x\|^2+1} \right\}$

other components to be infinite. Since it's quite unlikely in practice that a true 1 in the final component is the optimizing value, this seems to be only a minor issue.

The one major problem with this is that it isn't necessarily a well-behaved optimization problem. First, the function isn't differentiable, so derivative-based optimization algorithms (like BFGS) can't be employed. There also can be multiple modes, particularly when a constrained response is close to zero anyway—when that happens, fairly minor changes can flip the response at a horizon from one side to the other of zero, and do the opposite to a nearby horizon. Uhlig's initial recommendation was to try two sets of guess values and simply reject the draw if the two came to different optima. A later paper Mountford & Uhlig (2009) did a much broader search for the global optimum at each draw, which can be quite time-consuming.

This is demonstrated in Example 7.3. We again define a convenience function (`UhligPenalty`) for calculating the penalty function. This takes exactly the same parameters as `UhligAccept`, and uses the Cholesky draws from `RECT[SERIES] IMPULSES` and also a global `VECTOR` called `SCALES` which has the scale factors to standardize the responses. Both `IMPULSES` and `SCALES` have to be set before you can use the function. See Section 7.7 for more detail on how the function works.

Uhlig (2005) uses the standard error of the first difference of each variable as the scale factors. That's computed using:

```
dec vect scales(6)
dec real func
compute fill=0
dofor i = gdpcl gdpdef cprindex fedfunds bognonbr totresns
  diff i / diff1
  stats(noprint) diff1
  compute fill=fill+1,scales(fill)=sqrt(%variance)
end dofor i
```

A later paper used the diagonal elements from the covariance matrix of the VAR, which can be done with the much simpler:

```
compute scales=%sqrt(%xdiag(%sigma))
```

Inside the draw loop, we have the same code for generating the VAR covariance matrix and coefficients, and the Cholesky factor impulse responses. Now comes the optimization code:

```

find(noprint) min func
    compute a=%stereo(g)
    compute func=UhligPenalty(a,KMIN,KMAX+1,||+4,-3,-2,-5||)
end find
compute testfunc=func,testbeta=%beta
*
* Try the minimization again, starting from a standard
* set of values
*
compute g=%ones(nvar-1,1)
find(noprint) min func
    compute a=%stereo(g)
    compute func=UhligPenalty(a,KMIN,KMAX+1,||+4,-3,-2,-5||)
end find
if abs(testfunc-func)>.001
    goto reject

```

As mentioned earlier, Uhlig's original paper did two optimizations from different guess values, rejecting the draw if they didn't agree. The optimization is over the `NVAR-1` vector `G`. This is transformed by the `%STEREO` function into a point on the `NVAR` unit sphere. The following instructions *before* the loop define this and make it the parameter set for the non-linear optimizer:

```

dec vect g(nvar-1)
compute g=%ones(nvar-1,1)
nonlin g

```

The first optimization in the loop will start at the converged values from the previous draw, while the second uses a vector of 1's in `G`. The latter is fairly arbitrary and unlikely to be all that close to the optimum, so if both end up at the same place, there's a good chance that that is the global optimum.

The default optimization for `FIND` is the simplex method, which is quite reliable, if a bit slow. If a broader search procedure is wanted (at the cost of even slower execution), it's the genetic method. We found that the following alternative worked on more difficult problems:

```

find(noprint,pmethod=genetic,piters=50,method=simplex) min func

```

In practice, this is *much* slower than two separate simplex optimizations, so you should use it only if the simplex optimizations are disagreeing quite often.

7.3 Multiple Shocks

Uhlig's approach can also be extended to allow for multiple orthogonal shocks each defined using sign restrictions. The first is done as described above. We now have to create the second shock from the space that's orthogonal (in the proper space) to the first. This can be done by working in " α " space, but what we'll describe uses `@ForcedFactor` instead, since it does most of the work.

Suppose that we've generated impulse vector \mathbf{I}_1 . If we compute

```
@forcedfactor(force=column) sigma i1 f
```

the final $m - 1$ columns of \mathbf{F} will span the space of shocks orthogonal to \mathbf{I}_1 . An orthogonal impulse vector can be generated by postmultiplying that submatrix of \mathbf{F} by an $m - 1$ unit vector. If we accept another impulse vector \mathbf{I}_2 from that, then

```
@forcedfactor(force=column) sigma i1~i2 f
```

will give us the final $m - 2$ columns of \mathbf{F} spanning the space of shocks orthogonal to *both* \mathbf{I}_1 and \mathbf{I}_2 , etc. In order to use `UhligAccept` or `UhligPenalty`, we need to back-transform draws (since the inputs to those are weights on the Cholesky factor \mathbf{P}), thus the complete process is something like:

```
compute [vector] v1=%ransphere(nvar)
if UhligAccept(v1,KMIN,KMAX,||+1,+3,+8,+9||)==0
    goto reject
compute i1=p*v1
@forcedfactor(force=column) sigmad i1 f
compute r2=inv(p)*%xsubmat(f,1,nvar,2,nvar)
compute [vector] v2=r2*%ransphere(nvar-1)
if UhligAccept(v2,KMIN,KMAX,||+4,-5,-6,-7||)==0
    goto reject
compute i2=p*v2
```

That can be repeated as many times as necessary to pick additional orthogonal shocks, each time adding columns to the second parameter in `@FORCEDFACTOR` (i_1 , then $i_1 \sim i_2$, then $i_1 \sim i_2 \sim i_3$, etc.).

7.3.1 FORCEDFACTOR and the QR Decomposition

If $\mathbf{P}\mathbf{P}' = \mathbf{F}\mathbf{F}' = \Sigma$, that is, if both \mathbf{P} and \mathbf{Q} are factors of Σ , then

$$\mathbf{P}\mathbf{P}' = \mathbf{F}\mathbf{F}' \Rightarrow (\mathbf{F}^{-1}\mathbf{P}) (\mathbf{F}^{-1}\mathbf{P})' = \mathbf{I} \Rightarrow \mathbf{P} = \mathbf{F}\mathbf{Q}, \mathbf{Q} = \mathbf{F}^{-1}\mathbf{P}$$

\mathbf{Q} is an *orthogonal* matrix, which (for real-valued arrays like these) can also be called a *unitary* matrix, one whose inverse is its transpose: each row (or column) is length 1 and each row (column) is orthogonal to each other row (column). Thus, if we take any factor of Σ , any *other* factor can be obtained by post-multiplying by a unitary matrix.

In Rubio-Ramirez et al. (2010), the authors recommend handling multiple sign restrictions by generating an $m \times m$ matrix of independent $N(0, 1)$ elements, taking its QR decomposition, and using the orthogonal \mathbf{Q} matrix generated by that to weight the columns of the reference factor of Σ (typically the Cholesky factor). According to their Theorem 9, this gives a uniform distribution across the orthogonal matrices, and will thus have a uniform distribution across all possible factors.

The procedure that we have described is, in effect, identical to this. The QR method is described as being more efficient (by only doing one matrix decomposition per draw), while `@FORCEDFACTOR` does one for each additional restriction on each draw, but that isn't really a "hot spot" in the calculation and if the restrictions are done in an order so that the one least likely to be met is done first, the sequential procedure will only occasionally even get beyond the first matrix decomposition anyway.

7.4 Zero Constraints

You can also constrain a shock to have a true zero response at a (small) number of horizons. However, you can't blindly pick unit vectors as above and filter them out, since the probability of hitting true zero on a linear combination is zero. Instead, you have to pick from a restricted subspace of the unit sphere.

If the constraint is that the (initial) impact is zero for one or more series, there's a simple way to do this. Since the factorization P is arbitrary, pick a Cholesky factorization that has the variables that you want to constrain listed first. Say we want to constrain the impact responses on variables 1 and 2. Draw an α from $m - 2$ space and add two zeros at the front.

```
compute alpha=zeros(2,1)~~%ransphere(nvar-2)
```

In greater generality, the restrictions can be handled with a shrewd use of `@FORCEDFACTOR`. Suppose that the restrictions can be written in the form $RP\alpha = 0$. R is dimension $r \times m$, where r is the number of restrictions and is assumed to be full (row) rank, P is an arbitrary factor of Σ that you've chosen based upon convenience.⁵ Write the desired factor in the form:

$$F = PQ = P[Q_1|Q_2] \quad (7.1)$$

where Q_1 will be dimension $m \times r$, and (by the properties of orthogonal matrices) $Q_1'Q_2 = 0$. We can rewrite the condition $RP\alpha = 0$ as $RPQ\alpha^* = 0$ so α^* becomes weights on the (still-to-be-determined) factor matrix PQ . Then

$$RPQ = RP[Q_1|Q_2] = [RPQ_1|RPQ_2] \quad (7.2)$$

Choose Q_1 to have the same span as $(RP)'$. Then

$$Q_1 = (RP)' \Pi \quad (7.3)$$

where Π is a non-singular $r \times r$ matrix. Since $Q_1'Q_2 = 0$, we get

$$RPQ = [(RP) (RP)' \Pi | 0] \quad (7.4)$$

Since $(RP) (RP)' \Pi$ is (by assumption and construction) a non-singular $r \times r$ matrix, and the last $m - r$ columns of RPQ are zero, the only way that $RPQ\alpha^* = 0$ is if the first r components of α^* are zero.

⁵In practice, it's easier to figure out what you want for RP than R in isolation.

