The SVAR package

Jack Lucchetti

March 14, 2013

Contents

1 Introduction

The `SVAR` package is a collection of gretl scripts to estimate Structural VARs, or SVARs for short.

A bit of notation first[1]: we call "structural" a model in which we assume that the one-step-ahead prediction errors ε_t from a statistical model can be thought of as linear functions of the *structural shocks* u_t. In its most general form, a structural model is the pair of equations

$$\begin{aligned} \varepsilon_t &= y_t - E(y_t|\mathcal{F}_{t-1}) \quad (1) \\ A\varepsilon_t &= Bu_t \quad (2) \end{aligned}$$

In practically all cases, the statistical model is a a finite-order VAR and equation (1) specialises to

$$y_t = \mu' x_t + \sum_{i=1}^{p} A_i y_{t-i} + \varepsilon_t \qquad \text{or} \qquad A(L)y_t = \mu' x_t + \varepsilon_t \quad (3)$$

where the VAR may include an exogenous component x_t, which typically contains at least a constant term. The above model is referred to as the AB-model in Amisano-Giannini (1997).

The object of estimation are the square matrices A and B. Since the u_t are assumed mutually incorrelated with unit variance, the following relation must hold:

$$A\Sigma A' = BB' \quad (4)$$

If we define C as $A^{-1}B$, the relationship between prediction errors and structural shocks becomes

$$\varepsilon_t = Cu_t \quad (5)$$

and equation (4) can be written as

$$\Sigma = CC'.$$

This model can also arise when $A = I$ by assumption, in which case it is called a C model.

The matrix Σ can be consistently estimated via the covariance matrix of VAR residuals, but estimation of A and B is impossible unless some constraints are imposed on both matrices: $\hat{\Sigma}$ contains $\frac{n(n+1)}{2}$ distinct entries; clearly, the attempt to estimate $2n^2$ parameters violates an elementary order condition.

In Sims's (1980) original proposal, it was implicitly assumed that $A = I$ and B was lower triangular, so estimating B would just involve the Cholesky decomposition of $\hat{\Sigma}$. In general, however, one may wish to achieve identification by other means.[2]

The most immediate way to place enough constraints on the A and B matrices so to achieve identification is to specify a system of linear constraints; in other words, the restrictions on A and B take the form

$$\begin{aligned} R_a \operatorname{vec} A &= d_a \quad (6) \\ R_b \operatorname{vec} B &= d_b \quad (7) \end{aligned}$$

[1] I'll try to follow Amisano and Giannini (1997) as closely as possible.

[2] Necessary and sufficient conditions to achieve identification are stated in Lucchetti (2006), but the numerical procedures therein are not implemented in `SVAR` yet. Another interesting contribution in this area is Rubio-Ramirez et al. (2010).

This setup is perhaps overly general in most cases: the restrictions that are put almost universally on A and B are zero- or one-restrictions, that is constraints of the form, eg, $A_{ij} = 1$. In these cases, the corresponding row of R is a vector with a 1 in a certain spot and zeros everywhere else. However, generality is nice for exploring the identification problem.

The order condition demands that the number of restrictions is at least $2n^2 - \frac{n(n+1)}{2} = n^2 + \frac{n(n-1)}{2}$, so for the order condition to be fulfilled it is necessary that

$$\begin{aligned} 0 < \quad & \operatorname{rank}(R_a) & \leq n^2 \\ 0 < \quad & \operatorname{rank}(R_b) & \leq n^2 \\ n^2 + \frac{n(n-1)}{2} \leq \quad & \operatorname{rank}(R_a) + \operatorname{rank}(R_b) & \leq n^2 \end{aligned}$$

For the C model, $R_a = I_{n^2}$ and $d_a = \operatorname{vec} I_n$, so to satisfy the order condition $\frac{n(n-1)}{2}$ constraints are needed on on B: in practice, for a C model we have one set of constraints which pertain to B, or, equivalently in this context, to C:

$$R \operatorname{vec} C = d \tag{8}$$

The traditional choice of zeroing the the super-diagonal elements leads to the well-known solution where B is just the result of the Cholesky decomposition of Σ. In most other cases, explicit expressions for A and B are hard to find analytically and a numerical search procedure is necessary (more on this later). Of course, it is possible to estimate constrained models by placing some extra restrictions.

Estimation is carried out by maximum likelihood: under the assumption of normality, the average log-likelihood can be written as

$$\mathcal{L} = \text{const} - \ln|C| - 0.5\text{tr}(\hat{\Sigma}(CC')^{-1})$$

If the model is just-identified, $\hat{\Sigma}(CC')^{-1}$ will be the identity matrix and the log-likelihood simplifies to

$$\mathcal{L} = \text{const} - 0.5 \ln|\hat{\Sigma}| - 0.5n$$

For over-identified model, this makes it possible to test the over-identifying restrictions easily by means of a LR test.

Except for trivial cases, like the Cholesky decomposition, maximisation of the likelihood involves numerical iterations. Fortunately, analytical expressions for the score, the Hessian and the information matrix are available, which helps a lot[3]; once convergence has occurred, the covariance matrix for the unrestricted elements of A and B is easily computed via the information matrix.

Once estimation is completed, $\hat{A}$ and $\hat{B}$ can be used to compute the structural VMA representation of the VAR, which is the base ingredient for most of the subsequent analysis, such as Impulse Response Analysis and so forth. If the matrix polynomial $A(L)$ in eq. (3) is invertible, then (assuming $x_t = 0$ for ease of notation), y_t can be written as

$$y_t = A(L)^{-1}\varepsilon_t = \Theta(L)\varepsilon_t = \varepsilon_t + \Theta_1\varepsilon_{t-1} + \cdots$$

[3]As advocated in Amisano and Giannini, the scoring algorithm is used.

which is known as the VMA representation of the VAR. Note that in general the matrix polynomial $\Theta(L)$ is of infinite order.

From the above expression, one can write the *structural* VMA representation as

$$y_t = Cu_t + \Theta_1 Cu_{t-1} + \cdots = M_0 u_t + M_1 u_{t-1} + \cdots \tag{9}$$

From eq. (9) it is immediate to compute the impulse response functions:

$$\mathcal{I}_{i,j,h} = \frac{\partial y_{i,t}}{\partial u_{j,t-h}} = \frac{\partial y_{i,t+h}}{\partial u_{j,t}}$$

which in this case equal simply

$$\mathcal{I}_{i,j,h} = [M_h]_{ij}$$

The computation of confidence intervals for impulse responses could, in principle, be performed analytically by the delta method (see Lütkepohl (1990)). However, this has two disadvantages: for a start, it is quite involved to code. Moreover, the limit distribution has been shown to be a very poor approximation in finite samples, so the bootstrap is almost universally adopted, although in some cases it may be quite CPU-heavy.

Another quantity of interest that may be computed from the structural VMA representation is the Forecast Error Variance Decomposition (FEVD). The forecast error variance after h steps is given by

$$\Omega_h = \sum_{k=0}^{h} M_k M_k'$$

hence the variance for variable i is

$$\omega_i^2 = [\Omega_h]_{i,i} = \sum_{k=0}^{h} e_i' M_k M_k' e_i = \sum_{k=0}^{h} \sum_{l=1}^{n} ({}_k m_{i.l})^2$$

where ${}_k m_{i.l}$ is, trivially, the i, l element of M_k. As a consequence, the share of uncertainty on variable i that can be attributed to the j-th shock after h periods equals

$$\mathcal{VD}_{i,j,h} = \frac{\sum_{k=0}^{h} ({}_k m_{i.j})^2}{\sum_{k=0}^{h} \sum_{l=1}^{n} ({}_k m_{i.l})^2}.$$

2 C models

2.1 A simple example

As a trivial example, we will estimate a plain Cholesky model. The data are taken from Stock and Watson's sample data `sw_ch14.gdt`, and our VAR will include inflation and unemployment, with a constant and 3 lags. Then, we will compute the IRFs and their 90% bootstrap confidence interval[4].

