How to Estimate Structural Vector Autoregressions using Matlab

by Robert Vigfusson[1]

May 10, 1999

I have written a set of Matlab M-files to estimate structural vector autoregressions (VAR). This note describes how to use these files.

Each of the following sections is divided into two subsections: Theory and Matlab Code . The theory is a condensed description of the main concepts of structural VARs. The descriptions borrow heavily from Christiano, Eichenbaum and Evans (1998) Handbook chapter and Hamilton (1994)'s time series text.[2] The Matlab Code subsection should give the user enough information to use the procedures for his own work. At the end of the paper, I give examples of how to estimate specific structural VARs using the functions described here.

The sections are the following

- Set up and Estimation of Reduced Form VAR
- Fundamental Shocks and The Structural VAR
- Impulse Responses
- Constructing Confidence Intervals for Impulse Responses
- Variance Decomposition
- Examples

Software Requirements These programs require Matlab 5. Non-recursive estimation requires the optimization toolbox. The current code is written for optimization toolbox version 1.5.

1 Set up and Estimation of Reduced Form VAR

1.1 Theory

The basic building block is a vector time series Z_t with $nvars$ elements (variables). The vector Z_t evolves over time according to:.

$$Z_t = B_0 + B_1 Z_{t-1} + B_2 Z_{t-2} + ... + B_q Z_{t-q} + u_t$$

where $Eu_t u_t' = V$. The vector u_t is assumed to be uncorrelated with past values of Z_t.

Estimation of the VAR's coefficients $\{B_j\}_{j=0}^{q}$ can be done equation by equation by OLS. The matrix V can be estimated from the sample residuals $(1/T)\sum \hat{u}_t \hat{u}_t'$.

1.2 Matlab Code

The first step is to prepare the data. In a spreadsheet or statistical program, enter the data such that each column is a variable and each row is an observation. There should be no missing observations. Save this file to an ascii text file. In order that Matlab can read the file, each row should have the same number of entries. Also, do not include variable names with the data – just numbers. (I'll assume the file is named *datrep.txt.)* Next start Matlab and enter the following two commands. (Of course, this can be done in a script m-file.)

```
load datrep.txt
```

[1] My email address is r-vigfusson@nwu.edu. Updates to these programs will be made available on my webpage htttp://pubweb.nwu.edu/˜rjv541.

[2] One therefore should see these two sources for a more complete treatment. Of course, comments on how to improve the exposition here are welcome.

```
data = datrep;
```

The next step is to define a variable for the number of lags in the VAR. If the number of lags q is 4 then we define

$$\text{nlags } = 4;$$

By default, the program includes a constant in the VAR. If the VAR doesn't have a constant then define a variable *hascon* and set it equal to zero.

$$\texttt{hascon} = 0$$

If *hascon* is set equal to one, then a constant is included in the VAR.

The next line estimates the VAR. The syntax is

```
[betaz,sigma,residuals] = estimatevar(data,nlags,hascon);
```

The procedure `estimatevar` returns three variables. The first *betaz* is the coefficients of the VAR. The second *sigma* is the variance covariance matrix. The third *residuals* is a matrix of the residuals from the VAR. The *residuals* are organized the same way as the data matrix is organized. Each column is a variable and each row is an observation.

The matrix *betaz* has *nvars* rows and *nlags* times *nvars*+*hascon* columns. When a constant is included, the first column is the constant. Columns two to $nvars + 1$ are the estimated coefficients for the variables Z_{t-1}. The next *nvars* columns would be the estimated coefficients for the Z_{t-2}.Without a constant, the first nvars columns are the estimated coefficients for the variables Z_{t-1}.

For a 2 variable (bivariate) VAR with 3 lags, *betaz* would equal

$$\text{betaz} = \left[\begin{array}{cccc} B_0 & B_1 & B_2 & B_3 \end{array} \right]$$

where B_j is defined above.