To use `@FORCEDFACTOR` to handle most of the work in this, we need to force the first r columns to have the same span as the PQ_1 that we just computed. Since Q_1 needs the same span as $(RP)'$, PQ_1 has the same span as $P(RP)'$, so that's the matrix that we would input to `@FORCEDFACTOR`. If this is also the second or later shock being selected, we would need to combine it (using `~`) with the previously chosen shocks. A shock with the desired properties can then be sampled by post-multiplying the final $m - r$ columns of the factor matrix⁶ by a random unit vector.

It's important to note that, while the order is unimportant (except perhaps for efficiency) if you are doing multiple *unrestricted* shocks, that is no longer the case if you have multiple shocks with one or more having zero restrictions as well. It's easiest to see that if, for instance, you are doing a full set of four shocks in a four variable model. If one has a zero restriction, it simply *can't* be ordered last—if you've already chosen three shocks, there are no “degrees of freedom” left to choose that last shock. Thus, there is no way to force the zero restriction in addition to orthogonality to the first three shocks. It's possible for the restricted shock to be the first, second or third shock chosen, but the distribution will be different depending upon where you order it. This is unavoidable if you want a combination of multiple shocks with zero restrictions.

We kept the RP together through the derivation above because it's easiest to create the restriction matrix in that form, since $P\alpha$ can be computed as the responses to the (arbitrary) orthogonalized shocks which are generally needed for the sign-restriction calculations anyway. Suppose that we want the response of variable 1 to be zero in periods 1, 2 and 3. If the responses to the orthogonalized shocks with factor P are called $\{\Psi(k)\}$, then the response at step k to the vector of impact shocks α is $\Psi(k)\alpha$. The RP matrix for the constraints will be the first rows of $\Psi(1)$, $\Psi(2)$ and $\Psi(3)$. This can be constructed with:

```
compute KZERO=3
dec rect rp(KZERO,nvar)
ewise rp(i,j)=ix=%xt(impulses,i),ix(1,j)
```

(assuming, as we've done earlier, that `IMPULSES` are the `RESULTS` from an `IMPULSE` instruction applied to the Cholesky factor P .)

The factor matrix with the required structure is computed using

```
compute ir=p*tr(rp)
@forcedfactor(force=column) %sigma ir restrictedf
```

7.5 Fry-Pagan Critique

Fry & Pagan (2011) criticized the standard practice of VAR analysis with sign-restrictions on several grounds. The main argument is that the median of the

⁶ m less the number of columns in your input matrix to `@FORCEDFACTOR`, if you also needed orthogonality to previously chosen shocks.

“clouds” of accepted draws isn’t necessarily representative of the dynamics of any one response. That, of course, is true of almost any multi-dimensional process whose joint distribution is generated by simulation—we tend to describe the results of those with marginals, even though the modes of different parameters may not coincide. Where we suspect that (in particular) a pair of those parameters might have an odd joint distribution, we can do a contour graph or something similar.

Impulse responses, however, are a *very* high-dimensional space. If we graph an S -step IRF for responses to a single shock in an m dimensional VAR, we are showing the marginals of mS separate “parameters”. While a given draw is likely to be close to the same position in percentile terms for nearby horizons in a single response graph, there is no strong reason to believe that the same would be true when looking at different response graphs or at very different horizons.

Whether the Fry-Pagan suggestion of the *median target* IRF is particularly helpful, it’s very common for editors or referees to require that it be included as part of the analysis of sign-restricted models, so it’s important to understand how to compute it. Example 7.4 computes a median target response to the model from Uhlig (2005). One thing to note is that the Fry-Pagan response is computed by examining impulse vectors generated from the OLS estimates only—there is no “outer” loop drawing the covariance matrix and coefficients.

The first step is to draw a set of responses which meet the sign restrictions. Because there is no outer loop, the number of “subdraws” is increased dramatically: this plans to keep the first 1000 out of 50000 draws. (The original program has 200 subdraws inside 200 outer draws, so allows up to 40000).

```
compute n2=50000
compute nkeep=1000
```

Everything else for computing and saving draws is the same as in Example 7.2.

The median target shock is the one which minimizes the mean-squared deviations between itself and the median of the draws. This requires optimizing over the possible impulse vectors. Because each response to the shock is scale-dependent, it’s necessary to come up with some method to rescale the deviations to mitigate that effect. Fry and Pagan suggested using the standard deviation (computed separately for each response and horizon), but we would recommend using the interquartile range instead as a more robust estimator, since the distribution of responses can sometimes be fat-tailed. For each response and horizon, this computes the median (into `targets`), the 16% and 84%-iles (into `lower` and `upper`) and the reciprocal of the interquartile range (into `rescale`).

```

dec vect[series] targets(nvar)
dec vect[series] rescale(nvar)
dec vect[series] lower(nvar) upper(nvar)
do i=1,nvar
  set targets(i) 1 nstep = 0.0
  set rescale(i) 1 nstep = 0.0
  set lower(i) 1 nstep = 0.0
  set upper(i) 1 nstep = 0.0
  do k=1,nstep
    set work 1 accept = goodresp(t)(k,i)
    compute frac=%fractiles(work,||.25,.50,.75,.16,.84||)
    compute targets(i)(k)=frac(2)
    compute rescale(i)(k)=1.0/(frac(3)-frac(1))
    compute lower(i)(k)=frac(4)
    compute upper(i)(k)=frac(5)
  end do k
end do i

```

As with the penalty function approach, the median target response is found by minimizing across $m - 1$ space using the stereographic projection onto the unit circle in m space. In the code below `%xt(impulses,k)*a` is the k -step responses to all m variables to the impulse vector determined by the weights a . `%xt(targets,k)` are the medians of the cloud of responses (again, for all m variables) and `%xt(rescale,k)` are the scale factors. `%NORMSQR` returns the sum of squared elements of a vector so here it would be computing the contribution of the step- k responses to the objective function.

```

dec vect g(nvar-1)
compute g=%fill(nvar-1,1,1.0)
nonlin g
dec real gap
find(trace) min gap
  compute a=%stereo(g)
  compute gap=0.0
  do k=1,nstep
    compute gap=gap+%normsqr(%xt(rescale,k).*$
      (%xt(targets,k)-%xt(impulses,k)*a))
  end do k
end find

```

When the median target response is graphed with the upper and lower bounds, you get something like Figure 7.1. (It won't be exactly the same because this is optimized against simulated responses). Note well that there is no guarantee that the median target response will remain inside the bounds (which here are 16-84). Here, the only response where the median target isn't almost exactly where you would expect it is non-borrowed reserves. And also note that there is no guarantee that it will even meet all the *sign restrictions*. This is because it is optimized across the sum of a (possibly) very large number of deviations, so

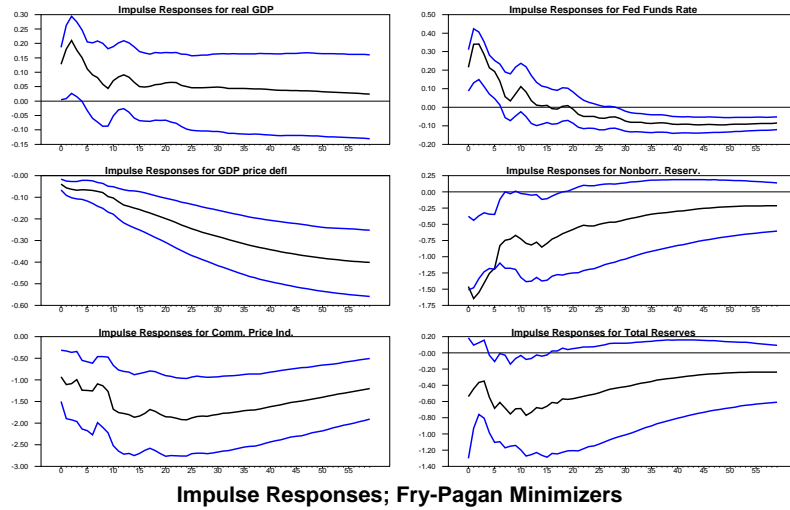


Figure 7.1: Impulse Responses: Fry-Pagan Minimizers

failing, even fairly badly, at a few may be best for that optimization. However, that may indicate a serious problem with your choice of restrictions.

7.6 Historical Decomposition

As we've seen, it's not hard to compute the impulse response function or the FEVD from isolated shocks. It's not quite so simple with the historical decomposition. Knowing how the isolated shock maps to the ε vector isn't enough information to transform an m -vector ε_t back into the scalar η_{1t} : a full rank F matrix is required for that. However, if we're requiring that (a scale multiple of) our shock be part of a factorization of a known matrix Σ , then η_{1t} will be invariant to the choice for completing that factorization.

