Chapter 6

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 x is a non-zero $m \times 1$ vectoror, then

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

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

Proof. For (a), factor $\Sigma = \mathbf{P}\mathbf{P}'$ (any factor will do). Generate an orthonormal matrix V with a scale of $\mathbf{P}'$x as the first column. PV is the desired factor. For (b), generate an orthonormal matrix U with a scale multiple of $\mathbf{P}^{-1}$x as the first column. Then PU 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, 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 λx 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 $\mathbf{PP}' = \mathbf{\Sigma}$, 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 and 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, Plosser, Stock, and Watson (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 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 `I1` and `I2`.[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 $\mathrm{BB}' = \Sigma$. The short-run constraints are that specific elements of B are zero. The long-run constraints are that specific elements of $\Psi(1)\mathrm{B}$ 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 Choleski 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,masums=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=R Perp 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,masums=masums,$
   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,iters=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 $\Psi_s\mathrm{B}$ 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:

$$\mathbf{R}vec(\mathbf{B}) = 0 \tag{6.1}$$

That, combined with $\mathrm{BB}' = \Sigma$ 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:

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

and one-step ahead constraints of

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

(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 **@IRFRESTRICT**. 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"

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

where θ is a VECTOR of free parameters and then converts the vectorized 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] lrfrml
*
frml lrfrml = %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 Choleski 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
cvmodel(b=lrfrml,factor=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 $\mathrm{BB}' = \Sigma$ 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 and 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-meaned 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, Hurn, and Harris (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] lrfrml
*
frml lrfrml = %vectorect(rp*theta,%nvar)
*
* Get guess values as closest to a Choleski 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, Hurn, and Harris (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,masum=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"||)
```