Chapter 3

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

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

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

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

The delta method is fairly commonly used in conventional maximum likelihood estimation, primarily when a particular parameterization is used for computational purposes. For instance, it's sometimes convenient 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 and 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:

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

where

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

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

[1] Nonlinear 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**. 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
```

[2]The 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
```

[3] Both 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.

3.3 Monte Carlo Integration

Monte Carlo integration is by far the most common method used to assess the statistical significance of the results generating 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 posterion.

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.2 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

[4] It'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.

scale matrix for the Wishart is $\left(T \times \Sigma(\hat{\mathbf{B}})\right)^{-1}$. The quickest way to factor a matrix is the Choleski 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:

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

While this is potentially a huge matrix (6 variables, 12 lags + constant is 432×432), 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.[5] 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 instead 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).

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

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

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

[5] The 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 larger matrix would take over 200 times as long as inverting the smaller one.

For most applications, you can use the **MCVARDoDraws** procedure, or a minor adaptation of it. 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 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 x shock x horizon), but RATS doesn't support that, so they're saved as (variable x shock) x horizon.

In **MCVARDoDraws**, the impulse responses are calculated with:

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

This does Choleski 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 do unit shocks, the only change you make is to replace the FACTOR option with FACTOR=%IDENTITY(NVAR). We would suggest that you make a copy of the original **MCVarDoDraws** before making a change like that.

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]

In the example in this section, this is

```
@MCVARDoDraws(model=varmodel,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, percentiles bands give a better picture in practice.

3.4 RATS Tips and Tricks

The function %CLOCK

%clock(draw,base) is like a "mod" function, but gives values from 1 to base, so %clock(draw,2) will be 1 for odd values of draw and 2 for even values.

The %MODEL function family

A MODEL is an object which organizes a collection of linear equations or formulas. The family of functions with names beginning with %MODEL take information out of and put it into a **MODEL**. These are all included in the *Model* category of the *Functions* wizard.

The most important of these apply to the case where the model is a set of linear equations like a VAR. For those, you can get and reset the coefficients of all equations at one time:

%modelgetcoeffs(model) returns a matrix of coefficients, with each column giving the coefficients in one of the equations.

%modelsetcoeffs(model,b) resets the coefficients of all equations in the model. The input matrix b can either have the same form as the one returned by %modelgetcoeffs (one column per equation) or can be a single "vec'ed" coefficient vector, with equation stacked on top of equation.

`%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.

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 bootdraws=2000
compute nvar  =3
compute nsteps=8
declare vect[rect] %%responses
dim %%responses(bootdraws)
declare rect[series] impulses(nvar,nvar)
declare vect ix
*
infobox(action=define,progress,lower=1,upper=bootdraws) $
   "Bootstrap Simulations"
do draw=1,bootdraws
   @VARBootDraw(model=varmodel,resids=resids) rstart rend
   *
   * Estimate the model with resampled data
   *
   estimate(noprint,noftests)
   impulse(noprint,model=bootvar,factor=%identity(3),$
     results=impulses,steps=nsteps)
   *
   * Store the impulse responses
   *
   dim %%responses(draw)(nvar*nvar,nsteps)
   ewise %%responses(draw)(i,j)=ix=%vec(%xt(impulses,j)),ix(i)
   infobox(current=draw)
end do draw
infobox(action=remove)
*
@mcgraphirf(model=varmodel,center=median,percent=||.025,.975||,$
  shocks=||"Investment","Income","Consumption"||,$
  footer="95% other-percentile bootstrap bands")
```

Example 3.3 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
*
* This uses a specialized MCVARDoDraws procedure which does unit shocks
* rather than Choleski shocks.
*
source mcvardodrawsunit.src
@MCVARDoDraws(model=varmodel,draws=2000,steps=8)
@mcgraphirf(model=varmodel,center=median,percent=||.025,.975||,$
  shocks=||"Investment","Income","Consumption"||,$
  footer="95% Monte Carlo bands")
```