Suppose that we start with a vector f . Then, by assumption, there is a scalar λ and an $m \times (m - 1)$ matrix H such that, if $F = [\lambda f | H]$, then $FF' = \Sigma$. If f is non-zero, such an H will always exist. Then

$$FF' = \Sigma \Rightarrow F^{-1} = F' \Sigma^{-1}$$

so

$$Fe(1)e(1)'F^{-1} = Fe(1)e(1)'F'\Sigma^{-1} = (\lambda f)(\lambda f)'\Sigma^{-1} \quad (7.5)$$

If G is any matrix so that $G\Sigma G' = I$ (equivalently $G'G = \Sigma^{-1}$), then

$$GFF'G' = I \Rightarrow (GF)(GF)' = I \quad (7.6)$$

Since

$$GF = [\lambda Gf | GH]$$

for (7.6) to hold, the first column must have length 1, so

$$\lambda^2 f' G' G f = 1 \Rightarrow \lambda^{-2} = f' \Sigma^{-1} f$$

Thus, the remapping matrix (7.5) from ε_t to $\varepsilon_t^{(1)}$ can be computed using only f and Σ , without need to compute the full factorization. In order to get the historical effects of the shock, we can just compute directly

$$\varepsilon_t^{(1)} = (\lambda f) (\lambda f)' \Sigma^{-1} \varepsilon_t \quad (7.7)$$

The base forecasts can be computed with **FORECAST**, and the forecasts with the $\varepsilon_t^{(1)}$ added in can be done with **FORECAST** with the **PATHS** option.

The calculation in (7.7) simplifies considerably when the shock is generated as a weighted sum of the columns of a factor of Σ . If $f = P\alpha$, then $\lambda = 1$, and

$$\varepsilon_t^{(1)} = (\lambda f) (\lambda f)' \Sigma^{-1} \varepsilon_t = P\alpha\alpha'P'\Sigma^{-1}\varepsilon_t = P\alpha\alpha'P^{-1}\varepsilon_t$$

The following shows the key calculations for this. It computes this for only one randomly selected unit vector (called A). `%xt(u,t)` extracts the OLS residual vector ε_t ; `%pt(upaths,t,vector)` fills into `upaths` at `t` the result $\varepsilon_t^{(1)}$. At the end, `history(1)` will be the historical response of the first series (in the model) to this shock, `history(2)` will be the response of the second, etc.

```
dec vect[series] upaths(nvar) base(nvar) baseplus(nvar) $
    history(nvar)
estimate(resids=u)
compute p=%decomp(%sigma)
compute a=%ransphere(nvar)
forecast(model=varmodel,from=hstart,to=hend,results=base)
compute remap=p*a*tr(a)*inv(p)
do t=hstart,hend
    compute %pt(upaths,t,remap*%xt(u,t))
end do t
forecast(model=varmodel,from=hstart,to=hend,$
    results=baseplus,paths)
# upaths
do i=1,nvar
    set history(i) hstart hend = baseplus(i)-base(i)
end do i
```

7.7 Convenience functions for sign restrictions

These are the functions on the `uhligfuncs.src` file that can be used for evaluating impulse vectors to see if they meet a set of sign restrictions.

`UhligAccept(q,first,last,constrained)` returns a 1 (accept) or 0 (reject) for the weight vector Q (applied to the impulse responses in the global `RECT[SERIES] IMPULSES`).

FIRST and **LAST** are the range of responses constrained (1 = impact response).

CONSTRAINED is a `VECT[INTEGER]` with the variable positions being constrained. Use `+slot` for a positivity constraint and `-slot` for a negativity con-

straint. That is, if you want the first 4 responses to be positive on the 2nd variable and negative on the 3rd, you would use the function

```
UhligAccept(q,1,4,||2,-3||)
```

If this returns 0, you would reject the draw, as not meeting the constraints.

```
function UhligAccept q first last constrained
type integer    UhligAccept
type vector     *q
type integer    first last
type vect[int]  constrained
*
local integer    i k
local real       func
local vector     ik
dec  rect[series] impulses
*
compute func=0.0
do k=first,last
  compute ik=%xt(impulses,k)*q
  do i=1,%rows(constrained)
    if constrained(i)<0
      compute value= ik(-constrained(i))
    else
      compute value=-ik(constrained(i))
    if value>0.0 {
      *
      * If we fail on the first slot being checked, just flip the sign
      * of q and ik and continue.
      *
      if k==first.and.i==1
        compute q=-1.0*q,ik=-1.0*ik
      else {
        compute UhligAccept=0
        return
      }
    }
  }
end do i
end do k
compute UhligAccept=1
end
```

UhligPenalty(q,first,last,constrained) returns the penalty function for the weight vector q (applied to the impulse responses in the global RECT[SERIES] impulses).

FIRST and LAST are the range of responses constrained (1 = impact response).

CONSTRAINED is a VECT[INTEGER] with the variable positions being constrained. Use +slot for a positivity constraint and -slot for a negativity constraint. That is, if you want the first 4 responses to be positive on the 2nd variable and negative on the 3rd, you would use the function

```
UhligPenalty(q,1,4,||2,-3||)
```

```
function UhligPenalty q first last constrained
type real      UhligPenalty
type vector    q
type integer    first last
type vect[int] constrained
*
local integer i k
local real      func
local vector    ik
dec  rect[series] impulses
dec  vector      scales
*
compute func=0.0
do k=first,last
  compute ik=(%xt(impulses,k)*q)./scales
  do i=1,%rows(constrained)
    if constrained(i)<0
      compute value= ik(-constrained(i))
    else
      compute value=-ik(constrained(i))
      compute func=func+%if(value<0.0,value,100*value)
    end do i
  end do k
compute UhligPenalty=func
end
```

Example 7.1 Sign Restrictions: Part I

These show some preliminary calculations from Uhlig (2005).

```
open data uhligdata.xls
calendar(m) 1965:1
data(format=xls,org=columns) 1965:1 2003:12 $
    gdpc1 gdpdef cprindex totresns bognonbr fedfunds
*
set gdpc1      = log(gdpc1)*100.0
set gdpdef     = log(gdpdef)*100.0
set cprindex   = log(cprindex)*100.0
set totresns   = log(totresns)*100.0
set bognonbr   = log(bognonbr)*100.0
*
system(model=sixvar)
variables gdpc1 gdpdef cprindex fedfunds bognonbr totresns
lags 1 to 12
end(system)
estimate(noprint)
*
dec vect[strings] vl(6)
compute vl=||"real GDP","GDP price defl","Comm. Price Ind.",$,
    "Fed Funds Rate","Nonborr. Reserv.,"Total Reserves"||
*
compute nvar   =6
compute nsteps=60
compute nkeep  =1000
compute n2     =50000
compute KMAX   =5
*
* Compute standard impulse response functions
*
compute atilde=%decomp(%sigma)
impulse(model=sixvar,steps=nsteps,results=impulses,$
    factor=atilde,noprint)
*
dec vect[series] mini(nvar) maxi(nvar)
dec series[vect] oneresp
dec vect a(nvar) atest ik
dec vect[series] impacts(nvar)
*
do i=1,nvar
    set impacts(i) 1 nkeep = %na
end do i
*
do i=1,nvar
    set mini(i) 1 nsteps = 1.e+10
    set maxi(i) 1 nsteps = -1.e+10
end do i
*
compute accept=0
infobox(action=define,progress,lower=1,upper=nkeep) $
```

```

"Monte Carlo Integration"
do draws=1,n2
*
* Take a draw from the unit sphere.
*
compute a=%ransphere(nvar)
*
* Transform back to the original variable space
*
compute atest=atilde*a
*
* Flip the sign if the FFR shock is negative
*
if atest(4)<0
compute a=-1.0*a

do k=1,KMAX+1
compute ik=%xt(impulses,k)*a
if ik(4)<0.or.ik(3)>0.or.ik(2)>0.or.ik(5)>0
goto reject
end do k
gset oneresp 1 nsteps = %xt(impulses,t)*a
do i=1,nvar
set maxi(i) 1 nsteps = %max(oneresp{0}(i),maxi(i))
set mini(i) 1 nsteps = %min(oneresp{0}(i),mini(i))
end do i
compute accept=accept+1
compute %pt(impacts,accept,atilde*a)
infobox(current=accept) %strval(100.0*accept/draws,"##.##")
if accept==nkeep
break
:reject
end do draws
infobox(action=remove)
{
if accept<nkeep
messagebox(style=alert) $
"Only "+accept+" draws met restrictions. You wanted "+nkeep
else
messagebox(style=alert) $
""+nkeep+" draws accepted in "+draws+" attempts"
}
*
spgraph(vfields=3,hfields=2,$
hlabel="Figure 2. Range of Impulse Responses with K=5")
do i=1,6
graph(nodates,number=0,picture="##.##",$
header="Impulse response for "+v1(i)) 2
# mini(i) / 1
# maxi(i) / 1
end do i
spgraph(done)
*
spgraph(vfields=3,hfields=2,$

```

```

    hllabel="Figure 3. Distribution of Impact Impulse Response")
do i=1,6
    density(type=hist,counts,maxgrid=20) impacts(i) / fx dx
    set dx = 100.0*dx/accept
    scatter(style=bar,vmin=0.0,vmax=12.0,$
        header="Impulse response for "+v1(i))
    # fx dx
end do i
spgraph(done)
*
* Standard impulse response functions (figure 5)
*
impulse(model=sixvar,steps=60,results=impulses,noprint)
*
spgraph(vfields=3,hfields=2,$
    hllabel="Figure 5. Responses to a Monetary Policy Shock")
do i=1,6
    graph(nodates,number=0,picture="##.##",$
        header="Impulse response for "+v1(i))
    # impulses(i,4)
end do i
spgraph(done)

```

Example 7.2 Sign Restrictions: Part II

This is the pure sign-restriction approach from Uhlig (2005).

