MATLAB
for Economics and Econometrics
A Beginners Guide

John C. Frain

TEP Working Paper No. 0414

November 2014

Trinity Economics Papers

Department of Economics

Trinity College Dublin

MATLAB
for Economics and Econometrics
A Beginners Guide

John C. Frain
Economics Department
Trinity College Dublin [1]

17th November 2014

[1] Comments are welcome. My email address is `frainj@tcd.ie`

Abstract

This beginners' guide to MATLAB for economics and econometrics is an updated and extended version of Frain (2010). The examples and illustrations here are based on Matlab version 8.3 (R2014a).

It describes the new MATLAB Desktop, contains an introductory MATLAB session showing elementary MATLAB operations, gives details of data input/output, decision and loop structures, elementary plots, describes the LeSage econometrics toolbox and shows how to do maximum likelihood estimation. Various worked examples of the use of MATLAB in economics and econometrics are also given. I see MATLAB not only as a tool for doing economics/econometrics but as an aid to learning economics/econometrics and understanding the use of linear algebra there. This document can also be seen as an introduction to the MATLAB on-line help, manuals and various specialist MATLAB books.

Contents

CHAPTER 1

Introduction

1.1 Preliminaries

These notes are a guide for students of economics/econometrics who wish to learn MATLAB. Throughout there is an emphasis on MATLAB as used in MS Windows. Apart from interaction with the operating system what is set out here transfers to MATLAB running under Linux. I have not used an Apple PC but I would presume that a similar statement holds.

To get the best benefit from these notes you should read them sitting in front of a computer entering the various MATLAB instructions in the examples and running them as you read the notes. The material in the first three chapters is elementary and will be required by all economists starting with MATLAB. The remaining sections contain some more advanced material and should be read as required.

In these notes I have used a mono-spaced font for MATLAB instructions and computer input/output. Often this material is set in boxes similar to those, for example, on page 30. Here the boxes are divided with the upper part containing MATLAB code and the lower the output arising from that code. Descriptive material, explanations and commentary on the computer input/output is given in the current font.

While the first aim of these notes is to get the reader started in the use of MATLAB for econometrics it should be pointed out that MATLAB has many uses in economics. In recent years it has been used widely in what is known as computational economics/finance. This has applications in macroeconomics, determination of optimal policies and in finance. Recent references include Cerrato (2012), Kienitz and Wetterau (2012), Anitţa et al. (2011), Huynh et al. (2008), Lim and McNelis (2008), Kendrick et al. (2006), Ljungqvist and Sargent (2004), Miranda and Fackler (2002) and Marimon and Scott (1999).

I do not know of any book on MATLAB written specifically for economics. Creel (2014) is a set of lecture notes on econometrics which can be downloaded from the web. This contains examples of econometric analysis using GNU Octave which has a syntax similar to MATLAB (see section 10.1). LeSage (1999) is a free econometrics toolbox available for download from `http://www.spatial-econometrics.com/`. This site also contains links to several other MATLAB resources useful in econometrics. A free econometrics for finance toolbox is available at `http://www.kevinsheppard.com/MFE_Toolbox`.

MathWorks, the composers of MATLAB have a list of books using MATLAB for Economics/Finance (`http://www.mathworks.co.uk/support/books/index_by_categorytitle.html?category=4`). They have also issued a new econometrics toolbox (see `http://www.mathworks.com/products/econometrics/`). The MathWorks overview of this toolbox indicates that is is targeted at econometric time series in finance.

For advanced applications in applied probability Paolella (2006, 2007) are comprehensive accounts of computational aspects of probability theory using MATLAB. Higham and Higham (2005) is a good book on MATLAB intended for all users of MATLAB. Pratap (2006) is a good general "getting started" book. There are also many excellent books covering MATLAB for Engineers and/or Scientists which you might find useful if you need to use MATLAB in greater depth. The file exchange section (`http://www.mathworks.co.uk/matlabcentral/fileexchange/index?utf8=%E2%9C%93&term=econometrics`) of the MathWorks website contains contributed toolboxes, functions and other files of interest to economists.

These notes can not give a comprehensive account of MATLAB. Your copy of MATLAB comes with one of the best on-line help systems available. Full versions of the manuals are available in portable document format on the web at `http:/www.mathworks.com`. The basic function reference for MATLAB runs to over 8000 pages. For economics you need only a small proportion of these commands. Here I describe commands and functions that are of interest to economists and give examples of how MATLAB might be used in more advanced work.

MATLAB started life, in the late 70's, as a computer program for handling matrix operations. Over the years it has been extended and the basic version of MATLAB now contains more than 1000 functions. Various "toolboxes" have also been written to add specialist functions to MATLAB. Anyone can extend MATLAB by adding their own functions and/or toolboxes. Any glance at an econometrics textbook shows that econometrics involves much matrix manipulation and MATLAB provides an excellent platform for implementing the various textbook procedures and other state of the art estimators. Before you use MATLAB to implement procedures from your textbook you must understand the matrix manipulations that are involved in the procedure. When you implement them you will understand the procedure better. Using a black box package may, in some cases, be easier but how often do you know exactly what the black box is producing. Using MATLAB for econometrics may appear to involve a lot of extra work but many students have found that it helps their understanding of both matrix theory and econometrics. They then are better equipped to make use of black box based approaches.

In MATLAB as it all other packages it makes life much easier if you organize your work properly. The procedure That I use is some variation of the following –

1. Set up a new directory for each project (e. g. `s:\MATLAB\project1`
2. Set up a short-cut for each project. The short-cut should specify that the program start in the data directory for the project. If all your work is on the same PC the short-cut is best stored on the desktop. If you are working on a PC in a computer lab you will not be able to use the desktop properly and the short-cut may be stored in the directory that you have set up for the project. If you have several projects in hand you should set up separate short-cuts and directories for each of them. Each short-cut should be renamed so that you can associate it with the relevant project.
3. Before starting MATLAB you are strongly advised to amend the options in Windows explorer so that full file names (including any file extensions allocated to programs) appear in Windows Explorer and any other Windows file access menus.

1.2 The MATLAB Desktop

The current MATLAB Graphical User Interface (GUI) follows the style of the tabs and ribbon interface introduced in Microsoft Office 2007 and developed in later versions of that program[1]. If you are familiar with the modern Microsoft Office interface you will find the MATLAB one easier to use.

When you start MATLAB you will be presented with the MATLAB desktop. The current default start-up will be similar to that displayed in figure 1.1. In this default 5 windows are displayed

1. **The Command Window** — This is where you can enter and execute MATLAB commands and display any output.
2. **The Editor Window** — This is where you edit MATLAB files. These files may be script files containing a sequence of Matlab instructions for later execution, definitions of user functions or other text files.
3. **The Command History Window** — This contains a list of commands issued from the command window.
4. **The Workspace or Variables Window** — Here the objects created during the current session are listed. Double clicking on an item in this window opens the item in the Editor where you may examine it or edit it.
5. **The Current or Work Folder Window** is your project directory. At start-up this is the directory that you should have specified in the MATLAB short-cut (see page 3).

A single click on a window makes that window the active window.

In the top left hand corner of each window you will see a $\bigtriangledown$ sign in a circle. Right clicking on this brings up a context menu that allows you to do several thing with the window. The full list of actions available depends on the particular window. In particular, you can use this menu to close a window.

In the default desktop the windows are docked to or fixed within the desktop. The windows can also be undocked using this context menu. If you have a smaller screen you may find that you have not got sufficient area to support all 5 screens. In such a case I would like a larger area for the editor and command windows. Undocking the editor

[1]The new interface was introduced in R2012b. There is an account of the previous MATLAB 7 interface in the earlier edition of these notes (Frain, 2010).

window removes it from the desktop and allows it to float on your screen. The editor window then takes up the space that was occupied by the editor window. The keyboard short cut Ctrl + Shift ⇑ + U undocks the active window while the sequence Ctrl + Shift ⇑ + D docks it again.

There are 6 tabs across the top of the MATLAB desktop in figure 1.1

1. **HOME**
2. **PLOTS**
3. **APPS**
4. **EDITOR**
5. **PUBLISH**
6. **VIEW**

If you are editing a variable in the editor window the **EDITOR** and **PUBLISH** tabs are replaced by a **VARIABLE** tab. If no file is open in the **EDITOR** and no data is being edited only the first three tabs are shown.

Immediately beneath the **Tabs** is the ribbon. The contents of the ribbon depend on which Tab is active. The contents are divided into groups. For example the **HOME** tab is divided into 6 groups -

1. **FILE** — Here you will find the resources necessary to manage your files
2. **VARIABLE** — This group contains the facilities to import/save/edit data
3. **CODE**
4. **SIMULINK**
5. **ENVIRONMENT** — Here you can set the MATLAB search path
6. **RESOURCES**

The **APPS** tab provides a menu access to MATLAB Apps. When you have entered your options on the menu(s) it generates the required MATLAB script and runs it. It can also generate a file containing the MATLAB script that it has generated. It is essential that you save this script. There are many options available in the **APPS** menus and it may be difficult to replicate your work if you depend only on your memory of what you did in the GUI.

YOU will make a lot of use of the **EDITOR** and some use of the **PUBLISH** tab and I will cover these in greater detail later. Note that the **EDITOR** tab also has a **FILE** group that duplicates some of the functions of the **HOME** tab. Thus there is no need

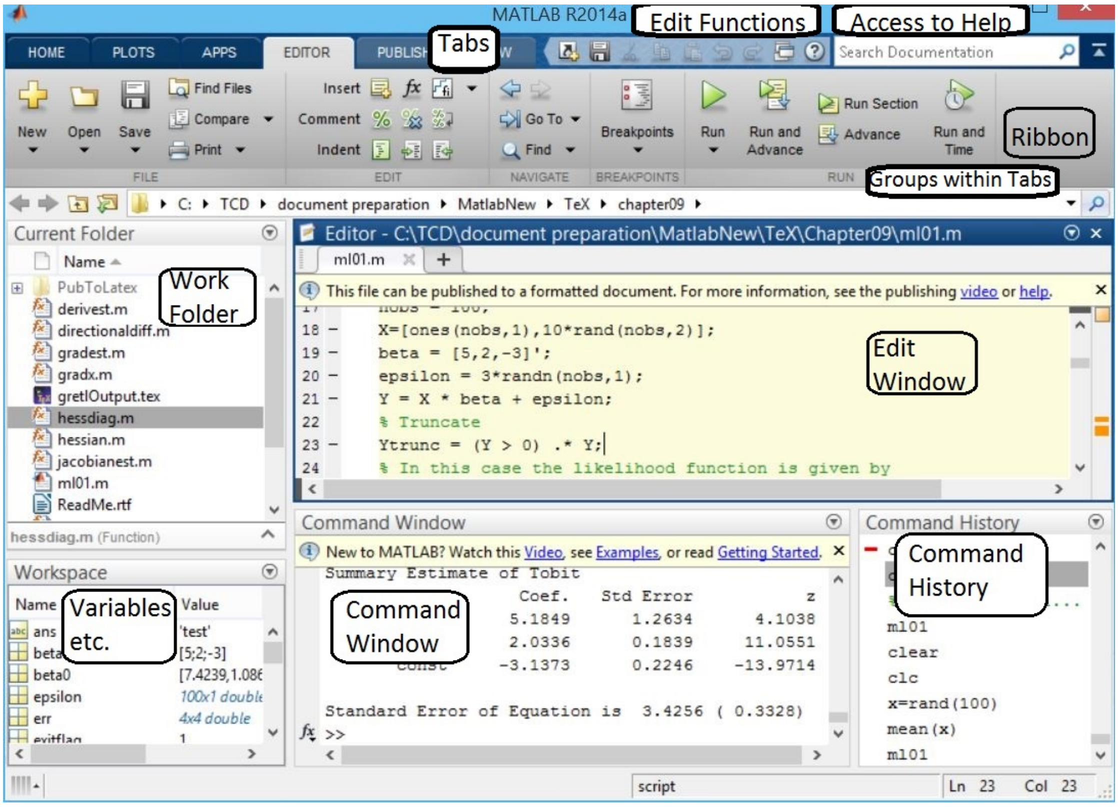


Figure 1.1: Basic Matlab GUI at start-up

to switch tabs when opening and saving files.

To the right of the tabs there is a set of icons giving quick access to some functions and HELP. to the right of this there is a search field for the help documentation.

While you can navigate the MATLAB desktop with a mouse, you can also easily navigate it from the keyboard. When you hold down the Alt key a series of letters/numbers appear across the tab bar as in figure 1.2. To select the tab or other item continue to hole the Alt key and press the key on the keyboard corresponding to the required item.

At this stage a further series of letters/numbers appear on the ribbon. Figure 1.3 shows the top left hand corner of the desktop when the **HOME** tab has been selected. Each item on the ribbon has been labelled with a letter. Just press that letter on the keyboard to access the relevant item.

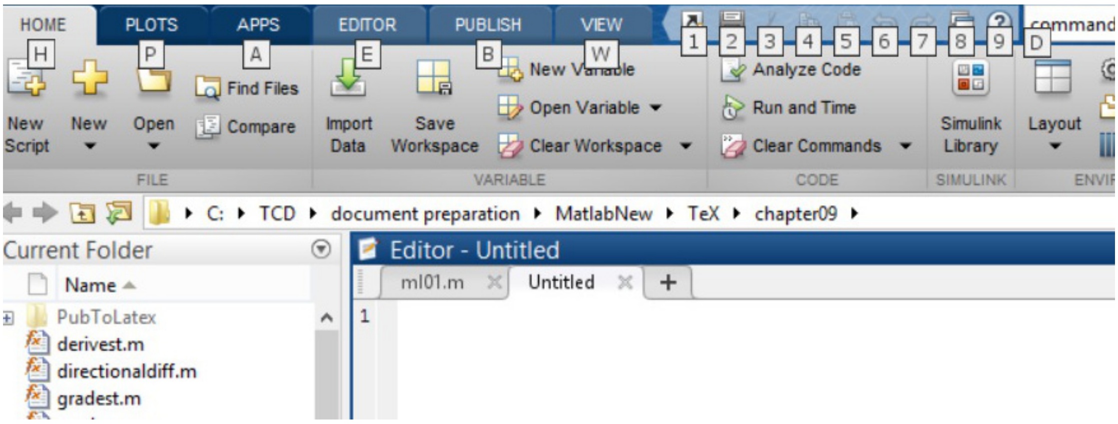


Figure 1.2: Use of Alt key to select tab

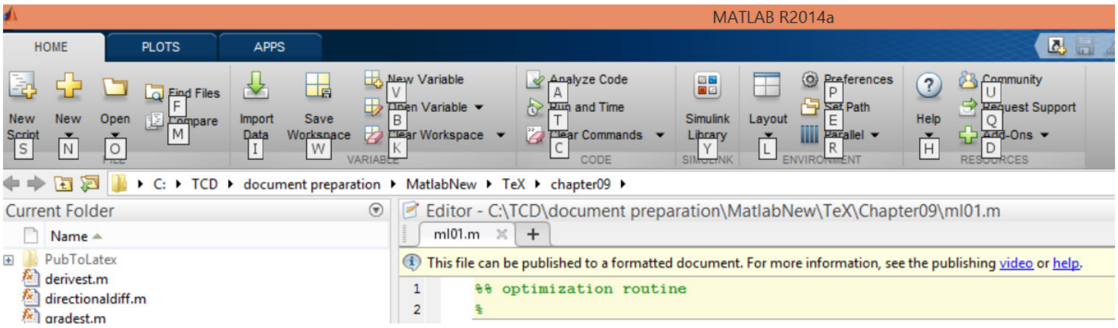


Figure 1.3: Use of Alt key after tab has been selected

1.3 Desktop Windows

1.3.1 The Command Window

The simplest use of the command window is as a calculator. With a little practice it may be as easy, if not easier, to use than a spreadsheet. Most calculations are entered almost exactly as one would write them.

```
>> 2+2
ans = 4

>> 3*2
ans = 6
```

The object `ans` contains the result of the last calculation of this kind. You may also create an object `a` which can hold the result of your calculation.

```
>> a=3^30
a = 27
>> a
a = 27
```

```
>> b=4^2+1
b = 17

>> b=4^2+1;
```

```

% continuation lines
>> 3+3 ...
+3
ans = 9
```

Type each instruction in the command window, press enter and watch the answer. Note

- The arithmetic symbols `+, -, *, /` and `^` have their usual meanings
- The assignment operator `=`
- the MATLAB command prompt `>>`
- A `;` at the end of a command suppresses output but any assignment is made or the command is completed
- If a statement will not fit on one line and you wish to continue it to a second type an ellipsis (...) at the end of the line to be continued.

Individual instructions can be gathered together in an m-file and may be run together from that file (or script). An example of a simple m-file is given in the description of the Edit Debug window below. You may extend MATLAB by composing new MATLAB instructions using existing instructions gathered together in a m-file (or function file).

You may use the up down arrow keys to recall previous commands (from the current or earlier sessions) to the Command Window. You may then edit the recalled command before running it. Further access to previous commands is available through the command window.

1.3.2 The Command History Window

If you now look at the Command History Window you will see that as each command was entered it was copied to the Command History Window. This contains all commands previously issued unless they are specifically deleted. To execute any command in the command history double click it with the left mouse button. To delete a commands from the history select them, right click the selection and select delete from the drop down menu.

At the prompt in the **Command Window** you may also access the **Command History** using the $\uparrow$ and $\downarrow$ keys. This places the commands one by one at the MATLAB prompt. When you have located the prompt you can use the $\leftarrow$, $\rightarrow$ keys or the mouse to position the cursor and edit the command. If you type the start of a command the $\uparrow$ and $\downarrow$ keys will only bring up previous commands that start with the fragment that you have entered.

1.3.3 Current Folder Window

This window displays the contents of the working or project directory. The name of this directory is given in the row below the ribbon. You can change the default by clicking on the part of the part of the displayed path that corresponds to the start of the new path and then negotiating to the new path in the Current Folder directory.

You can open m-files in the editor by double-clicking on the file name in the list.

1.3.4 The Editor Window

Clearly MATLAB would not be of much use if, every time you used it, one you had to re-enter or retrieve your commands one by one in the Command Window. You can save your commands in an m-file and run the entire set or a selection of the commands in the file. The MATLAB editor has facilities editing and saving your m-file, for deleting commands or adding new commands to the file before re-running it. Set up and run the simple example below. We shall be using more elaborate examples later.

You can set up a new m-file by selecting **new** and **script** from the **HOME** or **EDITOR** tab. Enter the following in the file[2].

```
% vol_sphere.m
% John C Frain revised 12 November 2006
% This is a comment line
% This M-file calculates the volume of a sphere
echo off
r=2
volume = (4/3) * pi * r^3;
string=['The volume of a sphere of radius ' ...
    num2str(r) ' is ' num2str(volume)];
disp(string)
% change the value of r and run again
```

In the **EDITOR** tab select **save** and **save as** and name the file as `vol_sphere`. (This will be saved in your default directory if you have set up things properly. Check that this is working properly).

Now return to the Command Window and enter `vol_sphere`. If you have followed the instructions properly MATLAB will process this as if it were a MATLAB instruction. You can change the value of the radius of the sphere in the editor and re-run the file.

The **EDITOR Window** is a programming text editor with various features colour coded. Comments are in green, variables and numbers in black, incomplete character strings in red and language key-words in blue. This colour coding helps to identify errors in a program.

The **EDITOR Window** also provides debug features for use in finding errors and verifying programs. In particular you may set a break point in the file and then run the commands in the file one by one. For example in if you have the `vol_sphere.m` open in the editor

[2]If you are reading this on a computer your pdf reader may allow you to copy and paste material from my boxes to the MATLAB editor. If some symbols do not copy and paste properly from the you may need to edit the file. Most of the examples in this book can be cut and pasted to the editor to save typing.

1. Notice that there is a — nest to each line that contains an executable MATLAB command.
2. Left-click on the — next to `echo off` and the — is replaced by a small red circle. This marks the breakpoint.
3. Now run the file from the ribbon. The script runs as far as the break point.
4. five new items have appeared on the ribbon
   Continue Continue running from breakpoint
   Step Run next line
   Step in Run next line and step into function.
   Step out Run until current function returns
   Run to Cursor Run to line containing cursor.
5. As you step through the m-file watch the output in the **COMMAND WINDOW** and the variables in the **WORKSPACE WINDOW**. You can enter various commands in the COMMAND WINDOW if you need to check that everything is going as expected.

You should return to the description of the **EDITOR WINDOW** when you start editing files.

1.3.5 Graphics Windows

This is used to display graphics generated in MATLAB. Details will be given later when we are dealing with graphics (chapter 5 on page 89).

1.3.6 The Workspace Browser

This is an option in the lower left hand corner of the desktop. Compare this with the material in the command window. Note that it contains a list of the variables already defined. Double clicking on an item in the workspace browser allows one to view and edit it.

The contents of the workspace can also be listed by the `whos` command

1.3.7 The Path Browser

MatLab comes with a large number of functions defined in m-files in various directories. You may also create your own functions and variables. MATLAB has various rules to find these functions, m-files and variables and if two of them have the same name to determine which has precedence.

1. When MATLAB encounters a name it looks first to see if it is a variable name. If it is a variable name the variable takes precedence and any function or m-file with the same name is blocked.

2. It then searches for the name as an m-file in the current directory. (This is one of the reasons to ensure that the program starts in the current directory). In this way you can redefine a MATLAB function and replace it with your own function.

3. The MATLAB search path is a list of directories that MATLAB searches sequentially for any other m-files or functions required. Starting at the first directory in the search path it uses the first such m-file or function found and ignores any in a later directory.

Thus, if one of your variables has the same name as an m-file or a MATLAB function you will not be able to access that m-file or MATLAB function. This is a common cause of problems. If, for example, you have name a variable `inv` or one of your own functions, in the current session, you will be unable to use the MATLAB `inv()` function. One way of checking that, for example, `inv()` is a MATLAB function would be to enter `help inv` on the command line. This will produce summary help file for `inv` if `inv()` is a MATLAB function. This is a common cause of problems in MATLAB but is easily fixed.

The MATLAB search path can be added to or changed at any stage by selecting **set path** in the **ENVIRONMENT** section or the **HOME** tab. Here you can make the following changes to the MATLAB path

`add Folder` Adds a directory to the MATLAB search path

`Add with subfolders` Adds a directory and its subdirectories to the MATLAB search path.

`Remove folder` Removes a directory from the MATLAB search path.

`Change the order of directories in the path` If there are two versions of a command in two different directories MATLAB will find the one closest to the top of the path, will use this and ignore the other.

If you need the changes in the current session only click on **Close**. If you want to make the changes permanent click on `Save`. The MATLAB command `addpath` can effect these changes from the **COMMAND WINDOW**.

The command `cd` changes the current working directory

1.3.8 The Help System

The help system in MATLAB is very good. It can be accessed in many ways. Perhaps the most obvious access to the help system is the *Search Documentation* invitation in the box on the extreme right hand side of the tab bar. For example suppose we want to find the inverse of a matrix and have forgotten the command. If you type `inverse` there you will be presented with the box displayed in figure 1.4. On this occasion the first item in the box shows that the `inv()` function calculates the inverse of a matrix.

To get additional details about the `inv()` function click on the first item on the list and you will be presented with the help window displayed in figure 1.5

Alternatively you can click the ? key next to the *Search Documentation* box (or press the F1 key) to access the product documentation.

The ? key is repeated on the ribbon for the **HOME** tab. Below this there is a drop-down menu giving access to a variety of introductory examples and videos.