The procedure is written with a default to include a constant. Therefore one does not have to include the *hascon* variable if you want to estimate the VAR with a constant.

```
[betaz,sigma]=estimatevar(data,nlags);
```

General Notes on Using Matlab The position and value of a variable and not its name is what is important when using a Matlab function. For example, *nlags*, the number of lags, was used in the `estimatevar` command.

```
[betaz,sigma]=estimatevar(data,nlags);
```

The value and not the name, however, is what is important. If *nlags* were equal to four then

```
[betaz, sigma] = estimatevar(data, 4);
```

would produce exactly the same output. I define the variable *nlags* as I use it in subsequent procedures.

Another thing to keep in mind is that one can use the command `help` to find out more about a function in Matlab. For example `help min` will give instructions on how to use Matlab's `min` command. The commands described here also have help comments. For example

```
help estimatevar
```

returns

```
ESTIMATEVAR Estimates a Vector Autoregression.
 [Beta,Sigma,UZ]=ESTIMATEVAR(DATA,nlags,hasconstant);
 Finds coefficients estimates BETA, variance covariance matrix SIGMA, and residuals UZ
 that are the results of estimate by OLS a VAR on the matrix DATA.
 The matrix DATA has each row being an observations
 from all the variables. NLAGS is the number of lags.
 HASCONSTANT is equal to one for a VAR with a constant and equal to zero otherwise.
```

2 Fundamental Shocks and The Structural VAR

2.1 Theory

The next step is to calculate the structural matrix A_0. The values of u_t are not the standard structural shocks. Instead we assume that the relationship between the VAR disturbances u_t and the fundamental economic shocks ε_t is given by

$$A_o u_t = \varepsilon_t$$

$$\begin{aligned} A_o Z_t &= A_o B_1 Z_{t-1} + A_o B_2 Z_{t-2} + ... + A_o B_q Z_{t-q} + A_o u_t \\ &= A_o B_1 Z_{t-1} + A_o B_2 Z_{t-2} + ... + A_o B_q Z_{t-q} + \varepsilon_t \end{aligned}$$

We need to know the value of A_o in order to calculate impulse responses and variance decompositions. To find a unique A_o however requires further assumptions. Many matrices might solve the equation

$$A_o^{-1} \left(A_o'\right)^{-1} = V$$

In general one can choose between two different approaches. The simpler and more common is the recursiveness assumption. One implements this assumption by defining the matrix A_o^{-1} to be the lower Cholesky decomposition of V.

The other approach is to not to assume recursiveness but rather assume that enough entries are zero or otherwise constrained that only a unique A_o solves the above equation and satisfies these further assumptions. However the uniqueness of this A_o can be difficult to establish.

2.2 Matlab Code

2.2.1 A_0 with the Recursive Assumption

Finding the value of A_0 is much easier with the recursiveness assumption than with the non-recursiveness assumptions. Recursiveness is straightforward. We just define A_0 to be

```
a0 = inv(chol(sigma)');
```

The transpose operator is necessary since Matlab's `chol` command creates an upper rather than lower triangular matrix. The inverse is used in order to define A_0 rather than A_0^{-1}.

2.2.2 A_0 without the Recursive Assumption

Non-recursiveness is not as straightforward. The matrix A_0 is found as the matrix that maximize the likelihood function (Hamilton p. 332)

$$L\left(A_o, V\right) = -\frac{T}{2}\log\left(2\pi\right) + \frac{T}{2}\log\left|A_o\right|^2 - \frac{T}{2}\text{trace}(A_o V A_o')$$

The solution is found by using the constrained minimization routine `constr`[3]. I use `constr` rather than `fminu` in order to impose the constraint that the diagonal elements of A_o are non-negative[4].

To estimate the matrix A_o, the user must write two functions. The first function forms the desired matrix A_o from a vector of parameters x. The second function generates a vector x of starting values that satisfy the constraints placed on A_0. This vector x will be used as starting values in the numerical optimization procedure.

An example of the first function that creates the A_o matrix of Sims and Zha is below.

[3]The computer solves the maximization problem by minimizing $-1 * L(A_0, V)$.