(same setup as Example 7.1)

```

source uhligfuncs.src
*
dec vect[strings] v1(6)
compute v1=||"real GDP","GDP price defl","Comm. Price Ind.",$
    "Fed Funds Rate","Nonborr. Reserv.,""Total Reserves"||
*
* n1 is the number of draws from the posterior of the VAR
* n2 is the number of draws from the unit sphere for each draw
* for the VAR
* nvar is the number of variables
* nstep is the number of IRF steps to compute
* KMAX is the "K" value for the number of steps constrained
*
compute n1=200
compute n2=200
compute nkeep=1000
compute nvar=6
compute nstep=60
compute KMIN=1
compute KMAX=5
*
* This is the standard setup for MC integration of an OLS VAR
*

```

```

compute fxx      =%decomp(%xx)
compute fwish    =%decomp(inv(%nobs*%sigma))
compute betaols=%modelgetcoeffs(varmodel)
compute ncoef    =%rows(fxx)
compute wishdof=%nobs-ncoef
*
* Most draws are going to get rejected. We allow for up to nkeep good
* ones. The variable accept will count the number of accepted draws.
* GOODRESP will be a RECT(nsteps,nvar) at each accepted draw.
*
declare vect[rect] goodresp(nkeep) goodfevd(nkeep)
declare vector ik a(nvar) ones(nvar)
declare series[rect] irfsquared
compute ones=%const(1.0)
*
compute accept=0
infobox(action=define,progress,lower=1,upper=n1) $
  "Monte Carlo Integration with subdraws"
do draws=1,n1
  *
  * Make a draw from the posterior for the VAR and compute its
  * impulse responses.
  *
  compute sigmad  =%ranwisharti(fwish,wishdof)
  compute fsigma  =%decomp(sigmad)
  compute betau   =%ranmvkron(fsigma,fxx)
  compute betadraw=betaols+betau
  compute %modelsetcoeffs(varmodel,betadraw)
  impulse(noprint,model=varmodel,decomp=fsigma,$
    results=impulses,steps=nstep)
  gset irfsquared 1 1      = %xt(impulses,t).^2
  gset irfsquared 2 nstep = irfsquared{1}+%xt(impulses,t).^2
  *
  * Do the subdraws over the unit sphere. These give the weights
  * on the orthogonal components.
  *
  do subdraws=1,n2
    compute a=%ransphere(nvar)
    *
    * Check that the responses have the correct signs for steps
    * 1 to KMAX+1 (+1 because in the paper, the steps used 0-based
    * subscripts, rather than the 1 based subscripts used by RATS).
    *
    if UhligAccept(a,KMIN,KMAX+1,||+4,-3,-2,-5||)==0
      goto reject
    *
    * This is an accepted draw. Copy the information out. If we
    * have enough good ones, drop out of the overall loop.
    *
    compute accept=accept+1
    dim goodresp(accept)(nstep,nvar) goodfevd(accept)(nstep,nvar)
    ewise goodresp(accept)(i,j)=ik=%xt(impulses,i)*a,ik(j)
    ewise goodfevd(accept)(i,j)=$
      ik=(irfsquared(i)*(a.^2))./(irfsquared(i)*ones),ik(j)

```

```

        if accept>=nkeep
            break
        :reject
    end do subdraws
    if accept>=nkeep
        break
    infobox(current=draws) $
        "Acceptance rate: "+%strval(100.0*accept/(draws*n2),"##.##")
end do draws
infobox(action=remove)
{
if accept<nkeep
    messagebox(style=alert) "Warning. Had only "+accept+" draws out of "+n1*n2
}
*
* Post-processing. Graph the mean of the responses along with the 16-
* and 84-%iles.
*
clear upper lower resp
*
spgraph(vfields=3,hfields=2,$
    footer="Figure 6. Impulse Responses with Pure-Sign Approach")
do i=1,nvar
    smpl 1 accept
    do k=1,nstep
        set work = goodresp(t)(k,i)
        compute frac=%fractiles(work,||.16,.84||)
        compute lower(k)=frac(1)
        compute upper(k)=frac(2)
        compute resp(k)=%avg(work)
    end do k
    *
    smpl 1 nstep
    graph(ticks,number=0,picture="##.##", $
        header="Impulse Responses for "+vl(i)) 3
    # resp
    # upper / 2
    # lower / 2
    smpl
end do i
*
spgraph(done)

spgraph(vfields=3,hfields=2,footer=$
    "Figure 8. Fraction of Variance Explained with Pure-Sign Approach")
do i=1,nvar
    compute minlower=maxupper=0.0
    smpl 1 accept
    do k=1,nstep
        set work = goodfevd(t)(k,i)
        compute frac=%fractiles(work,||.16,.50,.84||)
        compute lower(k)=frac(1)
        compute upper(k)=frac(3)
        compute resp(k)=frac(2)
    end do k
    *
    smpl
end do i
*
spgraph(done)

```



```

end do k
*
smpl 1 nstep
graph(ticks,number=0,min=0.0,picture="##.##",$
      header="Fraction Explained for "+v1(i)) 3
# resp
# upper / 2
# lower / 2
smpl
end do i
*
spgraph(done)

```

Example 7.3 Sign Restrictions: Part III

This is the penalty function approach from Uhlig (2005).

(same setup as the previous examples)

```

source uhligfuncs.src
*****
dec vect[strings] vl(6)
compute vl=|"real GDP","GDP price defl","Comm. Price Ind.",$
           "Fed Funds Rate","Nonborr. Reserv.,","Total Reserves"|
*
* Get the scale factors to be used for impulse responses
*
dec vect scales(6)
dec real func
compute fill=0
dofor i = gdpcl gdpdef cprindex fedfunds bognonbr totresns
  diff i / diff1
  stats(noprint) diff1
  compute fill=fill+1,scales(fill)=sqrt(%variance)
end dofor i
*
* nkeep is the desired number of draws from the posterior of the VAR
* nvar is the number of variables
* nstep is the number of IRF steps to compute
* KMAX is the "K" value for the number of steps constrained
*
compute nkeep=1000
compute nvar=6
compute nstep=60
compute KMIN=1
compute KMAX=5
declare vect[rect] goodresp(nkeep)
declare vector ik a(nvar)
*
* This uses the stereo projection of  $R^{(n-1)}$  to the unit sphere in  $R^n$ .
*
dec vect g(nvar-1)

```

```

compute g=%fill(nvar-1,1,1.0)
nonlin g
*
* This is the standard setup for MC integration of an OLS VAR
*
compute sxx     =%decomp(%xx)
compute svt     =%decomp(inv(%nobs*%sigma))
compute betaols=%modelgetcoeffs(varmodel)
compute ncoef   =%rows(sxx)
compute wishdof=%nobs-ncoef
*
infobox(action=define,progress,lower=1,upper=nkeep) $
  "Monte Carlo Integration"
*
* Plan for double the number of draws to allow for the ones which we
* reject.
*
compute accept=0
do draws=1,nkeep*2
  *
  * Make a draw from the posterior for the VAR and compute its
  * impulse responses.
  *
  compute sigmad =%ranwisharti(svt,wishdof)
  compute swish  =%decomp(sigmad)
  compute betau  =%ranmvkron(swish,sxx)
  compute betadraw=betaols+betau
  compute %modelsetcoeffs(varmodel,betadraw)
  impulse(noprint,model=varmodel,decomp=swish,$
    results=impulses,steps=nstep)
  *
  * Minimize the penalty function, starting from the last set of
  * minimizers.
  *
  find(noprint) min func
    compute a=%stereo(g)
    compute func=UhligPenalty(a,KMIN,KMAX+1,||+4,-3,-2,-5||)
  end find
  compute testfunc=func,testbeta=%beta
  *
  * Try the minimization again, starting from a standard
  * set of values,
  *
  compute g=%fill(nvar-1,1,1.0)
  find(noprint) min func
    compute a=%stereo(g)
    compute func=UhligPenalty(a,KMIN,KMAX+1,||+4,-3,-2,-5||)
  end find
  *
  * If the two estimates don't match, reject the draw. If they do,
  * copy out the impulse responses.
  *
  if abs(testfunc-func)>.001
    goto reject

```

```

compute accept=accept+1
dim goodresp(accept) (nstep,nvar)
ewise goodresp(accept) (i,j)=ik=%xt(impulses,i)*a,ik(j)
*
* If we've hit out desired number of accepted draws, break
*
if accept>=nkeep
    break
infobox(current=accept)
:reject
end do draws
infobox(action=remove)
*
* Post-processing. Graph the mean of the responses along with the
* 16% and 84%-iles.
*
clear upper lower resp
*
spgraph(vfields=3,hfields=2,hlabel=$
"Figure 14. Impulse Responses with Penalty Function Approach")
do i=1,nvar
    compute minlower=maxupper=0.0
    smpl 1 accept
    do k=1,nstep
        set work = goodresp(t)(k,i)
        compute frac=%fractiles(work,||.16,.84||)
        compute lower(k)=frac(1)
        compute upper(k)=frac(2)
        compute resp(k)=%avg(work)
    end do k
    *
    smpl 1 nstep
    graph(ticks,number=0,picture="##.##",$
        header="Impulse Responses for "+v1(i)) 3
    # resp
    # upper / 2
    # lower / 2
    smpl
end do i
*
spgraph(done)

```

Example 7.4 Sign Restrictions: Median Target Method

This demonstrates the Median Target method from Fry & Pagan (2011) applied to the model from Uhlig (2005) used in Example 7.2.