[4]Why not 95%? Well, keeping the number of bootstrap replications low is one reason. Anyway, it must be said that in the SVAR literature few people use 95%. 90%, 84% or even 66% are common choices.

```
# turn extra output off
set echo off
set messages off

# open the data and do some preliminary transformations
open sw_ch14.gdt
genr infl = 400*ldiff(PUNEW)
rename LHUR unemp
list X = unemp infl

var 3 unemp infl

Sigma = $sigma
C = cholesky(Sigma)
print Sigma C
```

Table 1: Cholesky example via the internal `gretl` command

```
VAR system, lag order 3
OLS estimates, observations 1960:1-1999:4 (T = 160)
Log-likelihood = -267.76524
Determinant of covariance matrix = 0.097423416
AIC = 3.5221
BIC = 3.7911
HQC = 3.6313
Portmanteau test: LB(40) = 162.946, df = 148 [0.1896]

Equation 1: u

             coefficient   std. error   t-ratio   p-value
  --------------------------------------------------------
  const       0.137300     0.0846842     1.621    0.1070
  u_1         1.56139      0.0792473    19.70     8.07e-44 ***
  u_2        -0.672638     0.140545     -4.786    3.98e-06 ***

...
Sigma (2 x 2)

    0.055341    -0.028325
   -0.028325       1.7749

C (2 x 2)

     0.23525       0.0000
    -0.12041       1.3268
```

Table 2: Cholesky example via the internal `gretl` command — Output

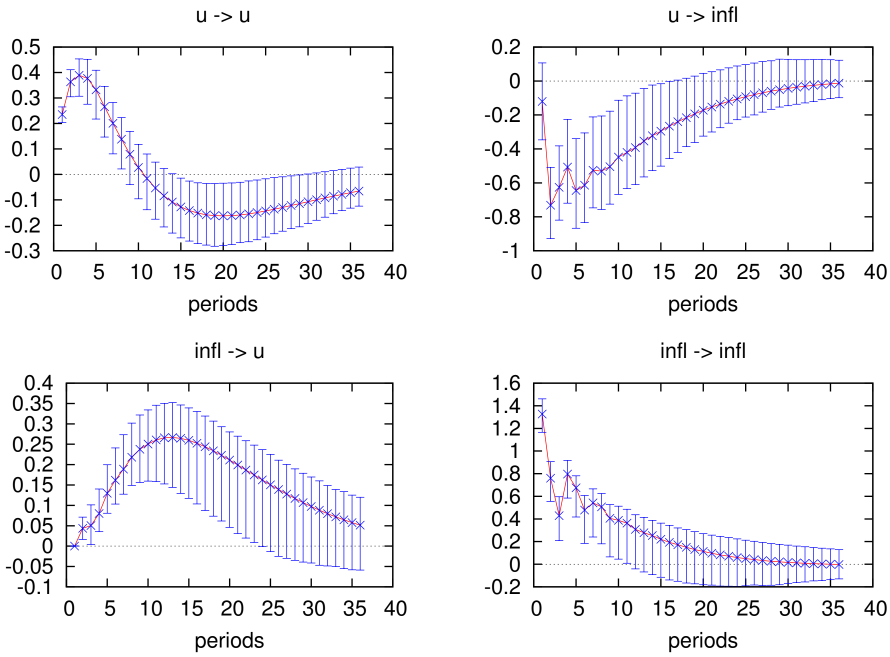

Figure 1: Impulse response functions for the simple Cholesky model (native)

2.1.1 Gretl native `var` command

In order to accomplish the above, note that we *don't* need to use the `SVAR` package, as a Cholesky SVAR can be handled by `gretl` natively. In fact, the script shown in Table 2.1 does just that: runs a VAR, collects $\hat{\Sigma}$ and estimates C as its Cholesky deomposition. Part of its output is in Table 2.1. The impulse responses as computed by `gretl`'s internal command can be see in figure 1. See the Gretl's User Guide for more details.

2.1.2 Base estimation via the `SVAR` package

We will now replicate the above example via the `SVAR` package; in order to do so, we need to treat this model as a special case of the C-model, where $\varepsilon_t = Cu_t$ and identification is attained by stipulating that C is lower-triangular, that is

$$C = \begin{bmatrix} c_{11} & 0 \\ c_{12} & c_{22} \end{bmatrix}. \tag{10}$$

Table 3 shows a sample script to estimate the example Cholesky model: the basic idea is that the model is contained in a gretl bundle[5]. In this example, the bundle is called `Mod`, but it can of course take any valid `gretl` identifier.

[5]Bundles are a fairly recent addition to the gretl weaponry: most likely, you'll want to take a look at the Gretl's User Guide, section 11.7. They may be briefly described as containers in which a certain object (a scalar, a matrix and so on) is associated to a "key" (a string). Technically speaking, a bundle is an associative array: these data structures are called "hashes" in Perl or "dictionaries" in Python.

```
# turn extra output off
set echo off
set messages off

# open the data and do some preliminary transformations
open sw_ch14.gdt
genr infl = 400*ldiff(PUNEW)
rename LHUR unemp
list X = unemp infl
list Z = const

# load the SVAR package
include SVAR.gfn

# set up the SVAR
Mod = SVAR_setup("C", X, Z, 3)

# Specify the constraints on C
SVAR_restrict(&Mod, "C", 1, 2, 0)

# Estimate
SVAR_estimate(&Mod)
```

Table 3: Simple C-model

After performing the same preliminary steps as in the example in Table 2.1, we load the package and use the `SVAR_setup` function, which initialises the model and sets up a few things. This function takes 4 arguments:

- a string, with the model type (`"C"` in this example);
- a list containing the endogenous variables y_t;
- a list containing the exogenous variables x_t (may be `null`);
- the VAR order p.

Once the model is set up, you can specify which elements you want to constrain to achieve identification; there are several ways to do this, but in most cases you'll want to use the `SVAR_restrict` function. A complete description can be found in appendix A; suffice it to say here that the result of the function

```
SVAR_restrict(&Mod, "C", 1, 2, 0)
```

is to ensure that $C_{1,2} = 0$ (see eq. 10).

The next step is estimation, which is accomplished via the `SVAR_estimate` function, which just takes one argument, the model to estimate. The output of the `SVAR_estimate` function is shown below[6]: note that, as an added benefit,

[6]For compatibility with other packages, $\hat{\Sigma}$ is estimated by dividing the cross-products of the VAR residuals by $T - k$ instead of T; this means that the actual figures will be slightly different from what you would obtain by running `var` and then `cholesky($sigma)`.

we get asymptotic standard errors for the estimated parameters (estimated via the information matrix).

```
Unconstrained Sigma:
     0.05676    -0.02905
    -0.02905     1.82044

             coefficient   std. error   z-stat    p-value
  --------------------------------------------------------
  C[ 1; 1]     0.238243    0.0131548    18.11     2.62e-73 ***
  C[ 2; 1]    -0.121939    0.105142     -1.160    0.2461
  C[ 1; 2]     0.00000     0.00000      NA       NA
  C[ 2; 2]     1.34371     0.0741942    18.11     2.62e-73 ***
```

At this point, the model bundle contains all the quantities that will need to be accessed later on, including the structural VMA representation; this is computed up to an order called the "horizon". The function SVAR_setup initalises automaticaly the horizon to 24 for monthly data and to 20 for quarterly data. To change it, you just assign the desired value to the "horizon" element of the bundle, as in (for example)

```
Mod["horizon"] = 40
```

More details on the internal organisation of the bundle can be found in section B in the appendix. Its contents can be accessed via the ordinary gretl syntactic constructs for dealing with bundles. For example, the number of observations used in estimating the model is stored as the bundle member "T", so if you ever need it you can just use the syntax Mod["T"], or Mod.T with newer versions of gretl (from 1.9.12 onwards).