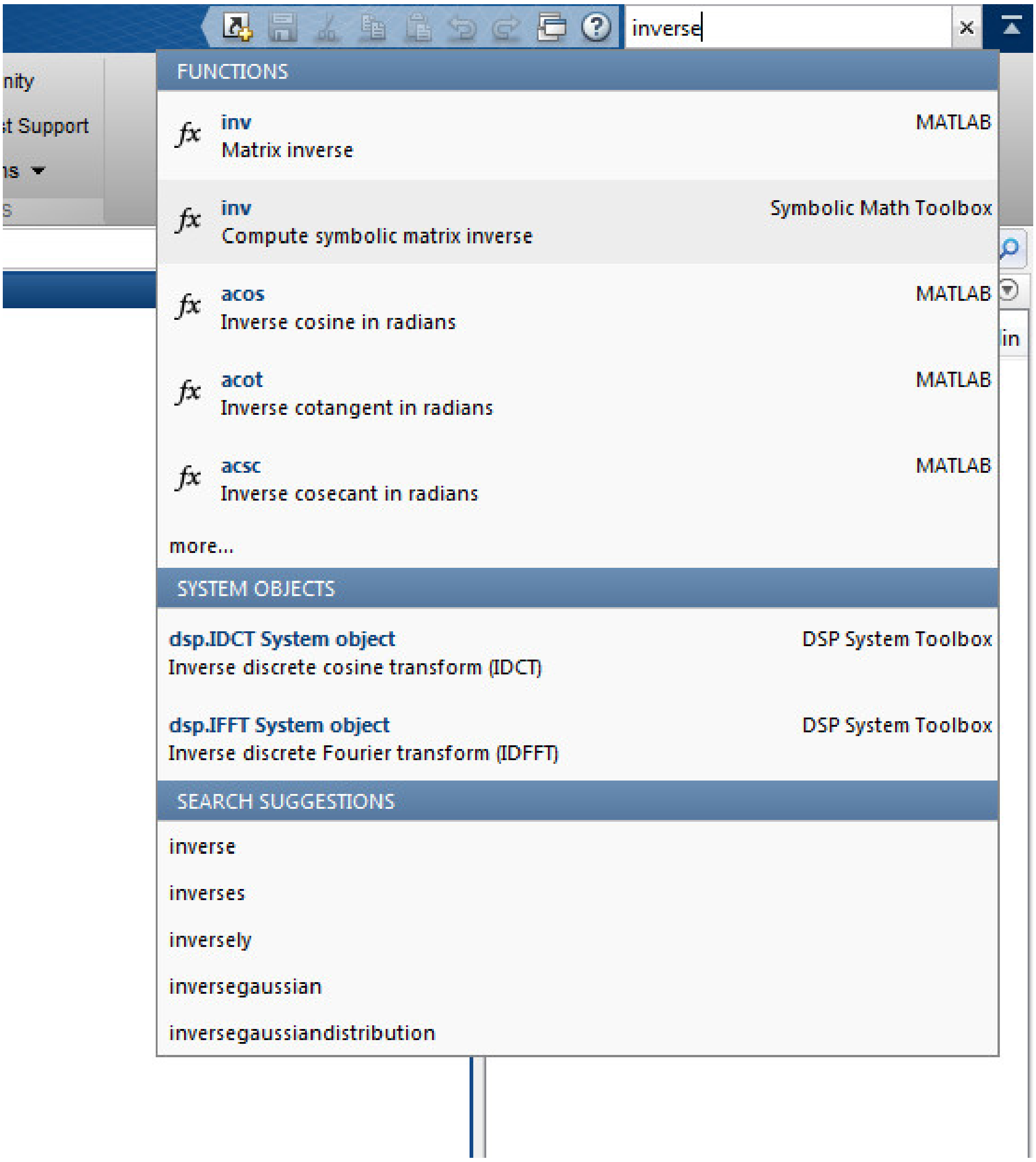


Figure 1.4: Using *Search Documentation* on Tab Bar

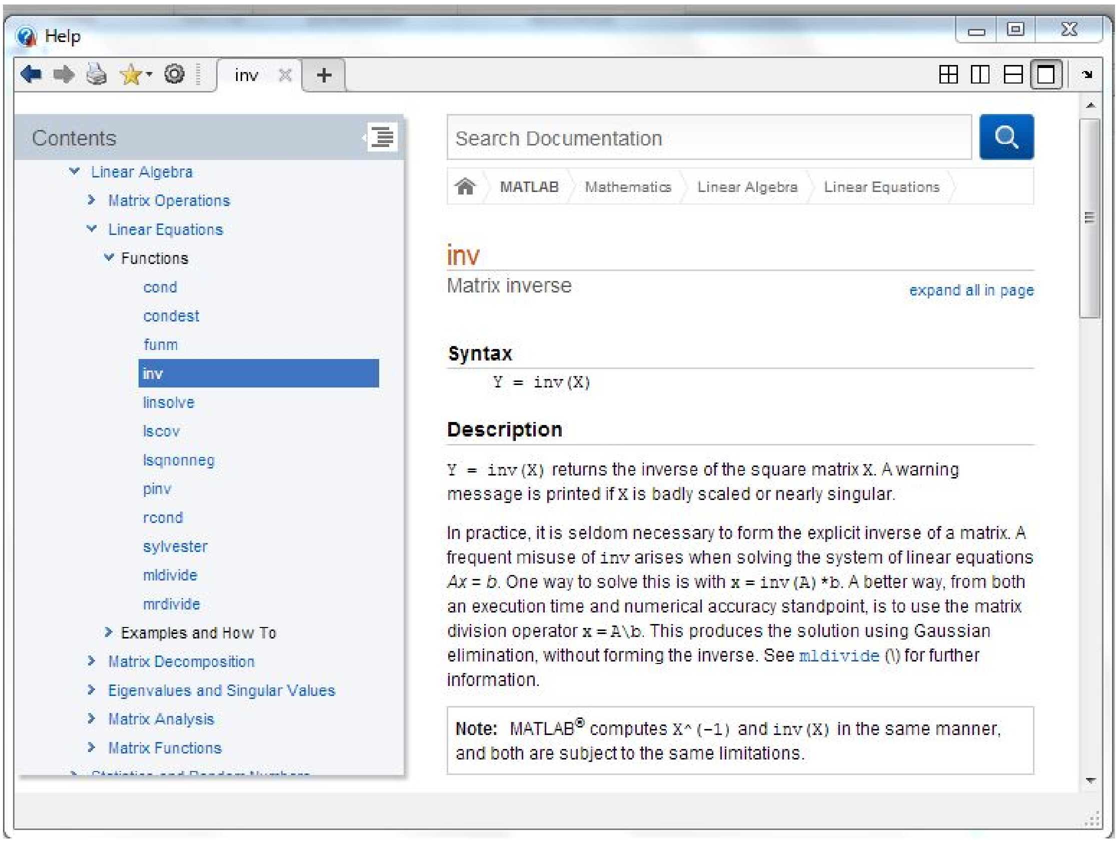


Figure 1.5: Help for `inv()` function

One can also type `help`[3] at the command prompt to get a list of available help topics for MATLAB and installed MATLAB toolboxes. For example the econometrics tool box is installed on this PC when I type `help` at the command prompt I get (with many lines on other topics deleted)

help on Command Line

```
        >> help
        HELP topics:

        Documents\MATLAB                - (No table of contents file)
        matlab\testframework            - (No table of contents file)
        matlab\demos                    - Examples.
        matlab\graph2d                  - Two dimensional graphs.
        matlab\graph3d                  - Three dimensional graphs.
        matlab\graphics                 - Handle Graphics.
***************lines deleted***************
        econ\econ                       - Econometrics Toolbox
        econ\econdemos                  - Econometrics Toolbox: Data,
                                          Demos, and Examples
***************lines deleted***************
        finance\finance                 - Financial Toolbox
        finance\calendar                - Financial Toolbox calendar
                                          functions.
        finance\finsupport              - (No table of contents file)
        finance\ftseries                - Financial Toolbox Times
```

[3] The help for the econometrics package described here is perhaps a little complicated. A simpler example would be to look at the help for the `inv()` function as in the box in this footnote. You can return to the help files for the econometrics package when you are reading section 8.3

```
        >> help inv
        inv    Matrix inverse.
        inv(X) is the inverse of the square matrix X.
        A warning message is printed if X is badly scaled or nearly singular.
**************deleted lines**************
        Reference page in Help browser
        doc inv
```

help on Command Line (cont.)

```
                                         Series Functions.
        finance\findemos                - Financial Toolbox Examples
***************lines deleted****************
        optim\optim                     - Optimization Toolbox
        optim\optimdemos                - Demonstrations.
***************lines deleted****************
        stats\stats                     - Statistics Toolbox
        stats\classreg                  - (No table of contents file)
        stats\clustering                - (No table of contents file)
        stats\statsdemos                - Statistics Toolbox --- Demos
```

You can now click on `econ\econ` to get a list of help items in that toolbox.

help on `econ\econ`

```
Econometrics Toolbox
Version 3.0 (R2014a) 30-Dec-2013

== Model Specification & Testing ==

adftest    - Augmented Dickey-Fuller test for a unit root
aicbic     - Akaike and Bayesian information criteria
archtest   - Engle test for residual heteroscedasticity
autocorr   - Sample autocorrelation
collintest - Belsley collinearity diagnostics
corrplot   - Plot variable correlations
crosscorr  - Sample cross-correlation
egcitest   - Engle-Granger cointegration test
hac        - Heteroscedasticity and autocorrelation consistent
covariance estimators
i10test    - Paired integration/stationarity tests
jcitest    - Johansen cointegration test
jcontest   - Johansen constraint test
kpsstest   - KPSS test for stationarity
lbqtest    - Ljung-Box Q-test for residual autocorrelation
```

help on econ\econ (cont.)

```
lmtest     - Lagrange multiplier test of model specification
lmctest    - Leybourne-McCabe test for stationarity
lratiotest - Likelihood ratio test of model specification
parcorr    - Sample partial autocorrelation
pptest     - Phillips-Perron test for a unit root
vratiotest - Variance ratio test for a random walk
waldtest   - Wald test of model specification

== Univariate Time Series Analysis ==

Data Filtering
hpfilter - Hodrick-Prescott filter for trend and cyclical components

ARIMAX/ARMAX/GARCH Specification
arima    - Create an ARIMA model
egarch   - Create an EGARCH conditional variance model
garch    - Create a GARCH conditional variance model
gjr      - Create a GJR conditional variance model
regARIMA - Create a regression model with ARIMA time series errors

ARIMAX/ARMAX/GARCH Modeling
arima/estimate    - Estimate ARIMA model parameters
egarch/estimate   - Estimate EGARCH model parameters
garch/estimate    - Estimate GARCH model parameters
gjr/estimate      - Estimate GJR model parameters
regARIMA/estimate - Estimate parameters of a regression model
                    with ARIMA errors

arima/filter    - Filter disturbances through an ARIMA model
egarch/filter   - Filter disturbances through an EGARCH(P,Q) model
garch/filter    - Filter disturbances through a GARCH(P,Q) model
gjr/filter      - Filter disturbances through a GJR(P,Q) model
regARIMA/filter - Filter disturbances through a regression model
                  with ARIMA errors
```

help on econ\econ (cont.)

```
arima/forecast    - Forecast ARIMA model responses and conditional
                    variances
egarch/forecast   - Forecast EGARCH model conditional variances
garch/forecast    - Forecast GARCH model conditional variances
gjr/forecast      - Forecast GJR model conditional variances
regARIMA/forecast - Forecast responses of a regression model with
                    ARIMA errors

arima/infer    - Infer ARIMA model innovations and conditional variances
egarch/infer   - Infer EGARCH model conditional variances
garch/infer    - Infer GARCH model conditional variances
gjr/infer      - Infer GJR model conditional variances
regARIMA/infer - Infer innovations of a regression model with ARIMA
                 time series errors

arima/simulate    - Simulate ARIMA model responses and conditional
                    variances
egarch/simulate   - Simulate EGARCH model conditional variances
garch/simulate    - Simulate GARCH model conditional variances
gjr/simulate      - Simulate GJR model conditional variances
regARIMA/simulate - Simulate a regression model with ARIMA time
                    series errors

arima/impulse    - Impulse response (dynamic multipliers) of an ARIMA
                   model
regARIMA/impulse - Impulse response (dynamic multipliers) of
                   regression with ARIMA errors

regARIMA/arima - Convert a regression model with ARIMA errors to an
                 ARIMAX model

Utilities
garchar       - Convert ARMA model to AR model
```

help on econ\econ (cont.)

```
garchma       - Convert ARMA model to MA model
lagmatrix     - Create matrix of lagged time series
price2ret     - Convert prices to returns
recessionplot - Add recession bands to time series plot
ret2price     - Convert returns to prices

== Multivariate Time Series Analysis ==

VARMAX Specification
vgxget - Get VARMAX model specification parameters
vgxset - Set VARMAX model specification parameters

VARMAX Modeling
vgxinfer - Infer VARMAX model innovations
vgxplot  - Plot VARMAX model responses
vgxpred  - Forecast VARMAX model responses
vgxproc  - Generate VARMAX model responses from innovations
vgxsim   - Simulate VARMAX model responses
vgxvarx  - Estimate VARX model parameters

VARMAX Utilities
vartovec  - Vector autoregression (VAR) to vector error-correction
            (VEC)
vectovar  - Vector error-correction (VEC) to vector autoregression
            (VAR)
vgxar     - Convert VARMA model to VAR model
vgxcount  - Count VARMAX model parameters
vgxdisp   - Display VARMAX model parameters and statistics
vgxloglik - VARMAX model loglikelihoods
vgxma     - Convert VARMA model to VMA model
vgxqual   - Test VARMAX model for stability/invertibility

SSM Specification
ssm   - Create a state-space model
```

help on econ\econ (cont.)

```
SSM Modeling
ssm/disp     - Display summary information of state-space models
ssm/estimate - Maximum likelihood parameter estimation of state-space
               models
ssm/filter   - Forward recursion of state-space models
ssm/forecast - Forecast states and observations of state-space models
ssm/refine   - Refine initial parameters to aid estimation of
               state-space models
ssm/simulate - Simulate observations and states of state-space models
ssm/smooth   - Backward recursion of state-space models

== Lag Operator Polynomials ==

LagOp              - Create a lag operator polynomial (LagOp) object
LagOp/filter       - Apply a lag operator polynomial to filter a
                                         time series
LagOp/isEqLagOp    - Determine if two LagOp objects are the same
                     mathematical polynomial
LagOp/isNonZero    - Find lags associated with non-zero coefficients of
                     LagOp objects
LagOp/isStable     - Determine the stability a lag operator polynomial
LagOp/minus        - Lag operator polynomial subtraction
LagOp/mldivide     - Lag operator polynomial left division
LagOp/mrdivide     - Lag operator polynomial right division
LagOp/mtimes       - Lag operator polynomial multiplication
LagOp/plus         - Lag operator polynomial addition
LagOp/reflect      - Reflect lag operator polynomial coefficients
                     around lag zero
LagOp/toCellArray  - Convert a lag operator polynomial object to a cell
                     array

>>
```

help on `arima\estimate`

estimate Estimate ARIMA model parameters
Syntax:

```
[EstMdl,EstParamCov,logL,info] = estimate(Mdl,Y)
[EstMdl,EstParamCov,logL,info] = estimate(Mdl,Y,param1,val1,...)
```

Description:

Given an observed univariate time series, estimate the parameters of an ARIMA model. The estimation process infers the residuals of the underlying response series and then fits the model to the response data via maximum likelihood.

Input Arguments:

```
Mdl - ARIMA model specification object, as produced by the ARIMA
constructor or arima/estimate method.

Y - Response data whose residuals and conditional variances are
inferred and to which the model Mdl is fit. Y is a column vector,
and therefore a single path of the underlying series. The last
observation of Y is the most recent.
```

Optional Input Parameter Name/Value Pairs:

'Y0' Presample response data, providing initial values for the model. Y0 is a column vector, and may have any number of rows, provided at least Mdl.P observations exist to initialize the model. If the number of rows exceeds Mdl.P, then only the most recent Mdl.P observations are used. If Y0 is unspecified, any necessary observations are backcasted (i.e., backward forecasted). The last row contains the most recent observation.

'E0' Mean-zero pre sample innovations, providing initial values for the model. E0 is a column vector, and may have any number of rows, provided sufficient observations exist to initialize the ARIMA model as well as any conditional variance model (the number of observations required is at least Mdl.Q, but may be more if a conditional variance model is included). If the number of rows exceeds the number necessary, then only the most recent observations

help on `arima\estimate` (cont.)

are used. If E0 is unspecified, any necessary observations are set to zero. The last row contains the most recent observation.

'V0' Positive pre sample conditional variances, providing initial values for any conditional variance model; if the variance of the model is constant, then V0 is unnecessary. V0 is a column vector, and may have any number of rows, provided sufficient observations exist to initialize the variance model. If the number of rows exceeds the number necessary, then only the most recent observations are used. If V0 is unspecified, any necessary observations are set to the average squared value of the inferred residuals. The last row contains the most recent observation.

'X' Matrix of predictor data used to include a regression component in the conditional mean. Each column of X is a separate time series, and the last row of each contains the most recent observation of each series. When pre sample responses Y0 are specified, the number of observations in X must equal or exceed the number of observations in Y; in the absence of pre sample responses, the number of observations in X must equal or exceed the number of observations in Y plus Mdl.P. When the number of observations in X exceeds the number necessary, only the most recent observations are used. If missing, the conditional mean will have no regression component regardless of the presence of any regression coefficients found in the model.

'Options' Optimization options created with OPTIMOPTIONS (or OPTIMSET).If specified, default optimization parameters are replaced by those in options. The default is an OPTIMOPTIONS object designed for the optimization function FMINCON, with'Algorithm' = 'sqp' and 'TolCon' = 1e-7. See documentation for OPTIMOPTIONS (or OPTIMSET) and FMINCON for details.

'Constant0' Scalar initial estimate of the constant of the model. If missing, an initial estimate is derived from standard time series techniques.

'AR0' Vector of initial estimates of non-seasonal autoregressivecoefficients. The number of coefficients in AR0 must equal the number of non-zero coefficients associated with the AR polynomial (excluding lag zero). If missing, initial estimates are derived from standard time series techniques.

'SAR0' Vector of initial estimates of seasonal autoregressive coefficients. The

help on `arima\estimate` (cont.)

number of coefficients in SAR0 must equal the number of non-zero coefficients associated with the SAR polynomial (excluding lag zero). If missing, initial estimates are derived from standard time series techniques.

'MA0' Vector of initial estimates of non-seasonal moving average coefficients. The number of coefficients in MA0 must equal the number of non-zero coefficients associated with the MA polynomial (excluding lag zero). If missing, initial estimates are derived from standard time series techniques.

'SMA0' Vector of initial estimates of seasonal moving average coefficients. The number of coefficients in SMA0 must equal the number of non-zero coefficients associated with the SMA polynomial (excluding lag zero). If missing, initial estimates are derived from standard time series techniques.

'Beta0' Vector of initial estimates of the regression coefficients. The number of coefficients in Beta0 must equal the number of columns in the predictor data matrix X (see above). If missing, initial estimates are derived from standard time series techniques.

'DoF0 Scalar initial estimate of the degrees-of-freedom parameter (used for t distributions only, and must exceed 2). If missing, the initial estimate is 10.

'Variance0 A positive scalar initial variance estimate associated with a constant-variance model, or a cell vector of parameter name-value pairs of initial estimates associated with a conditional variance model. As a cell vector, the parameter names must be valid coefficients recognized by the variance model. If missing, initial estimates are derived from standard time series techniques.

'Display' String or cell vector of strings indicating what information to display in the command window. Values are:

VALUE	DISPLAY
'off'	No display to the command window.
'params'	Display maximum likelihood parameter estimates, standard errors, and t statistics. This is the default.
'iter'	Display iterative optimization information.
'diagnostics'	Display optimization diagnostics.

help on `arima\estimate` (cont.)

'full' Display 'params', 'iter', and 'diagnostics'.

Output Arguments:

EstMdl - An updated ARIMA model specification object containing the parameter estimates.

EstParamCov - Variance-covariance matrix associated with model parameters known to the optimizer. The rows and columns associated with any parameters estimated by maximum likelihood contain the covariances of the estimation errors; the standard errors of the parameter estimates are the square root of the entries along the main diagonal. The rows and columns associated with any parameters held fixed as equality constraints contain zeros. The covariance matrix is computed by the outer product of gradients (OPG) method.

logL - Optimized loglikelihood objective function value.

info - Data structure of summary information with the following fields:

exitflag - Optimization exit flag (see FMINCON)
options - Optimization options (see OPTIMOPTIONS)
X - Vector of final parameter/coefficient estimates
X0 - Vector of initial parameter/coefficient estimates

Notes:

- Unspecified initial coefficient estimates are indicated by NaNs, which are derived from standard time series techniques.
- Missing values, indicated by NaNs, are removed from Y and X by listwise deletion (i.e., Y and X are merged into a composite series, and any row of the combined series with at least one NaN is removed), reducing the effective sample size. Similarly, missing values in the pre sample data Y0, E0, and V0 are also removed by listwise deletion (Y0, E0, and V0 are merged into a composite series, and any row of the combined series with at least one NaN is removed). The Y and X series, as well as the pre sample data, are also

help on `arima\estimate` (cont.)

synchronized such that the last (most recent) observation of each component series occurs at the same time.

- The parameters known to the optimizer and included in EstParamCov are ordered as follows:
  - Constant
  - Non-zero AR coefficients at positive lags
  - Non-zero SAR coefficients at positive lags
  - Non-zero MA coefficients at positive lags
  - Non-zero SMA coefficients at positive lags
  - Regression coefficients (models with regression components only)
  - Variance parameters (scalar for constant-variance models, vector of additional parameters otherwise)
  - Degrees-of-freedom (t distributions only)
- When 'Display' is specified, it takes precedence over the 'Diagnostics' and 'Display' selections found in the optimization 'Options' input.However, when 'Display' is unspecified, all selections related to the display of optimization information found in 'Options' are honoured

References:

1 Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. Time Series Analysis: Forecasting and Control. 3rd edition. Upper Saddle River, NJ: Prentice-Hall, 1994.

2 Enders, W. Applied Econometric Time Series. Hoboken, NJ: John Wiley & Sons, 1995.

3 Greene, W. H. Econometric Analysis. Upper Saddle River, NJ: Prentice Hall, 3rd Edition, 1997.

4 Hamilton, J. D. Time Series Analysis. Princeton, NJ: Princeton University Press, 1994.

See also arima, forecast, infer, simulate.

Reference page in Help browser

```
doc arima/estimate
```

When you are in the **COMMAND WINDOW** you have command completion and an

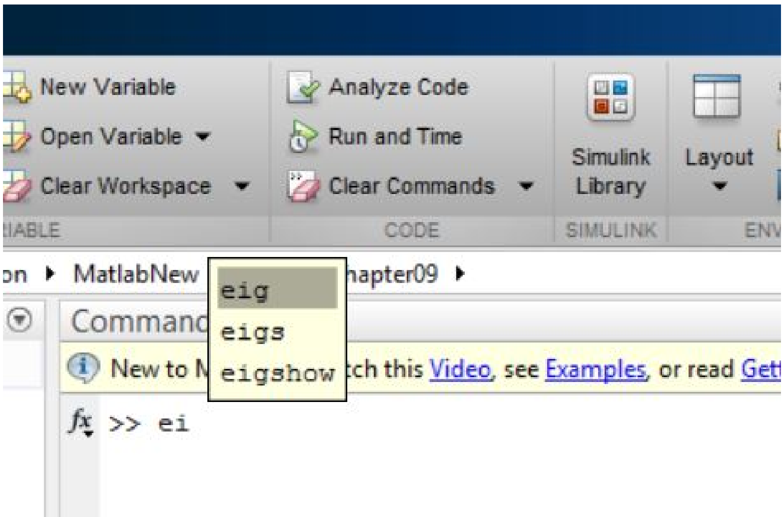
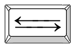


Figure 1.6

alternative access to the **HELP** system.

