# ATQM Lectures on GMM: Getting Started in Matlab

Mark Schaffer
March 2006

## 1. Setup

Matlab 7.0 is started from

Start > Programs > Matlab 7.0

and starts with the “Current Directory” set to the network drive M:\. You should create a new folder for your Matlab work called, say, “Matlabwork”. You can do this within Matlab; if the “Current Directory” window isn't visible, make it so with the View menu. Or you can simply do it in Windows.

Once you've created the folder, you need to add it to the “Path”, i.e., tell Matlab that it should look in that folder as well whenever it is looking for something. R-click on the folder and choose “Selected folder and subfolders”.

Now change folders to “Matlabwork” so it is your current directory. Double-clicking should do it.

## 2. Importing data

Matlab *data files* end with the extension “.mat” (for matrix). Matlab can import data from text files very easily. We will be using data from a famous early macro model by Klein. The model is often used in econometrics textbooks to illustrate IV and simultaneous equations sytems (see, e.g., Greene's textbook, *Econometric Analysis*), and for benchmarking, i.e., checking the accuracy of calculations. The Klein model benchmarks can be found on Clint Cummin's “Econometric Benchmarks” web page http://www.stanford.edu/~clint/bench/.

The data are in a text file called klein_data.txt, and it will be converted into a Matlab data file called klein_data.mat. You will also need to download some Matlab utilities. All are available either on WebCT and/or on my Matlab web page:

http://www.sml.hw.ac.uk/ecomes/matlab

The files you should download are below. Save them all in your Matlabwork folder. Don’t change the names of the files, and make sure the extensions (.txt, .m, .pdf) aren’t changed.

klein_data.txt
normcdf.m
norminv.m
chi2cdf.m
rows.m
cols.m
mprint.m
Matlabnotes.pdf

The files ending in “.m” are Matlab *programs* that do not come with the student version of Matlab; these are simple (and free) versions that you can use instead. “rows.m”, “cols.m” and “mprint.m” are Matlab programs that are part of James LeSage’s “Econometrics Toolbox”, a very large and freely-available collection of estimation routines and utilities available via his website, http://www.spatial-econometrics.com/. The “mprint.m” program is a utility that outputs regression results in a nice format. The last item in the list is this document.

The format of the Klein data text file is a line of variable names and then rows of figures. There are 22 lines (years) of data. The data are tab-delimited. To import the Klein dataset and convert it into a Matlab data file, use

```
File > Import data
```

and select klein_data.txt. You may need to set the number of text headers to 1 unless Matlab figures this out for itself. Proceed, and when given the choice, choose “Create vectors”. This makes the dataset load as a set of 22x1 column vectors, each with its own name. The variables are:

| | |
|---|---|
| Year | Runs from 1920 to 1941 |
| C | Consumption |
| P | Profits in the corporate sector |
| Wp | Wage income, private sector |
| I | Investment |
| K | Capital stock (start-year) |
| GNP | GNP |
| Wg | Wage income, public sector |
| G | Government spending |
| T | Taxes |

After importing the data, you should see the data in memory – click on the “Workspace” tab to confirm this.

To save the dataset as a Matlab data file:

```
File > Save Workspace as …
```

and use the name klein_data.mat.

## 3. Some simple calculations

Matlab is a matrix-oriented language with a very natural syntax. A single equals sign = is the assignment operator (a double == is the logical operator), so let's create a new variable W that is aggregate wage income for both the private and government sectors:

```
        Wpg=Wp+Wg
```

and you will see the new variable appears in the list of variables in the workspace. It is also printed to the screen. If you don't want the result of a calculation printed to the screen, add a semicolon ; to the end:

```
        Wpg=Wp+Wg;
```

Let's prepare to estimate a simple consumption function, in which consumption C is a function of total wage income Wpg, profits P, and a constant.

First create a variable for the number of observations. We will call it “nobs” (and reserve the name N for later). You can omit the ; to see the result.

```
nobs=length(Year)
```

The constant will be the coefficient on a vector of ones, so we need a new variable “one” that is a 22x1 vector of ones. We use the Matlab function ones(), which takes as its arguments the number of rows and number of columns needed.

```
one=ones(nobs,1)
```

We call our dependent variable Y, so create it:

```
Y=C
```

Our regressors X will be a constant term, profits, and wages. We create this 22x3 matrix with

```
X=[one P Wpg]
```

The matrix transpose operator is ', and the multiplication operator is *, so the following calculates the matrix crossproduct X'X. The result is a 3x3 matrix called XX:

```
XX=X'*X
```

The inverse function is inv(), so the following creates a new 3x3 matrix that is the inverse of X'X:

```
XXinv=inv(XX)
```

X is 22x3 and Y is 22x1, so X'Y will be a 3x1 matrix. Save this as a variable called XY:

```
XY=X'*Y
```

We can now calculate the OLS estimator for our simple macro consumption function, and save the coefficient as the variable beta:

```
beta=XXinv*XY
```

The vector beta is 3x1 with elements, and the following should appear on the screen:

```
beta =
   14.6427
    0.2598
    0.8381
```

The first element is the constant, the second is the coefficient on profits P, and the third is the coefficient on wage income Wpg.

We could also do it in one step (try it):

```
inv(X'*X)*X'*Y
```

or even using a Matlab trick that is sometimes useful, the matrix division operator \ (backslash):

```
(X'*X)\(X'*Y)
```

## 4. Creating an M-file "script"

A "script" is file that executes a series of Matlab command. Matlab scripts are saved in "M-files", i.e., files ending with the extension .m. Whenever you do data analysis in Matlab, you should *always* use a script. This enables you to track your analysis, make corrections and extensions as you go along, etc.

We will write a script file that calculates the OLS estimator for the simple model above.

To start a new M-file,

```
File > New > M-file
```

starts up the M-file editor.

If you want to add a comment in an M-file, put a % at the start of the line. Use comments frequently – it will help you to keep track of what you are doing.

Enter the following into your M-file

```
% Start with a blank workspace, then load data
clear
format compact
load M:\matlabwork\klein_data
% Create useful variables
nobs=length(Year)
one=ones(nobs,1)
Wpg=Wg+Wp
% OLS
Y=C
X=[one P Wpg]
XX=X'*X
XXinv=inv(XX)
XY=X'*Y
% Coefficient estimates
beta=XXinv*XY
```

Save the M-file with

```
File > Save As
```

and choose the name Klein_model.m.

To run a selected part of the M-file, highlight it, then right-click and choose "Evaluate selection". Try this by highlighting everything from the clear to the load command inclusive.