[4]A user who understands the constr function could add additional constraints by modifying the matlab function that calculates the likelihood and the constraints.

```
function [azero]=mkmatrix(x);
x=x';
azero=[x(1:7); 0 x(8:9) 0 -x(8) 0 -x(8); x(10:12)zeros(1,4); x(13) . . .
zeros(1,2) x(14:17);x(18) zeros(1,3) x(19:21) ;x(22) zeros(1,4) x(23:24);x(25) ...
zeros(1,5) x(26)];
```

This function creates a matrix that looks like

$$A = \begin{array}{ccccccc} x(1) & x(2) & x(3) & x(4) & x(5) & x(6) & x(7) \\ 0 & x(8) & x(9) & 0 & -x(8) & 0 & -x(8) \\ x(10) & x(11) & x(12) & 0 & 0 & 0 & 0 \\ x(13) & 0 & 0 & x(14) & x(15) & x(16) & x(17) \\ x(18) & 0 & 0 & 0 & x(19) & x(20) & x(21) \\ x(22) & 0 & 0 & 0 & 0 & x(23) & x(24) \\ x(25) & 0 & 0 & 0 & 0 & 0 & x(26) \end{array}$$

The second function creates a vector x that will be used as the initial guess to the maximization routine. An example for the Sims Zha matrix is below:

```
function avec = mkstart
%Generate Random Starting Values
avec=2*randn(26,1); %random starting values
%making the diagonal elements non-negative
avec([1 8 12 14 19 23 26]) = abs(avec([1 8 12 14 19 23 26]));
```

Numerical Maximization can be difficult. Common problems include failing to converge or converging at a local rather than a global maximum. To overcome these problems, I recommend to estimate the matrix A_0 many times using random starting values and then choose the estimate with the greatest likelihood value. The procedure `estnonreca` implements this strategy. Given the variance covariance matrix *sigma* and the user supplied procedures `mkmatrix` and `mkstart` supplied above, `estnonreca` estimates the A_0 matrix *numtries* times and returns the best estimate out of the set.

$$\texttt{a0best} = \texttt{estnonreca}(\texttt{sigma}, \texttt{numtries}, \texttt{'mkmatrix'}, \texttt{'mkstart'})$$

3 Impulse Responses

3.1 Theory

Having found the A_0 matrix, the next step is to calculate the impulse responses to a fundamental shock. The basic idea is the following. Suppose that the VAR has the following form.

$$Z_t = B_1 Z_{t-1} + B_2 Z_{t-2} + ... + B_q Z_{t-q} + A_o^{-1} \varepsilon_t$$

We suppose that the $j-th$ fundamental errors take on the value one while the other fundamental errors are set equal to zero. The impulse response is how the variables in the VAR respond to this shock. The impulse response in the first period is

$$\Gamma_j(1) = A_o^{-1} e_j$$

where the j-th element of e_j is equal to one and all other elements are zero. The vector $\Gamma_j(1)$ has length *nvars* with the same ordering as Z_t. The second period impulse response is

$$\Gamma_j(2) = B_1 A_o^{-1} e_j$$

The third is

$$\Gamma_j(3) = B_1 B_1 A_o^{-1} e_j + B_2 A_o^{-1} e_j$$

Note that if the matrix A_o were the identity matrix, then the impulse response functions for all shocks would be the moving average representation of the VAR.

3.2 Matlab Code

The Matlab syntax for calculating the impulse response function is the following. Define a variable that determines which error term will receive the shock. Define another variable for how many periods to calculate the impulse responses.

$$\begin{aligned} \text{errshk} &= 4; \\ \text{nstep} &= 15; \end{aligned}$$

The next step calls the procedure that actually calculate the impulse responses. The procedure returns a *nstep* by *nvars* matrix with each column corresponding to a variable in the original VAR and each row by the period.

$$\text{impzmat} = \text{mkimprep(betaz,a0,nlags,errshk,nstep);}$$

After discussing confidence intervals, I will describe how to plot these impulse responses.

4 Constructing Confidence Intervals for Impulse Responses

The procedures use simulation to calculate the confidence intervals around the impulse response functions. These confidence intervals are pointwise confidence intervals.

The procedure `mkimpci` constructs the confidence interval with the following steps. The first step is to generate the artificial data under the null hypothesis that the estimated model correctly represents the data. The artificial data is created by using the estimated coefficients for *beta* along with error terms. These error terms can be generated using either of two possible methods: Bootstrap or Monte Carlo. For the Bootstrap method, the error terms are generated by sampling randomly with replacement from the residuals of the original vector autoregression. (Note that the *nvars* residuals for a single time period are chosen together.) For the Monte Carlo method, the error terms are generated by multiplying random vectors drawn from a N(0,I) distribution by A_o^{-1}.