Say, for example we are interested in finding the eigenvalues and/or eigenvectors of a matrix. At the command line type `ei` and press the  key. A box appears showing a list of commands that stare with `ei`. This is illustrated in figure 1.6. Double click on `eig` in that box and the command is completed in the **COMMAND WINDOW**. Now that the full version of the command is in the **COMMAND WINDOW**.Type `(` after the function and you will be presented with a set of hints on the completion of the function. This is illustrated in figure 1.7

left click on the command

The more help at the end of the hints brings up the full help for the `eig` function.

Left-clicking on the `eig` function brings up a context menu which contains provides access to the **HELP** system.

The help facilities described above are also available in the `EDITOR WINDOW`.

1.3.9 Miscellaneous Commands

The following MATLAB commands will help in managing the MATLAB desktop

`clear` — Clears the MATLAB workspace.

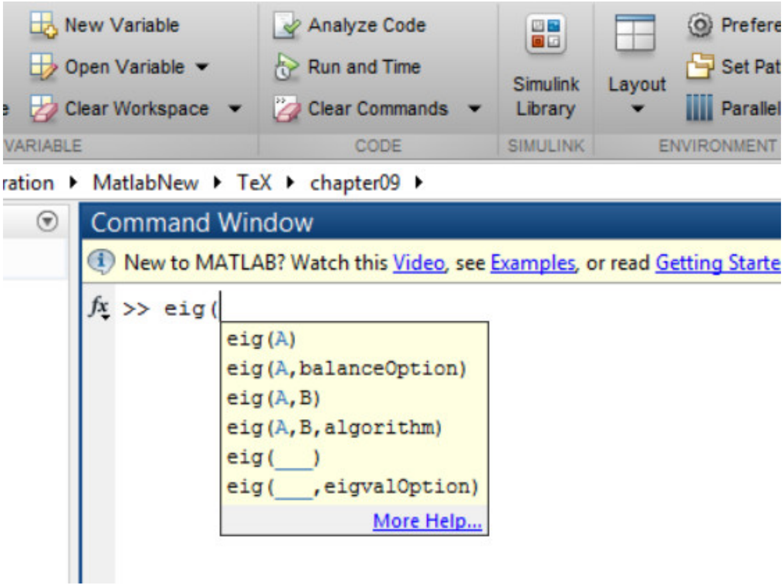


Figure 1.7

`clc` — Clears the contents of the Command Window

`clf` — Clears the contents of the Figure Window

If MATLAB appears to be caught in a loop and is taking too long to finish a command it may be aborted by Ctrl + C (Hold down the Ctrl key and press C). MATLAB will then return to the command prompt

`diary filename` After this command all input and most output is echoed to the specified file. The commands `diary off` and `diary on` will suspend and resume input to the diary (log) file.

CHAPTER 2

Basic Matlab and Some introductory Examples

The basic variable in MATLAB is an Array. (The numbers entered earlier can be regarded as (1×1) arrays, Column vectors as $(n \times 1)$ arrays and matrices as $(n \times m)$ arrays. MATLAB can also work with multidimensional arrays.

2.1 Sample MATLAB session in the Command Window

It is recommended that you work through the following sitting at a PC with MATLAB running and enter the commands in the Command window. Most of the calculations involved are simple and they can be checked with a little mental arithmetic. The upper section of each box is MATLAB input, the lower MATLAB output. In some cases the box is not divided and reports input and output in a command window. This session only covers a small proportion of the functionality of MATLAB

The ideas of this chapter can also help you with learning and understanding other aspects of linear algebra and their implementation in MATLAB. Simply compose a simple example and implement it both on paper and in MATLAB.

2.1.1 Entering Matrices

```
>> x=[1 2 3 4] % assigning values to a (1 by 4) matrix (row vector)
--------------------------------------------------------------------
x =
        1        2        3        4
```

```
>> x=[1; 2; 3; 0]  % A (4 by 1) (column) vector
--------------------------------------------------------------------
x =
        1
        2
        3
        4
```

```
>> x=[1,2,3;4,5,6] % (2 by 3) matrix
--------------------------------------------------------------------
x =
        1        2        3
        4        5        6
```

```
>> x=[]  %Empty array
--------------------------------------------------------------------
x = []
```

The matrix is entered row by row. The elements in a row are separated by spaces or commas. The rows are separated by semi-colons. The `</,<` is the MATLAB prompt. Anything on a line after a `%` sign is regarded as a comment. These are intended for you to read and you need not type these comments. When you are writing your own MATLAB programs you should use comments to explain what you are doing and why you are doing it. As already mentioned if you add a ; at the end of a command (before the comment sign %) the output will be suppressed.

2.1.2 Basic Matrix operations

. The following examples are simple. Check the various operations and make sure that you understand them. This will also help you revise some matrix algebra which you will need for your econometric theory.

```
>> x=[1 2;3 4]
--------------------------------
x =
        1        2
        3        4
```

```
>> y=[3 7;5 4]
--------------------------------
y =
        3        7
        5        4
```

```
        >> x+y  %addition of two matrices - same dimensions
--------------------------------
ans =
        4        9
        8        8
```

```
>> y-x  %matrix subtraction
--------------------------------
ans =
        2        5
        2        0
```

```
>> x*y  % matrix multiplication
--------------------------------
ans =
       13       15
       29       37
```

Note that when matrices are multiplied their dimensions must conform. The number of columns in the first matrix must equal the number of rows in the second otherwise

MATLAB returns an error. Try the following example. When adding matrices a similar error will be reported if the dimensions do not match

```
>> x=[1 2;3 4]
--------------------------------------------------------------
x =
     1     2
     3     4
```

```
>> z=[1,2]
--------------------------------------------------------------
z =
     1     2
```

```
>> x*z
--------------------------------------------------------------
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

```
>> inv(x) % find inverse of a matrix
--------------------------------------------------------------
ans =


  -2.00000   1.00000
   1.50000  -0.50000
```

```
>> x*inv(x) % verify inverse
--------------------------------------------------------------
ans =


  1.00000  0.00000
  0.00000  1.00000
```

```
>> y*inv(x) % multiply y by the inverse of x
ans =

   4.50000  -0.50000
  -4.00000   3.00000
```

```
>> y/x % alternative expression
ans =

   4.50000  -0.50000
  -4.00000   3.00000
```

```
>> inv(x)*y pre-multiply y by the inverse of x
ans =

  1.0e+01  *

  -0.10000  -1.00000
   0.20000   0.85000
```

```
>> x\y  % alternative expression - different algorithm - better for regression
ans =

  1.0e+01  *

  -0.10000  -1.00000
```

$\boldsymbol{x} = \boldsymbol{A}\backslash\boldsymbol{B}$ (`x=A\B`) solves $\boldsymbol{Ax} = \boldsymbol{B}$ Using Gaussian elimination. It is more robust numerically and more efficient than first calculating the inverse as in (`x = inv(A)*B`). The alternative expression should be used in calculating regression coefficients.

2.1.3 Kronecker Product

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \dots & a_{1m}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \dots & a_{2m}\boldsymbol{B} \\ \vdots & \vdots & & \vdots \\ a_{n1}\boldsymbol{B} & a_{n2}\boldsymbol{B} & \dots & a_{nm}\boldsymbol{B} \end{pmatrix}$$

```
x>> x=[1 2;3 4]
```

```
x =
     1     2
     3     4
```

```
>> I=eye(2,2) % Alternatively I = I(2)produces the
              % same result.
```

```
I =
     1     0
     0     1
```

```
>> kron(x,I)
```

```
ans =


     1     0     2     0
     0     1     0     2
     3     0     4     0
     0     3     0     4
```

```
>> kron(I,x)
--------------------------------------------------------------
ans =


     1     2     0     0
     3     4     0     0
     0     0     1     2
     0     0     3     4
```

2.1.4 Examples of number formats

This subsection gives examples of some of the ways in which the number formats in output cam be changed. the command

```
format
```

sets output to the default. The `format` command only changes the displayed output and does not effect how results are stored internally. Several examples follow of this command followed by a specifier.

```
>> x=12.345678901234567;
>> format loose % includes blank lines to space output
>> x
--------------------------------------------------------------
x =


   12.3457
```

```
>> format compact %Suppress blank lines
>> x
--------------------------------------------------------------
x =
   12.3457
```

```
>> format long %15 digits after decimal for double
>> x
---
x =
  12.345678901234567
```

```
>> format short e % exponential or scientific format
>> x
---
x =
  1.2346e+001
```

```
>> format long e
>> x
---
x =
    1.234567890123457e+001
```

```
>> format short g % decimal or exponential
>> x
---
x =
         12.346
```

```
>> format long g
>> x
---
x =
            12.3456789012346
```

```
>> format bank % currency format (2 decimals)
>> x
---
x =
           12.35
```

2.1.5 fprintf function

```
>> fprintf('%6.2f\n', x )
 12.35
>> fprintf('%6.3f\n', x )
12.346
>> fprintf('The number is %6.4f\n', x )
The number is 12.3457
```

Here `fprintf` prints to the command window according to the format specification '%6.4f\n'. In this format specification the % indicates the start of a format specification. There will be at least 6 digits displayed of which 4 will be decimals in floating point (f). The \n indicates that the cursor will then move to the next line. For more details see page 75.

2.1.6 element by element operations

The +, −, ∗ and / operators do standard matrix addition, subtraction etc, respectively. The dot operators .∗ and ./ provide element by element operations as in the following examples.

```
% .operations
>> x=[1 2;3 4];
>> y=[3 7;5 4];
>> x .* y  %element by element multiplication
---------------------------------------------------------------
ans =


        3       14
       15       16
```

```
>> y ./ x  %element by element division
---------------------------------------------------------------
ans =


  0.3333    0.2857
  0.6000    1.0000
```

```
>> z=[3 7;0 4];
>> x./z
---------------------------------------------------------------------------
Warning: Divide by zero.
ans =
    0.3333    0.2857
       Inf    1.0000
```

2.1.7 Mixed Scalar and Matrix Operations

In MATLAB adding a scalar to or multiplying a scalar by a matrix does that operation on each element of the matrix.

```
>> a=2;
>> x+a
---------------------------------------------------------------------------
ans =
        3        4
        5        6
```

```
>> x-a
---------------------------------------------------------------------------
ans =
        -1         0
         1         2
```

```
>> x*2
---------------------------------------------------------------------------
ans =
        2        4
        6        8
```

```
>> x/2
---------------------------------------------------------------------------
ans =
  0.5000   1.0000
  1.5000   2.0000
```

2.1.8 Exponents

In MATLAB it is possible to (i) raise a matrix to some power, (ii) raise each element of a matrix to a power and (iii) raise the elements to specified powers (possibly different for each element).

```
% x^a is x^2 or x*x i.e.
>> x^a
--------------------------------------------------------------------------
ans =
        7        10
       15        22
```

```
>> x .^ a % element by element to the power
--------------------------------------------------------------------------
ans =


1     4
9    16
```

```
% element by element exponent
>> z = [1 2;2 1]
>> x .^ z
--------------------------------------------------------------------------
ans =
        1        4
        9        4
```

2.1.9 Miscellaneous Functions

Some functions operate on each element of of a matrix.

```
>> x = [ 1 2 ; 3 4];
>> exp(x)
---------------------------------
    2.7183    7.3891
    20.0855   54.5982
```

```
>> log(x)
---------------------------------
ans =
    0         0.6931
    1.0986    1.3863
```

```
>> sqrt(x)
---------------------------------
ans =
    1.0000    1.4142
    1.7321    2.0000
```

Using negative numbers in the argument of logs and square-roots produces an error in many econometric/statistical packages. MATLAB returns complex numbers. Take care!! This is mathematically correct but may not be what you want.

```
>> z=[1 -2]
z =
1    -2
>> log(z)
ans =
0.0000 + 0.0000i   0.6931 + 3.1416i

>> sqrt(z)
ans =
1.0000 + 0.0000i   0.0000 + 1.4142i
```

the function `det(`$\boldsymbol{A}$`)` calculates the determinant of a matrix $\boldsymbol{A}$.

```
>> det(x)
---------------------------------------------------------------
ans =
    -2
```

The function `trace(`$\boldsymbol{A}$`)` calculates the trace of a matrix $\boldsymbol{A}$.

```
>> trace(x)
---------------------------------------------------------------
ans =
     5
```

The function `diag(`$\boldsymbol{X}$`)` where $\boldsymbol{X}$ is a square matrix puts the diagonal of $\boldsymbol{X}$ in a matrix. The function `diag(`$\boldsymbol{Z}$`)` where $\boldsymbol{Z}$ is a column vector outputs a matrix with diagonal $\boldsymbol{Z}$ and zeros elsewhere

```
>> z=diag(x)
---------------------------------------------------------------
z =
     1
     4
```

```
>> u=diag(z)
---------------------------------------------------------------
u =
     1     0
     0     4
```

The function `rank(`$\boldsymbol{Z}$`)` estimates the rank of a matrix.

```
>> a=[2 4 6 9
3 2 5 4
2 1 7 8]
---------------------------------------------------------------
a =
     2     4     6     9
     3     2     5     4
```

```
     2     1     7     8
```

```
>> rank(a)
ans =
     3
```

`sum(`$\boldsymbol{A}$`)` returns sums along different dimensions of an array. If $\boldsymbol{A}$ is a row or column vector, `sum(`$\boldsymbol{A}$`)` returns the sum of the elements. If $\boldsymbol{A}$ is a matrix, `sum(`$\boldsymbol{A}$`)` treats the columns of $\boldsymbol{A}$ as vectors, returning a row vector of the sums of each column.

```
>> x=[1 2 3 4]
--------------------------------------------------
x =
     1     2     3     4
```

```
>> sum(x)
--------------------------------------------------
ans =
    10
```

```
>> sum(x')
--------------------------------------------------
ans =
    10
```

```
>> x=[1 2;3 4]
--------------------------------------------------
x =
     1     2
     3     4
```

```
>> sum(x)
--------------------------------------------------
ans =
     4     6
```

The function `reshape(`$\boldsymbol{A}$`,m,n)` returns the $m \times n$ matrix $\boldsymbol{B}$ whose elements are taken

column-wise from $\boldsymbol{A}$. An error results if $\boldsymbol{A}$ does not have exactly mn elements

```
>> x=[1 2 3 ; 4 5 6]
x =
     1     2     3
     4     5     6
```

```
>> reshape(x,3,2)
ans =
     1     5
     4     3
     2     6
```

`blkdiag(`$\boldsymbol{A}$`,`$\boldsymbol{B}$`,`$\boldsymbol{C}$`)` constructs a block diagonal matrix from the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ $\boldsymbol{C}$ etc.

```
a =
     1     2
     3     4
>> b=5
b =
     5
>> c=[6 7 8;9 10 11;12 13 14]
c =
     6     7     8
     9    10    11
    12    13    14
>> blkdiag(a,b,c)
ans =
     1     2     0     0     0     0
     3     4     0     0     0     0
     0     0     5     0     0     0
     0     0     0     6     7     8
     0     0     0     9    10    11
     0     0     0    12    13    14
```

The function

2.1.10 Eigenvalues and Eigenvectors

The function `eig()` calculates eigenvectors and eigenvalues. The following are typical examples of the use of these functions. It is possible to specify outputs in greater detail or specify the algorithm to be used in the calculation — see the manual.

```
>> A=[54    45    68
    45    50    67
    68    67    95]
>> eig(A) % Vector with eigenvalues
--------------------------------------------------------------------
ans =
    0.4109
    7.1045
  191.4846
```

```
>> [V,D]=eig(A) % eigen vectors are columns of V
                % eiden values are on diagonal of D
--------------------------------------------------------------------
V =
    0.3970    0.7631    0.5100
    0.5898   -0.6378    0.4953
   -0.7032   -0.1042    0.7033
D =
    0.4109         0         0
         0    7.1045         0
         0         0  191.4846
```

```
>> Test=A*V
--------------------------------------------------------------------
Test =
    0.1631    5.4214   97.6503
    0.2424   -4.5315   94.8336
   -0.2890   -0.7401  134.6750
```

```
>> Test ./ V
--------------------------------------------
ans =
     0.4109    7.1045  191.4846
     0.4109    7.1045  191.4846
     0.4109    7.1045  191.4846
```

2.1.11 Transpose of a matrix

The transpose of a matrix is denoted by a quote mark. i. e. `A’` is the transpose of `A`.

```
>> A = [1 2 3; 4 5 6]
--------------------------------------------
A =


   1   2   3
   4   5   6
```

```
>> A'
--------------------------------------------
ans =


   1   4
   2   5
   3   6
```

2.1.12 Sequences

`[first:increment:last]` is a row vector whose elements are a sequence with first element `first`, second element `first+ increment` and continues while the new entry is less than `last`.

```
>> [1:2:9]
--------------------------------------------
ans =
         1         3         5         7         9
```

or

```
>> [2:2:9]
---------------------------------------------------------------
ans =
        2         4         6         8
```

If only two numbers are specified it is assumed that the increment is 1.

```
>> [1:4]
---------------------------------------------------------------
ans =
        1         2         3         4
```

To get a sequence in a column vector use the transpose operator defined in the previous subsection.

```
>> [1:4]'
---------------------------------------------------------------
ans =

        1
        2
        3
        4
```

2.1.13 Creating Special Matrices

The `eye(n)` creates an $n \times n$ identity matrix

```
>> x=eye(4)
---------------------------------------------------------------
x =
        1         0         0         0
        0         1         0         0
        0         0         1         0
        0         0         0         1
```

The `ones(n,m)` creates an $n \times m$ matrix with all its elements equal to one. If the required matrix is square $n \times n$ one has the option of using `n`.

```
>> x=ones(4)
------------------------------------------------------------
x =
        1        1        1        1
        1        1        1        1
        1        1        1        1
        1        1        1        1
```

```
>> x=ones(4,2)
------------------------------------------------------------
x =
        1        1
        1        1
        1        1
        1        1
```

The `zeros()` function is similar to the `ones()` function except that it creates matrices of zeros rather that ones.

```
>> x=zeros(3)
------------------------------------------------------------
x =
        0        0        0
        0        0        0
        0        0        0
```

```
>> x=zeros(2,3)
------------------------------------------------------------
x =
        0        0        0
        0        0        0
```

The `size` function returns the the number of elements in each dimension of a matrix. e. g. if ($\boldsymbol{A}$) is $n \times m$ then `size(A)` returns `n m`. The function `numel(A)` returns the number of elements in ($\boldsymbol{A}$) (nm).

```
>> size(x)
-----------------------------------------------------------
     2     3
```

```
>> numel(x)
-----------------------------------------------------------
ans =

     6
```

2.1.14 Random number generators

In MATLAB the basic random number generator is `rand()` which generates pseudo uniform random numbers in the range $[0, 1]$.

```
>> x=rand(5) % generate a 5 × 5 square matrix of random numbers.
-----------------------------------------------------------
x =
   0.81551  0.55386  0.78573  0.05959  0.61341
   0.58470  0.92263  0.78381  0.80441  0.20930
   0.70495  0.89406  0.11670  0.45933  0.05613
   0.17658  0.44634  0.64003  0.07634  0.14224
   0.98926  0.90159  0.52867  0.93413  0.74421
```

```
>> x=rand(5,1) % generate a 5 × 1 matrix of random numbers.
-----------------------------------------------------------
x =
   0.21558
   0.62703
   0.04805
   0.20085
   0.67641
```

```
>> x=randn(1,5) % generate a 1 × 5 matrix of random numbers.
---------------------------------------------------------------
x =
   1.29029   1.82176  -0.00236   0.50538  -1.41244
```

To generate a matrix of uniform random numbers on the interval $[a, b]$ use the expression

$$a + (b - a)x$$

where x is a matrix of uniform random numbers on the interval $[0, 1]$.

An economist is probably more interested in generating normal random numbers. The MATLAB function `randn()` generates normal random numbers with zero mean and unit variance. The syntax of this command is similar to that of the `rand` function. If you require normal random numbers with mean μ and variance σ^2 use something like

$$x = \mu + \sigma z$$

where z is an array of simulated standardized normal variates.

```
>> z=randn(2,3)  % generate a 2 × 3 matrix of standard normal random
                 % numbers.
>> z=randn(2,3)
---------------------------------------------------------------
z =

0.5377   -2.2588    0.3188
1.8339    0.8622   -1.3077
```

```
>> rng('default'); % re-initialize the random number generator
```

```
>> x = 2 + 4 * randn(2,3)  % generate a 1 × 5 matrix of standard
                           % normal random  numbers with mean  μ and
                           % variance σ^2.
---------------------------------------------------------------
x =
```

```
4.1507   -7.0354    3.2751
9.3355    5.4487   -3.2308
```

Every time MATLAB is started it initializes the the random number generator to the same state. Unlike many statistical packages every time MATLAB is started and the same instructions are given to generate a set or sets of random numbers, the same random numbers are generated.

At any stage in a MATLAB session you can initialize the random number generator to its initial state. use as In the middle box of the last three boxes above the command `rng('default')` achieves this.

It is also possible to "seed" the random generator with the command

```
>> rng(seed)
```

where `seed` is an integer between 0 and 2^{32}. This enables you to produce replicable simulations using random numbers other than the default.

The random number generators in earlier versions of MATLAB used different algorithms to generate random number sequences. If you do need to replicate an older analysis which used these older algorithms there are instructions in the documentation for the random number generators. My recommendation would be to stick with the new generators.

2.1.15 Extracting parts of a matrix, Joining matrices together to get a new larger matrix

Extract a single element from a row vector. command

```
>> arr1=[2 4 6 8 10];
>> arr1(3)
ans = 6
```

Extract a single element from a matrix. The first coordinate refers to the row number, the second to the column.

```
>> arr2=[1, 2, -3;4, 5, 6;7, 8, 9];
>> arr2(2, 2)
--------------------------------------------------------------
ans = 5
```

The : operator generates a sequence of coordinates to be extracted from a matrix (See subsection 2.1.12). Thus the first example extracts columns 2 to 3 of row 2 of the matrix.

```
>> arr2(2,2:3)

ans =

5     6
```

using : on its own extracts the entire column or row;

```
>> arr2(2,:)

ans =

4     5     6
```

If we are not sure how many elements are in a row or column we may use `end` to signify the last element in the matrix.

```
>> arr2(3,2:end)

ans =

8     9
```

2.1.16 Using sub-matrices on left hand side of assignment

The examples in this subsection show how to assign values to certain sub-matrices of a matrix.

