Matlab basics for the course: 'Applied Bayesian Econometrics'

Haroon Mumtaz

April 18, 2011

1 Introduction

This note provides a basic introduction to Matlab and introduces the key concept needed in dealing with the codes used in the course. Note that a number of alternative guides to matlab are available on the web and these may be used to supplement the material here. For example see

http://www.economics.ox.ac.uk/members/martin.ellison/Boe/dsge_day1_2.ppt.

All the code used in the examples above is included in the folder called code.

2 Getting started

Figure 1 shows a basic screen shot of Matlab. There are two main windows: (1) the editor window which is docked on top and the (2) command window which is docked at the bottom. The editor is where we type our code. Once the code is typed it can be saved as a file which has the extension .m. The code is run by clicking on the green run button or by simply typing in the name of the program file in the command window. The command window is where the output from running the code appears. Or alternatively, each line of the code can be run by typing it in the command window and pressing enter.

In figure 2 we show how to create a generic first program called helloworld.m which prints out the words Hello World. The code simply consists of the line 'Hello World' where the single quotes signify that this a string variable as opposed to a numeric variable (or a number). By clicking on run, the output appears in the command window. Alternatively one can run the line of the program containing the code by highlighting the line and pressing F9.

3 Matrix programming language

The key data type in Matlab is a matrix and Matlab supports numerous Matrix operations (multiplication, transposes etc).