Having generated the artificial data, the next step is to estimate a VAR on this data. Using the estimated variance covariance matrix, estimate a new A_0 matrix. The impulse response is calculated using these new estimates. These impulse responses are then stored in memory.[5]

The above steps are repeated many times to generate a large sample of impulse responses. The procedure then calculates the confidence intervals using two different methods. The first uses α and $1 - \alpha$ percentile values of the sample for the lower and upper bounds. The second relies on the normal approximation and calculates the confidence intervals as two times the square root of the sample variance around the estimated impulse response function.

4.1 Matlab code

Before calling the mkimpci procedure three variables need to be defined. The first variable *ndraws* defines how many simulations to use in constructing the confidence intervals. The second variable *nobs* defines how many observations each simulated time series should have. Typically this is taken to be the number of observations in the actual data. (To eliminate the effects of initial conditions, the procedure generates 2*nobs* observations and then uses only the last *nobs* of data.) The third variable *pctg* gives the value of α to be used in constructing the confidence intervals.

$$\begin{aligned} \text{pctg} &= 0.05; \\ \text{ndraws} &= 100 \\ \text{nobs} &= 120; \end{aligned}$$

[5]If memory constraints were a problem, the impulse responses could be stored to disk instead. This would require a modification of the `mkimpci` procedure.

4.1.1 Recursive

The confidence intervals are constructed by calling the procedure `mkimpci`. The first five input variables are the same as the `mkimprep`'s inputs. These are followed by the three defined above. The procedure returns the percentile lower and upper bounds first, followed by the upper and lower standard deviation bounds, and finally the sample variance of the impulse responses. Each matrix is organized the same way as the impulse responses (variables columns, periods rows).

```
[cilmc,ciumc,cilvarm,ciuvarm,varm] = mkimpci(betaz,a0,nlags,errshk,nstep,ndraws,nobs,pctg);
```

This first call will calculate the confidence intervals based upon the Monte Carlo simulated data. If the matrix of residuals from the `estimatevar` command is included as an input then the function calculates the confidence intervals based upon the Boot Strapped simulated data.

```
[cilbt,ciubt,cilvarb,ciuvarb,varb]=mkimpci(betaz,a0,nlags,errshk,nstep,ndraws,nobs,pctg,residuals);
```

4.1.2 Non-recursive

The non-recursive system is identical except that the four additional parameters from the `estnonreca` procedure are also included as inputs. In practice, one may want to use a much smaller value of *numtries* for the confidence intervals than the one used in the `estnonreca` procedure. Using the same number of *numtries* would be time consuming; as one would be making $ndraws * numtries$ maximization attempts.

To get Monte-Carlo-based confidence intervals pass an empty matrix [] for the *residuals*.

```
[cilmc,ciumc,cilvar,ciuvar,var]  =  mkimpci(betaz,a0,nlags,errshk,nstep, ...
                                    ndraws,nobs,pctg,[],sigma,numtries,'mkmatrix','mkstart');
```

```
[cilbt,ciubt,cilvar,ciuvar,var]  =  mkimpci(betaz,a0,nlags,errshk,nstep,...
                                    ndraws,nobs,pctg,residuals,sigma,numtries,'mkmatrix','mkstart');
```

Note that the procedure calls should be one line, however, in order to fit the page I put in a line break.

4.2 Making Plots In Matlab with An Application to Confidence Intervals

Matlab graphics can be very basic and easy to make. To plot the first series in data just type

```
plot(data(:,1))
```

A new figure window that contains the plot of the first column of data will appear. This graph however is somewhat plain.