```
>> arr4=[1 2 3 4;5 6 7  8 ;9 10 11 12]
--------------------------------------
arr4 =
        1         2         3         4
        5         6         7         8
        9        10        11        12
```

```
>> arr4(1:2,[1,4])=[20,21;22 23]
--------------------------------------
arr4 =
       20         2         3        21
       22         6         7        23
        9        10        11        12
```

```
>> arr4=[20,21;22 23]
--------------------------------------
arr4 =
       20        21
       22        23
```

```
>> arr4(1:2,1:2)=1
--------------------------------------
arr4 =
     1     1
     1     1
```

```
>> arr4=[1 2 3 4;5 6 7  8 ;9 10 11 12]
```

```
arr4 =
        1         2         3         4
        5         6         7         8
        9        10        11        12
```

```
>> arr4(1:2,1:2)=1
```

```
arr4 =
        1         1         3         4
        1         1         7         8
        9        10        11        12
```

2.1.17 Stacking Matrices

```
>> x=[1 2;3 4]
```

```
x =
        1         2
        3         4
```

```
>> y=[5 6; 7 8]
```

```
y =
        5         6
        7         8
```

```
>> z=[x,y,(15:16)'] % join matrices side by side
```

```
z =


        1         2         5         6        15
        3         4         7         8        16
```

```
>> z=[x',y',(15:16)']' % Stack matrices vertically
----------------------------------------------------------------
z =
        1        2
        3        4
        5        6
        7        8
       15       16
```

See also the help files for the MATLAB commands `cat`, `horzcat` and `vertcat`.

2.1.18 Special Values

```
>> pi % value of π
----------------------------------------------------------------

pi = 3.1416
```

```
>> exp(1) % value of e
----------------------------------------------------------------
ans =
2.7183
```

```
>> clock % Extract date and time
----------------------------------------------------------------
ans =
1.0e+03 *
2.0140   0.0110   0.0100   0.0220   0.0070   0.0475
  YEAR    Month      Day    Hours  Minutes  Seconds
```

The command `tic` starts a stopwatch. Subsequent `toc` commands measure the time since the last `tic` command.

2.2 Examples of Use of Command Window

Work through the following examples using MATLAB.

1. let $\boldsymbol{A} = \begin{pmatrix} 3 & 0 \\ 5 & 2 \end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix} 1 & 4 \\ 4 & 7 \end{pmatrix}$ Use MATLAB to calculate.

   (a) $\boldsymbol{A} + \boldsymbol{B}$

   (b) $\boldsymbol{A} - \boldsymbol{B}$

   (c) $\boldsymbol{AB}$

   (d) $\boldsymbol{AB}^{-1}$

   (e) $\boldsymbol{A}/\boldsymbol{B}$

   (f) $\boldsymbol{A}\backslash\boldsymbol{B}$

   (g) $\boldsymbol{A}. * \boldsymbol{B}$

   (h) $\boldsymbol{A}./\boldsymbol{B}$

   (i) $\boldsymbol{A} \otimes \boldsymbol{B}$

   (j) $\boldsymbol{B} \otimes \boldsymbol{A}$

   Use pen, paper and arithmetic to verify that your results are correct.

2. Let $\boldsymbol{A} = \begin{pmatrix} 1 & 4 & 3 & 7 \\ 2 & 6 & 8 & 3 \\ 1 & 3 & 4 & 5 \\ 4 & 13 & 15 & 15 \end{pmatrix}$

   Use the relevant MATLAB function to show that the rank of A is three. Why is it not four?

3. Solve $\boldsymbol{Ax} = \boldsymbol{a}$ for $\boldsymbol{x}$ where $\boldsymbol{A} = \begin{pmatrix} 1 & 4 & 3 & 7 \\ 2 & 6 & 8 & 3 \\ 1 & 3 & 4 & 5 \\ 2 & 1 & 7 & 6 \end{pmatrix}$ and $\boldsymbol{a} = \begin{pmatrix} 14 \\ 8 \\ 10 \\ 18 \end{pmatrix}$

4. Generate $\boldsymbol{A}$ which is a 4×4 matrix of uniform random numbers. Calculate the trace and determinant of A. Use MATLAB to verify that

(a) The product of the eigenvalues of $\boldsymbol{A}$ is equal to the determinant of $\boldsymbol{A}$

(b) The sum of the eigenvalues of $\boldsymbol{A}$ is equal to the trace of $\boldsymbol{A}$. (You might find the MATLAB functions `sum()` and **prod()** helpful - please see the relevant help files). Do these results hold for an arbitrary matrix A.

5. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two 4×4 matrices of independent N(0,1) random numbers. If tr($\boldsymbol{A}$) is the trace of $\boldsymbol{A}$. Show that

   (a) $\text{tr}(\boldsymbol{A} + \boldsymbol{B}) = \text{tr}(\boldsymbol{A}) + \text{tr}(\boldsymbol{B})$

   (b) $\text{tr}(4\boldsymbol{A}) = 4\text{tr}(\boldsymbol{A})$

   (c) $\text{tr}(\boldsymbol{A}') = \text{tr}(\boldsymbol{A})$

   (d) $\text{tr}(\boldsymbol{BA}) = \text{tr}(\boldsymbol{AB})$

   Which of these results hold for arbitrary matrices? Under what conditions would they hold for non-square matrices?

2.3 Regression Example

In this Example I shall use the instructions you have already learned to simulate a set of observations from a linear equation and use the simulated observations to estimate the coefficients in the equation. In the equation y_t is related to x_{2t} and x_{3t} according to the following linear relationship.

$$y_t = \beta_1 + \beta_2 x_{2t} + \beta_3 x_{3t} + \varepsilon_t, \quad t = 1, 2, \ldots, N$$

or in matrix notation

$$\boldsymbol{y} = \boldsymbol{X\beta}$$

where

- x_2 is a trend variable which takes the values (1,2, ... 50)
- x_3 is a random variable with uniform distribution on $[3, 5]$
- ε_t are independent identically distributed normal random variables with zero mean and constant variance σ^2.

- $\beta_1 = 5$, $\beta_2 = 1$ and $\beta_3 = 0.1$ and ε_t are iidn(0,.04) ($\sigma^2 = 0.04$)

1. Verify that the model may be estimated by OLS.
2. Use MATLAB to simulate 50 observations of each of x_3 and ε_t and thus of x_t.
3. Using the simulated values find OLS estimates of $\boldsymbol{\beta}$
4. Estimate the covariance matrix of $\boldsymbol{\beta}$ and thus the t-statistics for a test that the coefficients of $\boldsymbol{\beta}$ are zero.
5. Estimate the standard error of the estimate of $\boldsymbol{y}$
6. Calculate the F-statistic for the significance of the regression
7. Export the data to an econometric package of your choice and verify your result.
8. See page 48 for details on replication of simulation exercises.
9. Any answers submitted should be concise and short and should contain
   (a) A copy of the m-file used in the analysis. This should contain comments to explain what is being done
   (b) A short document giving the results of one simulation and any comments on the results. You might also include the regression table from your econometric package. This document should be less than one page in length.

A sample answer follows. First the program, then the output and finally some explanatory notes. The program is a good example of a MATLAB script. If Required you should be able to cut and past it into the **MATLAB EDITOR WINDOW**

Regression Example Using Simulated Data

```
% example1.m
% Regression Example Using Simulated Data
% John C Frain
% 16 August 2014
%values for simulation
clear;
rng(1234);
nsimul=50;
beta=[5,1,.1]';
```

Regression Example(cont.)

```
%
% Step 1 Prepare and process data for X and y matrices/vectors
%
x1=ones(nsimul,1); %constant
x2=[1:nsimul]';    %trend
x3=rand(nsimul,1)*2 +3;  % Uniform(3,5)
X=[x1,x2,x3];
e=randn(nsimul,1)*.2;  % N(0,.04)
y= X * beta +e ;         %5*x1 + x2 + .1*x3 + e;
%
[nobs,nvar]=size(X);
%
% Estimate Model
%{
```

Note that I have named my estimated variables `ols.betahat`, `ols.yhat`, `ols.resid` etc. The use of the `ols.` in front of the variable name has two uses. First if I want to do two different estimate I will call the estimates `ols1.` and `ols2.` or `IV.` etc. and I can easily put the in a summary table. Secondly this structure has a meaning that is useful in a more advanced use of MATLAB.

```
%}
ols.betahat=(X'*X)\X'*y;  % Coefficients
ols.yhat = X * ols.betahat;   % beta(1)*x1-beta(2)*x2-beta(3)*x;
ols.resid = y - ols.yhat; % residuals
ols.ssr = ols.resid'*ols.resid; % Sum of Squared Residuals
ols.sigmasq = ols.ssr/(nobs-nvar); % Estimate of variance
ols.covbeta=ols.sigmasq*inv(X'*X);  % Covariance of beta
ols.sdbeta=sqrt(diag(ols.covbeta));% st. dev of beta
ols.tbeta = ols.betahat ./ ols.sdbeta; % t-statistics of beta
ym = y - mean(y);
ols.rsqr1 = ols.ssr;
ols.rsqr2 = ym'*ym;
ols.rsqr = 1.0 - ols.rsqr1/ols.rsqr2; % r-squared
ols.rsqr1 = ols.rsqr1/(nobs-nvar);
```

Regression Example(cont.)

```
ols.rsqr2 = ols.rsqr2/(nobs-1.0);
if ols.rsqr2 ~= 0;
ols.rbar = 1 - (ols.rsqr1/ols.rsqr2); % rbar-squared
else
ols.rbar = ols.rsqr;
end;
ols.ediff = ols.resid(2:nobs) - ols.resid(1:nobs-1);
ols.dw = (ols.ediff'*ols.ediff)/ols.ssr; % durbin-watson
fprintf('R-squared      = %9.4f \n',ols.rsqr);
fprintf('Rbar-squared   = %9.4f \n',ols.rbar);
fprintf('sigma^2        = %9.4f \n',ols.sigmasq);
fprintf('S.E of estimate= %9.4f \n',sqrt(ols.sigmasq));
fprintf('Durbin-Watson  = %9.4f \n',ols.dw);
fprintf('Nobs, Nvars    = %6d,%6d \n',nobs,nvar);
fprintf('*****************************************************\n \n');
% now print coefficient estimates, SE, t-statistics and probabilities
%tout = tdis_prb(tbeta,nobs-nvar); % find t-stat probabilities - no
%tdis_prb in basic MATLAB - requires leSage toolbox
%tmp = [beta sdbeta tbeta tout];  % matrix to be printed
tmp = [ols.betahat ols.sdbeta ols.tbeta];  % matrix to be printed
% column labels for printing results
namestr = ' Variable';
bstring = '    Coef.';
sdstring= 'Std. Err.';
tstring = '  t-stat.';
cnames = strvcat(namestr,bstring,sdstring, tstring);
vname = ['Constant','Trend' 'Variable2'];
%{
```

The `fprintf` is used to produce formatted output. See subsection 3.6

```
%}
fprintf('%12s %12s %12s %12s \n',namestr, ...
bstring,sdstring,tstring)
fprintf('%12s %12.6f %12.6f %12.6f \n',...
'     Const',...
```

Regression Example(cont.)

```
ols.betahat(1),ols.sdbeta(1),ols.tbeta(1))
fprintf('%12s %12.6f %12.6f %12.6f \n',...
'     Trend',...
ols.betahat(2),ols.sdbeta(2),ols.tbeta(2))
fprintf('%12s %12.6f %12.6f %12.6f \n',...
'      Var2',...
ols.betahat(3),ols.sdbeta(3),ols.tbeta(3))



%{
```

The output of this program should look like

```
R-squared      =    0.9999
Rbar-squared   =    0.9999
sigma^2        =    0.0314
S.E of estimate=    0.1773
Durbin-Watson  =    1.9492
Nobs, Nvars    =     50,     3
****************************************************

Variable        Coef.    Std. Err.      t-stat.
   Const     4.911817     0.205177    23.939427
   Trend     1.001412     0.001760   568.997306
    Var2     0.119433     0.047365     2.521525
%}
```

If you are using a recent version of MATLAB you should be able to replicate these results. If you are using an old version you may not generate the same series of random numbers and you answers may differ slightly.

Explanatory Notes

Most of your MATLAB scripts or programs will consist of three parts

1. **Get and Process data.** Read in your data and prepare vectors or matrices of

your left hand side (**y**), Right hand side (**X**) and Instrumental Variables (**Z**)

2. **Estimation** Some form of calculation(s) like $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X}^{-1})\boldsymbol{X}'\boldsymbol{y}$ implemented by a MATLAB instruction like

```
betahat = (X'*X)\X*y
```

(where `X` and `y` have been set up in the previous step) and estimate of required variances, covariances, standard errors etc.

3. **Report** Output tables and Graphs in a form suitable for inclusion in a report.

4. Run the program with a smaller number of replications (say 25) and see how the t-statistic on y_3 falls. Rerun it with a larger number of replications and see how it rises. Experiment to find how many observations are required to get a significant coefficient for y_3 often. Suggest a use of this kind of analysis.

2.4 Simulation – Sample Size and OLS Estimates

This exercise is a study of the effect of sample size on the estimates of the coefficient in an OLS regression. The x values for the regression have been generated as uniform random numbers on the interval [0,100). The residuals are simulated standardized normal random variables. The process is repeated for sample sizes of 20, 100 500 and 2500 Each simulation is repeated 10,000 times.

Simulation of Effect of Sample Size on Regression Estimate

```
% example2.m
% MATLAB Simulation Example
% John C Frain
% September 2014
%
%{
The data files x20.csv, x100.csv, x500.csv and x2500.csv
were generated  using the code below
%}
%Generate Data
rng(56789);
```

Sample Size Example(cont.)

```
x20 = 100*rand(20,1);
save('x20.csv','x20','-ASCII','-double')
x100 = 100*rand(100,1);
save('x100.csv','x100','-ASCII','-double')
x500 = 100*rand(500,1);
save('x500.csv','x500','-ASCII','-double')
x2500 = 100*rand(2500,1);
save('x2500.csv','x2500','-ASCII','-double')
%
clear
nsimul=10000;
BETA20=zeros(nsimul,1); % vector - results of simulations with 20 obs.
x=load('-ascii', 'x20.csv'); % load xdata
X=[ones(size(x,1),1),x]; % X matrix note upper case X
beta = [ 10;2];  % true values of coefficients
%
for ii = 1 : nsimul;
eps = 20.0*randn(size(X,1),1); % simulated error term
y = X * beta + eps; % y values
betahat = (X'*X)\X'*y; % estimate of beta
BETA20(ii,1)=betahat(2);
end
fprintf('Mean and st. dev of 20 obs simulation %6.3f %6.3f\n' ...
,mean(BETA20),std(BETA20))
%hist(BETA,100)

BETA100=zeros(nsimul,1);
x=load('-ascii', 'x100.csv'); % load xdata
X=[ones(size(x,1),1),x]; % X matrix note upper case X
beta = [ 10;2];  % true values of coefficients
%
for ii = 1 : nsimul;
eps = 20.0*randn(size(X,1),1); % simulated error term
```

Sample Size Example(cont.)

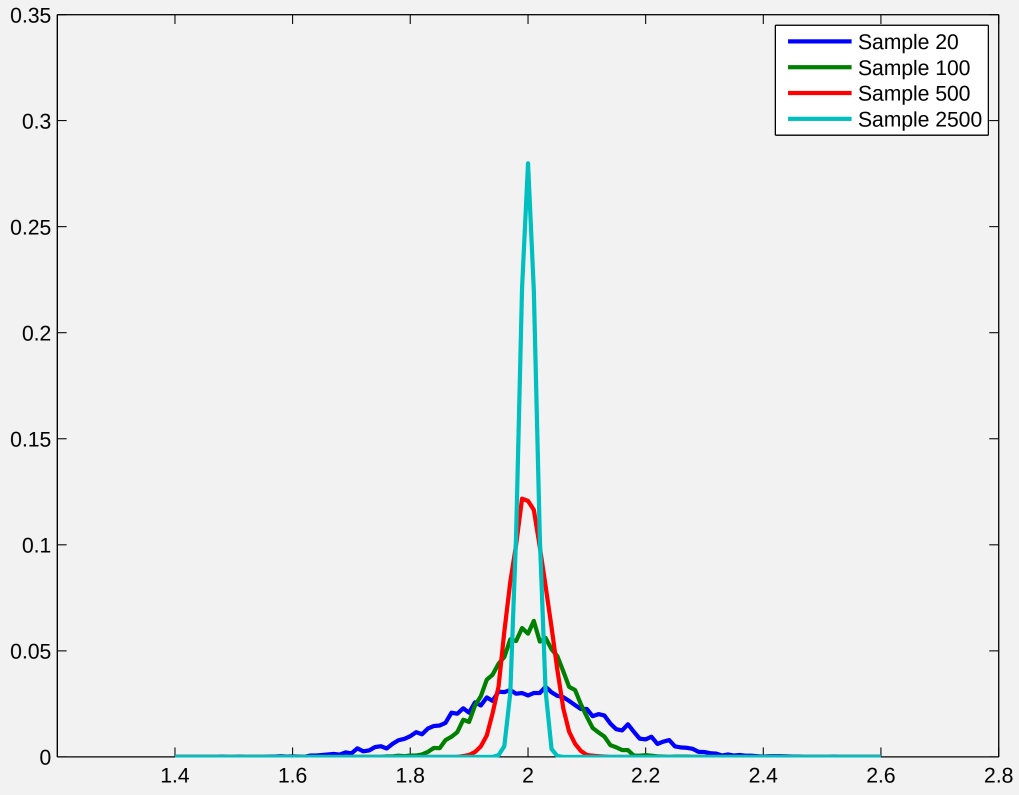
```
y = X * beta + eps; % y values
betahat = inv(X'*X)*X'*y; % estimate of beta
BETA100(ii,1)=betahat(2);
end
fprintf('Mean and st. dev of 100 obs simulation %6.3f %6.3f\n', ...
mean(BETA100),std(BETA100))

BETA500=zeros(nsimul,1);
x=load('-ascii', 'x500.csv'); % load xdata
X=[ones(size(x,1),1),x]; % X matrix note upper case X
beta = [ 10;2];  % true values of coefficients
%
for ii = 1 : nsimul;
eps = 20.0*randn(size(X,1),1); % simulated error term
y = X * beta + eps; % y values
betahat = inv(X'*X)*X'*y; % estimate of beta
BETA500(ii,1)=betahat(2);
end
fprintf('Mean and st. dev of 500 obs simulation %6.3f %6.3f\n', ...
mean(BETA500),std(BETA500))

BETA2500=zeros(nsimul,1);
x=load('-ascii', 'x2500.csv'); % load xdata note use of lower case x as vector
X=[ones(size(x,1),1),x]; % X matrix note upper case X
beta = [ 10;2];  % true values of coefficients
%
for ii = 1 : nsimul;
eps = 20.0*randn(size(X,1),1); % simulated error term
y = X * beta + eps; % y values
betahat = inv(X'*X)*X'*y; % estimate of beta
BETA2500(ii,1)=betahat(2);
end
fprintf('Mean and st. dev of 2500 obs simulation %6.3f %6.3f\n', ...
mean(BETA2500),std(BETA2500))
```

Sample Size Example(cont.)

```
n=hist([BETA20,BETA100,BETA500,BETA2500],1.4:0.01:2.6);
plot((1.4:0.01:2.6)',n/nsimul,'LineWidth',2);
h = legend('Sample 20','Sample 100','Sample 500','Sample 2500');
print -dpdf 'example2_3.pdf'
```

The output of this program will look like this. On your screen the graph will display coloured lines.

```
Mean and st. dev of 20 obs simulation  2.000  0.130
Mean and st. dev of 100 obs simulation  2.000  0.066
Mean and st. dev of 500 obs simulation  2.001  0.032
Mean and st. dev of 2500 obs simulation  2.000  0.014
```



2.5 Example – Macroeconomic Simulation with MATLAB

Simulation of Macroeconomic Model

Problem

This example is based on the macroeconomic system in Example 10.3 of Shone (2002). There are 10 equations in this economic model. The equations of the system are as follows

$$
\begin{aligned}
c_t &= 110 + 0.75yd_t \\
yd_t &= y_t - tax_t \\
tax_t &= -80 + 0.2y_t \\
i_t &= -4r_t \\
g_t &= 330 \\
e_t &= c_t + i_t + g_t \\
y_t &= e_{t-1} \\
md_t &= 20 + 0.25y_t - 10r_t \\
ms_t &= 470 \\
md_t &= ms_t
\end{aligned}
$$

While the aim in Shone (2002) is to examine the system algebraically, here we examine it numerically. Often this may be the only way to solve the system and MATLAB is a suitable tool for this work. The model is too simple to be of any particular use in macroeconomics but it does allow one to illustrate the facilities offered by MATLAB for this kind of work.

Initialise and Describe Variables

```
N = 15 ; % Number of periods for simulation
c = NaN * zeros(N,1); %real consumption
tax = NaN * zeros(N,1); %real tax
yd = NaN * zeros(N,1); %real disposible income
```

Macroeconomic Model(cont.)

```
i = NaN * zeros(N,1); % real investment
g = NaN * zeros(N,1); % real government expenditure
e = NaN * zeros(N,1); % real expenditure
y = NaN * zeros(N,1); % real income
md = NaN * zeros(N,1); % real money demand
ms = NaN * zeros(N,1); %real money supply
r = NaN * zeros(N,1); % interest rate
/{
```

Simulate

`g` and `ms` are the policy variables.

```
/}
t=(1:N)'; % time variable
g = 330 * ones(N,1);
ms = 470 * ones(N,1);
y(1) = 2000;
%{
```

The next step is to simulate the model over the required period. Here this is achieved by a simple reordering of the equations and inverting the money demand equation to give an interest rate equation. In the general case we might need a routine to solve the set of non linear equations or some routine to maximize a utility function. Note that the loop stops one short of the full period and then does the calculations for the final period (excluding the income calculation for the period beyond the end of the sample under consideration).

```
/}
for ii = 1:(N-1)
    tax(ii) = -80 + 0.2 * y(ii);
    yd(ii) = y(ii) - tax(ii);
    c(ii)  = 110 + 0.75 * yd(ii);
    md(ii) = ms(ii);
    r(ii) = (20 + 0.25* y(ii) -md(ii))/10; % inverting money demand
    i(ii) = 320 -4 * r(ii);
    e(ii) = c(ii) + i(ii) + g(ii);
```

Macroeconomic Model(cont.)