Once the model has been estimated, it becomes possible to retrieve estimates of the structural shocks, via the function GetShocks, as in:

```
series foo = GetShock(&Mod, 1)
series bar = GetShock(&Mod, 2)
```

If we append the two lines above to example 3, two new series will be obtained. The formula used is nothing but equation (5) in which the VAR residuals are used in place of ε_t.

2.2 Impulse responses and FEVD

```
fevdmat = FEVD(&Mod)
print fevdmat

IRFplot(&Mod, 1, 1)
```

Table 4: Simple C-model (continued)

As shown in Table 4, after the model has been estimated, it can be passed to another function called FEVD to compute the Forecast Error Variance Decomposition, which is subsequently printed. Its usage is very simple, since it only needs one input (a pointer to the model bundle).

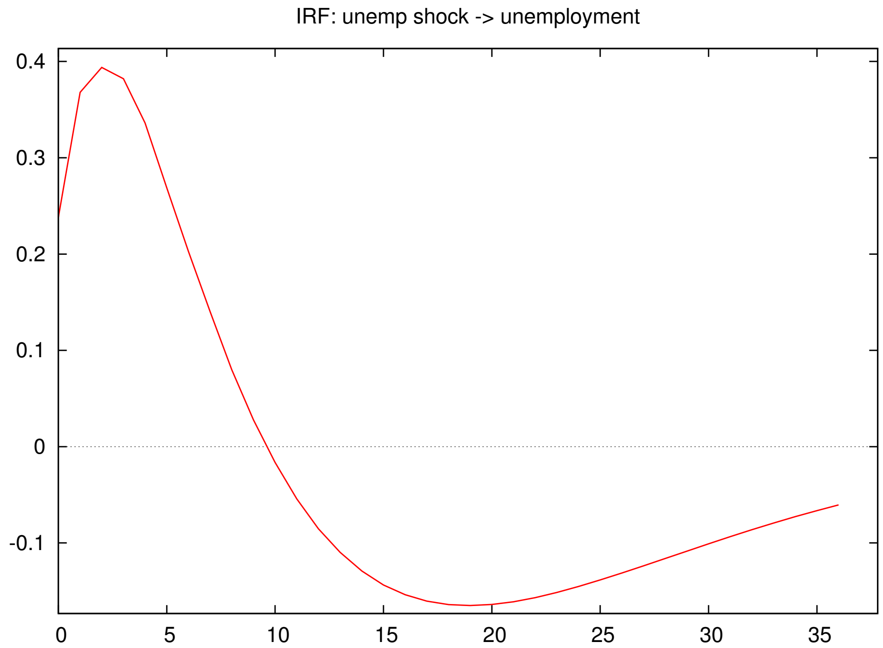

Figure 2: Impulse response functions for unemployment

The `SVAR` package provides a function called `IRFplot` for plotting the impulse response function on your screen, with a little help from our friend `gnuplot`; its syntax is relatively simple.[7] The three arguments used here are:

1. The model bundle (as a pointer);
2. the number of the structural shock we want the IRF to;
3. the number of the variable we want the IRF for.

The function can be used in a more sophisticated way than this (see later). Its output is presented in Figure 2. As can be seen, it's very similar to the one obtained by `gretl`'s native command (Figure 1).[8]

[7] A parallel function for the FEVD is not implemented, but wouldn't be difficult to do. Only boring.

[8] Warning: using the built-in GUI graph editor that gretl provides may produce 'wrong' results on the figures generated by the `IRFplot` function. All gretl's graphics are handled by creating a gnuplot script, executing it and then sending the result to the display. All this is done transparently. When you edit a graph, you modify the underlying gnuplot script via some GUI elements, so when you click "Apply" the graphic gets re-generated. However, gretl's GUI interface for modifying graphics can't handle arbitrary gnuplot scripts, but only those generated internally.

The figures generated by `IRFplot` contain a few extra features that the GUI editor doesn't handle, so invoking the GUI controls may mess up the graph. As an alternative, you can customise the graph by editing the gnuplot script directly: right-click on it and "Save [it] to session as icon". Then, in the icon view, right click on the graph icon and choose "Edit plot commands": you'll have the gnuplot source to the graph, that you can modify as needed.

2.3 Bootstrapping

```
bfail = SVAR_boot(&Mod, 1024, 0.90)

loop for i=1..2 -q
    loop for j=1..2 -q
        sprintf fnam "simpleC_%d%d.pdf", i, j
        IRFsave(fnam, &Mod, i, j)
    end loop
end loop
```

Table 5: Simple C-model (continued)

The next step is computing bootstrap-based confidence intervals for the estimated coefficients and, more interestingly, for the impulse responses[9]: as can be seen in Table 5, this task is given to the `SVAR_boot` function, which takes as arguments

1. The model bundle pointer;
2. the required number of bootstrap replications (1024 here)[10]
3. the desired size of the confidence interval α.

The function outputs a scalar, which keeps track of how many bootstrap replications failed to converge (none here). Note that this procedure may be quite CPU-intensive. The output contains a table similar to the output to `Cmodel`, which is used to display the bootstrap means and standard errors of the parameters:

```
Bootstrap results (1024 replications)
             coefficient   std. error      z       p-value
  --------------------------------------------------------
  C[ 1; 1]      0.232146    0.0183337    12.66      9.57e-37 ***
  C[ 2; 1]    -0.114610     0.143686     -0.7976    0.4251
  C[ 1; 2]      0.00000     0.00000      NA        NA
  C[ 2; 2]      1.30234     0.0853908    15.25      1.61e-52 ***

Failed = 0, Time (bootstrap) = 20.24
```

Finally, another `SVAR` function, `IRFsave()` is used to store plots the impulse responses into pdf[11] files for later use; its arguments are the same as `IRFplot()`, except that the first argument must contain a valid filename to save the plot into. In the above example, this function is used within a loop to save all impulse responses in one go. The output is shown in Figure 3.

[9]What is available at the moment is the most naive form of bootstrap. None of the fancy alternatives listed, for example, in Brüggemann (2006) are available. They are planned, though.

[10]There's a hard limit at 10000 at the moment; probably, it will be raised in the future. However, unless your model is very simple, anything more than that is likely to take forever and melt your CPU.

[11]Or PostScript...