To run the entire M-file, click on the "Run" icon at the top (the little piece of paper with the arrow pointing down).

## 5. Writing a Matlab function

One of the most useful features of Matlab is that it allows users to write their own functions. These can include econometric estimators. We will write our own OLS estimator.

Our OLS estimator will have the following syntax and uses.  If the user types

```
ols(Y,X)
```

then the estimator regresses Y on X and displays the results.  If the user types

```
[b V]=ols(Y,X)
```

then, in addition to the regression results, the function will create a new variable b that is the vector of coefficients, and a new variable V that is the variance-covariance matrix.

The user will be able to use different names for all these variables, e.g.,

```
[b_kleinmodel V_kleinmodel]=ols(C, C_regressors)
```

will also be perfectly valid.

We will use the Matlab M-file editor to write the OLS function, though we could just as easily used a Windows text editor (say Notepad or Wordpad).  It will be an M-file called ols.m.  We will continue to use the Matlab M-file editor for loading data and estimating the Klein model, including calls to our OLS function.

In the Matlab M-editor, start a new M-file with

```
File > New > M-file
```

and start the OLS function with the following lines:

```
function [b,V]=ols(Y,X)
```

This tells Matlab that the OLS function will take two arguments, Y and X, and will return two results, b and V.

The next block of code creates some useful variables/matrices:

```
N=length(Y);
XX=X'*X;
XXinv=inv(XX);
XY=X'*Y;
% Projection and annihilation matrices
PX=X*XXinv*X';
MX=eye(N)-PX;
```

Note the “eye” function creates an identity matrix I of dimension NxN.

The next block of code calculates the coefficients:

```
% Coefficient estimates
beta=XXinv*XY;
```

To cacluate the variance of the OLS estimator, we will need the residuals.  There are several routes to obtaining them.

```
% Predicted values, two methods
Yhat=X*beta
Yhat2=PX*Y
% Residuals, two methods
ehat=Y-Yhat
ehat2=MX*Y
```

Note that the semicolons have been left off so that you can see for yourselves when the program runs that the two sets of predicted values, and the two sets of residuals, are the same. We will put the semicolons back in later.

We also need the matrix of the sample moments X and its inverse:

```
% Sample moments
QXXhat=1/N*XX;
QXXhatinv=inv(QXXhat);
```

The final ingredient needed for the variance of the OLS estimator is the estimator of the covariance matrix of moment conditions, which we follow Hansen by denoting Ω (Hayashi uses S). We will use the version that assumes serial independence (i.e., no autocorrelation) but is robust to the presence of heteroskedasticity of arbitrary form.

The covariance matrix of moment conditions can be calculated in a number of ways, and it is instructive to see each one. The first method is perhaps the easiest to understand. We create an Nx1 vector of squared residuals, and then convert this into a matrix B that has these squared residuals running down the diagonal. The estimator of Ω is simply the matrix product X'BX times 1/N. Again, leave off the semicolon so that when the program runs, you can see omegahat.

```
% Method 1
ehatsq=ehat.*ehat;
B=diag(ehatsq);
omegahat=1/N*X'*B*X
```

The second method creates the NxL matrix g, i.e., the matrix of sample orthogonality conditions. Each row is the 1xL vector $g_i$, defined as $x_ie_i$ where $e_i$ is the residual. This method is useful because it also introduces you to several Matlab functions. The method operates column by column on X. The first line takes the first column of X and multiplies every row in it by the corresponding row of ehat. The : in X(:,1) means “every row in X”, so X(:,1) is an Nx1 matrix which is the first column of X. The operator .* is “array multiplication”, which is defined for matrices that have the same size. Here, every element in X(:,1) is multiplied by the corresponding element of ehat. The result is an Nx1 variable g.

The next block of 3 lines is a for loop that executes for the 2nd column of X onwards. The same array multiplication is done for each of the columns of X, and the result is appended to the variable g. After the loop runs once, g is Nx2; after the next loop, it's Nx3; etc. The last step is to calculate omega hat.

```
% Method 2
g=X(:,1).*ehat;
for i=2:cols(X)
    g=[g   X(:,i).*ehat];
end
omegahat2=1/N*g'*g
```

The third method also uses a for loop, but here it is to loop through each observation. The first line creates an empty LxL matrix of zeros. The loop then goes through each observation, and for each observation calculates $x_i x_i' e_i^2$. It sums all N of these, and then the last line divides by N to get omegahat.

```
% Method 3
omegahat3=zeros(cols(X),cols(X));
for i=1:N
     omegahat3=omegahat3+ehat(i)^2*X(i,:)'*X(i,:);
end
omegahat3=1/N*omegahat3
```

The last step is to calculate the estimate of the covariance matrix of beta. Recall that the standard statement of the consistency of the OLS estimator is that it is "root-N" consistent, i.e., in Hansen's notation,

$$E(x_i x_i' e_i^2) = \Omega$$

$$\sqrt{n}(\hat{\beta} - \beta) \rightarrow_d N(0, V)$$

$$V = Q^{-1} \Omega Q^{-1}$$

where Q gives the population moments of X (and our estimate of which is QXXhat). In practice, however, we conduct inference with and test hypotheses about beta, not root-N * beta, and consequently we work with 1/N*our estimate of V rather than with just V.

```
% Covariance matrix of root-N estimator
cov_beta_rootN=QXXhatinv*omegahat*QXXhatinv;
cov_beta=1/N*cov_beta_rootN;
```

Naturally, our estimator should include standard errors and so forth in its output. For simplicity, we will not employ small-sample corrections, i.e., we will work with z-stats and the standard Normal distribution rather than with t-stats. The output will also include p-values.