```
    y(ii+1) = e(ii);
end

tax(N) = -80 + 0.2 * y(N);
yd(N) = y(N) - tax(N);
c(N)  = 110 + 0.75 * yd(N);
md(N) = ms(N);
r(N) = (20 +  0.25* y(N) -md(N))/10;
i(N) = 320 -4 * r(N);
e(N) = c(N) + i(N) + g(N);
```

Now output results and save y for later use. note that the system is in equilibrium. Note that in printing we use the transpose of base

```
base = [t,y,yd,c,g-tax,i,r];
fprintf('      t      y     yd      c  g-tax     i       r\n')
fprintf('%7.0f%7.0f%7.0f%7.0f%7.0f%7.0f%7.0f\n',base')
ybase = y;
```

t	y	yd	c	g-tax	i	r
1	2000	1680	1370	30	300	5
2	2020	1696	1382	26	298	6
3	2030	1704	1388	24	297	6
4	2035	1708	1391	23	297	6
5	2038	1710	1393	23	296	6
6	2039	1711	1393	22	296	6
7	2039	1712	1394	22	296	6
8	2040	1712	1394	22	296	6
9	2040	1712	1394	22	296	6
10	2040	1712	1394	22	296	6
11	2040	1712	1394	22	296	6
12	2040	1712	1394	22	296	6
13	2040	1712	1394	22	296	6
14	2040	1712	1394	22	296	6
15	2040	1712	1394	22	296	6

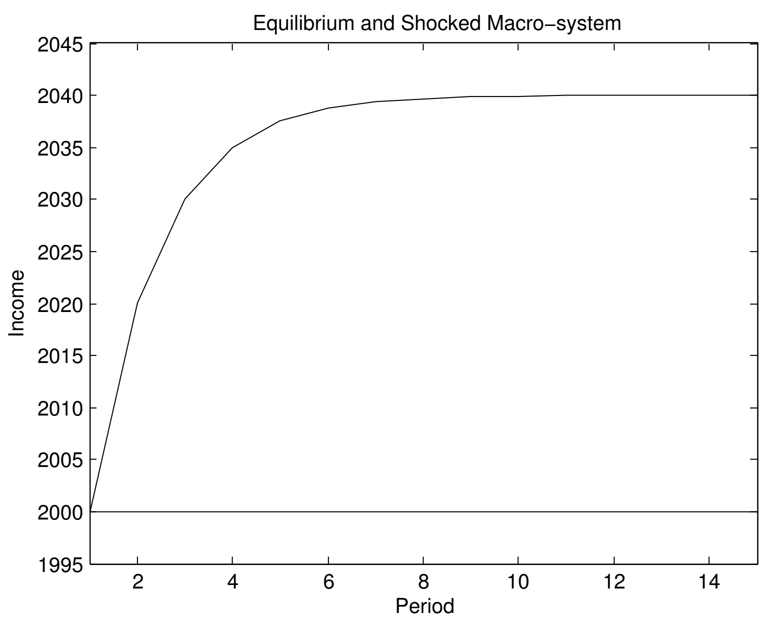
Now we compare results in a table and a graph. Note that income converges to a new limit.

Comparison of Simulations

```
fprintf('     t ybase ypolicy\n')
fprintf('%7.0f%7.0f%7.0f\n',[t,ybase, ypolicy]')

plot(t,[ybase,ypolicy])
title('Equilibrium and Shocked Macro-system')
xlabel('Period')
ylabel('Income')
axis([1 15 1995 2045])
```

```
      t  ybase ypolicy
      1   2000   2000
      2   2000   2020
      3   2000   2030
      4   2000   2035
      5   2000   2038
      6   2000   2039
      7   2000   2039
      8   2000   2040
      9   2000   2040
     10   2000   2040
     11   2000   2040
     12   2000   2040
     13   2000   2040
     14   2000   2040
     15   2000   2040
```

Comparison of Simulations (cont.)



CHAPTER 3

Data input/output

In this chapter I set out

1. How to import data from and export to an excel file.
2. How to import data from and export data to a text or comma separated value (csv) file.
3. Using native MATLAB format.
4. Formatting output
5. Producing material for inclusion in a paper

3.1 Importing from Excel format files

Figure 3.1 is the upper left-hand corner of the data file This file `g10xrate.xls`[1] contains daily observations on the US Dollar exchange rates of the G10 countries[2]. The ten

[1] This data file is embedded in this document. To extract it , please go to Appendix B. That appendix contains links to three embedded sample data files. The first is in EXCEL format, the second uses comma separated values (csv) and the third native MATLAB format.

[2] There are 11 G10 countries - Belgium, Canada, France, Germany, Italy, Japan, the Netherlands, Sweden, Switzerland, the United Kingdom and the United States. When the eleventh country joined

USXJPN	USXFRA	USXSUI	USXNLD	USXUK	···
0.331796	19.52248	26.51535	30.95017	234.72	···
0.331752	19.49508	26.57031	30.96455	234.87	···
0.332005	19.50991	26.54491	30.96263	235.03	···
0.331895	19.51981	26.55972	30.96263	235.17	···
0.330896	19.61015	26.53012	30.94251	235.1	···
0.331049	19.65486	26.57384	30.96455	235.14	···
0.331301	19.70249	26.60636	31.02988	235.25	···
0.331697	19.65602	26.58726	31.02218	235.13	···
0.331499	19.65988	26.58231	31.01352	235.18	···
0.331499	19.63017	26.56466	30.9703	235.09	···
0.331203	19.62246	26.57242	30.98277	235.09	···
0.331005	19.65757	26.59504	30.99526	235.27	···
0.330896	19.65486	26.59504	30.98757	235.31	···
0.330502	19.64984	26.60282	30.9751	235.25	···
0.3306	19.72503	26.88028	31.13519	235.35	···
0.331499	19.64984	26.80031	31.11	235.53	···
0.331005	19.69628	27.0636	31.17985	235.95	···
···	···	···	···	···	

Table 3.1: Top right-hand corner of excel file of g10 exchange rates

exchange rate series are in the ten columns columns. Each row then gives one observation of each series There are 6237 observations of each exchange rate. The name of each series is given in the first row at the head of each series. This construction is typical of many such files that are encountered in econometrics.

The MATLAB GUI provides the simplest way to import this data set. Use `|Home|Import Data|`, Navigate to file and open it. If the file is large MATLAB may take some time to load it. You will then be presented with a data import window similar to that in figure 3.1. Note that on this window you can

1. specify the range of data to be imported,
2. Specify which row contains the data names,
3. Import the data as column vectors or a single matrix
4. Specify how to deal with unimportable cells

When you have specified these items press the **|Import Selection|** button and the data are imported.

---

the original name was retained.

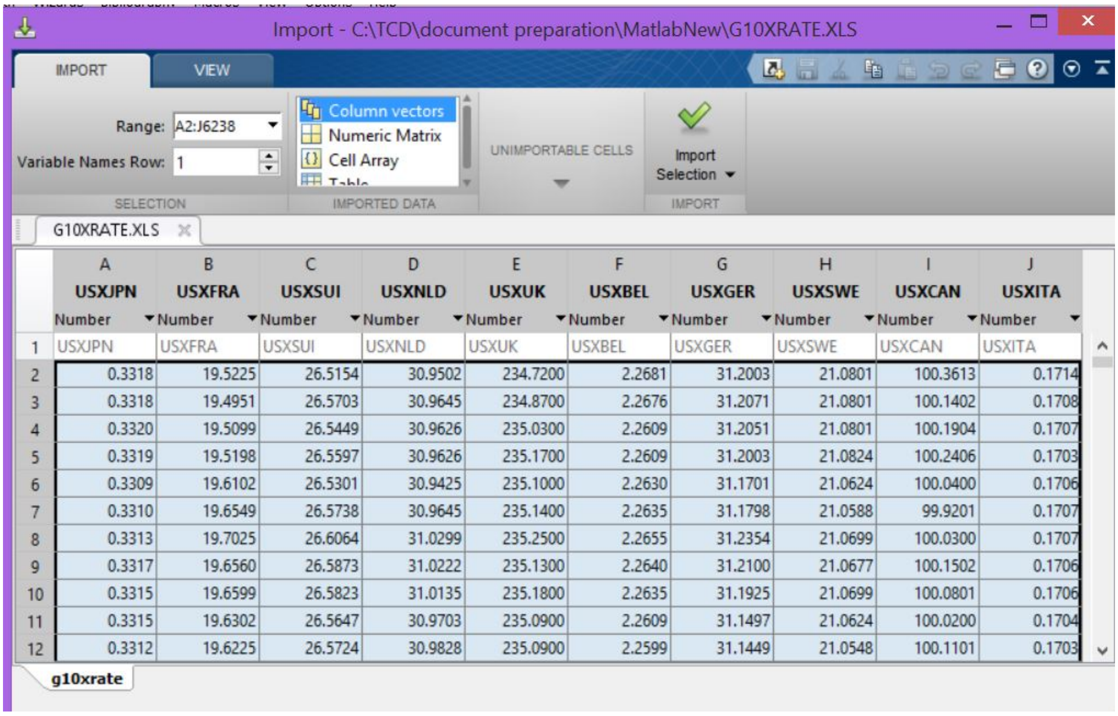


Figure 3.1: Matlab Excel Data Import Window

The dropdown menu on the |**Import Selection**| button allows one to display and save the actual MATLAB instructions that MATLAB is using to import the data. These are displayed in the box below. One could, of course, simply enter these commands in the command window or include them in a MATLAB program and produce the same result. In the interest of replication I would recommend that, if you use the GUI to import data, you also generate and save the MATLAB program file. The generated script in the current case is repeated in the box below. (Some of the commands are to long to fit on my printed line in the box and I have used continuation marks (...) to break the command and continue it on the next line.)

MATLAB Instructions to Import data file

```
%% Import data from spreadsheet
% Script for importing data from the following spreadsheet:
%
%    Workbook: C:\TCD\document preparation\MatlabNew\G10XRATE.XLS
%    Worksheet: g10xrate
```

MATLAB Instructions to Import data file (cont.)

```
%
% To extend the code for use with different selected data or a
% different spreadsheet, generate a function instead of a script.

% Auto-generated by MATLAB on 2014/09/03 23:56:37

%% Import the data
[~, ~, raw] = xlsread('C:\TCD\document preparation\MatlabNew\...
G10XRATE.XLS','g10xrate','A2:J6238');
raw(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw)) = {''};

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw);
% Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Create output variable
data = reshape([raw{:}],size(raw));

%% Allocate imported array to column variable names
USXJPN = data(:,1);
USXFRA = data(:,2);
USXSUI = data(:,3);
USXNLD = data(:,4);
USXUK = data(:,5);
USXBEL = data(:,6);
USXGER = data(:,7);
USXSWE = data(:,8);
USXCAN = data(:,9);
USXITA = data(:,10);

%% Clear temporary variables
clearvars data raw R;
```

The script uses the `xlsread` function to read the data, find non-numeric cells replace these with `NaN`, and extract and name the columns.

3.2 Reading data from text files

If your data are in comma separated value (`csv`) format the same procedure allows you to import the data. There are additional options available that allow you to specify the delimiter, combine delimiters and specify decimal point. You can also generate the script file that MATLAB used to import the data.

The MATLAB function `textscan` can read data files in various formats and and allows a greater degree of flexibility at the cost of greater complexity.

3.3 Native MATLAB data files

The instruction `save('`*`filename`*`')` saves the contents of the workspace in the file `'`*`filename`*`.mat'`. `save` used in the default manner saves the data in a binary format specific to MATLAB. The instruction `save('`*`filename`*`', '`*`var1`*`','`*`var2`*`')` saves *var1* and *var2* in the file '*`filename.mat`*'. Similarly the commands `load'(`*`filename`*`'` and `load('`*`filename`*`','`*`var1`*`', '`*`var2`*`')`. load the contents of '*filename.mat*' or the specified variables from the file into the workspace. In general `.mat` files are nor easily readable in most other software. They are ideal for use within MATLAB and for exchange between MATLAB users. (note that there may be some incompatibilities between different versions of MATLAB). These `.mat` files are binary and can not be examined in a text editor.

`.mat` is the default extension for a MATLAB data file. If you use another extension, say `.ext` you must specify the extension when using save or load functions. It is possible to use `save` and `load` to save and load text files but these instructions are very limited. If your data are in EXCEL or csv format the methods described above are better.

3.4 Exporting data to EXCEL and econometric/statistical packages

The command `xlswrite('`*filename*`',`**M**`)` writes the matrix **M** to an Excel format file *filename* in the current working directory. If **M** is $n \times m$ the numeric values in the matrix are written to the first n row and m columns in the first sheet in the spreadsheet. The command `csvwrite('`*filename*`',`**M**`)` writes the matrix **M** to the file *filename* in the current working directory.

Data in Excel or csv format can be read in most econometric/statistical packages.

The package `R.matlab` allows the R statistical system (R Core Team (2014))to read native MATLAB files. R is capable of outputting data in formats native to or consistent with many statistical/econometric packages.

3.5 Stat/Transfer

Another alternative is to use the Stat/Transfer package which allows the transfer of data files between a large number of statistical packages.

3.6 Formatted Output

The MATLAB function `fprintf()` may be used to produce formatted output on screen[3]. The following MATLAB program gives an example of the us of the `fprintf()` function.

Sample MATLAB program demonstrating Formatted Output

```
clear
degrees_c =10:10:100;
degrees_f = (degrees_c * 9 /5) + 32;
```

[3] `fprintf()` is only one of a large number of C-style input/output functions in C. These allow considerable flexibility in sending formatted material to the screen of to a file. The MATLAB help files give details of the facilities available. If further information is required one might consult a standard book on the C programming language

Sample MATLAB program demonstrating Formatted Output (cont.)

```
fprintf('\n\n Conversion from degrees Celsius \n');
fprintf('     to degrees Fahrenheit\n\n' );
fprintf('     Celsius  Fahrenheit\n');
for ii = 1:10;
    fprintf('%12.2f%12.2f\n',degrees_c(ii),degrees_f(ii));
end
%
fprintf(...
'\n\n%5.2f degrees Celsius is equivalent of %5.3f degrees Fahrenheit\n', ...
    degrees_c(1),degrees_f(1))
```

Output of Sample MATLAB program demonstrating Formatted Output

```

 Conversion from degrees Celsius
     to degrees Fahrenheit

     Celsius  Fahrenheit
       10.00       50.00
       20.00       68.00
       30.00       86.00
       40.00      104.00
       50.00      122.00
       60.00      140.00
       70.00      158.00
       80.00      176.00
       90.00      194.00
      100.00      212.00


10.00 degrees Celsius is equivalent of 50.000 degrees Fahrenheit
```

Note the following

- The first argument of the `fprintf()` function is a kind of format statement included within ' marks.

- The remaining arguments are a list of variables or items to be printed separated by commas
- Within the format string there is text which is produced exactly as set down. There are also statements like `%m.nf` which produces a decimal number which is allowed m columns of output and has n places of decimals. These are applied in turn to the items in the list to be printed.
- This `f` format is used to output floating point numbers there are a considerable number or other specifiers to output characters, strings, and number in formats other than floating point.
- If the list to be printed is too long the formats are recycled.
- Note the use of `\n` which means skip to the next line. This is essential.

I have already made some use of the `fprintf()` function to format the output of several examples in Chapter 2

A small amendment to this program allows one to print formatted output to a file.

Sample MATLAB program demonstrating Formatted Output

```
clear
fid = fopen('test01.txt','w');
degrees_c =10:10:100;
degrees_f = (degrees_c * 9 /5) + 32;
fprintf(fid,'\n\n Conversion from degrees Celsius \n');
fprintf(fid,'     to degrees Fahrenheit\n\n' );
fprintf(fid,'     Celsius  Fahrenheit\n');
for ii = 1:10;
fprintf(fid,'%12.2f%12.2f\n',degrees_c(ii),degrees_f(ii));
end
%
fprintf(fid,...
'\n\n%5.2f degrees Celsius is equivalent of %5.3f degrees Fahrenheit\n', ...
degrees_c(1),degrees_f(1));
fclose(fid);
```

There changes in this program are

1. The instruction `fid = fopen('test01.txt','w');` opens the file `test01.txt` for writing. The variable `fid` specifies the output file. If the file exists it will be overwritten. Substitute 'a' for 'w' if you wish to open a file and append output.
2. The variable `fid` has been added as a first argument to each call of the `fprintf` function.
3. The instruction `fclose(fid);` closes the file.
4. The output is the same as the previous case except that it is written to file rather than the command window,

3.7 Producing material for inclusion in a paper

In the MATLAB editor there is a tab labelled **PUBLISH**. The facilities under this tab allow you to produces output in WORD, Powerpoint, LaTeX , HTML etc. for inclusion in papers, presentations etc. This facility was used to produce some of the material in this set of notes. The facilities are described in the help files and may vary from version to version of MATLAB[4].

Section 2.5, including its graph, was subject to very minor editing before being added to this document. I have also used the facility to produce transparencies for lectures on the use of MATLAB in economics/econometrics.

The next box contains a sample MATLAB script which has been set up to prepare material for publication in LaTeX. Note the following

1. The script has been divided into sections by inserting two percent signs (%%) at the beginning of each section.
2. Next to each of these %% markers there is a section title
3. Add MATLAB comments to the script. These will provide a commentary in your finished document.
4. When writing your commentary there are facilities for
   (a) Bold, italic and mono-spaced text

[4]In older versions of MATLAB to access these facilities you had to first turn them on in the MATLAB Editor. This was done by **|Cell|Enable Cell Mode|** in the editor menu. Cell mode enabled you to divide your m-file into cells or sections. Cell mode in the editor was not to be confused with cell data structures in MATLAB. Enabling cell mode added a new set of buttons to the menu bar and enabled a set of items in the cell menu item. Many of the facilities described here were available but their implementation was different.

(b) Hyperlinks
(c) Inline LaTeX
(d) Bulleted/Numbered Lists
(e) Images
(f) Pre formatted text
(g) code
(h) Display LaTeX

Sample Publish routine in MATLAB

```
%% An example of publishing from matlab

%% The first section
% very simple
x=randn(15);
xbar = mean(x);
xsd = std(x);
% x has mean
xbar
% and standard deviation
xsd

%% An Equation
%
%%
%
% $$e^{\pi i} + 1 = 0$$
%

%% A bulleted list
%
%%
%
% * First bullet
% * second bullet
%
```

Sample Publish routine in MATLAB (cont.)

The next box displays the LaTeX code produced by this MATLAB script.

LaTeX code produced by Sample MATLAB Publish Routine

```
 This LaTeX was auto-generated from MATLAB code.
 To make changes, update the MATLAB code and republish this document.

\documentclass{article}
\usepackage{graphicx}
\usepackage{color}

\sloppy
\definecolor{lightgray}{gray}{0.5}
\setlength{\parindent}{0pt}

\begin{document}

\section*{An example of publishing from matlab}

\subsection*{Contents}

  \begin{itemize}
        \setlength{\itemsep}{-1ex}
          \item The first section
          \item An Equation
          \item A bulleted list
        \end{itemize}

\subsection*{The first section}
\begin{par}
        very simple
\end{par} \vspace{1em}
```

LaTeX code produced by Sample MATLAB Publish Routine (cont.)

```
\begin{verbatim}
x=randn(15);
xbar = mean(x);
xsd = std(x);
% x has mean
xbar
% and standard deviation
xsd
\end{verbatim}

\color{lightgray} \begin{verbatim}
xbar =

Columns 1 through 7

0.6203    0.4835   -0.3327    0.1027    0.0897    0.0213   -0.1768

Columns 8 through 14

-0.0929    0.0546   -0.4624    0.1530   -0.3130   -0.0270    0.2205

Column 15

-0.0829


xsd =

Columns 1 through 7

1.6560    0.8653    1.1796    0.8674    1.0939    0.9352    1.1211

Columns 8 through 14
```

LaTeX code produced by Sample MATLAB Publish Routine (cont.)

```
1.2128    1.2542    0.9056    1.0532    0.9412    0.7429    0.6715

Column 15

1.0677

\end{verbatim} \color{black}



\subsection*{An Equation}

        \begin{par}
                $$e^{\pi i} + 1 = 0$$
        \end{par} \vspace{1em}



\subsection*{A bulleted list}

        \begin{itemize}
                \setlength{\itemsep}{-1ex}
                \item First bullet
                \item second bullet
        \end{itemize}

\end{document}
```

The actual output produced by this LaTeX code is reproduced in figures 3.2 and 3.3. I have chosen LaTeX for this example as I am using LaTeX to type-set it. Other options can be configured by clicking on the **publish** icon under the **PUBLISH** tab and selecting **Edit Publishing Options**. There is another example of the use of these facilities in the MATLAB help files — search for ”Publishing MATLAB Code”.

An example of publishing from matlab

Contents

- The first section
- An Equation
- A bulleted list

The first section

very simple

```
x=randn(15);
xbar = mean(x);
xsd = std(x);
% x has mean
xbar
% and standard deviation
xsd
```

```
xbar =

  Columns 1 through 7

    0.6203    0.4835   -0.3327    0.1027    0.0897    0.0213   -0.1768

  Columns 8 through 14

   -0.0929    0.0546   -0.4624    0.1530   -0.3130   -0.0270    0.2205

  Column 15

   -0.0829


xsd =

  Columns 1 through 7

    1.6560    0.8653    1.1796    0.8674    1.0939    0.9352    1.1211

  Columns 8 through 14

    1.2128    1.2542    0.9056    1.0532    0.9412    0.7429    0.6715

  Column 15
```

1

Figure 3.2: First page of LaTeX output from Matlab generated script


1.0677

An Equation

$$e^{\pi i} + 1 = 0$$

A bulleted list

- First bullet
- second bullet

2

Figure 3.3: Second page of LaTeX output from Matlab generated script

CHAPTER 4

Decision and Loop Structures.

In composing MATLAB scripts, programs and functions you will often need to run a command or group of commands if certain conditions hold. On other occasions you may need to run the same command or group of commands a number of times.

In MATLAB there are four basic control (Decision or Loop Structures) available

if statements The basic form of the `if` statement is

```
if conditions
    statements
end
```

The `statements` are only processed if the conditions are true. The conditions can include the following operators

==	equal
~=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
&	logical and
&&	logical and (for scalars) short-circuiting
|	logical or
||	logical or and (for scalars) short-circuiting
xor	logical exclusive or
all	true if all elements of vector are nonzero
any	true if any element of vector is nonzero

The `if` statement may be extended

```
if conditions
    statements1
else
    statements2
end
```

in which case statements1 are used if conditions are true and statements2 if false. This `if` statement may be extended again

```
if conditions1
    statements1
elseif conditions2
    statements2
else
    statements3
end
```

with an obvious meaning (I hope).

for The basic form of the for group is

```
for variable = expression
    statements
end
```

Here `expression` is probably a vector. `statements` is processed for each of the values in expression. The following example shows the use of a loop within a loop

xxx

```
>> for ii = 1:3
for jj=1:3
total=ii+jj;
fprintf('%d + %d = %d \n',ii,jj,total)
end
end
```

```
1 + 1 = 2
1 + 2 = 3
1 + 3 = 4
2 + 1 = 3
2 + 2 = 4
2 + 3 = 5
3 + 1 = 4
3 + 2 = 5
3 + 3 = 6
```

while The format of the `while` statement is

```
while conditions
    statements
end
```

The `while` statement has the same basic functionality as the `for` statement. The `for` statement will be used when one knows precisely when and how many times an operation will be repeated. The statements are repeated so long as conditions are true

switch An example of the use of the `switch` statement follows

```
switch p
    case 1
        x = 24
    case 2
        x = 19
    case 3
        x = 15
    otherwise
        error('p must be 1, 2 or 3')
end
```

Use matrix statements in preference to loops. Not only are they more efficient but they are generally easier to use. That said there are occasions where one can not use a matrix statement.

If you wish to fill the elements of a vector or matrix using a loop it is good practice to initialise the vector or matrix first. For example if you wish to fill a 100×20 matrix, $\boldsymbol{X}$, using a series of loops one could initialise the matrix using one of the following commands

```
X = ones(100,20)
X = zeros(100,20)
X = ones(100,20)*NaN
X = NaN(100,20)
```

and then evaluate the elements of the matrix within the loop or other control structure.

CHAPTER 5

Elementary Plots

Simple graphs can be produced easily in MATLAB The following sequence

Script to graph regression residuals

```
%values for simulation
rng(1234); %same random numbers each time script is run
nsimul=50;
beta=[5,1,.1]';
%
x1=ones(nsimul,1); %constant
x2=[1:nsimul]';    %trend
x3=rand(nsimul,1)*2 +3;  % Uniform(3,5)
x=[x1,x2,x3];
e=randn(nsimul,1)*.2;  % N(0,.04)
y= x * beta +e ;         %5*x1 + x2 + .1*x3 + e;
%
[nobs,nvar]=size(x);
betahat=inv(x'*x)*x'*y; %g
```

Script to graph regression residuals (cont.)