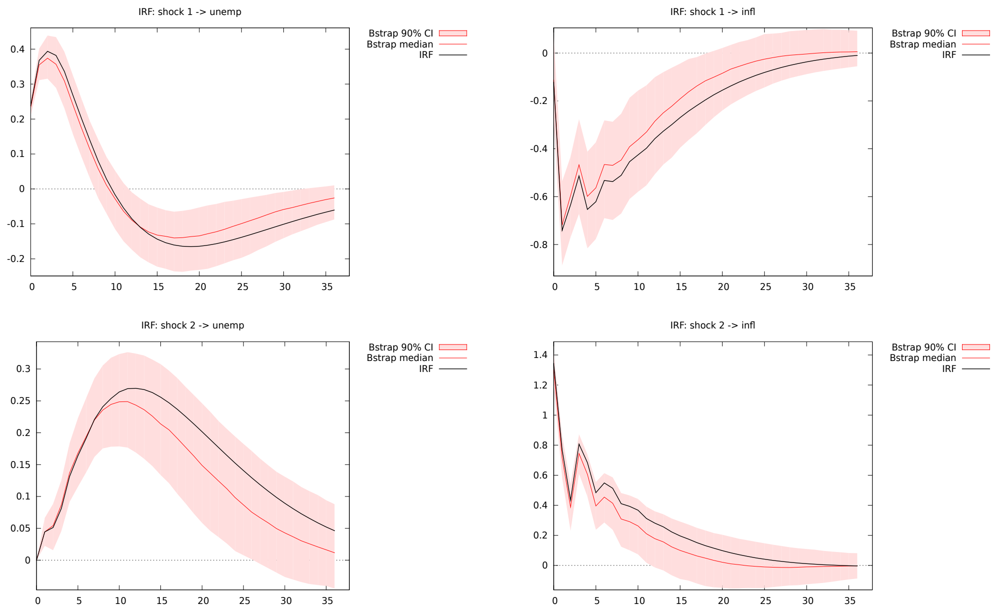

Figure 3: Impulse response functions for the simple Cholesky model

2.4 A shortcut

In many cases, a triangular, Cholesky-style specification for the C matrix like the one analysed in this section is all that is needed. When many variables are involved, the setting of the $\frac{n\times(n-1)}{2}$ restrictions via the `SVAR_restrict` function could be done quite boring, although easily done via a loop.

For these cases, the `SVAR` package provides an alternative way: if you supply the `SVAR_setup` function with the string `"plain"` as its first argument, the necessary restrictions are set up automatically. Thus, the example considered above in Table 3 could by modified by replacing the lines

```
Mod = SVAR_setup("C", X, Z, 3)
SVAR_restrict(&Mod, "C", 1, 2, 0)
```

with the one-liner

```
Mod = SVAR_setup("plain", X, Z, 3)
```

and leaving the rest unchanged.

2.5 Specifying restrictions

A key ingredient in a SVAR (arguably, *the* key ingredient) is the set of constraints we put on the structural matrices. `SVAR` handles these restrictions via their implicit form representation $R\theta = d$. Consider, for notational convenience, the matrix $R^* = [R|d]$. The `SVAR_restrict` function we used earlier does nothing but adding rows to R^*.

For a C model, the R^* matrix is stored as the bundle element `Rd1` and the number of its rows is kept as bundle element `nc1`. If you feel like building the matrix R^* via gretl's ordinary matrix functions, all you have to do is to fill up the bundle elements `Rd1` and `nc1` properly before calling `SVAR_estimate()`.

2.6 The GUI interface

Most of the above can be accomplished via the GUI interface, which can be accessed via the *Model > Time Series > Structural VAR* menu entry of the the graphical gretl client. While we recommend to use the script interface to use the full capabilities of the SVAR package, the GUI interface may be less intimidating for less experienced users.

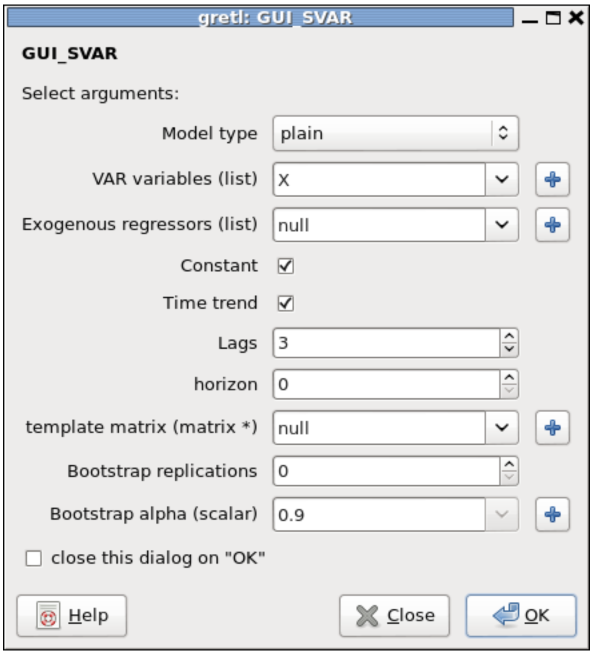

Figure 4: Plain Cholesky model through the GUI interface

The contents of the window displayed in figure 4 should be rather self-explanatory, with one exception, that is the "template" matrix. This can be used for estimating C models with a restriction scheme other than the lower diagonal one.

For example, suppose we wanted to estimate a C model like the one used as example so far, with the only difference that we want the C matrix to be *upper* triangular, rather than lower trangular. Via a script, you would use the function `SVAR_restrict()`, as in

```
# Force C_{2,1} to 0
SVAR_restrict(&Mod, "C", 2, 1, 0)
```

but you can do the same via the GUI interface by using a template matrix. A template matrix is a $n \times n$ matrix (that is, the same size as C) which contains valid numerical values for the corresponding restricted elements of C and NAs for unrestricted elements. This can be a pre-existing matrix or a matrix you define on the spot by clicking on the "+" button. In this case, you'll be presented with gretl's GUI window for creating matrices. Suppose we call the template matrix TMPL and that we select the option "Build Numerically" (of course, with 2 rows and 2 columns in this example). A further window will appear, that you can use for filling the matrix elements with the desired values, as in Figure 5 When you're done, you return to the main SVAR window (be sure to select C-

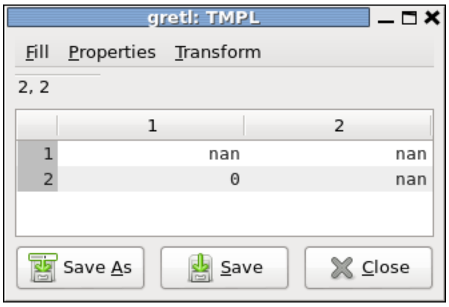

Figure 5: Template matrix

model as the model type). After clicking "OK", the results window will appear, as in Figure 6. Note that the estimated C matrix is now upper triangular.

From the output window, you can save the model bundle to the Icon view through the *File* menu and re-use it as needed for further processing.

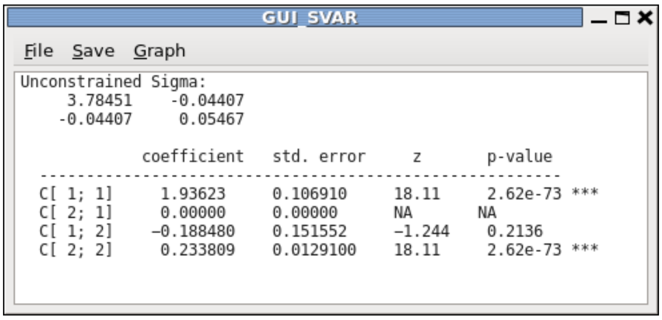

Figure 6: Template matrix

3 C-models with long-run restrictions (Blanchard-Quah style)

An alternative way to impose restrictions on C is to use long-run restrictions, as pioneered by Blanchard and Quah (1989). The economic rationale of imposing restictions on the elements of C is that C is equal to M_0, the instantaneous IRF. For example, Cholesky-style restrictions mean that the j-th shock has no instantaneous impact on the i-th variable if $i < j$. Assumptions of this kind are normally motivated by institutional factors such as sluggish adjustments, information asymmetries, and so on.