The standard errors are stored in the variable se, which like beta is an Lx1 column vector, and is simply the square root of the diagonal of the covariance matrix. The z-stats are just the ratio of the estimates of the coefficients beta to the corresponding standard errrors, and are calculated using the "array division" operator ./ (compare to "array multiplication" .* above). To calculate the p-values, we use the cumulative normal function. This function is not part of the student version of Matlab (it's part of the "Statistics Toolbox" that has to be purchased separately), so you need to have downloaded the version we've prepared for this workshop. Note that when this function is provided with an argument that is an RxC matrix, it returns a matrix that is also RxC, i.e., it operates on all the scalars in the matrix.

```
% Standard errors, z-stats, p-values
se=sqrt(diag(cov_beta));
zstat=beta./se;
pvalue=2*(1-normcdf(abs(zstat)));
```

The next block of code prints the output in a nice format. The way this is done in Matlab is to create a matrix out of the coefficients, the SEs, the z-stats and the p-values, and use the mprint function to print the matrix. The code also shows how the variable info can have various other variables attached to it. We attach the four pieces of text that will be the column names in the part of info called info.cnames, and mprint knows to look at the cnames part of info for the column names. Other formatting information can be attached to the variable info, but we don't do this. All of this is preceded by the display of a couple of lines of basic information about the estimates.

```
disp('Ordinary least squares estimates');
disp('Standard errors are heteroskedastic-robust');
olsout=[beta se zstat pvalue];
info.cnames = strvcat('coeff','SE', 'z-stat', 'p-value');
mprint(olsout, info);
```

The last step is to assign values to the variables that are returned by our function. Recall that the first line of the function (see above) is [b V]=ols(Y,X). We could have this last step if the first line of the function were instead [beta cov_beta]=ols(Y,X).

```
% Return arguments
b=beta;
V=cov_beta;
```

Now let's use the estimator. In the Matlab Command Window, type

```
ols(Y,X);
```

You should see the 3 different calcuations of omegahat printed, and they should all be the same, i.e.,

```
omegahat =
  1.0e+003 *
    0.0017    0.0296    0.0715
    0.0296    0.5354    1.2965
    0.0715    1.2965    3.1754
```

and this should be followed by the OLS output.

```
Ordinary least squares estimates
Standard errors are heteroskedastic-robust
     coeff       SE     z-stat     p-value
   14.6427     2.4378    6.0065     0.0000
    0.2598     0.0499    5.2104     0.0000
    0.8381     0.0634   13.2139     0.0000
```

Now go back and add the ; after the calculation of the omegahats, so that only the OLS output will appear. Next, save the coefficient and variance estimates by typing in the Command Window

```
[Klein_beta Klein_V]=ols(Y,X);
```

The workspace now has two new variables, Klein_beta and Klein_V. You can display the contents by simply typing these names in the Command Window, or by right-clicking on the names in the Workspace window.

## 6. Estimating the Klein model using lagged variables

In fact, in the Klein model, consumption is a function of wages, current profits, and *lagged* profits. Here is how to implement this.

In our Klein_model.m M-file, add the following lines. Leave off the semicolons so you can see the results of the assignments.

```
% Klein model proper
% First create variables
Y=C
P_lag=[NaN; P(1:nobs-1)]
X=[one P P_lag Wpg]
```

The special value NaN stands for "Not a Number"; it is Matlab's code for missing value. The vector P_lag contains lagged values of P for every year where these exist, i.e., row 2 (1921) through 22 (1941). The entry for the first year of data (1920) is NaN. This means that the first row cannot be used for estimation purposes, so we have to remove the first row of all the relevant matrices. After that, we can estimate the model.

```
% Then trim the first row of all matrices because of the
missing value in P_lag
Y=Y(2:nobs,:)
X=X(2:nobs,:)
% Estimate model
[Klein_beta Klein_V]=ols(Y,X);
```

The results should be

```
Ordinary least squares estimates
Standard errors are heteroskedastic-robust
     coeff       SE      z-stat     p-value
   16.2366     1.6179    10.0354     0.0000
    0.1929     0.0608     3.1715     0.0015
    0.0899     0.0660     1.3616     0.1733
    0.7962     0.0513    15.5272     0.0000
```

## 7. Writing a 2-step feasible efficient GMM (FEGMM) estimator

The discussion in this section will be less detailed, now that you are already familiar with the basics. Open a new M-file and call it fegmm.m. The estimator will have the following syntax:

```
[b V omegahat]=fegmm(Y,X,Z)
```

or, optionally,

```
[b V omegahat]=fegmm(Y,X,Z,q)
```

where X are the regressors, Z the instruments, and q is an optional argument indicating the bandwidth if a HAC (heteroskedastic and autocorrelation-consistent) covariance estimator is requested.

The first block of code announces the estimator and calculates some basic variables.

```
function [beta, cov_beta , J, omegahat]=fegmm(Y,X,Z,q)
disp('2-step feasible efficient GMM estimates');
N=length(Y);
ZZ=Z'*Z;
XZ=X'*Z;
ZZinv=inv(ZZ);
ZY=Z'*Y;
% Projection and annihilation matrices
PZ=Z*ZZinv*Z';
MZ=eye(N)-PZ;
```

Next, calcuate the first-step inefficient but consistent GMM estimator of beta, which in this application will be the IV estimator. Also calculate the IV predicted values Yhat and IV residuals ehat.

```
% First-step ineff but consistent coeff estimates are IV
beta=inv(XZ*ZZinv*XZ')*XZ*ZZinv*ZY;
% Predicted values and residuals
Yhat=X*beta;
ehat=Y-Yhat;
```

Next, calculate the sample moments for Z and X.

```
% Sample moments
QZZhat=1/N*ZZ;
QZZhatinv=inv(QZZhat);
QZXhat=1/N*Z'*X;
```

We use the third method described above for calculating the estimate of the covariance matrix of moment conditions, because it will be easy to extend it to calculate autocovariances when we turn to the HAC case:

```
% Covariance matrix of moment conditions
omegahat=zeros(cols(Z),cols(Z));
for i=1:N
     omegahat=omegahat+ehat(i)^2*Z(i,:)'*Z(i,:);
end
omegahat=1/N*omegahat;
omegahatinv=inv(omegahat);
```

Using the inverse of the estimated covariance matrix of orthogonality conditions, calculate the second-step efficient and consistent GMM estimator, where “efficient” means “efficient in the presence of arbitrary/unknown heteroskedasticity”.

```
% Second-step efficient and consistent GMM estimator
beta=inv(XZ*omegahatinv*XZ')*XZ*omegahatinv*ZY;
```

The covariance matrix is calculated much the same way as before. Again, in practice we work with 1/N times the root-N variance estimator:

```
% Covariance matrix of efficient root-N GMM estimator
cov_beta_rootN=inv(QZXhat'*omegahatinv*QZXhat);
cov_beta=1/N*cov_beta_rootN;
```