A Plain Graph

I have included a series of commands below that will make a graph that looks like this

A More Interesting Graph

This first step is to create a vector of series names. Using the Matlab command `char, we` will create a series vector *varnames*

```
varnames=char('Y','P','PCOM','FF','TOTR','NBR','M');
```

The next command `figure` creates a fresh graphics window.

```
figure
```

The next command `for` begins the loop

```
for zr = 1:4;
```

The next command `subplot` divides the page into pieces with 2 rows and 2 columns. The third argument tells Matlab where to put the current graph.

```
subplot(2,2,zr)
```

The next command `hold on` causes the `plot` command to put one plot on top of another without erasing the other plots.

```
hold on
```

The next two lines plot the confidence intervals and the impulse response for the *zr* variable. The third line just sets the axis limits to the range of the data.

```
plot([100*cilvarb( : ,zr)100*ciuvarb(:,zr)100*impzmat(:,zr)]);
   plot([100*cilb( : ,zr)100*ciub(:,zr)],'x-');
```

axis tight

The `title` command creates a title. The variable *varnames* is used to label the graph.

```
title(['Response by' varnames(zr,:)]);
```

The end command ends the loop.

end;

The final command suptitle is not a built in Matlab function but is included with the VAR programs[6]. . This function `suptitle` puts a title above the subplots.

```
suptitle('Shock to Fed Funds')
```

The Matlab manuals describes how to make graphs in much more detail.

5 Variance Decomposition

The variance decomposition is concerned with identifying the error in forecasting a VAR s periods into the future.

$$Z_{t+s} - Z_{t+s|s} = u_{t+s} + \Psi_1 u_{t+s-1} + \Psi_2 u_{t+s-2} + ... + \Psi_{s-1} u_{t+1}$$

where Ψ_j are the moving average coefficients and $Z_{t+s|s}$ is the forecast at time t of the value of Z_{t+s}.

The mean square error of the s-period forecast error is therefore

$$\text{MSE}\left(Z_{t+s|s}\right) = V + \Psi_1 V \Psi_1' + \Psi_2 V \Psi_2' + ... + \Psi_{s-1} V \Psi_{s-1}'$$

We are interested in knowing how much does each of the fundamental errors contributes to this error term.

$$Z_{t+s} - Z_{t+s|s} = A_o^{-1}\varepsilon_{t+s} + \Psi_1 A_o^{-1}\varepsilon_{t+s-1} + \Psi_2 A_o^{-1}\varepsilon_{t+s-2} + ... + \Psi_{s-1} A_o^{-1}\varepsilon_{t+1}$$

Define a_j to be the jth column of A_o^{-1} then since $A_o^{-1}A_o^{-1\prime}$ equals V we can decompose the MSE as

$$\text{MSE}\left(Z_{t+s|s}\right) = \sum a_j a_j' + \Psi_1 a_j a_j' \Psi_1' + \Psi_2 a_j a_j' \Psi_2' + ... + \Psi_{s-1} a_j a_j' \Psi_{s-1}'$$

so the contribution of any one shock can be defined as

$$a_j a_j' + \Psi_1 a_j a_j' \Psi_1' + \Psi_2 a_j a_j' \Psi_2' + ... + \Psi_{s-1} a_j a_j' \Psi_{s-1}'$$

The percentage of the s-period ahead forecast error explained by a particular shock can be easier to interpret. So for the confidence intervals we will calculate.

$$100 * \frac{a_j a_j' + \Psi_1 a_j a_j' \Psi_1' + \Psi_2 a_j a_j' \Psi_2' + ... + \Psi_{s-1} a_j a_j' \Psi_{s-1}'}{\text{MSE}\left(Z_{t+s|s}\right)}$$

[6] The program `suptitle` was written by Drea Thomas of Mathworks.

5.1 Matlab Code

This code consists of two procedures. The first `vardecomp` calculates the mean squared error and one error shock's contribution to the mean squared error. The second `mkvdci` calculates a confidence interval of the percentage contribution to the MSE.

The `vardecomp` procedure has the same inputs as the impulse response function `mkimprep` has:. the VAR coefficients, the factorization of the variance covariance matrix, the number of lags in the VAR, the choice of error term, and the number of forecast period steps. The vardecomp procedure returns two matrices. Both matrices *ms* and msdec are a three dimensional matrix. with dimensions *nvars* by *nvars* by *nstep*. The value $ms(j, k, s)$ is the s-period forecast covariance of the j-th and k-th elements of Z_{t+s}. The value of $msdecomp(j, k, s)$ is how much the *errshk*-th fundamental error contributes to the value of $ms(j, k, s)$. The procedure call is

[ms,msdec] = vardecomp(betaz,a0,nlags,errshk,nstep);

5.1.1 Confidence Intervals

Constructing confidence intervals for the variance decomposition is identical to how they are constructed for the impulse responses. The only difference is that the main procedure is `mkvdci` rather than `mkimpci` and the returns are different. Each gives the percentage of variance explained by the *errshk*. The returns are three dimensional matrices. with dimensions *nvars* by *nvars* by *nstep*.