Long-run restrictions, instead, stem from more theoretically-inclined reasoning: in Blanchard and Quah (1989), for example, it is argued that in the long run the level of GDP is ultimately determined by aggregate supply. Fluctuations in aggregate demand, such as those induced by fiscal/monetary policy, impact the level of GDP only in the short term. As a consequence, the impulse response of GDP with respect to demand shocks should go to 0 asymptotically, whereas the response of GDP to a supply shock shoud settle to some positive value.

3.1 A modicum of theory

To translate this intuition into formulae, assume that the bivariate process GDP growth-unemployment

$$x_t = \begin{bmatrix} \Delta Y_t \\ U_t \end{bmatrix}$$

is $I(0)$ (which implies that Y_t is $I(1)$), and that it admits a finite-order VAR representation

$$A(L)x_t = \varepsilon_t$$

where the prediction errors are assumed to be a linear combination of demand and supply shocks

$$\begin{bmatrix} \varepsilon_t^{\Delta Y} \\ \varepsilon_t^{U} \end{bmatrix} = C \begin{bmatrix} u_t^d \\ u_t^s \end{bmatrix},$$

Considering the structural VMA representation

$$\begin{aligned} \begin{bmatrix} \Delta Y_t \\ U_t \end{bmatrix} &= \Theta(L)\varepsilon_t = \varepsilon_t + \Theta_1\varepsilon_{t-1} + \cdots = \\ &= Cu_t + \Theta_1 C u_{t-1} + \cdots = M_0 u_t + M_1 u_{t-1} + \cdots , \end{aligned}$$

it should be clear that the impact of demand shocks on ΔY_t after h periods is given by the north-west element of M_h. Since x_t is assumed to be stationary, $\lim_{h\to\infty} \Theta_h = 0$ and the same holds for M_k, so obviously the impact of either shock on ΔY_t goes to 0. However, the impact of u_t on the *level* of Y_t is given by the *sum* of the corresponding elements of M_h, since

$$Y_{t+h} = Y_{t-1} + \sum_{i=0}^{h} \Delta Y_{t+i},$$

so

$$\frac{\partial Y_{t+h}}{\partial u_t^d} = \sum_{i=0}^{h} \frac{\partial \Delta Y_{t+i}}{\partial u_t^d} = \sum_{i=0}^{h} [M_i]_{11}$$

and in the limit

$$\lim_{h\to\infty} \frac{\partial Y_{t+h}}{\partial u_t^d} = \sum_{i=0}^{\infty} \frac{\partial \Delta Y_{t+i}}{\partial u_t^d} = \sum_{i=0}^{\infty} [M_i]_{11} ,$$

In general, if x_t is stationary, the above limit is finite, but needn't go to 0; however, if we assume that the long-run impact of u_t^d on Y_t is null, then

$$\lim_{k\to\infty} \frac{\partial Y_{t+k}}{\partial u_t^d} = 0$$

and this is the restriction we want. In practice, instead of constraining elements of M_0, we impose an implicit constraint on the whole sequence M_i.

How do we impose such a constraint? First, write $\sum_{i=0}^{\infty} \Theta_i$ as $\Theta(1)$; then, observe that

$$\Theta(1)C = \sum_{i=0}^{\infty} M_i;$$

the constraint we seek is that the north-west element of $\Theta(1)C$ equals 0. The matrix $\Theta(1)$ is easy to compute after the VAR coefficients have been estimated: since $\Theta(L) = A(L)^{-1}$, an estimate of $\Theta(1)$ is simply

$$\widehat{\Theta(1)} = \hat{A}(1)^{-1}$$

Of course, for this to work $A(1)$ needs to be invertible. This rules out processes with one or more unit roots. The cointegrated case, however, is an interesting related case and will be analysed in section 5.

The long-run constraint we seek can then be written as

$$R\,\mathrm{vec}[\Theta(1)C] = 0, \tag{11}$$

where $R = [1, 0, 0, 0]$; since

$$\mathrm{vec}[\Theta(1)C] = [I \otimes \Theta(1)]\,\mathrm{vec}(C),$$

the constraint can be equivalently expressed as

$$[\Theta(1)_{11}, \Theta(1)_{12}, 0, 0]\,\mathrm{vec}(C) = \Theta(1)_{11} \cdot c_{11} + \Theta(1)_{12} \cdot c_{21} = 0. \tag{12}$$

Note that we include in R elements that, stictly speaking, are not constant, but rather functions of the estimated VAR parameters. Bizarre as this may seem, this poses no major inferential problems under a suitable set of conditions (see Amisano and Giannini (1997), section 6.1).

3.2 Example

The way all this is handled in `SVAR` is hopefully quite intuitive: an example script is reported in Table 6. After reading the data in, the function `SVAR_setup` is invoked in pretty much the same way as in section 2.

Then, the `SVAR_restrict` is used to specify the identifying restriction. Note that in this case the code for the restriction type is `"lrC"`, which indicates that the restriction applies to the long-run matrix, so the formula (12) is employed. Next, we insert into the model the information that we will want IRFs for y_t, so those for Δy_t will have to be cumulated. This is done via the function `SVAR_cumulate()`, in what should be a rather self-explanatory way (the number 1 refers in this case to the position of ΔY_t in the list `X`). Finally, a cosmetic touch: we store into the model the string `"Supply Demand"`, which will be used to label the shocks in the graphs. Note that in this case there is no ad-hoc function, but we rely on the standard gretl syntax for bundles.

In Table 7 I reported the output to the example code in Table 6, while the pretty pictures are in Figure 7. Note that in the two calls to `IRFplot` which are used to plot the responses to a demand shock, the number to identify the shock is not 2, but rather -2. This is a little trick the plotting functions use to flip the sign of the impulse responses, which may be necessary to ease their interpretation (since the shocks are identified only up to their sign).

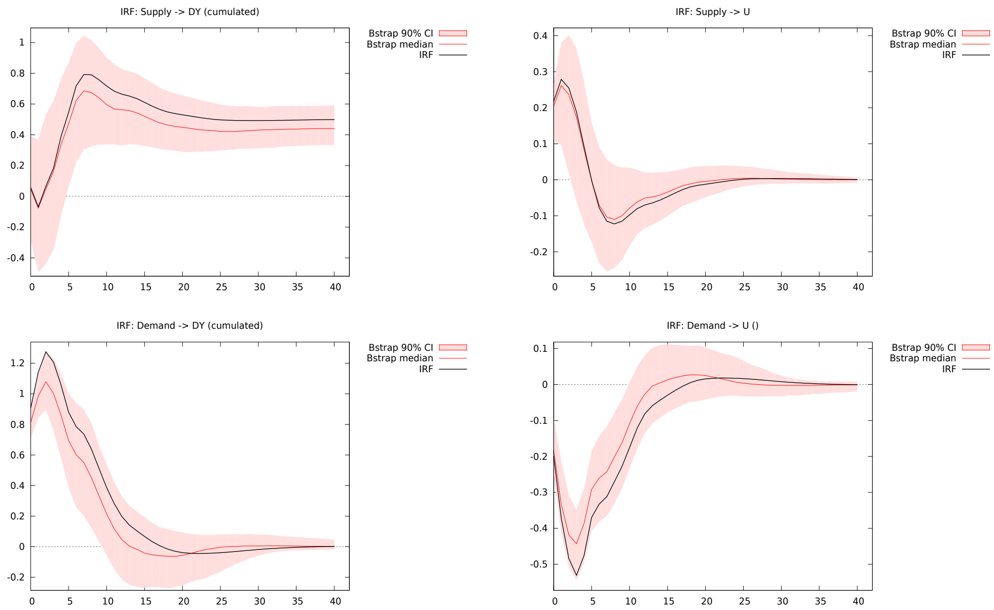

Figure 7: Impulse response functions for the Blanchard-Quah model

3.3 Combining short- and long-run restrictions

In the previous example, it turned out that the estimated coefficient for $c_{1,1}$ was seemingly insignificant; if true, this would mean that the supply shock has no instantaneous effect on ΔY_t; in other words, the IRF of output to supply starts from 0. Leaving the economic implications aside, from a statistical viewpoint this could have suggested an alternative identification strategy or, more interestingly, to combine the two hypotheses into one.