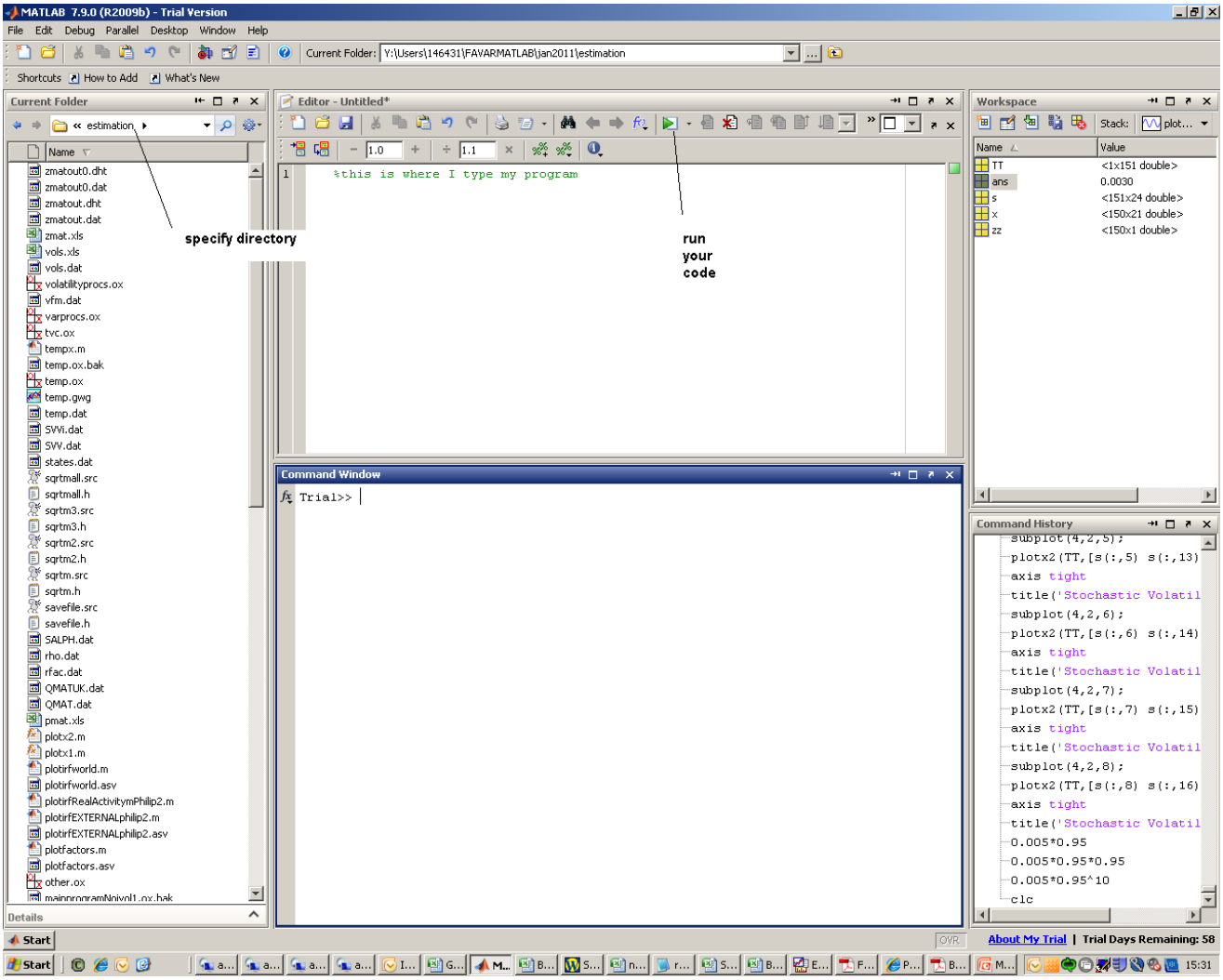

Figure 1: Matlab command window

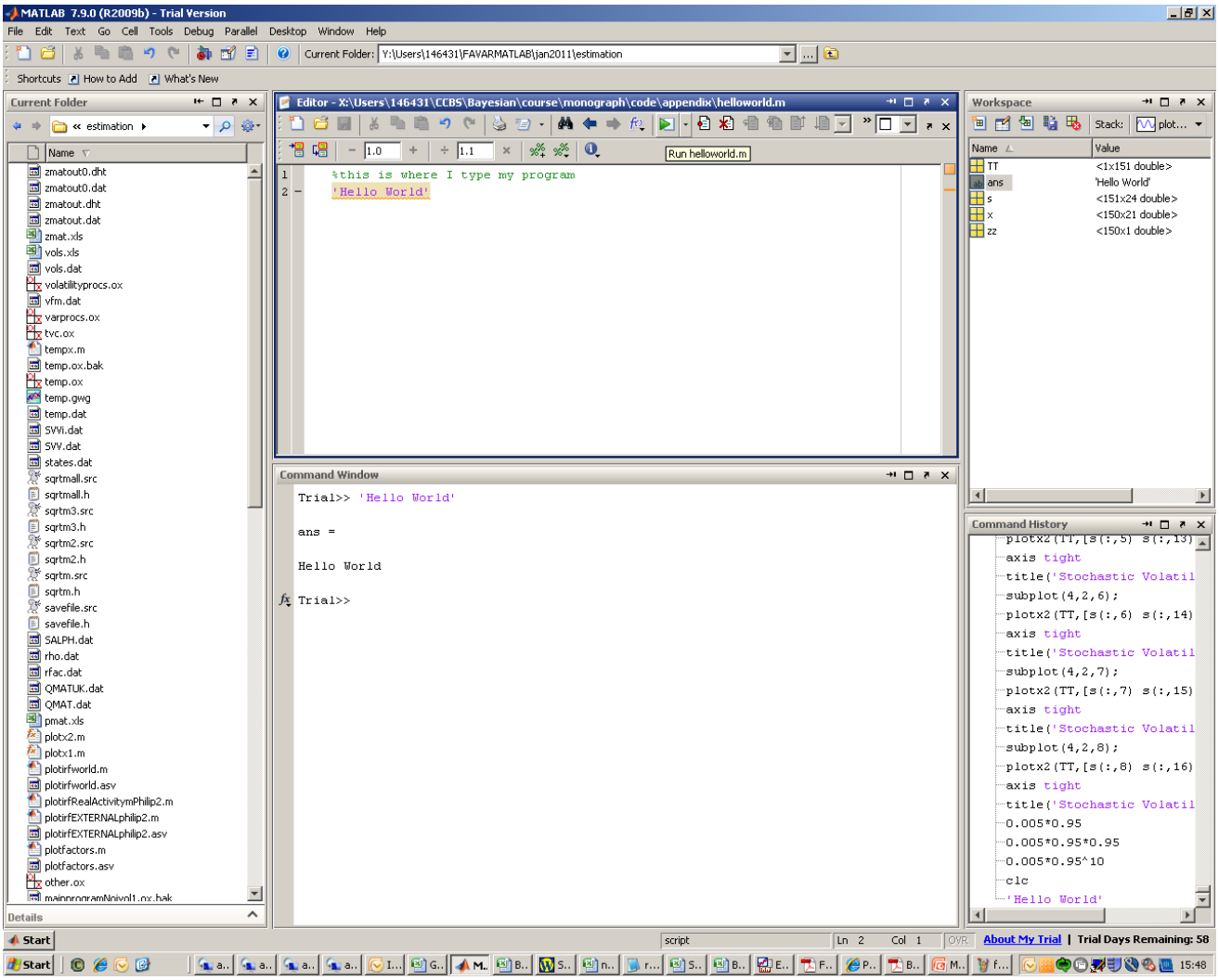

Figure 2: Hello World

```
1 %entering a matrix manually
2 X=[1 2 3 4;5 6 7 8];
3 Z=[2 3 4 5;7 8 9 10];
4 %print the matrices to screen
5 X
6 Z
```