```
yhat = x * betahat;   % beta(1)*x1-beta(2)*x2-beta(3)*x;
resid = y - yhat;
plot(x2,resid)
title('Graph Title')
xlabel('Time')
ylabel('Residual')
print -dpdf 'test.pdf'
```

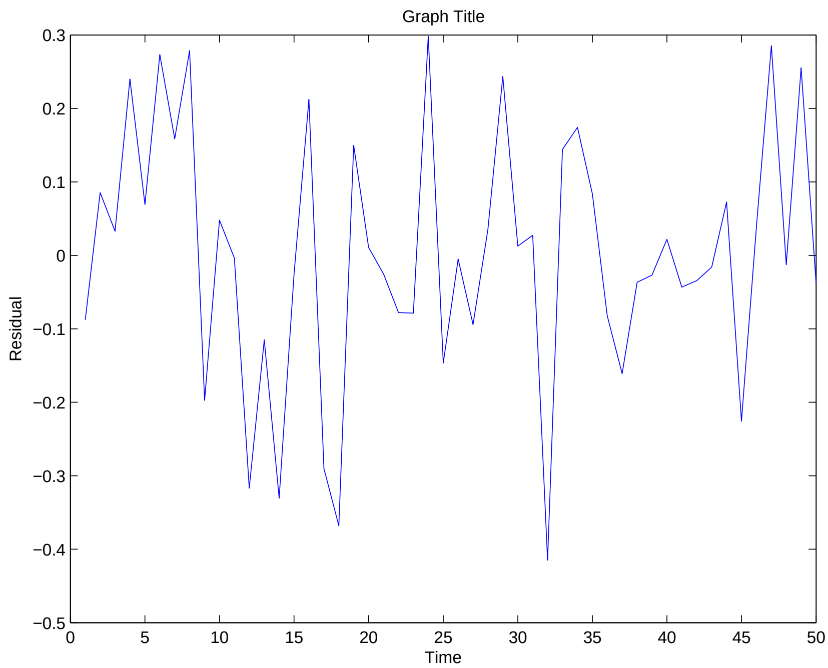


Figure 5.1: Residuals from a simulated regression

repeats the earlier OLS simulation (see page 57), opens a graph window, draws a graph of the residuals against the trend in the ols-simulation exercise, puts a title on the graph and labels the x and y axes. The vectors x2 and resid must have the same dimensions. The graph is then saved in pdf format and then imported into this document.

The following addition shows how to follow the previous plot of residuals by a plot of actual versus fitted values. the first command `figure;` opens a new graphics window. This is followed by two `plot` commands — one each for the actual and fitted values.

```
figure; % Command to open a new graph window
plot(y,x2); % plots actual
plot(yhat,x2); % plots fit on same graph
% more graphics commands for this graph
```

The **PLOTS** tab provides an easy way to generate a variety of Graphs. The following box illustrates the use of this tab to generate a semi log graph of Irish GDP over time. The file `NationalIncome.csv`[1] contains two columns, year and NI (National Income at constant Prices) downloaded from `www.cso.ie` on 13 October 2014.

1. I use the Import Data to generate a program to import the data. (see Chapter 3).
2. Now select the data in `year` and `NI`. Select the **PLOTS** tab and click on the **semilogy** item in the ribbon. The graph will be generated in a new window.
3. you can now embellish the graph with various features using the `insert` item on the menu of the graphics box.
4. You can now save the graph in a variety of formats using the **|File|Save|** item from the menu bar on the graph window.
5. You may also use the **|File|Generate Code...|** menu to generate a function which produces this graph. Save the function in the working directory. The generated file is displayed in the box below. You may need to refer to Chapter 7 for details of user generated functions in MATLAB.

Graph Code Generated by app

```
function createfigure(X1, Y1)
%CREATEFIGURE(X1, Y1)
%  X1:  vector of x data
%  Y1:  vector of y data

%  Auto-generated by MATLAB on 13-Oct-2014 21:53:10

% Create figure
figure1 = figure;

% Create axes
```

[1]This file is embedded in this document. To extract it, please go to appendix B

Graph Code Generated by app (cont.)

```
axes1 = axes('Parent',figure1,'YScale','log','YMinorTick','on');
box(axes1,'on');
hold(axes1,'all');

% Create semilogy
semilogy(X1,Y1);

% Create xlabel
xlabel({'Year'});

% Create ylabel
ylabel({'National','Income'});

% Create title
title({'National Income at Constant Prices'});
```

The next box contains a script which brings all these details together —

1. Load the data
2. Draw the Graph
3. Save the Graph

Generated Graph Using App

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\TCD\document
%    preparation\MatlabNew\TeX\Chapter05\NationalIncomeReal.csv
%
% To extend the code to different selected data or a different text
% file, generate a function instead of a script.

% Auto-generated by MATLAB on 2014/10/13 21:47:37
```

Generated Graph Using App (cont.)

```
%% Initialize variables.
filename = 'C:\TCD\document preparation...
\MatlabNew\TeX\Chapter05\NationalIncomeReal.csv';
delimiter = ',';
startRow = 2;

%% Format string for each line of text:
%   column1: double (%f)
%        column2: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the
% code from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, ...
'HeaderLines' ,startRow-1, 'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate
% the script.

%% Allocate imported array to column variable names
year1 = dataArray{:, 1};
```

Generated Graph Using App (cont.)

```
NI = dataArray{:, 2};

%% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;
createfigure(year1, NI)
print -dpdf 'NationalIncomeReal.pdf'
```

The graph produced by this script is reproduced in figure 5.2.

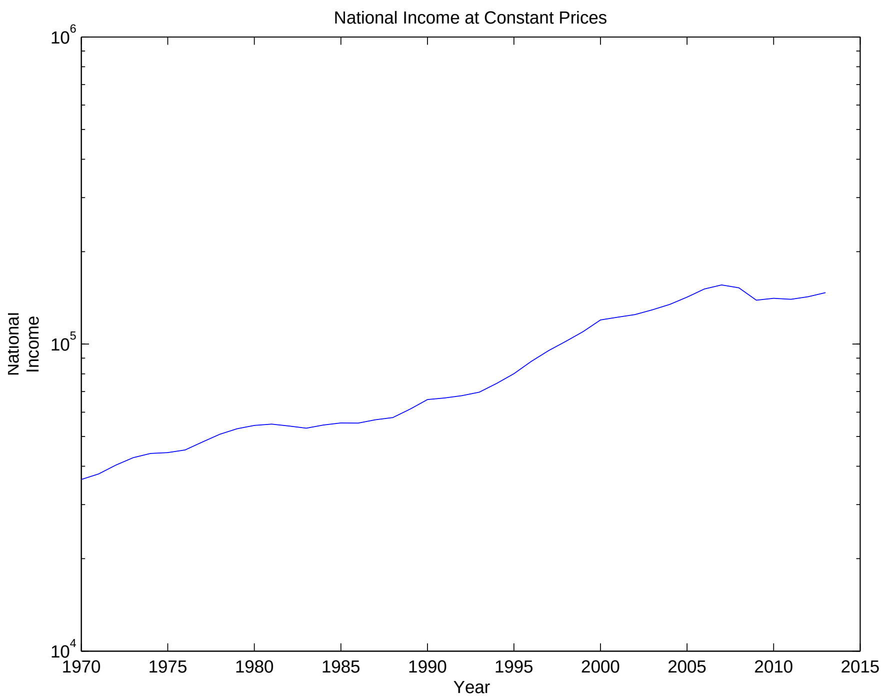


Figure 5.2: Graph produced by script

This chapter does not do justice to the graphics facilities available in MATLAB. The MATLAB graphics manual is currently just short of 600 pages. Many other graphs can be drawn using the **PLOT** tab in the GUI. Several of the additional MATLAB toolboxes that an economist might use also contain additional specialized graphics facilities.

CHAPTER 6

Systems of Regression Equations

6.1 Using MATLAB to estimate systems of regression equations

This section contains two examples of the estimation of systems of equations. The first is an examination of the classic Grunfeld investment data set. Many textbooks use this dataset to illustrate various features of system estimation. Green (2000) is the source of the data used here. Later editions of this book also examine these data but in less detail.

The MATLAB output also includes corresponding analysis using the le Sage econometrics package which is covered in section 8.2 of this note. As an exercise the user might extend the analysis to include various Likelihood Ratio tests of the restrictions imposed by the various estimation procedures.

Analysis of Grunfeld Investment data

Introduction

The basic system model that we are looking at here is

$$y_{ti} = \boldsymbol{X}_{ti}\boldsymbol{\beta}_i + \varepsilon_{ti}$$

where $1 \leqslant i \leqslant M$ represents the individual agent or country or item for which we are estimating some equation and $1 \leqslant t \leqslant T$ represents the t^{th} measurement on the i^{th} unit. We assume that the variance of ε_{ti}, σ_i^2 is constant for $1 \leqslant t \leqslant T$. Each $\boldsymbol{X}_i$ is $T \times k_i$. We may write these equations

$$\begin{aligned}
\boldsymbol{y}_1 &= \boldsymbol{X}_1\boldsymbol{\beta}_1 + \boldsymbol{\varepsilon_1} \\
\boldsymbol{y}_2 &= \boldsymbol{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon_2} \\
\ldots \\
\boldsymbol{y}_M &= \boldsymbol{X}_M\boldsymbol{\beta}_M + \boldsymbol{\varepsilon_M}
\end{aligned}$$

In this section we will assume that $\boldsymbol{X}_i$ is exogenous for all $1 \leqslant i \leqslant M$. By imposing various cross-equation restrictions on the $\boldsymbol{\beta}_i$ and the covariances of the ε_{ti} we obtain a variety of estimators (e. g. Pooled OLS, Equation by Equation OLS, SUR).

The variables included in the Grunfeld analysis are

- FIRM : There are 10 firms
- YEAR : Data are from 1935 to 1954 (20 years)
- I : Gross Investment
- F : Value of Firm
- C : Stock of Plant and Equipment

For more details see Green (2000, 2012) or the original references listed there.

- To reduce the amount of detail we shall restrict analysis to 5 firms
- Firm no 1 : GM - General Motors

- Firm no 4 : GE - General Electric
- Firm no 3 : CH - Chrysler
- Firm no 8 : WE - Westinghouse
- Firm no 2 : US - US Steel

The file `Grunfeld.xlsx` was generated from the Green data and transformed to `xlsx` format by importing the original csv data file into **OpenOffice CALC** and exporting it in the required format. While the script shows the command to import the data from the excel file the user may prefer to import it using the GUI.

The data set contains five columns. Row one contains variable names. Data for each firm is then given in 20 10 blocks each of 20 columns. The variable names on the data set are not imported as I wish to define the variables myself. The data is imported as a matrix which I have named `data`.

```
load data; %
```

to reload and process the data

Load and Generate Data

```
data = xlsread('Grunfeld.xlsx','A2:E201');
Y_GM = data(1:20, 3); % I
X_GM = [ones(20,1),data(1:20,[4 5])]; % constant F C
Y_GE = data(61:80, 3); % I
X_GE = [ones(20,1),data(61:80,[4 5])]; % constant F C
Y_CH = data(41:60, 3); % I
X_CH = [ones(20,1),data(41:60,[4 5])]; % constant F C
Y_WE = data(141:160, 3); % I
X_WE = [ones(20,1),data(141:160,[4 5])]; % constant F C
Y_US = data(21:40, 3); % I
X_US = [ones(20,1),data(21:40,[4 5])]; % constant F C
```

We now estimate the coefficients imposing various restrictions. Each estimation involves the following steps

1. Set up the required stacked $\boldsymbol{y}$ and $\boldsymbol{X}$ matrices.
2. Estimate the required coefficients.

3. Estimate standard errors, t-statistics etc.
4. Report.

6.1.1 Pooled OLS

The restrictions imposed by Pooled OLS are that corresponding coefficients are the same across equations. We also assume that the variance of the disturbances is constant across equations.[1] Thus, in this case $k_i = k$, for all i We can therefore assume that each observation on each unit is one more observation from the same single equation system. We may write the system as follows

$$\underset{(MT \times 1)}{\begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \cdots \\ \boldsymbol{y}_M \end{pmatrix}} = \underset{(MT \times k)}{\begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \cdots \\ \boldsymbol{X}_M \end{pmatrix}} \quad \underset{(k \times 1)}{\boldsymbol{\beta}} \quad + \quad \underset{(MT \times 1)}{\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \cdots \\ \varepsilon_M \end{pmatrix}}$$

or, more compactly, using the obvious notation

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \varepsilon$$

and $\boldsymbol{\beta}$ may be estimated by $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}$ etc. This is implemented in MATLAB as follows –

Pooled OLS

```
Y = [Y_GM', Y_GE', Y_CH', Y_WE', Y_US']'; % leave out ; for testing if
% necessary delete during run or output will be unreadable
X = [X_GM', X_GE', X_CH', X_WE', X_US']';

pols.beta = (X'*X)\X'*Y;
pols.uhat = Y - X*pols.beta ;
pols.sigsq = (pols.uhat'*pols.uhat)/(size(X,1)-size(X,2));%(T-k)
pols.sdbeta  = sqrt(diag(inv(X'*X))*pols.sigsq);
pols.tbeta = pols.beta ./ pols.sdbeta;
```

[1] We could of course relax this condition and estimate Heteroskedastic Consistent Standard Errors

Pooled OLS (cont.)

```
pols.se = sqrt(pols.sigsq);
label = ['Constant  '; 'F         '; 'C         '];
disp('OLS Results using stacked matrices')
disp('                coef        sd    t-stat')
for ii=1:size(X,2)
fprintf('%s%10.4f%10.4f%10.4f\n',label(ii,:),pols.beta(ii),pols.sdbeta(ii), pols.tbeta(
end
fprintf('Estimated Standard Error %10.4f\n\n\n',pols.se)

OLS Results using stacked matrices
                coef        sd    t-stat
Constant    -47.0237   21.6292   -2.1741
F             0.1059    0.0114    9.2497
C             0.3014    0.0437    6.8915
Estimated Standard Error   128.1429
%
% Verification using Lesage package
%
pooled = ols(Y, X);
vnames= ['I         ';
         'Constant  ';
         'F         ';
         'C         '];
prt(pooled,vnames)

Ordinary Least-squares Estimates
Dependent Variable =       I
R-squared      =    0.7762
Rbar-squared   =    0.7716
sigma^2        = 16420.6075
Durbin-Watson  =    0.3533
Nobs, Nvars    =    100,     3
***************************************************************
```

Pooled OLS (cont.)

Variable	Coefficient	t-statistic	t-probability
Constant	-47.023691	-2.174080	0.032132
F	0.105885	9.249659	0.000000
C	0.301385	6.891475	0.000000

6.1.2 Equation by equation OLS

This section assumes that the coefficients vary across units. (In panel data estimation we assume that only the constant terms vary across units). We also assume that there is no contemporaneous correlation between the disturbances in the system. We may write the system of equations as

$$\begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_M \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{X}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{X}_M \end{pmatrix} \boldsymbol{\beta} + \begin{pmatrix} \boldsymbol{\varepsilon}_1 \\ \boldsymbol{\varepsilon}_2 \\ \vdots \\ \boldsymbol{\varepsilon}_M \end{pmatrix}$$

or more compactly using the obvious substitutions

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{y}$ and $\boldsymbol{\varepsilon}$ are $TM \times 1$, $\boldsymbol{X}$ is $TM \times kM$ and $\boldsymbol{\beta}$ is $kM \times 1$. $\boldsymbol{y}$, $\boldsymbol{\varepsilon}$, and $\boldsymbol{\beta}$ are stacked versions of $\boldsymbol{y}_i$, $\boldsymbol{\varepsilon}_i$, and $\boldsymbol{\beta}_i$. The variance of $\boldsymbol{\varepsilon}$ is given by

$$\begin{aligned} \Omega &= E\left[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}'\right] \\ &= \begin{pmatrix} \sigma_1^2 \boldsymbol{I}_T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \sigma_2^2 \boldsymbol{I}_T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \sigma_M^2 \boldsymbol{I}_T \end{pmatrix} \\ &= \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_M^2 \end{pmatrix} \otimes \boldsymbol{I}_T \end{aligned}$$

The coding of this example should be relatively clear. Perhaps the most difficult part is the estimation of the variances. The procedure here is very similar to that used in the first step of the SUR estimation procedure except that the contemporaneous correlation is used to improve the estimates. It should be noted that, in this case different variables are likely to be used in different equations, The only change required is in the calculation of the $\boldsymbol{X}$ matrix.

Equation by Equation OLS

```
% Y as before
X=blkdiag(X_GM ,X_GE , X_CH , X_WE , X_US);
eqols.beta = (X'*X)\X'*Y;
eqols.uhat = Y - X*eqols.beta ;
eqols.temp = reshape(eqols.uhat,size(X_GM,1),5); %residuals for
                                                % each firm in a column
eqols.sigsq1 =eqols.temp'*eqols.temp/(size(X_GM,1)-size(X_GM,2));
eqols.sigsq = diag(diag(eqols.sigsq1)); % Remove non-diagonal elements
%eqols.sdbeta = sqrt(diag(inv(X'*X)*X'*kron(eye(size(X_GM,1)),eqols.sigsq)*X*inv(X'*X))
eqols.covarbeta = inv(X'*X)*kron(eqols.sigsq,eye(3));
eqols.sdbeta = diag(sqrt(eqols.covarbeta));
eqols.tbeta=eqols.beta ./ eqols.sdbeta;
eqols.se=sqrt(diag(eqols.sigsq));
%
% Write results
%
disp('OLS equation by equation using stacked matrices')
disp('OLS estimates GE equation')

firm = ['GE';
        'GM';
        'CH';
        'wE';
        'US'];
for jj = 1:5 % Loop over firms
    fprintf('\n\n\n')
    disp('                coef        sd    t-stat')
```

Equation by Equation OLS (cont.)

```
    for ii=1:3 %Loop over coefficients
        fprintf('%10s%10.4f%10.4f%10.4f\n',label(ii), ...
            eqols.beta(ii+(jj-1)*3),eqols.sdbeta(ii+(jj-1)*3), ...
            eqols.tbeta(ii+(jj-1)*3))
    end
    fprintf('Standard Error is %10.4f\n',eqols.se(jj));
end

OLS equation by equation using stacked matrices
OLS estimates GE equation



                coef        sd    t-stat
         C -149.7825  105.8421   -1.4151
         F    0.1193    0.0258    4.6172
         C    0.3714    0.0371   10.0193
Standard Error is    91.7817



                coef        sd    t-stat
         C   -6.1900   13.5065   -0.4583
         F    0.0779    0.0200    3.9026
         C    0.3157    0.0288   10.9574
Standard Error is    13.2786



                coef        sd    t-stat
         C   -9.9563   31.3742   -0.3173
         F    0.0266    0.0156    1.7057
         C    0.1517    0.0257    5.9015
Standard Error is    27.8827
```

Equation by Equation OLS (cont.)

```
                coef         sd    t-stat
         C   -0.5094    8.0153   -0.0636
         F    0.0529    0.0157    3.3677
         C    0.0924    0.0561    1.6472
Standard Error is    10.2131



                coef         sd    t-stat
         C  -49.1983  148.0754   -0.3323
         F    0.1749    0.0742    2.3566
         C    0.3896    0.1424    2.7369
Standard Error is    96.4345

% % Verify using le Sage Toolbox

olsestim=ols(Y_US,X_US);
prt(olsestim, vnames);

Ordinary Least-squares Estimates
Dependent Variable =       I
R-squared      =    0.4709
Rbar-squared   =    0.4086
sigma^2        = 9299.6040
Durbin-Watson  =    0.9456
Nobs, Nvars    =     20,     3
***************************************************************
Variable        Coefficient      t-statistic    t-probability
Constant         -49.198322        -0.332252         0.743761
F                  0.174856         2.356612         0.030699
C                  0.389642         2.736886         0.014049
```

Equation by Equation OLS (cont.)

6.1.3 SUR Estimates

Suppose that we have a random sample of households and we are have time series data on expenditure on holidays (y_{it}) and relevant explanatory variables. Suppose that we have sufficient data to estimate a single equation for each person in the sample. We also assume that there is no autocorrelation in each equation (often a rather heroic assumption). During the peak of the business cycle it is likely that many of the persons in the sample spend above what they do at the trough. Thus it is likely that there will be contemporaneous correlation between the errors in the system.

$$E[\varepsilon_{ti}\varepsilon_{sj}] = \begin{cases} \sigma_{ij} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Thus we may write the contemporaneous covariance matrix ($\boldsymbol{\Sigma}$) as

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1M} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \cdots & \sigma_{MM} \end{pmatrix}$$

The total covariance matrix is, in this case, given by

$$\begin{aligned} \Omega &= \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Sigma} \end{pmatrix} \\ &= \boldsymbol{\Sigma} \otimes \boldsymbol{I}_T \end{aligned}$$

If $\boldsymbol{\Omega}$ were known we would use GLS to get optimum estimates of $\boldsymbol{\beta}$. In this case we can obtain a consistent estimate of $\boldsymbol{\Sigma}$ from the residuals in the equation by equation OLS estimate that we have just completed. We can then use this consistent estimate in Feasible GLS.