(same setup as the previous examples)

```

source uhligfuncs.src
*

```

```

dec vect[strings] vl(6)
compute vl=||"real GDP","GDP price defl","Comm. Price Ind.",$
    "Fed Funds Rate","Nonborr. Reserv.,""Total Reserves"||
*
* n2 is the number of draws from the unit sphere for each draw for the VAR
* nvar is the number of variables
* nstep is the number of IRF steps to compute
* KMAX is the "K" value for the number of steps constrained
*
compute n2=50000
compute nkeep=1000
compute nvar=6
compute nstep=60
compute KMIN=1
compute KMAX=5
*
* Most draws are going to get rejected. We allow for up to nkeep good
* ones. The variable accept will count the number of accepted draws.
* GOODRESP will be a RECT(nsteps,nvar) at each accepted draw.
*
declare vect[rect] goodresp(nkeep)
declare vector ik a(nvar) ones(nvar)
*
impulse(noprint,model=varmodel,factor=%decomp(%sigma),$
    results=impulses,steps=nstep)
compute accept=0
infobox(action=define,progress,lower=1,upper=n2) $
    "Monte Carlo Integration"
do subdraws=1,n2
    *
    * Do the subdraws over the unit sphere. These give the weights on the
    * orthogonal components.
    *
    compute a=%ransphere(nvar)
    *
    * Check that the responses have the correct signs for steps 1 to
    * KMAX+1 (+1 because in the paper, the steps used 0-based
    * subscripts, rather than the 1 based subscripts used by RATS).
    *
    if UhligAccept(a,KMIN,KMAX+1,||+4,-3,-2,-5||)==0
        goto reject
    *
    * This is an accepted draw. Copy the information out. If we have
    * enough good ones, drop out of the overall loop.
    *
    compute accept=accept+1
    dim goodresp(accept)(nstep,nvar)
    ewise goodresp(accept)(i,j)=ik=%xt(impulses,i)*a,ik(j)
    infobox(current=subdraws)
    if accept>=nkeep
        break
:reject
end do subdraws
infobox(action=remove)

```

```

*
* Compute Fry-Pagan minimizers. This uses the interquartile range of the
* responses at each horizon rather than standard error for rescaling the
* difference between a test response and the median.
*
dec vect[series] targets(nvar)
dec vect[series] rescale(nvar)
dec vect[series] lower(nvar) upper(nvar)
do i=1,nvar
  set targets(i) 1 nstep = 0.0
  set rescale(i) 1 nstep = 0.0
  set lower(i) 1 nstep = 0.0
  set upper(i) 1 nstep = 0.0
  do k=1,nstep
    set work 1 accept = goodresp(t)(k,i)
    compute frac=%fractiles(work,|.25,.50,.75,.16,.84|)
    compute targets(i)(k)=frac(2)
    compute rescale(i)(k)=1.0/(frac(3)-frac(1))
    compute lower(i)(k)=frac(4)
    compute upper(i)(k)=frac(5)
  end do k
end do i
*
* This does the optimization. It parameterizes the unit circle using the
* stereographic mapping. The function being minimized is the sum of
* squared standardized gaps between the impulse responses given the test
* rotation and median sign-restricted response where the standardization
* is 1/IQR.
*
dec vect g(nvar-1)
compute g=%fill(nvar-1,1,1.0)
nonlin g
dec real gap
find(trace) min gap
  compute a=%stereo(g)
  compute gap=0.0
  do k=1,nstep
    compute gap=gap+%normsqr(%xt(rescale,k).*$
      (%xt(targets,k)-%xt(impulses,k)*a))
  end do k
end find
*
* Graph with 16-84% bounds. Note that there is no reason the Fry-Pagan
* minimizing IRF has to be within the bounds at all horizons since it's
* chosen to minimize the sum across (in this case) 360 squared
* deviations from the median responses.
*
compute a=%stereo(g)
set resp 1 nstep = 0.0
spgraph(vfields=3,hfields=2,$
  hlabel="Impulse Responses; Fry-Pagan Minimizers")
do i=1,nvar
  set resp 1 nstep = ik=%xt(impulses,t)*a,ik(i)
  graph(ticks,number=0,picture="##.##",$

```

```
        header="Impulse Responses for "+v1(i)) 3
    # resp      1 nstep
    # lower(i)  1 nstep 2
    # upper(i)  1 nstep 2
end do i
*
spgraph(done)
```

Probability Distributions

A.1 Multivariate Normal

Parameters	Mean (μ), Covariance matrix (Σ) or precision (H)
Kernel	$ \Sigma ^{-1/2} \exp \left(-\frac{1}{2} (x - \mu)' \Sigma^{-1} (x - \mu) \right)$ or $ H ^{1/2} \exp \left(-\frac{1}{2} (x - \mu)' H (x - \mu) \right)$
Support	\mathbb{R}^n
Mean	μ
Variance	Σ or H^{-1}
Main uses	Distribution of multivariate error processes. Asymptotic distributions. Prior, exact and approximate posteriors for a collection of parameters with unlimited ranges.
Density Function	<code>%LOGDENSITY(sigma,u)</code> . To compute $\log f(x \mu, \Sigma)$ use <code>%LOGDENSITY(sigma,x-mu)</code> . (The same function works for univariate and multivariate Normals).
Distribution Function	<code>%BICDF(x,y,rho)</code> returns $P(X \leq x, Y \leq y)$ for a bivariate Normal with mean zero, variance 1 in each component and correlation rho.
Random Draws	<code>%RANMAT(m,n)</code> draws a matrix of independent $N(0, 1)$. <code>%RANMVNORMAL(F)</code> draws an n -vector from a $N(0, FF')$, where F is any factor of the covariance matrix. This setup is used (rather than taking the covariance matrix itself as the input) so you can do the factor just once if it's fixed across a set of draws. To get a single draw from a $N(\mu, \Sigma)$, use <code>MU+%RANMVNORMAL(%DECOMP(SIGMA))</code> <code>%RANMVPOST</code> , <code>%RANMVPOSTCMOM</code> , <code>%RANMVKRON</code> and

`%RANMVKRONCMOM` are specialized functions which draw multivariate Normals with calculations of the mean and covariance matrix from other matrices.

A.2 Chi-Squared Distribution

Parameters	Degrees of freedom (ν).
Kernel	$x^{(\nu-2)/2} \exp(-x/2)$
Range	$[0, \infty)$
Mean	ν
Variance	2ν
Main uses	Asymptotic distribution. Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model.
Density function	<code>%CHISQRDENSITY(x, nu)</code> is the (non-logged) density with <code>nu</code> degrees of freedom at <code>x</code> .
Tail Probability	<code>%CHISQR(x, nu)</code> returns the probability that a chi-squared with <code>nu</code> degrees of freedom exceeds <code>x</code> .
Inverse Tail Probability	<code>%INVCHISQR(p, nu)</code> returns the critical value for probability p .
Random Draws	<code>%RANCHISQR(nu)</code> draws one or more (depending upon the target) independent chi-squareds with <code>NU</code> degrees of freedom.

A.3 (Scaled) Inverse Chi-Squared Distribution

Parameters	Degrees of freedom (ν) and scale (τ^2). An inverse chi-squared is the reciprocal of a chi-squared combined with scaling factor which represents a “target” variance that the distribution is intended to represent. (Note that the mean is roughly τ^2 for large degrees of freedom.)
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a / \Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{\nu\tau^2}{\nu-2}$ if $\nu > 2$
Variance	$\frac{2\nu^2\tau^4}{(\nu-2)^2(\nu-4)}$ if $\nu > 4$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. The closely-related inverse gamma (Appendix A.5) can be used for that as well, but the scaled inverse chi-squared tends to be more intuitive.
DensityFunction	<code>%LOGINVCHISQRDENSITY(x, nu, tausq)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvChisqrParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((ν, τ^2) in that order) for the parameters of an inverse chi-squared with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $\nu = 4$, which is the largest value of ν which gives an infinite variance.
Random draws	You can use <code>(nu*tausq)/%ranchisqr(nu)</code> . Note that you divide by the random chi-squared.

A.4 Gamma Distribution

Parameters	shape (a) and scale (b), alternatively, degrees of freedom (ν) and mean (μ). The RATS functions use the first of these. The relationship between them is $a = \nu/2$ and $b = \frac{2\mu}{\nu}$. The chi-squared distribution with ν degrees of freedom is a special case with $\mu = \nu$.
Kernel	$x^{a-1} \exp\left(-\frac{x}{b}\right)$ or $x^{(\nu/2)-1} \exp\left(-\frac{x\nu}{2\mu}\right)$
Support	$[0, \infty)$
Mean	ba or μ
Variance	b^2a or $\frac{2\mu^2}{\nu}$
Main uses	Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model
Density function	<code>%LOGGAMMADENSITY(x, a, b)</code> . For the $\{\nu, \mu\}$ parameterization, use <code>%LOGGAMMADENSITY(x, .5*nu, 2.0*mu/nu)</code>
Random Draws	<code>%RANGAMMA(a)</code> draws one or more (depending upon the target) independent Gammas with unit scale factor. Use <code>b*%RANGAMMA(a)</code> to get a draw from $Gamma(a, b)$. If you are using the $\{\nu, \mu\}$ parameterization, use <code>2.0*mu*%RANGAMMA(.5*nu)/nu</code> . You can also use <code>mu*%RANCHISQR(nu)/nu</code> .
Moment Matching	<code>%GammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((a, b) parameterization) for a gamma with the given mean and standard deviation.