Figure 3: Example 1

3.1 Creating matrices

3.1.1 Entering a matrix manually

Suppose we want to create the following two matrices

$$X = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

$$Z = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 7 & 8 & 9 & 10 \end{pmatrix}$$

Figure 3 shows the matlab code required to do this (example1.m). Note that the numbers in the first column are line numbers which are shown here for convenience and will not appear in the actual code. Note also that any code starting with % is a comment and is ignored by matlab. Line 2 shows how X is created.[1] The numbers have square brackets around them. Each column of the matrix is seperated by a space and rows by a semi-colon. Lines 2 and 3 finish with a semi-colon. This stops Matlab from priniting out X and Z when the code is run. Instead we print the matrices by typing X and Z without a semi-colon on lines 5 and 6.

3.1.2 Entering a matrix using built in commands

Line 2 in figure 4 creates a 10×10 matrix of zeros (the file is example2.m). The first argument in the function zeros denotes the number of rows, the second denotes the number of columns required. Line 4 creates a 10×20 matrix of ones. Line 6 creates a 10×10 identity matrix. Line 8 creates a 10×10 matrix where each element is drawn from a N(0,1) distribution. Similarly, line 9 creates a matrix from the standard uniform distribution.

[1] Matlab is case sensitive. Therefore X is treated as a different variable than x.

```
%create a matrix of zeros
X=zeros(10,10);
%create a matrix of ones
Y=ones(10,20);
%create an identity matrix
I=eye(10);
%create a 10 by 10 matrix from N(0,1)
N=randn(10,10);
%create a 10 by 10 matrix from U(0,1)
U=rand(10,10);
```

Figure 4: Example 2

```
%read in data from an excel file
[data,names]=xlsread('\data\data.xls');
```

Figure 5: Example 3

```
1 %entering a matrix manually
2 X=[1 2 3 4;5 6 7 8];
3 Z=[2 3 4 5;7 8 9 10];
4 % set X(2,1)=30
5 X(2,1)=30;
6 % vertical concatenation
7 M=[X;Z];
8 %horizontal concatenation
9 N=[X Z];
10 %set the second row of N to -10
11 N(2,1:end)=-10;
```

Figure 6: Example 4

3.1.3 Reading in data from an excel file

In practice we will read in matrices (i.e. data) from an outside source. As an example we have stored data on UK GDP growth and inflation in an excel file called data.xls in the folder called data. Figure 5 shows the matlab code used to read the excel file. The command xlsread reads the data into a matrix called data. The variable names are read into a string variable called names. Note that text files can be read by using the command load. Suppose one wants to read in data from a text file dalled data.txt, one simply needs to type load data.txt. Type 'help load' in the command window for further information.

3.2 Manipulating matrices

Lines 2 and 3 in figure 6 create two matrices X and Z shown in example 1 above. Line 5 sets the element (2,1), i.e. the element on the second row first column to 30. Line 7 creates a new 4×4 matrix by vertically concatenating X and Z. This done by the command M=[X;Z] where the semi-colon denotes vertical concatenation. That is, it creates

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 30 & 6 & 7 & 8 \\ 2 & 3 & 4 & 5 \\ 7 & 8 & 9 & 10 \end{pmatrix}$$

Line 9 of the code creates a new 2×8 matrix N by horizontally concatenating X and Z. This done by the command N=[X Z] where the space denotes horizontal concatenation. Finally Line 11 of

the code shows how to set the entire second row of N equal to -10. Note that argument 1:end in N(2,1:end)=-10 selects columns 1 to 10. One can delete elements by setting them equal to [ ]. For example N(2,:)=[ ] deletes the entire second row.