Lastly, we must calculate the Sargan-Hansen, or J, statistic. This is both the value of the maximised GMM objective function, and a statistic that tests the validity of the orthogonality conditions, i.e., it is a test of the "overidentifying restrictions" (if the model is exactly identified, the value of J is zero and nothing can be tested). To calculate J, we need the sample orthogonality conditions (denoted g-bar in the lecture notes). These use the *new* residuals, i.e., the residuals from our 2-step feasible efficient GMM estimator. The sample orthogonality conditions can then be calculated in one Matlab line

```
    % Sample moment conditions
    % Need moment conditions from new FEGMM estimator
    % Predicted values
    Yhat=X*beta;
    % Residuals
    ehat=Y-Yhat;
    gbar=1/N*Z'*ehat;
```

and so

```
    % J statistic
    J=N*gbar'*omegahatinv*gbar;
```

The J statistic is distributed as chi-squared with degrees of freedom = degree of overidentification = number of instruments minus number of regressors, so

```
    % Degrees of freedom
    [r, L]=size(Z);
    [r, K]=size(X);
    Jdof=L-K;
    Jpvalue=1-chi2cdf(J,Jdof);
```

The code for standard errors, z-stats and p-values are the same as in the OLS estimator

```
    % Standard errors, zstats, pvalues
    se=sqrt(diag(cov_beta));
    zstat=beta./se;
    pvalue=2*(1-normcdf(abs(zstat)));
```

The final block of code prints out the results:

```
    fegmmout=[beta se zstat pvalue];
    info.cnames = strvcat('coeff', 'SE', 'z-stat', 'p-value');
    mprint(fegmmout, info);
    disp('Sargan-Hansen J statistic');
    disp(J);
    disp('J degrees of freedom');
    disp(Jdof);
    disp('J p-value');
    disp(Jpvalue);
```

Save the file, and test it by adding the following lines to the Klein_model.m M-file. This is the classic Klein consumption function. The explanatory variables are as before, but now wage income W and current profits P are treated as endogenous. The excluded instruments are a time trend (traditionally called "A", for some reason), government spending, taxes, capital stock, public sector wage income, and lagged GNP.

```
% Klein model with endogenous vars and excluded IVs
A=Year-1919
P_lag=[NaN; P(1:nobs-1)]
GNP_lag=[NaN; GNP(1:nobs-1)]
Y=C
X=[one P P_lag Wpg]
Z=[one P_lag G T A Wg K GNP_lag]
Y=Y(2:nobs)
X=X(2:nobs,:)
Z=Z(2:nobs,:)
% Estimate model
[Klein_beta Klein_V J omegahat]=fegmm(Y,X,Z);
```

The output should be:

```
2-step feasible efficient GMM estimates
Standard errors are heteroskedastic-robust
     coeff          SE      z-stat    p-value
   14.7443      1.1596     12.7149     0.0000
    0.0758      0.0936      0.8100     0.4179
    0.1663      0.0825      2.0159     0.0438
    0.8494      0.0356     23.8544     0.0000

Sargan-Hansen J statistic
    4.8358
J degrees of freedom
    4
J p-value
    0.3046
```

Current profits are insignificant but the other variables are significant at the 5% level. The J statistic has a p-value of 0.3046, suggesting that the model orthogonality conditions are valid.

## 8. Heteroskedastic and Autocorrelation-Consistent GMM estimation

We want our estimates to be robust to arbitrary autocorrelation as well as heteroskedasticity. We will use the Bartlett kernel (a.k.a. Newey-West), to guarantee a positive-definite covariance matrix of orthogonality conditions. Here is how to code it. Between the lines

```
omegahat=1/N*omegahat;
omegahatinv=inv(omegahat);
```

add the following code:

```
% If bandwidth supplied, calculate HAC covariance matrix
if nargin==4
% Start of HAC block
     disp('Standard errors are het. and autocorr.-robust');
% q is 4th argument and provides the bandwidth
     for j=1:q
% Autocovariance matrices
          Gq=zeros(rows(omegahat),cols(omegahat));
          for k=j+1:N
               Gq=Gq+ehat(k)*ehat(k-j)*Z(k,:)'*Z(k-j,:);
          end
          Gq=1/N*Gq;
          omegahat=omegahat+(1-j/q)*(Gq+Gq');
     end
     msg=['Bandwidth=' num2str(q)];
     disp(msg);
% End of HAC block
else
     disp('Standard errors are heteroskedastic-robust');
end
```

The line with nargin checks whether the bandwidth q was supplied as the 4th argument to fegmm. If it was, the HAC block of code is started, and the message that the standard errors are HAC is printed; if not, the message that the SEs are only heteroskedastic-robust is printed. Note the double equal == after nargin (the single = is the assigment operator).

A reminder of the lecture notes presentation: this is the notation used by Hayashi to denote the "truncated kernel" or Hansen-White HAC estimator. Hayashi's S is our Ω.

$$\hat{\Gamma}_j = \frac{1}{n}\sum_{t=j+1}^{n}\hat{g}_t\hat{g}'_{t-j} \quad j = 0,1,...,n-1$$

$$\hat{g}_t \equiv z_t\hat{\varepsilon}_t$$

$$\Gamma_j = 0 \text{ for } j > q, q \text{ known \& finite}$$

$$\hat{S} = \hat{\Gamma}_0 + \sum_{j=1}^{q}(\hat{\Gamma}_j + \hat{\Gamma}'_j)$$

The core of the HAC bock of code is the loop that calculates the autocovariance matrices. In the lecture notes these are denoted $\Gamma_1$ , $\Gamma_2$ , $\Gamma_3$ , …, $\Gamma_q$ . Recall that the jth population autocovariance matrix is $E(g_tg'_{t-j})$ and $E(g_tg'_t)$ is just omega, i.e., $\Gamma_0$, which we already estimated in our code and is available as omegahat. The loop starts with j=1 and runs through j=1, and each time through the loop the code calculates an estimated autocovariance matrix, calling it Gq.

The first step in calculating the jth autocovariance matrix is setting up a blank LxL matrix of zeros. The next step is another loop that starts at j+1 and runs through all the remaining observations, calculating $e_te_{t-j}z_tz'_{t-j}$. When this sum is complete, we divide it by 1/N to get our estimated autocovariance matrix, Gq. The last step is to add Gq and its transpose, Gg', to omegahat.

Note the weight (1-j/q) in front of (Gq+Gq'). If we omitted this, i.e., the weight was always one, then we would get the truncated kernel (Hansen-White) estimator. The weight (1-j/q) is the formula for the Bartlett kernel (Newey-West estimator). Important: say the bandwidth q is 3. Then the weights on the autocovariance matrices will be (1-1/3)=2/3, (1-2/3)=1/3, and (1-3/3)=0. A weight of zero means, of course, that the autocovariance matrix for the lag j=3 actually drops out of the calculations entirely. In other words, with the Bartlett kernel, the autocovariance matrix for a lag length of q, i.e., the last autocovariance matrix used in the formula, actually gets a weight of zero. This is true of some kernel estimators but not of others; the obvious counter-example is the truncated kernel estimator, where it gets a weight of one.