A.5 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 A.3 with parameters ν (degrees of freedom) and τ^2 (scale parameter) which has $a = \nu/2$ and $b = \nu\tau^2/2$).
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a/\Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{b}{(a-1)}$ if $a > 1$
Variance	$\frac{b^2}{(a-1)^2(a-2)}$ if $a > 2$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. For these purposes, it's usually simpler to directly use the scaled inverse chi-squared variation.
Density Function	<code>%LOGINVGAMMADENSITY(x, a, b)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvGammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters (a, b) parameterization) for the parameters of an inverse gamma with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $a = 2$, which is the largest value of a which gives an infinite variance.
Random draws	You can use <code>b/%rangamma(a)</code> . A draw from a scaled inverse chi-squared is typically done with <code>nu*tausqr/%ranchisqr(nu)</code> .

A.6 Wishart Distribution

Parameters	Scaling \mathbf{A} (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and \mathbf{A} is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}\left(\mathbf{A}^{-1}\mathbf{X}\right)\right) \mathbf{X} ^{\frac{1}{2}(\nu-n-1)}$
Support	Positive definite symmetric matrices
Mean	$\nu\mathbf{A}$
Main uses	Prior, exact and approximate posterior for the precision matrix (inverse of covariance matrix) of residuals in a multivariate regression, though that is mainly in the inverse form (Appendix A.7) since that would be the distribution of the covariance matrix itself.
Random Draws	<p><code>%RANWISHART (n, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{I}$ and degrees of freedom ν.</p> <p><code>%RANWISHARTF (F, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{F}\mathbf{F}'$ and degrees of freedom ν. \mathbf{F} can be any factor of \mathbf{A}, but would typically be computed as the Cholesky factor using <code>%DECOMP</code>.</p>

A.7 Inverse Wishart Distribution

Parameters	Scaling Ψ (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and Ψ is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}(\Psi\mathbf{X}^{-1})\right) \mathbf{X} ^{-\frac{1}{2}(\nu+n+1)}$
Support	Positive definite symmetric matrices
Mean	$\frac{1}{\nu-n-1}\Psi$
Main uses	Prior, exact and approximate posterior for the covariance matrix of residuals in a multivariate regression with Gaussian residuals.
Diffuse versions	The density function is improper if $\nu < n - 1$, but the improper prior with $\nu = 0$ and $\Psi = 0$ has kernel $ \mathbf{X} ^{-\frac{1}{2}(n+1)}$ which forms the Jeffrey's prior for inference on the covariance matrix.
Combining Densities	If $\mathbf{X} \sim IW(\Psi_1, \nu_1)$ and $\mathbf{X} \sim IW(\Psi_2, \nu_2)$ are inverse Wishart densities for \mathbf{X} , then the posterior from combining them has $\mathbf{X} \sim IW(\Psi_1 + \Psi_2, \nu_1 + \nu_2 + n + 1)$.
Random Draws	<code>%RANWISHARTI(F, nu)</code> draws a single $n \times n$ inverse Wishart matrix with $\mathbf{F}\mathbf{F}' = \Psi^{-1}$ and degrees of freedom ν . Note that \mathbf{F} needs to be a factor of the <i>inverse</i> . \mathbf{F} can be any factor matrix, but is typically the Cholesky factor, computed using <code>%DECOMP</code> .
Notes	The basic result has the data evidence on the covariance matrix of Gaussian residuals summarized as an inverse Wishart with $\Psi = T\hat{\Sigma}$, where T is the number of observations and $\hat{\Sigma}$ the sample covariance matrix of residuals (thus Ψ itself is the sum of the outer products of the residuals). The degrees of freedom for the inverse Wishart from the data itself are typically $T - (n + 1)$ (sometimes less some additional adjustments for regressors, depending upon the form of conditioning). The $n + 1$ is needed because you don't really have the ability to estimate a covariance matrix until you have that many observations. Combining data with the Jeffrey's prior "corrects" the degrees of freedom so the posterior value of $\nu = T$.

An informative prior is generally based upon a prior belief on the value of \mathbf{X} . Because \mathbf{X} is typically the covariance matrix Σ , call this Σ_0 . The corresponding value of Ψ for this is $\alpha\Sigma_0$ where the prior degrees of freedom are $\alpha + n + 1$. Combined with data, this gives an inverse Wishart with $T + \alpha$ degrees of freedom and Ψ matrix which is $T\hat{\Sigma} + \alpha\Sigma_0$, thus (roughly) sample and non-sample information on Σ weighted by the number of actual and “dummy” observations.

Appendix B

VAR Likelihood Function

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.

It's most convenient to write this in terms of the precision matrix $\mathbf{H} \equiv \Sigma^{-1}$. 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

$$y_t = \mathbf{B} x_t' + \varepsilon_t \tag{B.1}$$

where x_t is the k vector of explanatory variables, \otimes is the Kroneker product, β 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 β , that is, the first k elements of β 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 β , while the second will be better for analyzing the precision matrix, and for computing the likelihood function. For analyzing β , 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 β :

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

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

$$\begin{aligned} \sum (\mathbf{I}_m \otimes x_t)' \mathbf{H} y_t &\Rightarrow \sum (\mathbf{H} \otimes x_t)' y_t \Rightarrow \\ \sum (\mathbf{H} \otimes \mathbf{I}_k) (\mathbf{I}_m \otimes x_t)' (y_t \otimes \mathbf{I}_1) &\Rightarrow \sum (\mathbf{H} \otimes \mathbf{I}_k) (y_t \otimes x_t') \end{aligned} \quad (\text{B.2})$$

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 β is

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

and the mean is

$$\hat{\beta} = \left(\mathbf{H} \otimes \sum x_t' x_t \right)^{-1} (\mathbf{H} \otimes \mathbf{I}_k) \sum (y_t \otimes x_t') = \left(\mathbf{I}_m \otimes \sum x_t' x_t \right)^{-1} \sum (y_t \otimes x_t') \quad (\text{B.3})$$

Since $\sum (y_t \otimes x_t')$ is just a vec'ed 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} . If we *didn't* have identical explanatory variables, that is, if we have a more 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 (B.2).

The classical (non-Bayesian) result here is that the maximum likelihood estimator is OLS equation by equation with covariance matrix

$$\hat{\mathbf{H}}^{-1} = \Sigma \otimes \left(\sum x_t' x_t \right)^{-1} \quad (\text{B.4})$$

If we use the matrix form (B.1), we can rewrite the likelihood as:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{B}, \mathbf{H}) \propto |\mathbf{H}|^{T/2} \exp \left(-\frac{1}{2} \text{trace} \mathbf{H} (T \times \Sigma(\mathbf{B})) \right) \quad (\text{B.5})$$

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} . For the classical result, we can take the log of this to get

$$\frac{T}{2} \log |\mathbf{H}| - \frac{T}{2} \text{trace } \mathbf{H} \Sigma(\mathbf{B}) \quad (\text{B.6})$$

The derivative of $\log |\mathbf{H}|$ with respect to its elements is \mathbf{H}'^{-1} and the derivative of $\text{trace } \mathbf{H} \Sigma(\mathbf{B})$ with respect to the elements of \mathbf{H} is just $\Sigma(\mathbf{B})$. Since \mathbf{H} is symmetric, the solution for the first order conditions is $\mathbf{H}^{-1} = \Sigma(\mathbf{B})$ or $\Sigma = \Sigma(\mathbf{B})$. For a Bayesian analysis of this, part of (B.5) 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 $\left| \sum x'_t x_t \right|$ 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 β 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 (B.5) as:

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

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

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

This is a Wishart with $T - k$ degrees of freedom and scale matrix $(T \times \Sigma(\hat{\mathbf{B}}))^{-1}$.

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. What we end up needing for all other purposes is actually $\Sigma \equiv \mathbf{H}^{-1}$, so we’ll actually use the `%RANWISHARTI` function, which draws an inverse Wishart.

Properties of Multivariate Normals

The density function for a jointly Normal random n -vector \mathbf{x} with mean μ and covariance matrix Σ 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 μ and Σ . 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 β , 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 β 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 β . Thus, the posterior for β 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.

Deriving the Schwarz criterion

We'll take a look at how the Schwarz criterion is derived. The Bayesian approach to deciding between hypotheses H_1 and H_2 given data Y is to formulate likelihood functions $f_1(Y|\Theta_1)$ and $f_2(Y|\Theta_2)$, with priors $g_1(\Theta_1)$ and $g_2(\Theta_2)$ over the respective parameter spaces. The two hypotheses themselves have prior probabilities $P(H_1)$ and $P(H_2)$.

Let ℓ_1 be the loss in choosing H_2 if H_1 is true and ℓ_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

$$\frac{P(Y|H_1)}{P(Y|H_2)} > \frac{\ell_2 P(H_2)}{\ell_1 P(H_1)} \quad (\text{D.1})$$

The left side of this is known as the *Bayes factor*, which summarizes the data evidence regarding the two hypotheses. The Schwarz criterion aims at approximating the Bayes factor for large samples.

If we look at

$$P(Y|H_1) = \int f_1(Y|\Theta_1)g_1(\Theta_1)d\Theta_1$$

we see that the integrand is the product of the likelihood, which changes with the data set size, and the prior, which doesn't. We expect (under standard regularity conditions) that the likelihood will become more and more peaked. As long as the prior is non-zero and continuous near that peak, the posterior should also become peaked in the same region.