SUR Estimates

```
Omega = kron(eqols.sigsq1,eye(20,20)); % Page page 256
eqsur.beta= inv(X'*inv(Omega)*X)*X'*inv(Omega)*Y;
eqsur.yhat = X * eqsur.beta;
eqsur.uhat = Y - eqsur.yhat;
eqsur.temp = reshape(eqsur.uhat,20,5);
eqsur.omega = eqsur.temp' * eqsur.temp /size(X_GM,1); %(size(X_GM,1)-size(X_GM,2));
eqsur.covar = inv(X'*inv(kron(eqsur.omega, eye(20)))*X);
eqsur.sdbeta = sqrt(diag(eqsur.covar));
eqsur.tbeta = eqsur.beta ./ eqsur.sdbeta;
eqsur.se = sqrt(diag(eqsur.omega));
%print results
fprintf('SUR estimates\n');
for jj = 1:5 % Loop over firms
    fprintf('\n\n\n')
    disp('                coef        sd    t-stat')
    for ii=1:3 %Loop over coefficients
        fprintf('%s%10.4f%10.4f%10.4f\n',label(ii), ...
            eqsur.beta(ii+(jj-1)*3),eqsur.sdbeta(ii+(jj-1)*3), ...
            eqsur.tbeta(ii+(jj-1)*3))
    end
    fprintf('Standard Error is %10.4f\n',eqsur.se(jj));
end

SUR estimates


                coef        sd    t-stat
C -168.1134   84.9017   -1.9801
F    0.1219    0.0204    5.9700
C    0.3822    0.0321   11.9109
Standard Error is    84.9836
```

SUR Estimates (cont.)

```
                coef        sd    t-stat
C    0.9980   11.5661    0.0863
F    0.0689    0.0170    4.0473
C    0.3084    0.0260   11.8766
Standard Error is    12.3789



                coef        sd    t-stat
C  -21.1374   24.3479   -0.8681
F    0.0371    0.0115    3.2327
C    0.1287    0.0212    6.0728
Standard Error is    26.5467



                coef        sd    t-stat
C    1.4075    5.9944    0.2348
F    0.0564    0.0106    5.3193
C    0.0429    0.0382    1.1233
Standard Error is     9.7420



                coef        sd    t-stat
C   62.2563   93.0441    0.6691
F    0.1214    0.0451    2.6948
C    0.3691    0.1070    3.4494
Standard Error is    90.4117

% % SUR in LeSage toolbox

y(1).eq = Y_GM;
y(2).eq = Y_GE;
```

SUR Estimates (cont.)

```
y(3).eq = Y_CH;
y(4).eq = Y_WE;
y(5).eq = Y_US;
XX(1).eq = X_GM;
XX(2).eq = X_GE;
XX(3).eq = X_CH;
XX(4).eq = X_WE;
XX(5).eq = X_US;
neqs=5;
sur_result=sur(neqs,y,XX);
prt(sur_result)

Seemingly Unrelated Regression -- Equation   1
System R-sqr   =    0.8694
R-squared      =    0.9207
Rbar-squared   =    0.9113
sigma^2        = 458183.2995
Durbin-Watson  =    0.0400
Nobs, Nvars    =     20,     3
***************************************************************
Variable          Coefficient      t-statistic    t-probability
variable   1      -168.113426        -1.980094         0.064116
variable   2         0.121906         5.969973         0.000015
variable   3         0.382167        11.910936         0.000000


Seemingly Unrelated Regression -- Equation   2
System R-sqr   =    0.8694
R-squared      =    0.9116
Rbar-squared   =    0.9012
sigma^2        = 8879.1368
Durbin-Watson  =    0.0310
Nobs, Nvars    =     20,     3
***************************************************************
```

SUR Estimates (cont.)

```
Variable          Coefficient      t-statistic    t-probability
variable   1         0.997999         0.086286         0.932247
variable   2         0.068861         4.047270         0.000837
variable   3         0.308388        11.876603         0.000000


Seemingly Unrelated Regression -- Equation   3
System R-sqr   =    0.8694
R-squared      =    0.6857
Rbar-squared   =    0.6488
sigma^2        = 11785.8684
Durbin-Watson  =    0.0202
Nobs, Nvars    =     20,     3
***************************************************************
Variable          Coefficient      t-statistic    t-probability
variable   1       -21.137397        -0.868140         0.397408
variable   2         0.037053         3.232726         0.004891
variable   3         0.128687         6.072805         0.000012


Seemingly Unrelated Regression -- Equation   4
System R-sqr   =    0.8694
R-squared      =    0.7264
Rbar-squared   =    0.6943
sigma^2        = 2042.8631
Durbin-Watson  =    0.0323
Nobs, Nvars    =     20,     3
***************************************************************
Variable          Coefficient      t-statistic    t-probability
variable   1         1.407487         0.234802         0.817168
variable   2         0.056356         5.319333         0.000056
variable   3         0.042902         1.123296         0.276925
```

SUR Estimates (cont.)

```
Seemingly Unrelated Regression -- Equation   5
System R-sqr   =    0.8694
R-squared      =    0.4528
Rbar-squared   =    0.3884
sigma^2        = 173504.8346
Durbin-Watson  =    0.0103
Nobs, Nvars    =     20,     3
***************************************************************
Variable          Coefficient      t-statistic    t-probability
variable   1        62.256312         0.669105         0.512413
variable   2         0.121402         2.694815         0.015340
variable   3         0.369111         3.449403         0.003062

Cross-equation sig(i,j) estimates
equation      eq 1      eq 2      eq 3      eq 4      eq 5
eq 1     7222.2204 -315.6107  601.6316  129.7644 -2446.3171
eq 2     -315.6107  153.2369    3.1478   16.6475  414.5298
eq 3      601.6316    3.1478  704.7290  201.4385 1298.6953
eq 4      129.7644   16.6475  201.4385   94.9067  613.9925
eq 5     -2446.3171  414.5298 1298.6953  613.9925 8174.2798


Cross-equation correlations
equation      eq 1      eq 2      eq 3      eq 4      eq 5
eq 1        1.0000   -0.3000    0.2667    0.1567   -0.3184
eq 2       -0.3000    1.0000    0.0096    0.1380    0.3704
eq 3        0.2667    0.0096    1.0000    0.7789    0.5411
eq 4        0.1567    0.1380    0.7789    1.0000    0.6971
eq 5       -0.3184    0.3704    0.5411    0.6971    1.0000
```

6.2 Exercise – Using MATLAB to estimate a simultaneous equation system

Consider the demand-supply model

$$q_t = \beta_{11} + \beta_{21}x_{t2} + \beta_{31}x_{t2} + \gamma_{21}p_t + u_{t1} \quad (6.1)$$

$$q_t = \beta_{12} + \beta_{42}x_{t4} + \beta_{52}x_{t5} + \gamma_{22}p_t + u_{t2}, \quad (6.2)$$

where q_t is the log of quantity, p_t is the log of price, x_{t2} is the log of income, x_{t3} is a dummy variable that accounts for demand shifts x_{t4} and x_{t5} are input prices. Thus equations (6.1) and (6.2) are demand and supply functions respectively. 120 observations generated by this model are in the file `demand-supply.csv`

1. Comment on the identification of the system. Why can the system not be estimated using equation by equation OLS. For each of the estimates below produce estimates, standard errors and t-statistics of each coefficient. Also produce standard errors for each equation.
2. Estimate the system equation by equation using OLS.
3. Estimate the system equation by equation using 2SLS. Compare the results with the OLS estimates.
4. Set up the matrices of included variables, exogenous variables required to do system estimation.
5. Do OLS estimation using the stacked system and compare results with the equation by equation estimates.
6. Do 2SLS estimation using the stacked system and compare results with the equation by equation estimates.
7. Do 3SLS estimation using the stacked system and compare results with the 2SLS estimates.
8. Comment on the identification of the system.
9. How can the method be generalized to estimate other GMM estimators? Estimate the optimum GMM estimator for the system and compare your results with the previous estimators.

CHAPTER 7

User written functions in MATLAB

7.1 Function m-files

One of the most useful facilities in MATLAB is the facility to write one's own functions and use them in the same way as a native MATLAB functions. We are already familiar with m-files which contain MATLAB instructions. Such files are known as script files and allow us to do repeat an analysis without having to retype all the instructions. The file `normdensity.m` contains a function that estimates the density function of a normal distribution,

$$\frac{1}{\sqrt{2\pi}\,\sigma} \exp - \frac{(x-\mu)^2}{2\sigma^2},$$

Example of MATLAB Function

```
function f = normdensity(z, mu, sigma);
% Calculates the Density Function of the Normal Distribution
% with mean mu
% and standard deviation sigma
% at a point z
```

Example of MATLAB Function (cont.)

```
% sigma must be a positive non-zero real number
if sigma  <= 0
    fprintf('Invalid input\n');
    f = NaN;
else
    f = (1/(sqrt(2*pi)*sigma))*exp(-(z-mu)^2/(2*sigma^2));
end
```

Note the following

1. The file starts with the keyword function. This is to indicate that this m-file is a function definition.
2. In the first line the f indicates that the value of f when the file has been “run” is the value that will be returned.
3. The function is called with normdensity(z, mu, sigma) where z mu and sigma are given values in calling the function.
4. The commented lines immediately after the first line are a help system for the function
5. All variables within a function are local to the function. Thus if there is a variable within a function called x and one in the program with the same name the one in the function is used when the function is in operation. Once the program has been run the value in the function is forgotten and the value outside the program is used.
6. It is possible to have more than one function stored in a function file. Only the first function (the main function) in the file is visible to the calling program. Second and subsequent function in a function file can only be seen by this main function.
7. The file should be saved in the current working directory or in a folder on the current MATLAB search path. You may change the search path using the **set path** icon in the **HOME** tab in the GUI.
8. Your function operates in the same way as the official functions distributed with MATLAB.

The use of the function normdensity can be demonstrated as follows –

Get help for normdensity function.

```
help normdensity
```

```
Calculates the Density Function of the Normal Distribution
with mean mu
and standard deviation sigma
at a point z
sigma must be a positive non-zero real number
```

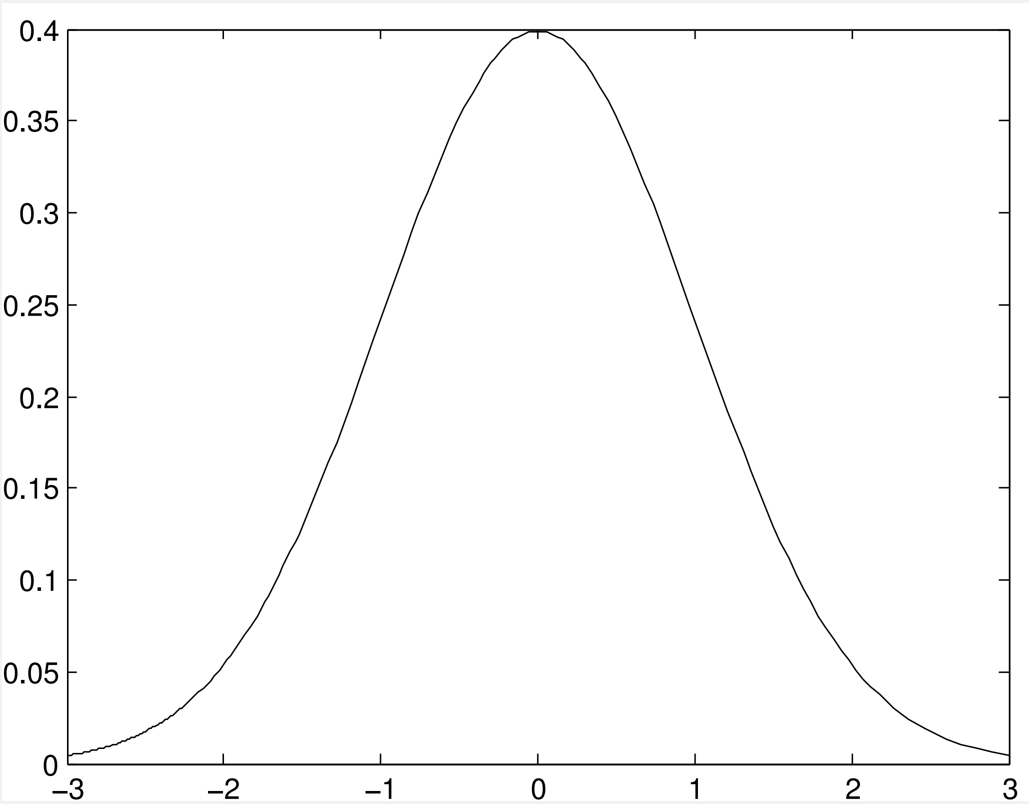
Evaluate Standard Normal density function at zero

```
normdensity(0,0,1)
```

```
ans =
    0.3989
```

Plot standard normal density function

```
fplot('normdensity(x,0,1)',[-3 3])
```



7.2 Anonymous functions

MATLAB Version 7 introduced the idea of an anonymous function. This is illustrated in the box below which shows how to define the normal distribution function in the previous example as an anonymous function.

```
f1 = @(z,mu,sigma) (1/(sqrt(2*pi)*sigma))*exp(-(z-mu)^2/(2*sigma^2));

f1 ( 0, 0, 1)
```

```
ans =
0.3989
```

The anonymous function is defined in the first line of the box.

1. The `f1` is a function handle for the anonymous function. This can be passed as an input argument to another function.
2. The `@` character constructs the function. The parentheses '()' which follow contain the input arguments of the function within brackets.
3. This is then followed by the definition of the function with a single MATLAB statement.
4. You may use variables that are not in the argument list in the definition of the function. If the value of these variables is changed after the function is defined the function is not revised. If you wish to use the revised values of these variables you must redefine the function.
5. One advantage of anonymous functions is that you do not have to maintain a separate function file.
6. Perhaps more useful, the file handle can be passed as an input argument to another function. For example,in estimating maximum likelihood estimates we may need to pass the likelihood function to be maximised to a general optimisation routine

CHAPTER 8

Econometric Toolboxes

8.1 Introduction

If you are accustomed to using one of the many packages that deal specifically with econometrics you may think that MATLAB takes a long time to do simple things. It is also clear that many or the more difficult tasks are often easier in MATLAB than in these packages. MATLAB is less of a "'black box" than many of the other programs. One must really learn and understand the algebra before one can use MATLAB for econometrics. One also knows exactly what one is doing when one has written the routines in MATLAB

The big problem is the lack of elementary econometric facilities in MATLAB. Two toolboxes are available to economists —

1. **The LeSage econometric toolbox.** This covers most of the econometric routines required for an econometrics course which is part of an economics primary or master's degree
2. **The MathWorks econometrics toolbox**. To use this toolbox you must also have installed the **statistics**, **optimization** and **financial** toolboxes. The MATLAB

econometrics toolbox concentrates to a large extent on the type of time series econometrics used in finance and does not cover much of the material that is in the LeSage toolbox.

8.2 LeSage Toolbox

1. Download the file `jplv7.zip` to a directory on your PC using the link on `http://www.spatial-econometrics.com/`.
2. Unzip the file to this directory. This creates 13 directories
3. If you have administrative rights create a subdirectory in the toolbox subdirectory of your MATLAB installation. I called this directory `LeSage` as I also had an econ directory holding the MathWorks toolbox. If you do not have permissions to create the `LeSage` directory here create it some where in your user directory.
4. In your working directory create a file named startup.m with the following content. You may, if necessary copy theses directories to a subdirectory on a flash drive.

```
addpath(genpath('c:\Program Files\MATLAB\R2014a\toolbox\LeSage\'))
```

   On my PC I copied the LeSage econometrics sub-directories to the directory `'c:\Program Files\MATLAB\R2014a\toolbox\LeSage\'`. If you copied them to another directory or to a flash drive then you must amend this path. This instruction adds the LeSage directory and all its subdirectories at the front of the MATLAB path during the current session. Thus, if a function or variable in LeSage conflicts with a similar function in MATLAB or in a MATLAB toolbox the LeSage one takes precedence. I do not save the new MATLAB path. In this way if I start MATLAB in a directory which has not got that command in a `startup.m` it will start with the default MATLAB path[1].
5. Download the manual from the link on `http://www.spatial-econometrics.com/`.
6. All the programs, functions etc. in the LeSage toolbox contain excellent comments. The code is well written and relatively easy to understand. The commentary in the manual is also helpful. It should be relatively easy to extend the functions or to add new functions if the need arises.

[1] When MATLAB finds a startup.m file in the working directory at start-up it executes any commands in that file. In this way you can automatically load the data for your project or do any other initializations that you find useful.

The current version of this toolbox is optimized for MATLAB version 7. According to the material on the web-site the manual was last up-dated in September, 1999. It does give a good account of the reasoning behind the functions in the toolbox. The examples distributed with the code for the functions are more up to date and should be used rather than the examples in the manual. All of the examples in the distribution end in `...d.m`. Thus the function to estimate a vector autoregression is `vare.m` and the demonstration of that function is `vare_d.m`.

In each subdirectory of the toolbox there is a file `contents.html` which lists the functions, demonstrations etc. in that subdirectory. The material in appendix A has been taken from these files.

One can summarize the contents of the subdirectories as follows

1. **Regression/Estimation Functions in table A.1**. These cover estimation of OLS, HCSE Models, Box-Cox Models, Limited Dependent Variable Models, Simultaneous Equations, Bayesian Models, Panel Models etc.
2. **Diagnostics in table A.2** Included are a variety of residual and specification tests e. g. ARCH, Breusch-Pagan, Q-test, cusum, BKW, recursive residuals.
3. **Unit Roots and Cointegration in A.3** Covers Dickey-Fuller, Philips-Peron, HEGY seasonal unit roots and Johansen procedures.
4. **Vector Autoregression in table A.4**. Covers standard VAR estimation routines, impulse response functions, causality tests and various Bayesian procedures.
5. **Markov Chain Monte Carlo in table A.5** A Gibbs sampling library of utility functions along with various estimation procedures is included here. functions are described in this chapter
6. **Time Series Aggregation/Disaggregation in table A.6** These functions appear to have been contributed after the toolbox manual was written. This is the best collection of interpolation disaggregation and aggregation routines that I have encountered. The subdirectory contains two pdf files describing the methodologies. The demonstration files in this directory show how to run the functions within MATLAB. (The pdf documentation in the directory is based on calling the MATLAB functions from within EXCEL which I do not recommend.)
7. **Optimization in table A.7** The optimization function `fminsearch` in recent version of MATLAB employed as in Chapter 9 can satisfy most of an economist's requirements.
8. **Plots and Graphs in table A.8**. This section contains functions that produce

graphs of interest in econometrics.

9. **Statistical Functions in table A.9**. The functions listed in table A.8 calculate density functions, distribution functions, quantiles and simulate random samples for beta, binomial, chisquared, F, gamma, Hypergeometric, lognormal, logistic, normal, left- and right-truncated normal, multivariate normal, Poison, t and Wishart distributions. This is essential if you do not have access to the statistics toolbox in MATLAB.
10. **Utilities in table A.10**. This is a collection of functions that an economist might find useful. There are four kinds of utilities — (i) functions for working with time series and dates, (ii) general functions for printing and plotting matrices as well as producing LaTeX formatted output of matrices for tables, (iii) econometric data transformations and (iv) functions that mimic functions in other software which are not available in MATLAB.
11. **Spatial Econometrics**. The toolbox also contains a collection of functions for spatial econometrics. For details see `http://www.spatial-econometrics.com/`
12. The **USCD-GARCH** package is now depreciated and has been replaced by the MFE Toolbox (see section 8.4).

You can use the `help` function in the **COMMAND WINDOW** to get help on the LeSage functions. This is illustrated in the following box which shows the help on the LeSage `ols` function.

Help on `ols` function in LeSage toolbox

```
>> help ols
```

```
PURPOSE: least-squares regression
---------------------------------------------------
USAGE: results = ols(y,x)
where: y = dependent variable vector    (nobs x 1)
x = independent variables matrix (nobs x nvar)
---------------------------------------------------
RETURNS: a structure
results.meth  = 'ols'
results.beta  = bhat     (nvar x 1)
results.tstat = t-stats  (nvar x 1)
results.bstd  = std deviations for bhat (nvar x 1)
```

Help on `ols` function in LeSage toolbox (cont.)

```
results.yhat  = yhat     (nobs x 1)
results.resid = residuals (nobs x 1)
results.sige  = e'*e/(n-k)   scalar
results.rsqr  = rsquared     scalar
results.rbar  = rbar-squared scalar
results.dw    = Durbin-Watson Statistic
results.nobs  = nobs
results.nvar  = nvars
results.y     = y data vector (nobs x 1)
results.bint  = (nvar x2 ) vector with 95% confidence intervals on beta
---------------------------------------------------
SEE ALSO: prt(results), plt(results)
---------------------------------------------------
```

The next box shows the demonstration program for the `ols()` function.

ols demonstration program

```
% PURPOSE: An example using ols(),
%                            prt(),
%                            plt(),
% ordinary least-squares estimation
%---------------------------------------------------
% USAGE: ols_d
%---------------------------------------------------

rng(12345) %JCF
nobs = 1000;
nvar = 15;
beta = ones(nvar,1);

xmat = randn(nobs,nvar-1);

x = [ones(nobs,1) xmat];
evec = randn(nobs,1);
```

ols demonstration program (cont.)

```
y = x*beta + evec;

vnames = strvcat('y-vector','constant','x1',';x2','x3','x4','x5','x6', ...
'x7','x8','x9','x10','x11','x12','x13','x14');

% do ols regression
result = ols(y,x);

% print the output
prt(result,vnames);
title('title string')
% plot the predicted and residuals
plt(result);
print -dpdf 'ols_d_01.pdf';%JCF
% pause; %JCF

% recover residuals
resid = result.resid;

% print out tstats
fprintf(1,'tstatistics = \n');
result.tstat

% plot actual vs. predicted
tt=1:nobs;
figure % JCF
plot(tt,result.y,tt,result.yhat,'--');
title('Actual and Predicted')
print -dpdf 'ols_d_02.pdf';%JCF
```

```
>> ols_d

Ordinary Least-squares Estimates
```

ols demonstration program (cont.)

```
Dependent Variable =         y-vector
R-squared      =    0.9328
Rbar-squared   =    0.9319
sigma^2        =    0.9954
Durbin-Watson  =    2.0224
Nobs, Nvars    =   1000,    15
***************************************************************
Variable      Coefficient      t-statistic    t-probability
constant         0.984796        31.085490         0.000000
x1               1.048733        33.075864         0.000000
;x2              0.980482        29.999217         0.000000
x3               1.052899        32.421795         0.000000
x4               0.996301        31.395523         0.000000
x5               0.970616        31.216304         0.000000
x6               1.016998        33.323262         0.000000
x7               1.005818        32.351407         0.000000
x8               0.995431        31.215225         0.000000
x9               0.990791        31.155604         0.000000
x10              1.003293        31.389013         0.000000
x11              0.993757        29.455230         0.000000
x12              1.014091        31.328235         0.000000
x13              0.984371        29.961983         0.000000
x14              1.047248        33.044102         0.000000

tstatistics =

ans =

31.0855
33.0759
29.9992
32.4218
31.3955
31.2163
```

ols demonstration program (cont.)

```
33.3233
32.3514
31.2152
31.1556
31.3890
29.4552
31.3282
29.9620
33.0441
```

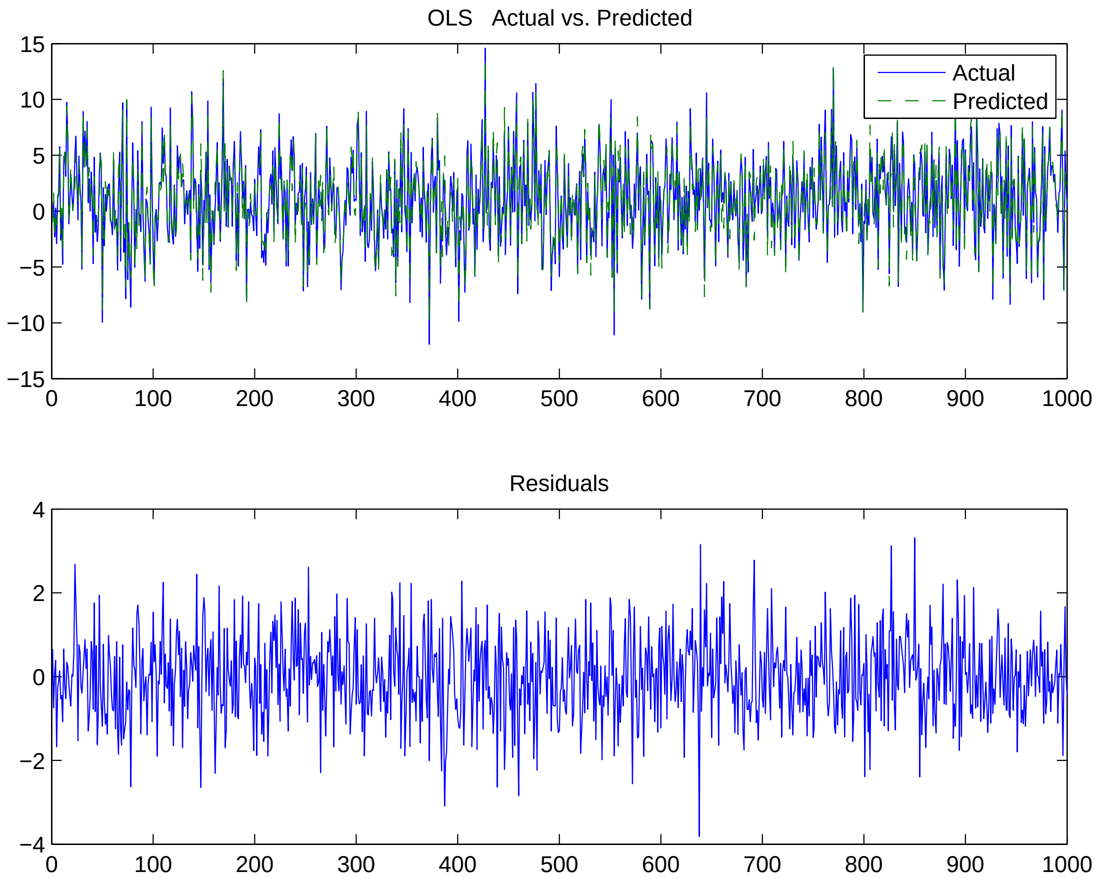


Figure 8.1: First Graph produced by ols_d

I have made some small amendments to the distributed `ols_d()` program.