The last step is to inform the user that the length of the bandwidth is q. The code shows how to get it to print on one line: first, create a string the combines the text with the number, and then print the string. Note that the number has to be converted to a string before it is combined with the text.

We want to obtain GMM estimates of the Klein model that are efficient in the presence of arbitrary heteroskedasticity and autocorrelation. We need to choose a bandwidth. The Bartlett kernel requires the bandwidth to grow at the rate $N^{1/3}$, the sample size is 21, $21^{1/3}$=2.8, so we choose a bandwidth of 3. Add the following lines to your M-file and execute them:

```
% HAC estimation, bandwidth=3
[Klein_beta Klein_V J omegahat]=fegmm(Y,X,Z,3);
```

The result should be

```
2-step feasible efficient GMM estimates
Standard errors are het. and autocorr.-robust
Bandwidth=3
     coeff          SE     z-stat    p-value
   15.2448      1.0602    14.3787     0.0000
    0.0542      0.1282     0.4228     0.6724
    0.1800      0.1004     1.7930     0.0730
    0.8395      0.0396    21.1993     0.0000

Sargan-Hansen J statistic
    3.5582
J degrees of freedom
    4
J p-value
    0.4691
```

The main effect is to make the coefficient estimates of P and P_lag slightly less significant. The J statistic is still low enough (3.56, p-value=0.47) for us to conclude that the orthogonality conditions are valid.

## 9. An endogeneity test that is heteroskedastic and autocorrelation-robust

Can we treat the endogeneous variables Wpg and P as exogenous? We can test this using a "GMM distance" or "C" statistic. We add Wpg and P to the orthogonality conditions and test whether this additional subset of orthogonality conditions is valid. In this second GMM estimation, there are no endogenous variables, so it is an example of the "HOLS" estimator. The test statistic is just the difference in J statistics for the two GMM estimations. It is distributed as chi-squared with degrees of freedom equal to the number of additional orthogonality conditions being tested, here 2. A large value means the two extra orthogonality conditions are not valid.

Add the following lines to your M-file and execute them:

```
% HAC endogeneity test
Y2=C
X2=[one P P_lag Wpg]
Z2=[one P_lag G T A Wg K GNP_lag P Wpg]
Y2=Y2(2:nobs)
X2=X2(2:nobs,:)
Z2=Z2(2:nobs,:)
[Klein_beta2 Klein_V2 J2 omegahat2]=fegmm(Y2,X2,Z2,3);
```

The estimation results should be

```
2-step feasible efficient GMM estimates
Standard errors are het. and autocorr.-robust
Bandwidth=3
     coeff          SE      z-stat    p-value
   16.1427      0.5732    28.1645     0.0000
    0.2258      0.0358     6.3160     0.0000
    0.0647      0.0308     2.0958     0.0361
    0.7962      0.0196    40.5413     0.0000

Sargan-Hansen J statistic
    5.0460
J degrees of freedom
    6
J p-value
    0.5379
```

Now J has the value from the first GMM estimation (3.5582) and J2 has the value from the second GMM estimation (5.0460). Our test statistic is simply J2-J, i.e., we are using the simple form where a positive test statistic is not guaranteed; see the lecture notes for how to guarantee this (it takes some programming). Add the following lines to the M-file and execute them. Don't use semicolons.

```
C=J2-J
Cpvalue=1-chi2cdf(C,2)
```

The result is

```
C =
    1.4878
Cpvalue =
    0.4753
```

A p-value of 0.4753 suggests that the variables P and Wpg can be treated as exogenous, i.e., we can them to the orthogonality conditions. Note that this has important implications for the interpretation of the model, because the coefficient on P is now much bigger (0.2258) and very significant.

## 10. Interpreting the HOLS estimator

What does the J statistic for the HOLS estimator mean? It is a test of the orthogonality conditions of the model, but in this case there are no endogenous regressors. This means that it is simply a test of whether nor not the excluded instruments can be excluded from the model as regressors, i.e., it is a Lagrange multiplier test of the joint significant of these variables as regressors. Asymptotically, it is equivalent to including these instruments as regressors and testing their joint significance using a Wald test. The J statistic of 5.046 with a p-value of 0.5379 means that we can conclude that these variables do not belong as regressors in the model.

## 11. Obtaining OLS estimates using fegmm

It is very easy to obtain OLS estimates from our fegmm program: simply specify that the instruments and regressors are the same. Add the following lines to your M-file and execute them

```
% OLS again
[Klein_beta3 Klein_V3 J3 omegahat3]=fegmm(Y,X,X,3);
```

The results should be

```
2-step feasible efficient GMM estimates
Standard errors are het. and autocorr.-robust
Bandwidth=3
     coeff         SE     z-stat    p-value
   16.2366     1.5277    10.6283     0.0000
    0.1929     0.0734     2.6280     0.0086
    0.0899     0.0644     1.3961     0.1627
    0.7962     0.0510    15.6189     0.0000

Sargan-Hansen J statistic
  2.3838e-020
J degrees of freedom
    0
J p-value
    0
```

The coefficients are the same OLS coefficients as before, but the standard errors are different, because they are now heteroskedastic AND autocorrelation-robust.

Note the Sargan-Hansen statistic is essentially zero. This is because in an exactly-identified model, the maximised value of the GMM objective function is zero (the GMM coefficients can be chosen so that sample moment conditions equal exactly what the theoretical moment conditions predict, namely zero).

## 12. The “continuously updated GMM estimator”, CUE

In this section, we explore an alternative to 2-step feasible efficient GMM, namely CUE.

Recall that the two steps of 2-step FEGMM are (1) obtain a consistent estimate of the covariance matrix of orthogonality conditions Ω, and (2) use the inverse of our estimate of Ω as the weighting matrix W in the GMM objective function,

$$\hat{\beta}_{GMM} = \underset{\beta}{\operatorname{argmin}}\, J_n(\beta) = \underset{\beta}{\operatorname{argmin}}\left[n\bar{g}_n(\beta)' W_n \bar{g}_n(\beta)\right]$$

Note that in the minimization behind 2-step FEGMM, the weighting matrix $W_n$ is treated as a constant.