The biggest problem in applying Bayesian methods in general is that while it may be possible to write down the posterior as an explicit function, the integrals such as the above aren't tractable. A common attack on it is to expand the log of the posterior density around its mode. Under the proper conditions, we can approximate this by a Taylor expansion:

$$\log f(Y|\Theta)g(\Theta) \approx \log f(Y|\Theta_M)g(\Theta_M) - \frac{1}{2}(\Theta - \Theta_M)'J(\Theta - \Theta_M)$$

where Θ_M is the maximizer of the posterior. Since

$$\int \exp\left(-\frac{1}{2}(\Theta - \Theta_M)'J(\Theta - \Theta_M)\right)d\Theta = (2\pi)^{k/2}|J|^{-1/2}$$

(regardless of Θ_M), the data evidence can be approximated by

$$\int f(Y|\Theta)g(\Theta)d\Theta \approx f(Y|\Theta_M)g(\Theta_M)(2\pi)^{k/2}|J|^{-1/2}$$

Under the regularity conditions for asymptotic normality of maximum likelihood, $\frac{1}{T}J \rightarrow \mathcal{I}$ (again, because the likelihood will dominate the posterior). So, since $|J| = |\frac{1}{T}J| T^k$

$$\log P(Y|H) \approx \log f(Y|\Theta_M) + \log g(\Theta_M) + \frac{k}{2} \log 2\pi - \frac{k}{2} \log T - \frac{1}{2} \log |\mathcal{I}|$$

The second, third and fifth terms do not depend upon T , so will be asymptotically negligible, leaving just $\log P(Y|H) \approx \log f(Y|\Theta_M) - \frac{k}{2} \log T$. And since, in large samples, the mode of the posterior will be almost the same as the mode of the likelihood, we get the final simplification to $\log P(Y|H) \approx \log f(Y|\hat{\Theta}) - \frac{k}{2} \log T$. Multiplying this by T gives the SBC, which is smaller for the hypothesis favored by the data. Since this should be $O(T)$, the decision between the hypotheses will be dominated by the Bayes factor, and not the fixed values on the right side of (D.1).

Delta method

The *delta method* is used to estimate the variance of a non-linear function of a set of already estimated parameters. The basic result is that if θ are the parameters and we have

$$\sqrt{T} \left(\hat{\theta} - \theta \right) \xrightarrow{d} N(0, \Sigma_{\theta}) \quad (\text{E.1})$$

and if $f(\theta)$ is continuously differentiable, then, by using a first order Taylor expansion

$$\left(f(\hat{\theta}) - f(\theta) \right) \approx f'(\theta) \left(\hat{\theta} - \theta \right)$$

Reintroducing the \sqrt{T} scale factors and taking limits gives

$$\sqrt{T} \left(f(\hat{\theta}) - f(\theta) \right) \xrightarrow{d} N \left(0, f'(\theta) \Sigma_{\theta} f'(\theta)' \right)$$

In practice, this means that if we have

$$\hat{\theta} \approx N(\theta, \mathbf{A}) \quad (\text{E.2})$$

then

$$f \left(\hat{\theta} \right) \approx N \left(f(\theta), f'(\hat{\theta}) \mathbf{A} f'(\hat{\theta})' \right) \quad (\text{E.3})$$

(E.1) is the type of formal statement required, since the \mathbf{A} in (E.2) collapses to zero as $T \rightarrow \infty$. It's also key that (E.1) implies that $\hat{\theta} \xrightarrow{p} \theta$, so $f'(\hat{\theta}) \xrightarrow{p} f'(\theta)$ allowing us to replace the unobservable $f'(\theta)$ with the estimated form in (E.3). So the point estimate of the function is the function of the point estimate, at least as the center of the asymptotic distribution. If $\hat{\theta}$ is unbiased for θ , then it's almost certain that $f(\hat{\theta})$ will *not* be unbiased for $f(\theta)$ so this is *not* a statement about expected values.

To compute the asymptotic distribution, it's necessary to compute the partial derivatives of f . For scalar functions of the parameters estimated using a RATS instruction, that can usually be most easily done using the instruction **SUMMARIZE**.

Gibbs Sampling and Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) techniques allow for generation of draws from distributions which are too complicated for direct analysis. The simplest form of this is Gibbs sampling. This simulates draws from the density by means of a correlated Markov Chain. Under the proper circumstances, estimates of sample statistics generated from this converge to their true means under the actual posterior density.

Gibbs Sampling

The idea behind Gibbs sampling is that we partition our set of parameters into two or more groups: for illustration, we'll assume we have two, called Θ_1 and Θ_2 . We want to generate draws from a joint density $f(\Theta_1, \Theta_2)$, but don't have any simple way to do that. We can always write the joint density as $f(\Theta_1|\Theta_2)f(\Theta_2)$ and as $f(\Theta_2|\Theta_1)f(\Theta_1)$. In a very wide range of cases, these conditional densities *are* tractable. It's the unconditional densities of the *other* block that are the problem.

The intuition behind Gibbs sampling is that if we draw Θ_1 given Θ_2 , and then Θ_2 given Θ_1 , the pair should be closer to their unconditional distribution than before. Each combination of draws is known as a *sweep*. Repeat sweeps often enough and it should converge to give draws from the joint distribution. This turns out to be true in most situations. Just discard enough at the beginning (called the *burn-in* draws) so that the chain has had time to converge to the unconditional distribution. Once you *have* a draw from $f(\Theta_1, \Theta_2)$, you (of necessity) have a draw from the marginals $f(\Theta_1)$ and $f(\Theta_2)$, so now each sweep will give you a new draw from the desired distribution. They're just not *independent* draws. With enough "forgetfulness", however, the sample averages of the draws will converge to the true mean of the joint distribution.

Gibbs sampling works best when parameters which are (strongly) correlated with each other are in the same block, otherwise it will be hard to move both since the sampling procedure for each is tied to the other.

Metropolis-Hastings/Metropolis within Gibbs

Metropolis within Gibbs is a more advanced form of MCMC. Gibbs sampling requires that we be able to generate the draws from the conditional densities. However, there are only a handful of distributions for which we can do that—things like Normals, gammas, Dirichlets. In many cases, the desired density

is the likelihood for a complicated non-linear model which doesn't have such a form.

Suppose, we want to sample the random variable θ from $f(\theta)$ which takes a form for which direct sampling is difficult.¹ Instead, we sample from a more convenient density $q(\theta)$. Let $\theta^{(i-1)}$ be the value from the previous sweep. Compute

$$\alpha = \frac{f(\theta)}{f(\theta^{(i-1)})} \times \frac{q(\theta^{(i-1)})}{q(\theta)} \quad (\text{F.1})$$

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

The first ratio in (F.1) makes perfect sense. We want, as much as possible, to have draws where the posterior density is high, and not where it's low. The second counterweights (notice that it's the ratio in the opposite order) for the probability of drawing a given value. Another way of looking at the ratio is

$$\alpha = \frac{f(\theta)}{q(\theta)} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)})}$$

f/q is a measure of the relative desirability of a draw. The ones that really give us a strong "move" signal are where the target density (f) is high and the proposal density (q) is low—we may not see those again, so when we get a chance we should move. Conversely, if f is low and q is high, we might as well stay put—we may revisit that one at a later time.

What this describes is *Independence Chain Metropolis*, where the proposal density doesn't depend upon the last draw. Where a set of parameters is expected to have a single mode, a good proposal density often can be constructed from maximum likelihood estimates, using a Normal or multivariate t centered at the ML estimates, with the covariance matrix some scale multiple (often just 1.0) of the estimate coming out of the maximization procedure.

If the actual density has a shape which *isn't* a good match for a Normal or t , this is unlikely to work well. If there are points where f is high and q is relatively low, it might take a very large number of draws before we find them, and once we get there we'll likely stay for a while. A more general procedure has the proposal density depending upon the last value: $q(\theta|\theta^{(i-1)})$. The acceptance criterion is now based upon:

$$\alpha = \frac{f(\theta)}{q(\theta|\theta^{(i-1)})} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)}|\theta)} \quad (\text{F.2})$$

¹What we're describing here is Metropolis-Hastings or M-H for short. In all our applications, the densities will be conditional on the other blocks of parameters, which is formally known as Metropolis within Gibbs.

Note that the counterweighting is now based upon the ratio of the probability of moving from $\theta^{(i-1)}$ to θ to the probability of moving back. The calculation simplifies greatly if the proposal is a mean zero Normal or t added to $\theta^{(i-1)}$. Because of symmetry, $q(\theta|\theta^{(i-1)}) = q(\theta^{(i-1)}|\theta)$, so the q cancels, leaving just:

$$\alpha = f(\theta) / f(\theta^{(i-1)})$$

This is known as *Random Walk Metropolis*, which is probably the most common choice for Metropolis within Gibbs. Unlike Independence Chain, when you move to an isolated area where f is high, you can always move back by, in effect, retracing your route.

Avoiding Overflows

The f that we've been using in this is either the sample likelihood or the posterior (sample likelihood times prior). With many data sets, the sample *log* likelihood is on the order of 100's or 1000's (positive or negative). If you try to convert these large log likelihoods by taking the exp, you will likely overflow (for large positive) or underflow (for large negative), in either case, losing the actual value. Instead, you need to calculate the ratios by adding and subtracting in log form, and taking the exp only at the end.