Recursive Monte Carlo

[vdcilmc,vdciumc,vdcilvarmc,vdciuvarmc,vdvarmc] = mkvdci(betaz,a0,nlags,errshk,nstep,ndraws,nobs,pctg);

Bootstrap

[vdcilbt,vdciubt,vdcilvarbt,vdciuvarbt,vdvarbt]=mkvdci(betaz,a0,nlags,errshk,nstep,ndraws,nobs,pctg,residuals);

Non-recursive Monte Carlo

[vdcilmc,vdciumc,vdcilvarmc,vdciuvarmc,vdvarmc] = mkvdci(betaz,a0,nlags,errshk,nstep, ...
ndraws,nobs,pctg,[],sigma,`numtries`, `'mkmatrix'`, `'mkstart'`);

Bootstrap

[vdcilbt,vdciubt,vdcilvarbt,vdciuvarbt,vdvarbt] = mkvdci(betaz,a0,nlags,errshk,nstep, ...
ndraws,nobs,pctg,residuals,`sigma`, `numtries`, `'mkmatrix'`, `'mkstart'`

6 Examples

6.1 Recursive example

We want to estimate a model with 7 variables with 4 lags. The data has already been prepared. We are interested in the impulse response to the fourth shock. and want to have the 5 and 95% confidence intervals. We also want to create the first column of Table 3 which shows the Variance Decomposition.

```
load datrep;
data=datrep;
%('Y','P','PCOM','FF','TOTR','NBR','M');
```

```
nlags  = 4;
hasconst = 1;

[betaz,sigma,residuals]=estimatevar(data,nlags,hasconst);

a0rec=inv(chol(sigma)');

errshk = 4;
nstep = 15;

impzmat=mkimprep(betaz,a0rec,nlags,errshk,nstep);
pctg=0.05;
ndraws=100;
nobs=120;
%[cilmc,ciumc,cilvar,ciuvar,var]=mkimpci(betaz,a0rec,nlags,errshk,nstep,ndraws,nobs,pctg);
[cilb,ciub,cilvarb,ciuvarb,varb]=mkimpci(betaz,a0rec,nlags,errshk,nstep,ndraws,...
nobs,pctg,residuals);

[mse,msedecomp]=vardecomp(betaz,a0rec,nlags,errshk,nstep);

vdpctg = msedecomp./mse;

[cilvd,ciuvd,cilvarbv,ciuvarbv,varbv]=mkvdci(betaz,a0rec,nlags,errshk,nstep,ndraws,nobs,pctg,residua
kstep = [2 4 8 12];

for zk=1:4;
   lowci(:,zk) = diag(squeeze(cilvd(:,:,kstep(zk))));
   lowvar(:,zk) = diag(squeeze(cilvarbv(:,:,kstep(zk))));

table(:,zk) = diag(squeeze(vdpctg(:,:,kstep(zk))));
highci(:,zk) = diag(squeeze(ciuvd(:,:,kstep(zk))));
highvar(:,zk) = diag(squeeze(ciuvarbv(:,:,kstep(zk))));

end;

%The first column of Table 3 from CEE
100*[lowci(:,1) table(:,1) highci(:,1)]

varnames=char('Y','P','PCOM','FF','TOTR','NBR','M');

figure
for zr=1:4;
   subplot(2,2,zr)
   hold on
   plot([ 100*cilvarb(:,zr) 100*ciuvarb(:,zr) 100*impzmat(:,zr)]);
   plot([100*cilb(:,zr) 100*ciub(:,zr)],'x-');
   title(['Response by ' varnames(zr,:)]);
   axis tight
end;

suptitle('Shock to Fed Funds')
figure
for zr=5:7;
```

```
   subplot(2,2,zr-4)
   hold on
      plot([ 100*cilvarb(:,zr) 100*ciuvarb(:,zr) 100*impzmat(:,zr)]);
   plot([100*cilb(:,zr) 100*ciub(:,zr)],'x-');
   title(['Response by ' varnames(zr,:)]);
   axis tight
end;
suptitle('Shock to Fed Funds')
```

The result is the following impulse responses.