In the general HAC case (see the lecture notes) we obtain an estimate of Ω as follows:

$$\hat{\Gamma}_j = \frac{1}{n}\sum_{t=j+1}^{n} \hat{g}_t \hat{g}'_{t-j} \quad j = 0,1,...,n-1$$

$$\hat{g}_t \equiv x_t \hat{e}_t$$

$$\hat{e}_t = y_t - x'_t\hat{\beta}$$

$\hat{\beta}$ consistent for $\beta$

i.e., from any consistent first-step estimator.

$\Gamma_j = 0$ for $j > q$, q known & finite

$$\hat{\Omega} = \hat{\Gamma}_0 + \sum_{j=1}^{q}(\hat{\Gamma}_j + \hat{\Gamma}'_j)$$

$$= \sum_{j=-q}^{q} \hat{\Gamma}_j$$

The heteroskedastic case with no autocorrelation is just a special case of the above: $\hat{\Omega} = \hat{\Gamma}_0$. After we have an estimate of Ω, we invert it to obtain our weighting matrix W.

In 2-step FEGMM, we are therefore working with two different estimates of β: one which is used in the first step and must be consistent (but might be inefficient), and one which is the result of the second step, namely the 2-step FEGMM estimator itself.

In the CUE estimator, we have only one estimator, the CUE estimator itself. It is used to construct the weighting matrix W, and it is the result of the minimization. We can make this explicit if we rewrite the CUE GMM objective function as:

$$\hat{\beta}_{CUE\text{-}GMM} = \underset{\beta}{\operatorname{argmin}}\, J_n(\beta) = \underset{\beta}{\operatorname{argmin}}\left[n\bar{g}_n(\beta)'\Omega(\beta)^{-1}\bar{g}_n(\beta)\right]$$

so that the weighting matrix is now explicitly a function of the same β that we are minimizing over.

The CUE and the FEGMM estimators are asymptotically equivalent – both are asymptotically efficient – but there is some recent evidence that CUE has better finite sample performance, particularly when instruments are "weak", i.e., not strongly correlated with the endogenous regressors. In the special case of conditional homoskedasticity, the CUE reduces to the limited information maximum likelihood or LIML estimator.

The CUE normally requires numerical methods to calculate, i.e., unlike 2-step FEGMM applied to a linear model, there is no simple closed-form solution. We will use this as a gentle introduction to numerical maximization methods in Matlab.

To implement this, we will need to write two Matlab functions: an objective function that returns the value of J, the GMM objective function, for some given values of the parameter vector β, and "parent" function that minimizes the value of the objective function, i.e., that minimizes J (plus does some other housekeeping).

We will call the objective function "gmmobjfn". Here is the code; save it as "gmmobjfn.m".

```
function J=gmmobjfn(beta,Y,X,Z,q)
% make these accessible to both functions
global omegahat omegahatinv QZXhat
N=length(Y);
% Residuals
ehat=Y-X*beta;
% Sample moments
QZZhat=1/N*Z'*Z;
QZZhatinv=inv(QZZhat);
QZXhat=1/N*Z'*X;
gbar=1/N*Z'*ehat;
% Covariance matrix of moment conditions
omegahat=zeros(cols(Z),cols(Z));
for i=1:N
     omegahat=omegahat+ehat(i)^2*Z(i,:)'*Z(i,:);
end
omegahat=1/N*omegahat;
% If bandwidth supplied, calculate HAC covariance matrix
if nargin==5
% Start of HAC block
% q is 5th argument and provides the bandwidth
     for j=1:q
% Autocovariance matrices
          Gq=zeros(rows(omegahat),cols(omegahat));
          for k=j+1:N
               Gq=Gq+ehat(k)*ehat(k-j)*Z(k,:)'*Z(k-j,:);
          end
          Gq=1/N*Gq;
          omegahat=omegahat+(1-j/q)*(Gq+Gq');
     end
% End of HAC block
end
omegahatinv=inv(omegahat);
% Sargan-Hansen J statistic
J=N*gbar'*omegahatinv*gbar;
```

Several comments:

- The function returns a scalar, J, and takes as its arguments a parameter vector beta, the data matrices, and an optional bandwidth argument if HAC is desired. The beta can have any values.
- Some of the matrices calculated by this function can be used by the parent function. It makes life easier if the parent function can access them. This is achieved by the "global" statement at the top.
- Everything else is straightforward and corresponds to portions of our FEGMM estimator: first calculate the residuals based on the supplied beta, then the sample moments, calculate the estimate of omega, invert it and finally calculate J.

The parent function that calls the objective function is called "cuegmm". It will have the same syntax as our "fegmm" estimator. Save the code in the file "cuegmm.m".

```
function [beta,cov_beta,J,omegahat]=cuegmm(Y,X,Z,q)
% These matrices are calculated in the GMM obj function.
% Make them accessible to both functions.
global omegahat omegahatinv QZXhat
disp('Starting values for CUE estimator...');
if nargin==4
    [beta_0 V_0 J_0 omegahat_0]=fegmm(Y,X,Z,q);
    [beta, J] = fminsearch(@gmmobjfn, beta_0, [], Y, X, Z, q);
else
    [beta_0 V_0 J_0 omegahat_0]=fegmm(Y,X,Z);
    [beta, J] = fminsearch(@gmmobjfn, beta_0, [], Y, X, Z);
end
% Add a blank line (don't forget the blank space)
disp(' ')
N=length(Y);
% Covariance matrix of efficient root-N GMM estimator
cov_beta_rootN=inv(QZXhat'*omegahatinv*QZXhat);
cov_beta=1/N*cov_beta_rootN;
% Degrees of freedom
[r, K]=size(X);
[r, L]=size(Z);
Jdof=L-K;
Jpvalue=1-chi2cdf(J,Jdof);
% Standard errors, zstats, pvalues
se=sqrt(diag(cov_beta));
zstat=beta./se;
pvalue=2*(1-normcdf(abs(zstat)));
disp('CUE GMM estimates');
% Print appropriate message about robustness
if nargin==4
     disp('Standard errors are het. and autocorr.-robust');
     msg=['Bandwidth=' num2str(q)];
     disp(msg);
%else
     disp('Standard errors are heteroskedastic-robust');
end
cuegmmout=[beta se zstat pvalue];
info.cnames = strvcat('coeff', 'SE', 'z-stat', 'p-value');
mprint(cuegmmout, info);
disp('Sargan-Hansen J statistic');
disp(J);
disp('J degrees of freedom');
disp(Jdof);
disp('J p-value');
disp(Jpvalue);
```

The global command at the top ensures that cuegmm can access the matrices that gmmobjfn designated global.