Diagnostics

There are two types of diagnostics: those on the behavior of the sampling methods for subsets of the parameters, and those on the behavior of the overall chain. When you use Independence Chain Metropolis, you would generally like the acceptance rate to be fairly high. In fact, if $q = f$ (which means that you're actually doing a simple Gibbs draw), everything cancels and the acceptance is $\alpha = 1$. Numbers in the range of 20-30% are generally fine, but acceptance rates well below 10% are often an indication that you have a bad choice for q —your proposal density isn't matching well with f . When you use Random Walk Metropolis, an acceptance probability near 100% *isn't* good. You will almost never get rates like that unless you're taking very small steps and thus not moving around the parameter space sufficiently. Numbers in the 20-40% range are usually considered to be desirable. You can often tweak either the variance or the degrees of freedom in the increment to move that up or down. Clearly, in either case, you need to count the number of times you move and compare with the total number of draws.

If you're concerned about the overall behavior of the chain, you can run it several times and see if you get similar results. If you get decidedly different results, it's possible that the chain hasn't run long enough to converge, requiring a greater number of burn-in draws. The `@MCMCPOSTPROC` procedure also computes a CD measure which does a statistical comparison of the first part of the accepted draws with the last part. If the burn-in is sufficient, these should be asymptotically standard Normal statistics (there's one per parameter). If you

get values that have absolute values far out in the tails for a Normal (such as 4 and above), that's a strong indication that you either have a general problem with the chain, or you just haven't done a long enough burn-in.

Pathologies

Most properly designed chains will *eventually* converge, although it might take many sweeps to accomplish this. There are, however, some situations in which it won't work no matter how long it runs. If the Markov Chain has an *absorbing state* (or set of states), once the chain moves into this, it can't get out. This is particularly common with switching models, where once the probability of a regime is low enough you get no data points which actually are considered to fall into it, therefore the probability is pushed even more towards zero.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

`<https://fsf.org/>`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated

herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or

XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that

carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together

- with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
 - D. Preserve all the copyright notices of the Document.
 - E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
 - F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
 - G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
 - H. Include an unaltered copy of this License.
 - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
 - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
 - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
 - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document

does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliography

- Bernanke B 1986 *Carnegie-Rochester Conference Series on Public Policy* **25**(1), 49–99.
- Bernanke B & Mihov I 1998 *Quarterly Journal of Economics* **113**(3), 869–902.
- Blanchard O & Quah D 1989 *American Economic Review* **79**(4), 655–673.
- Burbidge J & Harrison A 1985 *Journal of Monetary Economics* **16**(1), 45–54.
- Canova F & De Nicro G 2002 *Journal of Monetary Economics* **49**(6), 1131–1159.
- Diebold F X & Yilmaz K 2012 *International Journal of Forecasting* **28**(1), 57–66.
- Fry R & Pagan A 2011 *Journal of Economic Literature* **49**(4), 938–960.
- Gali J 1992 *Quarterly Journal of Economics* **107**(2), 709–738.
- Hamilton J 1994 *Time Series Analysis* Princeton: Princeton University Press.
- Kilian L 1998 *Review of Economics and Statistics* **80**(2), 218–230.
- King R G, Plosser C I, Stock J & Watson M 1991 *American Economic Review* **81**(4), 819–40.
- Luetkepohl H 1990 *Review of Economics and Statistics* **72**(1), 116–125.
- Luetkepohl H 2006 *New Introduction to Multiple Time Series* Berlin: Springer.
- Martin V, Hurn S & Harris D 2012 *Econometric Modelling with Time Series: Specification, Estimation and Testing* Cambridge: Cambridge University Press.
- Mountford A & Uhlig H 2009 *Journal of Applied Econometrics* **24**(6), 960–992.
- Pesaran M H & Shin Y 1998 *Economics Letters* **58**(1), 17–29.
- Rubio-Ramirez J F, Waggoner D F & Zha T 2010 *Review of Economic Studies* **77**(2), 665–696.
- Sims C 1980a *American Economic Review* **70**(2), 250–7.
- Sims C 1980b *Econometrica* **48**(1), 1–48.

Sims C 1986 *FRB Minneapolis Quarterly Review* pp. 2–16.

Sims C & Zha T 1999 *Econometrica* **67**(5), 1113–1156.

Tsay R 1998 *Journal of American Statistical Association* **93**(443), 1188–1202.

Uhlig H 2005 *Journal of Monetary Economics* **52**, 381–419.

Waggoner D F & Zha T 2003 *Journal of Economic Dynamics and Control* **28**(2), 349–366.

Index

- ~ operator, 103
- Absorbing state, 166
- AIC, 4
- Akaike Information criterion, 4
- Antithetic acceleration, 36
- Bayes factor, 160
- Bernanke, B., 65
- %BETASYS matrix, 9
- BIC, 4
- %BICDF function, 145
- Blanchard, O., 101
- BOOT** instruction, 33
 - BLOCK option, 43
- Bootstrapping, 33
- %BQFACTOR function, 103
- Burn-in, 163
- Canova, F., 116
- Chi-squared distribution, 147, 148
- Cholesky factorization, 21
- Contemporaneous correlation, 14
- Counterfactual simulation, 53
- CVMODEL** instruction, 62
- De Nicolo, G., 116
- %DECOMP function, 35
- Decomposition of variance, 15
- Delta method, 30, 162
- Diebold, F. X., 58
- %DMULT function, 109
- EIGEN** instruction, 56
- ERRORS** instruction, 24
- ESTIMATE** instruction, 7
- FEVD, 23
- FIND** instruction, 124
- @**FORCEDFACTOR** procedure, 102
- Fry, R., 127
- Gali, J., 105
- Gamma distribution, 149, 150
- General to specific, 5
- Generalized impulse responses, 57
- Gibbs sampling, 163
- Hannan-Quinn criterion, 4
- Historical decomposition, 51
- HISTORY** instruction, 52
- HQ, 4
- Impact responses, 16
- Importance sampling, 71
- IMPULSE** instruction, 16
- Impulse vector, 101
- Independence chain M-H, 164
- INFOBOX** instruction, 119
- Information criteria, 4
- Inverse Wishart distribution, 152
- @**IRFRESTRIC**T procedure, 106
- Kilian, L., 34
- @**KilianBootSetup** procedure, 34
- %LOGDENSITY function, 145
- M-H, *see* Metropolis-Hastings
- MAR, 14
- Markov Chain Monte Carlo, 163
- @**MCGraphIRF** procedure, 38
- MCMC, *see* Markov Chain Monte Carlo
- @**MCMCPOSTPROC** procedure, 165
- @**MCPROCESSIRF** procedure, 40
- @**MCVARDoDraws** procedure, 36
- Median target method, 128
- Metropolis-Hastings, 164
- Mihov, I., 65
- MODEL data type, 36
- %MODELCOMPANION function, 32
- %MODELGETCOEFFS function, 42
- %MODELLABEL function, 42
- %MODELLAGSUMS function, 103
- %MODELLAGSUMS function, 42

- `%MODELSETCOEFFS` function, 42
- Mountford, A., 123
- Moving average representation, 14
- Multivariate Normal distribution, 145, 157
- Normal distribution
 - multivariate, 145, 157
- `%NREG` variable, 8
- `%NREGSYSTEM` variable, 8
- `%NVAR` variable, 9
- Orthogonal matrix, 125
- Orthogonalization, 21
- Orthogonalized innovations, 15
- Overflow, 165
- Pagan, A., 127
- Parametric bootstrap, 33
- `PARMSET`, 84
- `%PARMSPEEK` function, 84
- `%PARMSPOKE` function, 84
- `PATHS` option, 34
- Pesaran, H., 57
- Precision, 158
- Precision matrix, 35
- Probability distributions
 - chi-squared, 147
 - gamma, 149
 - inverse chi-squared, 148
 - inverse gamma, 150
 - inverse Wishart, 152
 - multivariate normal, 145, 157
 - Wishart, 151
- Pseudo-code, 119
- `%PT` function, 131
- Quah, D., 101
- Random walk M-H, 165
- `%RANMAT` function, 145
- `%RANMVKRON` function, 36
- `%RANMVNORMAL` function, 145
- `%RANSPPHERE` function, 117
- `%RANWISHART` function, 151
- `%RANWISHARTF` function, 151
- `%RANWISHARTI` function, 35, 152, 156
- `RESIDUALS` option, 9
- `%%RESPONSES` matrix, 37
- Rubio-Ramirez, J., 125
- SBC, 4
- Schwarz Bayesian criterion, 4
- Shin, Y., 57
- `@SHORTANDLONG` procedure, 104
- `%SIGMA` matrix, 9
- `%SIGN` function, 109
- Sims, C., 62
- Standardized shocks, 19
- `%STEREO` function, 124
- `@StructResids` procedure, 67
- Structural residuals, 67
- Sweep
 - in MCMC, 163
- Uhlig, H., 101, 116, 123
- `UhligAccept` function, 121
- `UhligPenalty` function, 123
- Underflow, 165
- Unitary matrix, 125
- `@VARBootDraw` procedure, 33
- `@VARBootSetup` procedure, 33
- `@VARIRF` procedure, 22
- `@VARIRFDelta` procedure, 32
- `@VARLagSelect` procedure, 6
- `%VARLAGSUMS` matrix, 103
- Waggoner, D., 81, 125
- Wishart distribution, 151
 - inverse, 152
- `%XT` function, 131
- `%XX` matrix, 9
- Yilmaz, K., 58
- `@YuleVAR` procedure, 10
- Zha, T., 62, 81, 125