Modification. Change the ordering with the first variable being interchanged with the fifth and the sixth variable being interchanged with the seventh. This is very easy to do in Matlab. Replace the line `data=datrep;` with the line `data=datrep(:,[5 2 3 4 1 7 6];`. Nothing else in the program needs to be modified. Of course, the results will be different.

Modification: Suppose we wanted to interchange the third and fourth variables but were still interested in following the effects of the shock to what is currently the fourth shock. First replace `data=datrep;` with the line `data=datrep(:,[1:2 4 3 5:7]);`. Second the line `errshk = 4`; needs to be changed to `errshk = 3`;. Again that is all.

Modification Only using a subperiod of the data. Suppose instead of using the 1+nlags to nobs observations you only wanted to use a subset of the data from say abeg to aend. As long as abeg is greater than or equal to 1+ *nlags* and aend is less than or equal to *nobs* then this modification can be again made at the data statement. Insert the line `data=datrep(abeg-nlags:aend,:);`

Modification Only using a subset of the variables. want to exclude the 2nd and the 6th variables Set `data=datrep(:,[1 3 4 5 7]); Change errshk = 3.`

6.1.1 Nonrecursive example

We now estimate the Sims Zha model described in CEE. Keep in mind that nonrecursive models take much much longer to estimate than recursive models. This program assumes that you have created the two files `mksimzha` and `svsz` to create the A_0 matrix and the starting values. A listingof these two files is below the current file.

nonrecmain.m

```
load c:\robswork\identeich\szdat_dm.dat;
data=szdat_dm;
data(:,1)=data(:,1)./sqrt(10.0);
nl = 4;
[bz,sig,rz]=estimatevar(data,nl);
a0=estnonreca(sig,20,'mksimzha','svsz')
eshk=3;
nstp = 15;
```

```
 impz=mkimprep(bz,a0,nl,eshk,nstp);
pc=0.05;
nd=10; %should be more like a 100
nb=120;
[lp,up,lv,uv]=mkimpci2(bz,a0,nl,eshk,nstp,nd,nb,pc,rz,sig,3,'mksimzha','svsz');
[mse,msed]=vardecomp(bz,a0,nl,eshk,nstp);
vdpctg = msed./mse;
[vlp,vup,vlv,vuv]=mkvdci(bz,a0,nl,eshk,nstp,nd,nb,pc,rz,sig,3,'mksimzha','svsz');
kstep = [2 4 8 12];
for zk=1:4;
 lowci(:,zk) = diag(squeeze(vlp(:,:,kstep(zk))));
 lowvar(:,zk) = diag(squeeze(vlv(:,:,kstep(zk))));
table(:,zk) = diag(squeeze(vdpctg(:,:,kstep(zk))));
highci(:,zk) = diag(squeeze(vup(:,:,kstep(zk))));
highvar(:,zk) = diag(squeeze(vuv(:,:,kstep(zk))));
end;


varnames=char('Y','P','PCOM','FF','TOTR','NBR','M');
for zr=1:7;
 subplot(4,2,zr)
 hold on
 plot([impz(:,zr) cilb(:,zr) ciub(:,zr)],'.');
 % title(['Response by ' varnames(zr,:)]);
 axis tight
end;
```

mksimzha.m

function [azero]=mksimzha(x);
x=x';
azero=[x(1:7); 0 x(8:9) -x(8) 0 -x(8) 0; x(10:12)zeros(1,4); x(13) zeros(1,2) x(14:17);...
x(18) zeros(1,3) x(19:21) ;x(22) zeros(1,4) x(23:24);x(25) zeros(1,5) x(26)];

svsz.m

function avec=svsz;
avec=2*randn(26,1);
avec([1 8 12 14 19 23 26]) = abs(avec([1 8 12 14 19 23 26]));