The script presented in Table 6 is very easy to modify to this effect: in this case, we simply need to insert the line

```
SVAR_restrict(&BQModel, "C", 1, 1, 0)
```

somewhere between the `SVAR_setup` and the `SVAR_estimate` function. The rest is unchanged, and below is the output.

```
            coefficient   std. error     z       p-value
---------------------------------------------------------
C[ 1; 1]     0.00000      0.00000       NA      NA
C[ 2; 1]    -0.230192     0.0128681    -17.89    1.45e-71 ***
C[ 1; 2]    -0.909033     0.0508165    -17.89    1.45e-71 ***
C[ 2; 2]     0.199859     0.0111725     17.89    1.45e-71 ***

Overidentification LR test = 0.642254 (1 df, pval = 0.422896)
```

Note that, since this model is over-identified, `SVAR` automatically computes a LR test of the overidentifying restrictions. Of course, all the subsequent steps (bootstrapping and IRF plotting) can be performed just like in the previous example if so desired.

4 AB models

4.1 A simple example

AB models are more general than the C model, but more rarely used in practice. In order to exemplify the way in which they are handled in the SVAR package, I will replicate the example given in section 4.7.1 of Lütkepohl and Krätzig (2004). See Table 8.

This is an empirical implementation of a standard Keynesian IS-LM model in the formulation by Pagan (1995). The vector of endogenous variables includes output q_t, interest rate i_t and real money m_t; the matrices A and B are

$$A = \begin{bmatrix} 1 & a_{12} & 0 \\ a_{21} & 1 & a_{31} \\ 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & 0 & 0 \\ 0 & b_{22} & 0 \\ 0 & 0 & b_{33} \end{bmatrix}$$

so for example the first structural relationship is

$$\varepsilon_t^q = -a_{12}\varepsilon_t^i + u_t^{IS} \tag{13}$$

which can be read as an IS curve. The LM curve is the second relationship, while money supply is exogenous.

The model is set up via the function `SVAR_setup`, like in the previous section. Note, however, that in this case the model code is `"AB"` rather than `"C"`. The base VAR has 4 lags, with the constant and a linear time trend as exogenous variables. The horizon of impulse response analysis is set to 48 quarters.

The constraints on the matrices A and B can be set up quite simply by using a the function `SVAR_restrict` via a special syntax construct: the line

```
SVAR_restrict(&ISLM, "Adiag", 1)
```

sets up a system of constraints such that all elements on the diagonal of A are set to 1. More precisely, `SVAR_restrict(&Model, "Adiag", x)` sets all diagonal elements of A to the value x, unless x is `NA`. In that case, all *non-diagonal* elements are constrained to 0, while diagonal elements are left unrestricted; in other words, the syntax

```
SVAR_restrict(&ISLM, "Bdiag", NA)
```

is a compact form for saying "B is diagonal". The other three constraints are set up as usual.

Estimation is then carried out via the `SVAR_estimate` function; as an example, Figure 8 shows the effect on the interest rate of a shock on the IS curve. This example also shows how to retrieve extimated quantities from the model: after estimation, the bundle elements `"S1"` and `"S2"` contain the estimated A and B matrices; the C matrix is then computed and printed out.

The output is shown below:

```
             coefficient   std. error      z       p-value
  -------------------------------------------------------
  A[ 1; 1]     1.00000       0.00000      NA         NA
  A[ 2; 1]   -0.144198       0.280103    -0.5148    0.6067
  A[ 3; 1]     0.00000       0.00000      NA         NA
```

```
  A[ 1; 2]   -0.0397571     0.155114    -0.2563    0.7977
  A[ 2; 2]    1.00000       0.00000     NA         NA
  A[ 3; 2]    0.00000       0.00000     NA         NA
  A[ 1; 3]    0.00000       0.00000     NA         NA
  A[ 2; 3]    0.732161      0.146135     5.010     5.44e-07 ***
  A[ 3; 3]    1.00000       0.00000     NA         NA


           coefficient   std. error      z      p-value
  -------------------------------------------------------
  B[ 1; 1]   0.00671793    0.000473619   14.18    1.15e-45 ***
  B[ 2; 1]   0.00000       0.00000       NA      NA
  B[ 3; 1]   0.00000       0.00000       NA      NA
  B[ 1; 2]   0.00000       0.00000       NA      NA
  B[ 2; 2]   0.00858125    0.000581359   14.76    2.63e-49 ***
  B[ 3; 2]   0.00000       0.00000       NA      NA
  B[ 1; 3]   0.00000       0.00000       NA      NA
  B[ 2; 3]   0.00000       0.00000       NA      NA
  B[ 3; 3]   0.00555741    0.000371320   14.97    1.21e-50 ***

Estimated contemporaneous impact matrix (x100) =
  0.675666  0.034313 -0.016270
  0.097430  0.863073 -0.409238
  0.000000  0.000000  0.555741

Bootstrap results (2000 replications)

           coefficient   std. error      z       p-value
  --------------------------------------------------------
  A[ 1; 1]    1.00000       0.00000     NA         NA
  A[ 2; 1]   -0.0909784     0.395312    -0.2301    0.8180
  A[ 3; 1]    0.00000       0.00000     NA         NA
  A[ 1; 2]   -0.0377229     0.228185    -0.1653    0.8687
  A[ 2; 2]    1.00000       0.00000     NA         NA
  A[ 3; 2]    0.00000       0.00000     NA         NA
  A[ 1; 3]    0.00000       0.00000     NA         NA
  A[ 2; 3]    0.782728      0.181538     4.312     1.62e-05 ***
  A[ 3; 3]    1.00000       0.00000     NA         NA


           coefficient   std. error      z       p-value
  --------------------------------------------------------
  B[ 1; 1]   0.00635862    0.000850539    7.476    7.66e-14 ***
  B[ 2; 1]   0.00000       0.00000       NA        NA
  B[ 3; 1]   0.00000       0.00000       NA        NA
  B[ 1; 2]   0.00000       0.00000       NA        NA
  B[ 2; 2]   0.00814276    0.00111305     7.316    2.56e-13 ***
  B[ 3; 2]   0.00000       0.00000       NA        NA
  B[ 1; 3]   0.00000       0.00000       NA        NA
  B[ 2; 3]   0.00000       0.00000       NA        NA
  B[ 3; 3]   0.00512819    0.000478826   10.71     9.14e-27 ***
```