3.3 Matrix Algebra

Matlab supports numerous matrix functions. We demonstrate some of these by writing code to estimate an OLS regression using data in the file data.xls used in example 3 above (example5.m). Recall the formulas for an OLS regression $Y = XB + E$ are

$$\begin{aligned} \hat{B} &= (X'X)^{-1}(X'Y) \\ VAR(E) &= S = \frac{(Y - XB)'(Y - XB)}{T - K} \\ VAR\left(\hat{B}\right) &= S \times (X'X)^{-1} \end{aligned}$$

Where T denotes the number of observations and K are the number of regressors. We assume Y is the first column of data.xls and X is the second column of data.xls and a constant term. Line 3 of the code shown in Figure 7 loads the data. Line 5 shows the function size to find the number of rows in the data. Line 7 assigns the first column of data to a $T \times 1$ matrix called Y. Line 9 creates the $T \times 2$ X matrix where the first column is the constant and the second column is the second column of data.

Line 11 calculates the OLS esimate of the coefficients using the formula $\hat{B} = (X'X)^{-1}(X'Y)$. This line shows three matrix functions. First the the transpose of X in matlab is simply X'. One can multiply conformable matrices by using the * key. The inverse of matrix is calculated in matlab using the inv function. Line 13 calculates the residuals while Line 15 calculates $VAR(E)$ and line 17 calculates $VAR\left(\hat{B}\right) = S \times (X'X)^{-1}$. Note that S is a scalar and $(X'X)^{-1}$ is a matrix. In general a scalar can be multiplied by each element of matrix by the '**.***' key which denotes element by element multiplication. So this can also be used for element by element multiplication of two matrices. Finally, the standard errors of the coefficients are given as the square root of the diagonal of the matrix $S \times (X'X)^{-1}$. Line 19 uses the diag command to extract the diagonal and then takes the square root by raising them to the power 0.5. This is done by the command .^ which raises each element of a matrix to a specified power.

Other useful matrix functions include the Cholesky decomposition (command chol, type help chol in the command window for details). A list of matrix functions can be seen by typing help matfun in the command window.

4 Program control

4.1 For Loops

One of the main uses of programming is to use the code to repeat tasks. This is used intensively in Bayesian simulation methods. The main type of loop we use is the *For loop*. To take a trivial example suppose we need to create a 100×1 matrix called Z where each element is equal to row number raised to power 2. So Z(1,1)=1, Z(2,1)=2^2, Z(3,1)=3^2 and so on. The code is shown in figure 8. Line 2 sets the total number of rows in the matrix Z. Line 3 creates the empty matrix Z.

```
clear %clears all variables in memory
%read in data from an excel file
[data,names]=xlsread('\data\data.xls');
%dimensions of the data
T=size(data,1); %number of rows in the data
%assign data
Y=data(:,1); %Y is the first column of data
%X matrix with a constant
X=[ones(T,1) data(:,2)];
%OLS coefficients
B=inv(X'*X)*(X'*Y);
%Residuals
E=Y-X*B;
%Variance of Error term
S=(E'*E)/(T-2);
%Covariance of B
V=S.*inv(X'*X);
%standard errors
SE=diag(V).^0.5;
```

Figure 7: Example 5

```
clear
REPS=100; % number of repetitions
Z=zeros(REPS,1);
for i=1:REPS
    Z(i,1)=i^2;
end
```

Figure 8: Example 6

```
clear
T=1000; %Simulate for a 1000 periods
Y=zeros(T,1);
V=randn(T,1);
RHO=0.99;
for i=2:T
Y(i)=Y(i-1)*RHO+V(i,1);
end
plot(Y);
```

Figure 9: Example 7

```
1 clear
2 T=1000; %Simulate for a 1000 periods
3 Y=zeros(T,1);
4 V=randn(T,1);
5 RHO=0.99;
6 for i=2:T
7 temp=Y(i-1)*RHO+V(i,1);
8 if temp>=0
9     Y(i)=temp;
10 end
11 end
12 plot(Y);
```

Figure 10: Example 8

Line 4 begins the for loop and instructs matlab to repeat the instruction on line 5 REPS times. That is the loop starts with i=1 and repeats the instruction below (instructions on lines before the end statement) until i=REPS. It increases i by 1 in each iteration. On line 5 the ith row of Z is set equal to i squared. Therefore when i=1, Z(1,1)=1, when i=2, Z(2,1)=2^2, when i=3, Z(3,3)=3^2 and so on. The instruction end on line 6 closes the for loop. Note if we had typed on line 4 *for i=1:-1:REPS*, i would be decreased by 1 in each iteration. If we had typed on line 4 *for i=1:2:REPS*, i would be increased by 2 in each iteration.