The first substantial action of the estimator is to calculate *starting values* for beta. Numerical optimization works by searching, and starting at a good place makes the search go much more smoothly. We start our search at the 2-step FEGMM parameter estimates, and we obtain these simply by calling our FEGMM estimator. Note that this will mean that the FEGMM estimates get output to the screen first.

The next and most important step is the minimization. We use the built-in general purpose Matlab minimizer "fminsearch". This takes the following arguments: (1) the function to minimize, preceded by "@"; (2) the starting values (which we saved as beta_0 from the call to FEGMM); (3) options – which we don't use, so we supply an empty list of options, "[]"; (4) the additional parameters required by the function we are minimizing, namely gmmobjfn – it needs the data matrices plus the optional bandwidth argument.

When "fminsearch" is done, it will save the values of the parameters that minimize the objective function in "beta", and the value of the minimized objective function in "J". This beta is the CUE beta, and now that we have it and J, the rest of the program is straightforward – it just calculates the covariance matrix, degrees of freedom, p-values, etc., just like the FEGMM estimator.

Add the following lines to your M-file and execute them:

```
% CUE estimator, bandwidth=3
[CUE_beta CUE_V J omegahat]=cuegmm(Y,X,Z,3);
```

and the results should be

```
Starting values for CUE estimator...
2-step feasible efficient GMM estimates
Standard errors are het. and autocorr.-robust
Bandwidth=3
     coeff         SE     z-stat    p-value
   15.2448     1.0602    14.3787     0.0000
    0.0542     0.1282     0.4228     0.6724
    0.1800     0.1004     1.7930     0.0730
    0.8395     0.0396    21.1993     0.0000

Sargan-Hansen J statistic
    3.5582
J degrees of freedom
     4
J p-value
    0.4691

CUE GMM estimates
Standard errors are het. and autocorr.-robust
Bandwidth=3
Standard errors are heteroskedastic-robust
     coeff         SE     z-stat    p-value
   14.0065     0.6025    23.2478     0.0000
    0.1010     0.0347     2.9073     0.0036
    0.0707     0.0425     1.6633     0.0963
    0.8920     0.0184    48.5450     0.0000

Sargan-Hansen J statistic
    3.1618
J degrees of freedom
     4
J p-value
    0.5311
```

**Appendix: m-files**

**ols.m**

```
function [b,V]=ols(Y,X)
N=length(Y);
XX=X'*X;
XXinv=inv(XX);
XY=X'*Y;
% Projection and annihilation matrices
PX=X*XXinv*X';
MX=eye(N)-PX;
% Coefficient estimates
beta=XXinv*XY;
% Predicted values, two methods
Yhat=X*beta;
Yhat2=PX*Y;
% Residuals, two methods
ehat=Y-Yhat;
ehat2=MX*Y;
% Sample moments
QXXhat=1/N*XX;
QXXhatinv=inv(QXXhat);
% Covariance matrix of moment conditions, several methods
% Method 1
ehatsq=ehat.*ehat;
B=diag(ehatsq);
omegahat=1/N*X'*B*X;
% Method 2
g=X(:,1).*ehat;
for i=2:cols(X)
    g=[g X(:,i).*ehat];
end
omegahat2=1/N*g'*g;
% Method 3
omegahat3=zeros(cols(X),cols(X));
for i=1:N
      omegahat3=omegahat3+ehat(i)^2*X(i,:)'*X(i,:);
end
omegahat3=1/N*omegahat3;
% Covariance matrix of efficient root-N GMM estimator
cov_beta_rootN=QXXhatinv*omegahat*QXXhatinv;
cov_beta=1/N*cov_beta_rootN;
% Standard errors, z-stats, p-values
se=sqrt(diag(cov_beta));
zstat=beta./se;
pvalue=2*(1-normcdf(abs(zstat)));
disp('Ordinary least squares estimates');
disp('Standard errors are heteroskedastic-robust');
olsout=[beta se zstat pvalue];
info.cnames = strvcat('coeff','SE', 'z-stat', 'p-value');
mprint(olsout, info);
% Return arguments
b=beta;
V=cov_beta;
```

**fegmm.m**

```
function [beta,cov_beta,J,omegahat]=fegmm(Y,X,Z,q)
disp('2-step feasible efficient GMM estimates');
N=length(Y);
ZZ=Z'*Z;
XZ=X'*Z;
ZZinv=inv(ZZ);
ZY=Z'*Y;
% Projection and annihilation matrices
PZ=Z*ZZinv*Z';
MZ=eye(N)-PZ;
% First-step ineff but consistent coeff estimates are IV
beta=inv(XZ*ZZinv*XZ')*XZ*ZZinv*ZY;
% Predicted values and residuals
Yhat=X*beta;
ehat=Y-Yhat;
% Sample moments
QZZhat=1/N*ZZ;
QZZhatinv=inv(QZZhat);
QZXhat=1/N*Z'*X;
% Covariance matrix of moment conditions
omegahat=zeros(cols(Z),cols(Z));
for i=1:N
     omegahat=omegahat+ehat(i)^2*Z(i,:)'*Z(i,:);
end
omegahat=1/N*omegahat;
% If bandwidth supplied, calculate HAC covariance matrix
if nargin==4
% Start of HAC block
     disp('Standard errors are het. and autocorr.-robust');
% q is 4th argument and provides the bandwidth
     for j=1:q
% Autocovariance matrices
          Gq=zeros(rows(omegahat),cols(omegahat));
          for k=j+1:N
               Gq=Gq+ehat(k)*ehat(k-j)*Z(k,:)'*Z(k-j,:);
          end
          Gq=1/N*Gq;
          omegahat=omegahat+(1-j/q)*(Gq+Gq');
     end
     msg=['Bandwidth=' num2str(q)];
     disp(msg);
% End of HAC block
else
     disp('Standard errors are heteroskedastic-robust');
end
omegahatinv=inv(omegahat);
% Second-step efficient and consistent GMM estimator
beta=inv(XZ*omegahatinv*XZ')*XZ*omegahatinv*ZY;
% Covariance matrix of efficient root-N GMM estimator
cov_beta_rootN=inv(QZXhat'*omegahatinv*QZXhat);
cov_beta=1/N*cov_beta_rootN;
% Sargan-Hansen J statistic
% Need moment conditions from new FEGMM estimator for this
% Predicted values
```

```
Yhat=X*beta;
% Residuals
ehat=Y-Yhat;
% Sample moment conditions
gbar=1/N*Z'*ehat;
% J statistic
J=N*gbar'*omegahatinv*gbar;
% Degrees of freedom
[r, K]=size(X);
[r, L]=size(Z);
Jdof=L-K;
Jpvalue=1-chi2cdf(J,Jdof);
% Standard errors, zstats, pvalues
se=sqrt(diag(cov_beta));
zstat=beta./se;
pvalue=2*(1-normcdf(abs(zstat)));
fegmmout=[beta se zstat pvalue];
info.cnames = strvcat('coeff', 'SE', 'z-stat', 'p-value');
mprint(fegmmout, info);
disp('Sargan-Hansen J statistic');
disp(J);
disp('J degrees of freedom');
disp(Jdof);
disp('J p-value');
disp(Jpvalue);
```