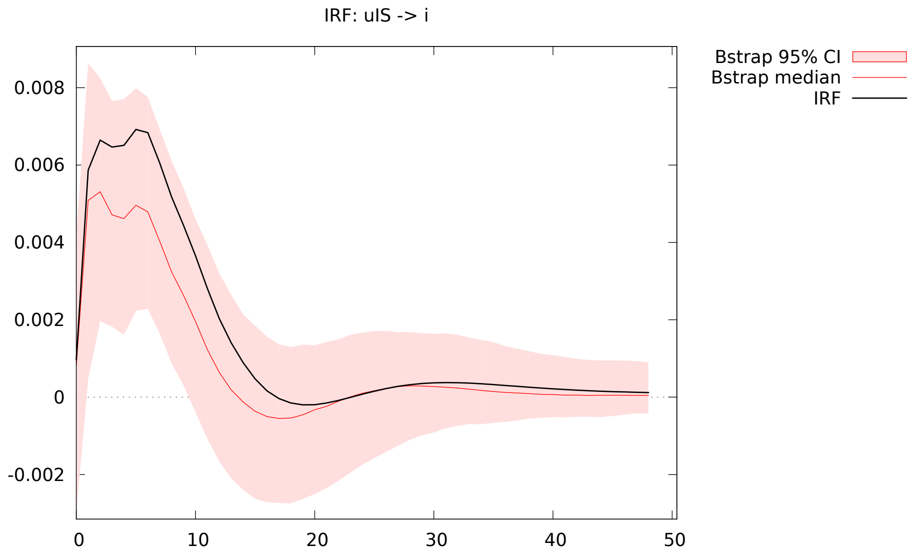

Figure 8: $u^{IS} \to i$

5 Structural VECMs

The functions for these models aren't ready for production use yet. Hopefully, they will be soon. If you're feeling brave, have a look at the `awm.inp` file in the examples directory.

This class of models was first proposed in King et al. (1991). A SVECM is basically a C-model in which the interest is centred on classifying structural shocks as permanent or transitory by exploiting the presence of cointegration.

Suppose we have an n-dimensional system with cointegration rank r. Apart from the usual ECM representation

$$\Gamma(L)\Delta y_t = \alpha\beta' y_{t-1} + \varepsilon_t$$

it is also possible to express Δy_t as a vector moving average process

$$\Delta y_t = C(L)\varepsilon_t. \tag{14}$$

The main consequence of cointegration for eq. (14) is that $C(1)$ is a singular matrix. As Granger's representation theorem shows, the $C(1)$ matrix satisfies

$$\begin{aligned} \beta' C(1) &= 0 \\ C(1)\alpha &= 0; \end{aligned}$$

thus, the rank of $C(1)$ is $n - r$.

The implications of the above on structural estimation stem from the consideration that the ij-th element of $C(1)$ can be thought of as the long-run response of $y_{i,t}$ to $\varepsilon_{j,t}$ or, more precisely

$$C(1)_{i,j} = \lim_{k\to\infty} \frac{\partial y_{i,t+k}}{\partial \varepsilon_{j,t}}.$$

The response of y_t to structural shocks is easily seen (via eq. 5) to be $C(1) \cdot C$. Now, define a transitory shock as a structural shock that has no long-run effect on any variable: hence, the corresponding column of $C(1) \cdot C$ must be full of zeros. But this, in turn, implies that the corresponding column of C must be a linear combination of the columns of α. Since α has r linearly independent columns, the vector of structural shocks must contain r transitory shocks and $n - r$ permanent ones.

By ordering the structural shocks with the permanent ones first,

$$u_t = \begin{bmatrix} u_t^p \\ u_t^t \end{bmatrix}$$

it's easy to see that identification of the permanent shocks can be achieved by imposing that the last r columns of C lie in the space spanned by α; in formulae,

$$\alpha'_\perp C J = 0, \tag{15}$$

where J is the matrix

$$J = \begin{bmatrix} 0_{n-r \times r} \\ I_{r \times r} \end{bmatrix}$$

and $\perp$ is the "nullspace" operator, that is: if M is an $r \times c$ matrix, with $r > c$ and $\text{rank}\,(M) = c$, then $M_\perp$ is some matrix[12] such that $M'_\perp M = 0$.

Equation (15) can be expressed in vector form as

$$(J' \otimes \alpha'_\perp) \operatorname{vec}(C) = 0;$$

since $\alpha_\perp$ has $n - r$ columns, this provides $r \cdot (n - r)$ constraints of the type $R \operatorname{vec}(C) = d$, that we know how to handle. Since $0 < r < n$, this system of constraints is not sufficient to achieve identification, apart from the special case $n = 2, r = 1$.

For this type of models, the keyword to use in the `SVAR_setup` function is `KPSW`.

The matrix β is not estimated within `SVAR` and must be supplied by the user and put into the `"aux"` element of the model bundle. This allows you to estimated the cointegration vectors by any method you like (or pre-set them to some theory-derived value). Note that the `$jbeta` standard gretl accessor makes it painless to fetch it from a Johansen-style VECM if necessary.

References

Amisano, G. and Giannini, C. (1997). *Topics in structural VAR econometrics*. Springer-Verlag, 2nd edition.

Blanchard, O. and Quah, D. (1989). The dynamic effects of aggregate demand and aggregate supply shocks. *American Economic Review*, 79(4):655–73.

Brüggemann, R. (2006). Finite sample properties of impulse response intervals in svecms with long-run identifying restrictions. Sfb 649 discussion papers, Sonderforschungsbereich 649, Humboldt University, Berlin, Germany.

[12]Note: $M_\perp$ is not unique.

King, R. G., Plosser, C. I., Stock, J. H., and Watson, M. (1991). Stochastic trends and economic fluctuations. *American Economic Review*, 81(4):819–40.

Lütkepohl, H. and Krätzig, M., editors (2004). *Applied Time Series Econometrics*. Cambridge University Press.

Lucchetti, R. (2006). Identification of covariance structures. *Econometric Theory*, 22(02):235–257.

Lütkepohl, H. (1990). Asymptotic distributions of impulse response functions and forecast error variance decompositions of vector autoregressive models. *The Review of Economics and Statistics*, 72(1):116–25.

Pagan, A. (1995). Three econometric methodologies: An update. In Oxley, L., Roberts, C., George, D., and Sayer, S., editors, *Surveys in Econometrics*, pages 30–41. Basil Blackwell.

Rubio-Ramirez, J., Waggoner, D., and Zha, T. (2010). Structural vector autoregressions: Theory of identification and algorithms for inference. *Review of Economic Studies*, 77(2):665–696.

Sims, C. A. (1980). Macroeconomics and reality. *Econometrica*, 48:1–48.

A Alphabetical list of functions

`FEVD(bundle *SVARobj)`

Computes the Forecast Error Variance Decomposition from the structural IRFs, as contained in the model `SVARobj`. Returns an $h \times n^2$ matrix.

`GetShock(bundle *SVARobj, scalar i)`

Retrieves, as a series, the estimate of i-th structural shock of the system via equation (2), in which VAR residuals are used instead of the one-step-ahead prediction errors ε_t. If the bundle `SVARobj` contains a non-null string `snames` with shock names, those are used in the description for the generated series.