Figure 9 shows a second example where we simulate an $AR(1)$ process for 1000 periods $Y_t = \rho Y_{t-1} + V_t, t = 1...1000$. Where $V_t \tilde{} N(0,1)$. Line 3 of the code creates a $T \times 1$ matrix of zeros Y. Line 4 draws the error term from the standard normal distribution. Line 5 sets the AR coefficient $\rho = 0.99$. Line 6 starts the loop from period 2 going up to period T=1000. Line 7 simulates each value of $Y_t, t = 1....1000$ and line 8 ends the for loop. Line 9 plots the simulated series.

4.2 Conditional statements

Conditional statements instruct Matlab to carry out commands if a condition is met. Suppose, in example 7 above, we want to simulate the AR model but only want to retain values for Y which are greater than or equal to zero. We can do this by using the if statement in matlab. Figure 10 shows the matlab code. Lines 1 to 5 are exactly as before. Line 6 begins the for loop. Line 7 sets a variable $temp = \rho Y_{t-1} + V_t$. Line 8 begins the if statement and instructs matlab to carry out the

```
1 clear
2 T=1000; %Simulate for a 1000 periods
3 Y=zeros(T,1);
4 V=randn(T,1);
5 RHO=0.99;
6 i=2;
7 while i<T
8 Y(i)=Y(i-1)*RHO+V(i,1);
9 i=i+1;
10 end
11 plot(Y);
```

Figure 11: Example 9

command on line 9 if the condition temp>=0 is true. Line 10 ends the if statement. Line 11 ends the for loop. If we wanted to retain negative values only, line 8 would change to *if temp<0.* If we wanted to retain values equal to zero, line 8 would change to *if temp==0.* If we wanted to retain all values not equal to zero, line 8 would change to *if temp~=0.* If we wanted to retain values greater than 0 but less than 1, line 8 would change to if temp>0 && temp<1. If we wanted to retain values greater than 0 or greater than 1 line 8 would change to if temp>0 || temp >1.

4.3 While Loops

While loops are loops which repeat matlab statements until a condition is met. The code in figure 11 re-formulates the for loop in example 7 using the while loop. Line 6 sets the starting point of the lopp at i=2. Line 7 starts the while loop and instructs matlab to perform tasks before the end statement until the condition i<T is true. On line 8 we simulate the AR(1) model as before. Line 9 increase the value of i by 1 in each iteration. Note that unlike the For loop, the index variable is not incremented automatically and this has to be done manually.

4.4 Functions

As our code becomes longer and more complicated, it is good practice to transfer parts of the code into seperate files called functions which can then be called from a main program in exactly the same way as built in matlab functions (like inv() etc) . Suppose we want to create a function called AR which takes as inputs the value of AR coefficient ρ and number of observations T and returns as

```
1 function Y=AR(RHO,T)
2 Y=zeros(T,1);
3 V=randn(T,1);
4 for i=2:T
5 Y(i)=Y(i-1)*RHO+V(i,1);
6 end
```

Figure 12: Example 10 ar.m

output simulated $T \times 1$ matrix of data from this AR model. We can convert the code from example 7 into this function in a simple way. The code is shown in figure 12. The function begins with the word function. Then one specifies the output of the function (Y), the name of the function (AR) and the inputs (RHO,T). Lines 2 to 6 remain exactly the same. This function should be saved with the file name AR.m. The function (for e.g. with $\rho = 0.99, T = 100$)can be called from the command window (or from another piece of code) as Y=AR(0.99,100).

5 Numerical optimisation

A key tool in Matlab is the ability find the maximum/minimum of a function numerically. This is important as we need these numerical tools to find the maximum of the likelihood functions. There are several built in optmising routines in Matlab. Here we focus on a minimisation routine written by Chris Sims called CSMINWEL (available from http://sims.princeton.edu/yftp/optimize/mfiles/. We have saved these files in the folder func). This routine has been known to work well for the type of models we consider in this course.