**gmmobjfn.m**

```
function J=gmmobjfn(beta,Y,X,Z,q)
% make these accessible to both functions
global omegahat omegahatinv QZXhat
N=length(Y);
% Residuals
ehat=Y-X*beta;
% Sample moments
QZZhat=1/N*Z'*Z;
QZZhatinv=inv(QZZhat);
QZXhat=1/N*Z'*X;
gbar=1/N*Z'*ehat;
% Covariance matrix of moment conditions
omegahat=zeros(cols(Z),cols(Z));
for i=1:N
      omegahat=omegahat+ehat(i)^2*Z(i,:)'*Z(i,:);
end
omegahat=1/N*omegahat;
% If bandwidth supplied, calculate HAC covariance matrix
if nargin==5
% Start of HAC block
% q is 5th argument and provides the bandwidth
      for j=1:q
% Autocovariance matrices
            Gq=zeros(rows(omegahat),cols(omegahat));
            for k=j+1:N
                  Gq=Gq+ehat(k)*ehat(k-j)*Z(k,:)'*Z(k-j,:);
            end
            Gq=1/N*Gq;
            omegahat=omegahat+(1-j/q)*(Gq+Gq');
      end
% End of HAC block
end
omegahatinv=inv(omegahat);
% Sargan-Hansen J statistic
J=N*gbar'*omegahatinv*gbar;
```

**cuegmm.m**

```
function [beta,cov_beta,J,omegahat]=cuegmm(Y,X,Z,q)
% These matrices are calculated in the GMM obj function.
% Make them accessible to both functions.
global omegahat omegahatinv QZXhat
% Call optimization routine after obtaining start values
% Starting values are FEGMM estimates
% first argument points to GMM objective function, to be
minimized
% second argument is starting values for minimization
% third argument is options (not used)
% remaining arguments are needed by GMM obj function to get J
% Returns values of parameters that minimize J, and J
disp('Starting values for CUE estimator...');
if nargin==4
    [beta_0 V_0 J_0 omegahat_0]=fegmm(Y,X,Z,q);
    [beta, J] = fminsearch(@gmmobjfn, beta_0, [], Y, X, Z, q);
else
    [beta_0 V_0 J_0 omegahat_0]=fegmm(Y,X,Z);
    [beta, J] = fminsearch(@gmmobjfn, beta_0, [], Y, X, Z);
end
% Add a blank line (don't forget the blank space)
disp(' ')
N=length(Y);
% Covariance matrix of efficient root-N GMM estimator
cov_beta_rootN=inv(QZXhat'*omegahatinv*QZXhat);
cov_beta=1/N*cov_beta_rootN;
% Degrees of freedom
[r, K]=size(X);
[r, L]=size(Z);
Jdof=L-K;
Jpvalue=1-chi2cdf(J,Jdof);
% Standard errors, zstats, pvalues
se=sqrt(diag(cov_beta));
zstat=beta./se;
pvalue=2*(1-normcdf(abs(zstat)));
disp('CUE GMM estimates');
% Print appropriate message about robustness
if nargin==4
      disp('Standard errors are het. and autocorr.-robust');
      msg=['Bandwidth=' num2str(q)];
      disp(msg);
%else
      disp('Standard errors are heteroskedastic-robust');
end
cuegmmout=[beta se zstat pvalue];
info.cnames = strvcat('coeff', 'SE', 'z-stat', 'p-value');
mprint(cuegmmout, info);
disp('Sargan-Hansen J statistic');
disp(J);
disp('J degrees of freedom');
disp(Jdof);
disp('J p-value');
disp(Jpvalue);
```

**Klein_model.m**

```
% Start with a blank workspace, then load data
clear
format compact
load M:\Matlabwork\klein_data
% Create useful variables
nobs=length(Year)
one=ones(nobs,1)
Wpg=Wp+Wg
% OLS
Y=C
X=[one P Wpg]
XX=X'*X
XXinv=inv(XX)
XY=X'*Y
% Coefficient estimates
beta=XXinv*XY

% Use our estimator
ols(Y,X)
ols(Y,X);

% Klein model proper
% First create variables
Y=C
P_lag=[NaN; P(1:nobs-1)]
X=[one P P_lag Wpg]
% Then trim the first row of all matrices
% because of the missing value in P_lag
Y=Y(2:nobs,:)
X=X(2:nobs,:)
% Estimate model
[Klein_beta Klein_V]=ols(Y,X);

% Klein model with endogenous vars and excluded IVs
A=Year-1919
P_lag=[NaN; P(1:nobs-1)]
GNP_lag=[NaN; GNP(1:nobs-1)]
Y=C
X=[one P P_lag Wpg]
Z=[one P_lag G T A Wg K GNP_lag]
Y=Y(2:nobs)
X=X(2:nobs,:)
Z=Z(2:nobs,:)
% Estimate model
[Klein_beta Klein_V J omegahat]=fegmm(Y,X,Z);
% HAC estimation, bandwidth=3
[Klein_beta Klein_V J omegahat]=fegmm(Y,X,Z,3);

% HAC endogeneity test
Y2=C
X2=[one P P_lag Wpg]
Z2=[one P_lag G T A Wg K GNP_lag P Wpg]
Y2=Y2(2:nobs)
X2=X2(2:nobs,:)
```

```
Z2=Z2(2:nobs,:)
[Klein_beta2 Klein_V2 J2 omegahat2]=fegmm(Y2,X2,Z2,3);
C=J2-J
Cpvalue=1-chi2cdf(C,2)

% OLS again
[Klein_beta3 Klein_V3 J3 omegahat3]=fegmm(Y,X,X,3);

% CUE estimator, bandwidth=3
[CUE_beta CUE_V J omegahat]=cuegmm(Y,X,Z,3);
```