`IRFplot(bundle *obj, scalar snum, scalar vnum)`

Plots an impulse response function on screen. Its arguments are:

1. a bundle holding the model
2. the progressive number of the shock (may be negative, in which case the IRF is flipped)
3. the progressive number of the variable

```
IRFsave(string outfilename, bundle *obj, scalar snum, scalar vnum)
```

Saves an impulse response function to a graphic file, whose format is identified by its extension. Its arguments are:

1. The graphic file name
2. a bundle holding the model
3. the progressive number of the shock (may be negative, in which case the IRF is flipped)
4. the progressive number of the variable

```
SVAR_boot(bundle *obj, scalar rep, scalar alpha)
```

Perform a bootstrap analysis of a model. Returns the number of bootstrap replications in which the model failed to converge. Its arguments are:

1. a bundle holding the model
2. the number of bootstrap replications
3. the quantile used for the confidence bands

```
SVAR_cumulate(bundle *b, scalar nv)
```

Stores into the model the fact that the cumulated IRFs for variable `nv` are desired. This is typically used jointly with long-run restrictions.

```
SVAR_estimate(bundle *obj, int quiet)
```

Estimates the model by maximum likelihood. Its second argument is a scalar, which controls the verbosity of output. If omitted, estimation proceeds silently.

```
SVAR_restrict(bundle *b, string code, scalar r, scalar c, scalar d)
```

Sets up contraints for an existing model. The function which takes at most five arguments:

1. A pointer to the model for which we want to set up the restriction(s)
2. A code for which type of restriction we want:

"C" Applicable to C models. Used for short-run restrictions.

"lrC" Applicable to C models. Used for long-run restrictions.

"A" Applicable to AB models. Used for constraints on the A matrix.

"B" Applicable to AB models. Used for constraints on the B matrix.

"Adiag" Applicable to AB models. Used for constraints on the whole diagonal of the A matrix (see below).

"Bdiag" Applicable to AB models. Used for constraints on the whole diagonal of the B matrix (see below).

3. An integer:

   case 1 : applies to the codes "C", "lrC", "A" and "B". Indicates the row of the restricted element.

   case 2 : applies to the codes "Adiag" and "Bdiag". Indicates what kind of restriction is to be placed on the diagonal: any valid scalar indicates that the diagonal of A (or B) is set to that value. Almost invariably, this is used with the value 1. IMPORTANT: if this argument is `NA`, all *non-diagonal* elements are constrained to 0, while diagonal elements are left unrestricted.

4. An integer: the column of the restricted element, for the codes "C", "lrC", "A" and "B". Otherwise, unused.

5. A scalar: for the codes "C", "lrC", "A" and "B", the fixed value the matrix element should be set to (may be omitted if 0). Otherwise, unused.

A few examples:

- `SVAR_restrict(&M, "C", 3, 2, 0)`; in a C model called `M`, sets $C_{3,2} = 0$. As a consequence, the IRF for variable number 3 with respect to the shock number 2 starts from zero.

- `SVAR_restrict(&foo, "A", 1, 2, 0)`; in an AB model called `foo`, sets $A_{1,2} = 0$.

- `SVAR_restrict(&MyMod, "lrC", 5, 3, 0)`; in a C model called `MyMod`, restricts C such that the long-run impact of shock number 3 on variable number 5 is 0. This implies that the cumulated IRF for variable 5 with respect to shock 3 tends to zero.

- `SVAR_restrict(&bar, "Adiag", 1)`; in an AB model called `bar`, sets $A_{i,i} = 1$ for $1 \leq i \leq n$.

- `SVAR_restrict(&baz, "Bdiag", NA)`; in an AB model called `baz`, sets $B_{i,j} = 0$ for $i \neq j$.

`SVAR_setup(string type, list Y, list X, int varorder)`

Initialises a model: the function's output is a bundle. The function arguments are:

1. A type string: at the moment, valid values are "C", "plain" and "AB";
2. a list containing the endogenous variables;
3. a list containing the exogenous variables;
4. a positive integer, the VAR order.

B Contents of the model bundle

Basic setup	
`step`	done so far
`type`	string, model type
`n`	number of endogenous variables
`p`	VAR order
`k`	number of exogenous variables
`T`	number of observations
`Xlist`	list of exogenous variables (as matrix)
`X`	exogenous variables data matrix
`Ylist`	list of endogenous variables (as matrix)
VAR	
`VARpar`	autoregressive parameters
`mu`	coefficients for the deterministic terms
`E`	residuals from base VAR (as matrix)
`Sigma`	unrestricted covariance matrix
SVAR setup	
`Rd1`	main matrix of constraints
`aux`	depends
`nc1`	number of constraints in `Rd1`
`nc2`	number of constraints in `aux`
`horizon`	horizon for structural VMA
`cumul`	vector of cumuland variables
`ncumul`	number of cumuland variables
`snames`	string, Names for shocks (may be empty)
SVAR post-estimation	
`S1`	estimated A (or C)
`S2`	estimated B
`theta`	coefficient vector
`IRFs`	IRFs
Bootstrap-related	
`nboot`	number of bootstrap replications
`boot_alpha`	bootstrap confidence level
`bootdata`	output from the bootstrap

```
set echo off
set messages off
include SVAR.gfn
open BlQuah.gdt

list X = DY U
list exog = const time
maxlag = 8

# set up the model
BQModel = SVAR_setup("C", X, exog, maxlag)
BQModel["horizon"] = 40

# set up the long-run restriction
SVAR_restrict(&BQModel, "lrC", 1, 2, 0)

# cumulate the IRFs for variable 1
SVAR_cumulate(&BQModel, 1)

# set up names for the shocks
BQModel["snames"] = "Supply Demand"

# do estimation
SVAR_estimate(&BQModel)

# retrieve the demand shocks
dShock = GetShock(&BQModel, 2)

# bootstrap
bfail = SVAR_boot(&BQModel, 1024, 0.9)

# page 662
IRFsave("bq_Yd.pdf", &BQModel,  1, 1)
IRFsave("bq_ud.pdf", &BQModel, -2, 1)
IRFsave("bq_Ys.pdf", &BQModel,  1, 2)
IRFsave("bq_us.pdf", &BQModel, -2, 2)
```

Table 6: Blanchard-Quah example

```
             coefficient   std. error      z       p-value
  --------------------------------------------------------
  C[ 1; 1]     0.0575357   0.0717934      0.8014   0.4229
  C[ 2; 1]     0.217542    0.0199133     10.92     8.80e-28 ***
  C[ 1; 2]    -0.907210    0.0507146    -17.89     1.45e-71 ***
  C[ 2; 2]     0.199459    0.0111501     17.89     1.45e-71 ***

Bootstrap results (1000 replications)

             coefficient   std. error      z      p-value
  -------------------------------------------------------
  C[ 1; 1]      0.232452   0.285316      0.8147   0.4152
  C[ 2; 1]      0.191064   0.0786388     2.430    0.0151   **
  C[ 1; 2]     -0.829021   0.113861     -7.281    3.31e-13 ***
  C[ 2; 2]      0.222009   0.0648956     3.421    0.0006   ***
```

Table 7: Output for the Blanchard-Quah model

```
set echo off
set messages off
include SVAR.gfn
open IS-LM.gdt

list X = q i m
list Z = const time

ISLM = SVAR_setup("AB", X, Z, 4)
ISLM["horizon"] = 48

SVAR_restrict(&ISLM, "Adiag", 1)
SVAR_restrict(&ISLM, "A", 1, 3, 0)
SVAR_restrict(&ISLM, "A", 3, 1, 0)
SVAR_restrict(&ISLM, "A", 3, 2, 0)
SVAR_restrict(&ISLM, "Bdiag", NA)
ISLM["snames"] = "uIS uLM uMS"
SVAR_estimate(&ISLM)

Amat = ISLM["S1"]
Bmat = ISLM["S2"]

printf "Estimated contemporaneous impact matrix (x100) =\n%10.6f", \
  100*inv(Amat)*Bmat

rej = SVAR_boot(&ISLM, 2000, 0.95)
IRFplot(&ISLM, 1, 2)
```

Table 8: Estimation of an AB model — example from Lütkepohl and Krätzig (2004)