As an example we are going to maximise the likelihood function for the linear regression model considered in example 5. The log likelihood function is given by

$$\ln lik = -T/2 \ln \left(2\pi\sigma^2\right) - \frac{1}{2}\left(\frac{(Y - XB)'(Y - XB)}{\sigma^2}\right)$$

We need to maximise this with respect to B and σ^2. To use CSMINWEL we proceed in two steps

STEP1: We first need to write a function that takes in as input a set of values for B and σ^2 and returns the value of $\ln lik$ at that particular value of the parameters. The code to calculate the likelihood function is shown in 13. The function is called loglikelihood. It takes as input, the parameters theta, and the data series Y and X. The parameter vector needs to be the first argument. Line 5 extracts the regression coefficients. Line 6 extracts the standard deviation of the error term σ and squares it. Thus we optimise with respect to σ (and not σ^2). Line 6 ensures that the value of σ^2 will always be positive. Lines 7 and 8 calculate the likelihood function for a given B

```
1 function lik=loglikelihood(theta,Y,X)
2 %size of Time series
3 T=size(Y,1);
4 %extract parameters
5 beta=theta(1:2); %coefficients
6 sigma=theta(3)^2; %variance of the error term
7 E=Y-X*beta; %calculate residuals
8 lik=(-T/2)*log(2*pi*sigma)-(0.5*(E'*E)/sigma);
9 if isnan(lik) || isinf(lik) || 1-isreal(lik)
10     lik=100000;
11 else
12     lik=-lik;
13 end
```

Figure 13: Example 11

and σ^2. The consitional statement on line 9 checks for numerical problems. In particular, it checks if the log likelihood is not a number (isnan(lik), or it equals infinity (isinf(lik)) or is a complex number (1-isreal(lik) – isreal(lik) equals 1 if lik is real and thus 1-isreal(lik) equals 1 if lik is not a real number). In case of numerical problems, the negative of the log likelihood is set to a large number. If there are no numerical problems the function returns the negative of the calculated log likelihood. The function returns the negative of the log likelihood as CSMINWEL is a minimiser (i.e. we minimise the negative of the log likelihood and this is equivalent to maximising the log likelihood).

```
clear %clears all variables in memory
addpath('func') %adds the functions in folder func
%read in data from an excel file
[data,names]=xlsread('\data\data.xls');
%dimensions of the data
T=size(data,1); %number of rows in the data
%assign data
Y=data(:,1); %Y is the first column of data
%X matrix with a constant
X=[ones(T,1) data(:,2)];
%specify starting values
theta=[0;0;0.1];
% call csminwel
[fhat,xhat,ghat,Hhat,itct,fcount,retcodehat] = csminwel('loglikelihood',theta,eye(3)*.5,[] ,1e-14,100,Y,X);
```

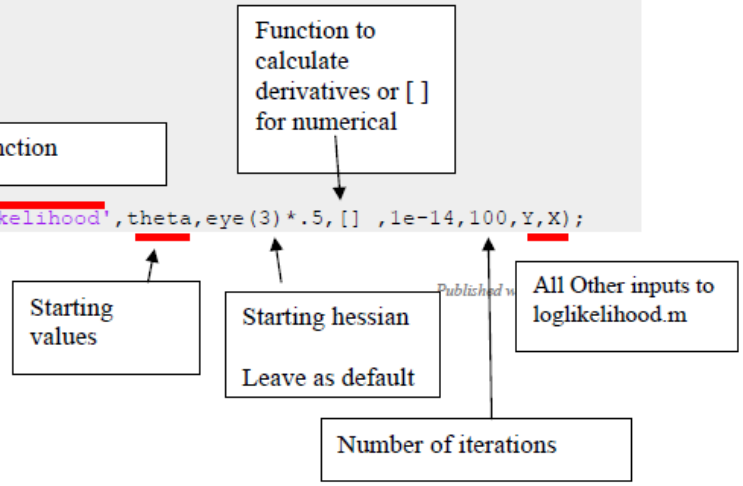

Figure 14: Example 12

STEP2: We use CSMINWEL to minimise the negative log likelihood calculated by loglikelihood.m. This code can be seen in figure 14. Line 2 ensures that the files required for CSMINWEL in the folder func can be found by Matlab. Lines 4 to 10 load the data and create the Y and X matrix. Line 12 specifies the initial values of the K parameters. Line 14 calls the function csminwel. The first input argument is the name of the function that calculates the log likelihood. The second input are the starting values. The tird input is the starting value of the inverse hessian. This can be left as default as a $K \times K$ matrix with diagonal elements equal to 0.5. If the next argument is set equal to [ ] csminwel uses numerical derivitives in the optimisation. This is the default un all out applications. The next argument is the convergence tolerance which should be left as defaults. The next input argument are the number of maximum iterations. The remaining inputs are passed directly to the function loglikelihood.m after the parameter values. The function returns the minimum of the negative log likelihood in fhat, the values of the parameters at the minimum in xhat and the inverse hessian as Hhat.Retcodehat=0 if convergence occurs. Running this code produce the same value of B as the OLS formula in the examples above.