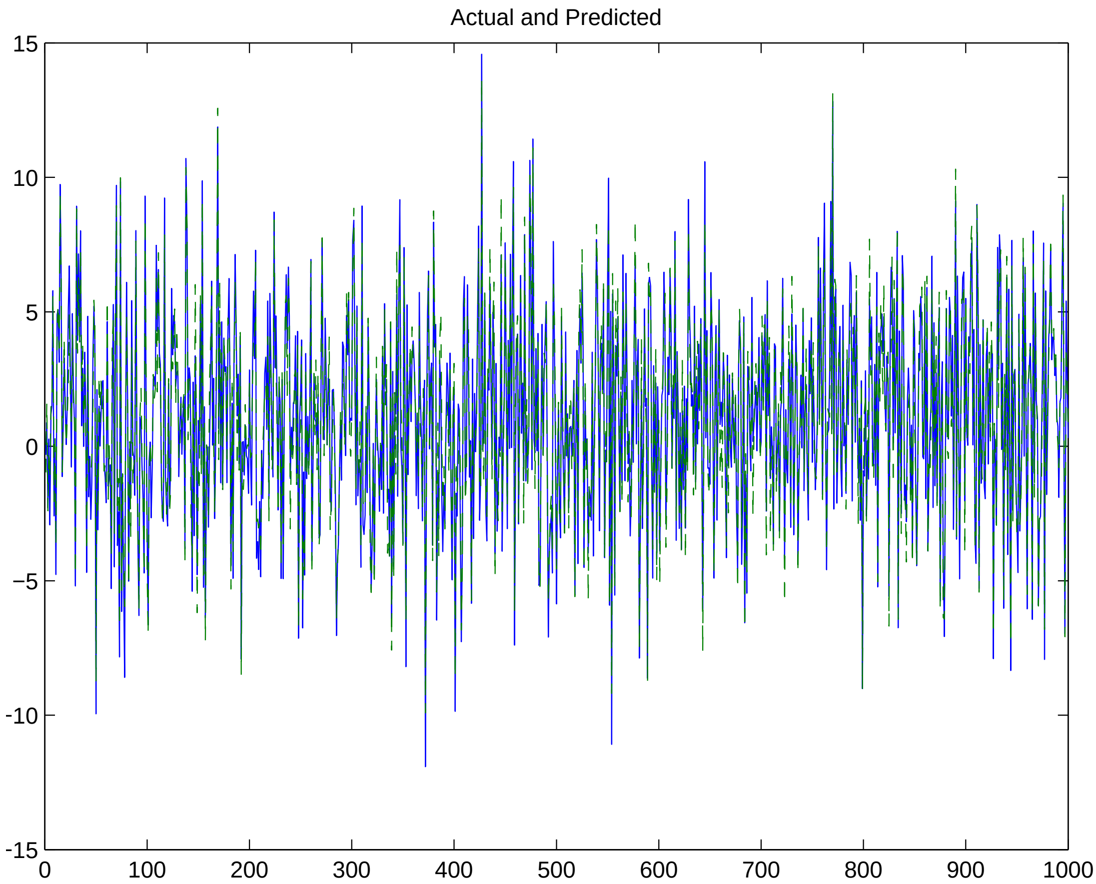


Figure 8.2: Second Graph produced by ols_d

These amendments are indicated by `%JCF` in the box above.

1. The '`rng(12345)`' command ensures that the same random numbers are generated each time the program is run. If you are working through LeSage demonstrations involving simulated data you may consider including a similar statement to ensure that you can replicate results.
2. Many of the LeSage demonstrations include '`pause`' statements. I prefer to remove these.
3. The '`print -dpdf 'ols_d_01.pdf';%JCF`' has been added to save the graph ion pdf format for inclusion here. I have also save the second graph in the same way.
4. I have added a '`figure`' statement before the second graph is drawn.
5. I have added a title to the second graph.

8.3 MATLAB Econometrics package

http://www.mathworks.co.uk/help/econ/product-description.html contains the following overview of the econometrics toolbox. The help files and contents of this toolbox were described in more detail in subsection 1.3.8

Model and analyse financial and economic systems using statistical methods

Econometrics Toolbox provides functions for modelling economic data. You can select and calibrate economic models for simulation and forecasting. For time series modelling and analysis, the toolbox includes univariate ARMAX/GARCH composite models with several GARCH variants, multivariate VARMAX models, and cointegration analysis. It also provides methods for modelling economic systems using state-space models and for estimating using the Kalman filter. You can use a variety of diagnostic functions for model selection, including hypothesis, unit root, and stationarity tests.

Key Features

- Univariate ARMAX/GARCH composite models, including EGARCH, GJR, and other variants
- Multivariate simulation and forecasting of VAR, VEC, and cointegrated models
- State-space models and the Kalman filter for estimation
- Tests for unit root (Dickey-Fuller, Phillips-Perron) and stationarity (Leybourne-McCabe, KPSS)
- Statistical tests, including likelihood ratio, LM, Wald, Engle's ARCH, and Ljung-Box Q Cointegration tests, including Engle-Granger and Johansen
- Diagnostics and utilities, including AIC/BIC model selection and partial-, auto-, and cross-correlations
- Hodrick-Prescott filter for business-cycle analysis

This toolbox is of interest to those working in the financial econometrics and computational finance. It should be noted that the econometrics toolbox requires that the statistics, optimization and financial toolboxes be also installed.

8.4 Oxford MFE Toolbox

The **Oxford MFE** can be downloaded from `http://www.kevinsheppard.com/MFE_Toolbox`. This toolbox covers many time-series and other econometric methods used in financial econometrics and quantitative finance. It also contains some relevant functions similar to those in the MATLAB statistics and financial toolboxes that can be substituted if the MATLAB versions are not available.

Unless you are sure about what you are doing you should nit use the LeSage and MFE toolboxes together. The following details of the functions in the MFE toolbox are taken from its website.

The **USCD-GARCH** package is now depreciated and has been replaced by the MFE Toolbox. The new toolbox and a manual can be downloaded from `http://www.kevinsheppard.com/MFE_Toolbox`. The MFE toolbox covers many time-series procedures used in financial econometrics. I would avoid loading the both the LeSage and MFE toolboxes.

Details of MFE toolbox

High Level List of Functions

- Regression
- ARMA Simulation
- ARMA Estimation
  - Heterogeneous Autoregression
  - Information Criteria
- ARMA Forecasting
- Sample autocorrelation and partial
- autocorrelation
- Theoretical autocorrelation and partial autocorrelation
- Testing for serial correlation
  - Ljung-Box Q Statistic
  - LM Serial Correlation Test
- Filtering
  - Baxter-King Filtering
  - Hodrick-Prescott Filtering
- Regression with Time Series Data

Details of MFE toolbox (cont.)

- Long-run Covariance Estimation
  - Newey-West covariance estimation
  - Den Hann-Levin covariance estimation
- Nonstationary Time Series
  - Unit Root Testing
  - Augmented Dickey-Fuller testing
  - Augmented Dickey-Fuller testing with automated lag selection
- Vector Autoregressions
  - Granger Causality Testing: grangercause
  - Impulse Response function calculation
- Volatility Modeling
  - ARCH/GARCH/AVARCH/TARCH/ZARCH Simulation
  - EGARCH Simulation
  - APARCH Simulation
  - FIGARCH Simulation
  - GARCH Model Estimation
  - ARCH/GARCH/GJR-GARCH/TARCH/AVGARCH/ZARCH Estimation
  - EGARCH Estimation
  - APARCH Estimation
  - AGARCH and NAGARCH estimation
  - IGARCH estimation
  - FIGARCH estimation
  - HEAVY models
- Density Estimation
  - Kernel Density Estimation
- Distributional Fit Testing
  - Jarque-Bera Test
  - Kolmogorov-Smirnov Test
  - Berkowitz Test
- Bootstraps
  - Block Bootstrap
  - Stationary Bootstrap

Details of MFE toolbox (cont.)

- Multiple Hypothesis Tests
  - Reality Check and Test for Superior Predictive Accuracy
  - Model Confidence Set
- Multaivariate GARCH
  - CCC MVGARCH
  - Scalar Variance Targetting VECH
  - MATRIX GARCH
  - DCC and ADCC
  - OGARCH
  - GOGARCH
  - RARCH
- Realized Measures
  - Realized Variance
  - Realized Covariance
  - Realized Kernels
  - Multivariate Realized Kernels
  - Realized Quantile Variance
  - Two-scale Realized Variance
  - Multi-scale Realized Variance
  - Realized Range
  - QMLE Realized Variance
  - Min Realized Variance, Median Realized Variance (MinRV, MedRV)
  - Integrated Quarticity Estimation

Functions Missing available from Previous UCSD GARCH Toolbox

The following list of function have not been updated and so if needed, you should continue to use the UCSD_GARCH code.

- GARCH in mean
- IDCC MVGARCH
- Shapirowilks
- Shapirofrancia

CHAPTER 9

Maximum Likelihood Estimation using Numerical Techniques

In many cases of maximum likelihood estimation there is no analytic solution to the optimisation problem and one must use numerical techniques. On some occasions the analytic solution may be very complicated and numerical techniques might be easier to use. The aim of this section is to demonstrate these techniques. The example used is based on the estimation of a tobit process.

This basic maximum likelihood algorithm could be applied in many econometric packages. In most cases, as in theory, one works with the log-likelihood rather than the likelihood. MATLAB, like many other packages contain a minimisation routines rather than maximisation. Thus one uses the equivalent procedure of minimising the negative of the log-likelihood.

The steps involved are as follows –

1. Load and process your data.
2. Write a MATLAB function to estimate the log-likelihood.
3. Calculate an initial estimate of the parameters to be estimated. You may use OLS, previous studies or even guesses. This will serve as starting values for the

optimisation routine.

4. Check the defaults for the optimization routine (e./,g. maximum number of iterations, convergence criteria). If your initial attempt does not converge you may have to change these and/or use different starting values.

5. Call the optimisation routine.

6. Check for convergence. After a specified number of iterations the optimisation routine will stop and out put results. If the routine has not converged the results, however good they look, are worthless. You must go back and look at your starting values or change the number of iterations or consider whether your model is appropriate. Many economists try to estimate models with many parameters and little data and the likelihood function may be almost flat over a wide region.

7. Estimate standard errors of your coefficients.

8. Print out results.

I shall use the `fminsearch` MATLAB function to carry out the optimisation. This routine uses what is known as the Nelder-Mead simplex[1] method and does not require the use of derivatives. It is generally regarded as slow but sure. For a description of this and alternative minimisation routines I would recommend Press et al. (2007). Earlier versions of this book are available on-line at `www.nr.com`.

The MATLAB optimisation toolbox, **optim**, provides a wide range of optimisation functions. The Le Sage `econometrics` package also provides some other optimisation functions.

The data to be used in this example is a random sample drawn from the tobit process

$$
\begin{aligned}
Y_i^* &= 5 + 2X_{2i} - 3X_{3i} + \varepsilon_i \\
Y_i &= 0 \quad \text{if} \quad y_i^* \leqslant 0 \\
Y_i &= 1 \quad \text{if} \quad y_i^* > 0
\end{aligned}
\tag{9.1}
$$

where ε_i follows a normal distribution with mean 0 and variance 9. The code in the following box generates a sample of 100 from this distribution.

[1]This has nothing to do with the simplex method in linear programming

Generation of Tobit Sample

```
rng(2468);
nobs = 100;
X=[ones(nobs,1),10*rand(nobs,2)];
beta = [5,2,-3]';
epsilon = 3*randn(nobs,1);
Y = X * beta + epsilon;
% Truncate
Ytrunc = (Y > 0) .* Y;
```

Using the usual notation the log-likelihood for such a tobit process can then be written as

$$\sum_{i=1}^{N}\left[d_i(-\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}(y_i - \boldsymbol{x}_i\boldsymbol{\beta})^2) + (1-d_i)\ln\left(1-\Phi\left(\frac{\boldsymbol{x\beta}}{\boldsymbol{\sigma}}\right)\right)\right]$$

where d_i is a dummy variable where

$$d_t = \begin{cases} 1 & \text{if } y_t > 0 \\ 0 & \text{if } y_t \leqslant 0 \end{cases}$$

The MATLAB function (`tobit_like(b,x,y)`) programmed in the following box calculates the likelihood of such a tobit process. It is an amended version of the corresponding program from the Le Sage econometrics toolbox. If you have access to the MATLAB statistics toolbox the probability density and distribution functions there can be used to simplify this function.

tobit_like.m

```
function tobitlike =  tobit_like(b,y,x);
% PURPOSE: evaluate tobit log-likelihood
%          to demonstrate optimization routines
%-----------------------------------------------------
% USAGE:    like = tobit_like(b,y,x)
% where:     b = parameter vector (k x 1) to be estimated
%                   b contains beta parameters and variance of
```

tobit_like.m (cont.)

```
%                    disturbance term (b(k)) - see note below
%             y = dependent variable vector (n x 1)
%             x = explanatory variables matrix (n x m)
%-----------------------------------------------------
% NOTE: this function returns a scalar equal to the
%       negative of the log likelihood function
%       or a scalar sum of the vector depending
%       on the value of the flag argument
%       k ~= m because we may have additional parameters
%           in addition to the m bhat's (e./,g. sigma)
%-----------------------------------------------------

% error check
if nargin ~= 3,error('wrong # of arguments to to_like1'); end;
[m1 m2] = size(b);
if m1 == 1
b = b';
end;

h = .000001;             % avoid sigma = 0
[m junk] = size(b);
beta = b(1:m-1);         % pull out bhat
sigma = max([b(m) h]);   % pull out sigma
xb = x*beta;
% next two lines amended
llf1 =   -0.5*log(2*pi) - 0.5*log(sigma^2) -((y-xb).^2)./(2*sigma^2);
xbs = xb./sigma; cdf = .5*(1+erf(xbs./sqrt(2)));
llf2 = log(h+(1-cdf));
llf = (y > 0).*llf1 + (y <= 0).*llf2;
tobitlike = -sum(llf); % scalar result
```

The next step is to estimate starting values. In the next box I use OLS to estimate these starting values.

tobit_like.m

```
beta0 = (X'*X)'\(X'*Ytrunc);
beta0 = beta0';
%sd = sqrt( Ytrunc);

sd=sqrt(((Ytrunc-X*beta0')'* (Ytrunc-X*beta0'))/(nobs-length(beta0)));
parm0 = [beta0 sd];
```

The next box gives the call to the minimisation routine and the values of the output of that routine.

parm The values of the parameters as the routine finishes. The values found are close to the values used to simulate the process

fval In this case the negative of the maximum likelihood

exitflag An exit flag of 1 indicates that the process has converged. (0 indicates that it has not and that you have more work to do)

output This gives some details of the minimisation process.

Minimisation

```
[parm, fval, exitflag, output] =...
 fminsearch(@(parm)tobit_like(parm,Y,X),parm0)
fval
exitflag
output
```

```
parm =
5.1849    2.0336   -3.1373    3.4256

fval =
145.0262

exitflag =
1

output =
```

Minimisation (cont.)

```
iterations: 132
funcCount: 226
algorithm: 'Nelder-Mead simplex direct search'
message: 'Optimization terminated:
the current x satisfies the ter...'
```

The next box shows how to obtain estimates of the standard errors of these estimates. The method used here involves usinf numerical methods to estimate the i,j$^{\text{th}}$ element of the information matrix

$$I(\boldsymbol{\theta})_{ij} = -\frac{\partial^2 l(\boldsymbol{\theta})}{\partial\theta_i \partial\theta_j}$$

This matrix is then inverted to estimate the variance covariance matrix of the estimators. Their standard errors are given by the square roots of the diagonal elements of the inverted matrix. Numerical differentiation is calculated using the routines for the MATLAB `hessian` function available on the MATLAB exchange site at `http://www.mathworks.com/matlabcentral/fileexchange/13490-adaptive-robust-numerical-differentiation`

Standard Errors

```
tobit_like2 = @(parm)-tobit_like(parm, Y, X);

[hess,err] = hessian(tobit_like2,parm');

varcov = -inv(hess);
separm = sqrt(diag(varcov));
tparm  = parm' ./ separm;
```

The next box summarises the results in a format similar to that produced by an econometric package.

Printing Results

```
fprintf('\nSummary Estimate of Tobit\n')
fprintf('                  Coef.   Std Error          z\n')
fprintf('%12s%12.4f%12.4f%12.4f\n',...
```

Printing Results (cont.)

```
'const', parm(1,1),separm(1),tparm(1))
fprintf('%12s%12.4f%12.4f%12.4f\n',...
'const', parm(1,2),separm(2),tparm(2))
fprintf('%12s%12.4f%12.4f%12.4f\n\n',...
'const', parm(1,3),separm(3),tparm(3))

fprintf('Standard Error of Equation is %7.4f (%7.4f)\n',...
parm(1,4),separm(4))
```

```
Summary Estimate of Tobit
Coef.   Std Error           z
const      5.1849      1.2634      4.1038
const      2.0336      0.1839     11.0551
const     -3.1373      0.2246    -13.9714

Standard Error of Equation is  3.4256 ( 0.3328)
```

The next box gives the results of using the Gretl econometric package (Cottrell and Lucchetti, 2014) to estimate this Tobit process. As you can see both results are in close agreement.

Output of GRETL Estimate of Tobit

Model 1: Tobit, using observations 1–100
Dependent variable: Ytrunc
Standard errors based on Hessian

	Coefficient	Std. Error	z	p-value
const	5.18485	1.26341	4.1038	0.0000
X2	2.03357	0.183971	11.0537	0.0000
X3	−3.13735	0.224831	−13.9542	0.0000

Chi-square(2)	253.2255	p-value	1.03e–55
Log-likelihood	−145.0263	Akaike criterion	298.0525
Schwarz criterion	308.4732	Hannan–Quinn	302.2700

$\hat{\sigma} = 3.42557\ (0.338551)$

Output of GRETL Estimate of Tobit (cont.)

Left-censored observations: 48
Right-censored observations: 0

Test for normality of residual –
Null hypothesis: error is normally distributed
Test statistic: $\chi^2(2) = 0.389122$
with p-value = 0.823196

Exercise

In Frain (2010) I illustrated this procedure by replicating the tobit analysis of tobacco expenditure on in Table 7.9 on page 237 of Verbeek (2008). Verify the results obtained there using the routines described in this chapter[2].

[2] The relevant file, in native MATLAB format is embedded in this document. To extract it, please go to appendix B

CHAPTER 10

Octave, Scilab and R

MATLAB is an excellent program and is widely used in finance science and engineering. The documentation and user interface for MATLAB are excellent. There is also a lot of material on MATLAB available on the internet.

There have been many occasions when I needed access to MATLAB and was forced to find an alternative. For security reasons, your employer may place various restrictions on the programs you can use on a work computer. If you want to use MATLAB you may need local management approval, IT management approval and purchase through a central purchasing unit. Your employer's MATLAB license may not allow you to use MATLAB on a home computer where you want to do some work. In some cases you may only use MATLAB on an occasional basis and the cost of licenses for MATLAB and a variety of toolboxes may not be justified.

In such cases, you might consider Octave, Scilab or R which are free programs with similar functionality to MATLAB.

10.1 Octave

Octave is free software distributed under the Gnu General Public License. A new version containing a modern GUI is under development. For the moment instructions for downloading and installing an unofficial version of Octave 3.8.2 for windows can be obtained from `http://wiki.octave.org/Octave_for_Windows#Octave_3.8_MXE_Builds`. If you are running Windows 8 be sure that you follow the instructions as you may have problems running the GUI in Windows 8. Several toolboxes are available with this distribution. The file

Octave is largely compatible with MATLAB to the extent that most programs written in base MATLAB will also run in base Octave. In base Octave there are some additional options that are not available in MATLAB. Some MATLAB toolboxes have full or partial implementation as Octave toolboxes. The file `README.html` in the install directory of this distribution describes how to install these packages and where to get additional packages.

A program for base MATLAB will run in Octave with at most minor changes and, in all likelihood with none. The original drafts of these note were completed at home with Octave as I had no access to MATLAB there at the time. Programs written in Octave may not run in base MATLAB as base Octave contains many functions similar to those in add-on MATLAB toolboxes. These make Octave a better package for econometrics than base MATLAB. Creel (2014) is a set of econometrics notes based with applications in Octave. Examples, data-sets and programs are available on the web with the notes.

Two useful references for those making the transition from MATLAB to octave are

1. `http://wiki.octave.org/FAQ#How_is_Octave_different_from_Matlab.3F` and
2. Differences between Octave and MATLAB (Wikibooks.)

Many of the programs and functions in the LeSage package can be made to run in Octave. Many of the LeSage programs will crash and report a missing `fcnchk()` function. Create a file containing the following and save it on the Octave path. I have saved it in the `util` subdirectory of the LeSage toolbox.

```
function f=fcnchk(x, n)
f = x;
end
```

You will also get "Short-circuit & and | operators" warnings when you are using the LeSage toolbox. To avoid these add

```
do_braindead_shortcircuit_evaluation(1)
```

to your .octavrc file or otherwise run this instruction before you call any LeSage functions.

10.2 Scilab

Scilab is another free matrix manipulation language available from `www.scilab.org`. Scilab has the same basic functionality as MATLAB. Scilab syntax is different to that of MATLAB. Scilab programs will need considerable editing before they could be used in MATLAB. Scilab contains a utility for translating MATLAB programs to Scilab. Campbell et al. (2006) is a good introduction to Scilab and contains a lot of tutorial material. There is also an econometrics toolbox for Scilab called GROCER. While this package is partly derived from the LeSage package it has been updated and includes a lot of additional features.

One may well ask which is the best program. MATLAB is definitely the market leader in the field. It is very much embedded in the scientific/engineering fields with branches in Finance. It has applications in advanced macroeconomics and is a suitable tool for empirical research. Octave is very similar to MATLAB. Octave has better facilities than base MATLAB. The combination of Scilab and GROCER makes a most interesting tool for economics and deserves to be better known. If I was working in a MATLAB environment where good support was available in-house I would not try anything else. If I wanted a program to run my MATLAB programs at home and did not want to go to the expense of acquiring a licence for basic MATLAB and tools I would try Octave first. If I was just interested in some private empirical research Scilab would be worth trying. Scilab and Grocer should be used more widely in economics/econometrics Experience gained in programming Octave or Scilab would transfer easily to MATLAB.

10.3 R

The third alternative that I have used for this kind of work is R (R Core Team, 2014). R has been more suitable for much of the work that I have been doing recently.

R is "GNU S", a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and non-linear modelling, statistical tests, time series analysis, classification, clustering, etc. More information is available from The Comprehensive R Archive Network (CRAN) at `http://www.r-project.org/` or at one of its many mirror sites. Not only does R cover all aspects of statistics but it has most of the computational facilities of MATLAB. It is the package in which most academic statistical work is being completed. There is a large amount of free tutorial material available on CRAN and an increasing number of textbooks on R have been published in recent years.

If it can not be done in basic R then one is almost certain to find a solution in one of the 6000+ "official packages" on CRAN or the "unofficial packages" on other sites. R is regarded as having a steep learning curve but there are several graphical interfaces that facilitate the use of R. Summary information of the use of R in economics and finance can be seen on the Task views on the CRAN web site or on one of its many mirrors. Kleiber and Zeileis (2008) is a good starting point for econometrics in R.

APPENDIX A

Functions etc. in LeSage Econometrics Toolbox

A.1 Regression

The regression function library is in a subdirectory `regress`. This covers estimation of OLS, HCSE Models, Box-Cox Models, Limited Dependent Variable Models, Simultaneous Equations, Bayesian Models, Panel Models etc.

Table A.1: Regression in LeSage Econometrics Toolbox

Programs and Demonstrations	
ar1_like	evaluate ols model with AR1 errors log-likelihood
ar_g	MCMC estimates Bayesian heteroscedastic AR(k) model
ar_gd	An example using ar_g(),
box_lik	evaluate Box-Cox model likelihood function
boxc_trans	compute box-cox transformation
boxcox	box-cox regression using a single scalar transformation
boxcox_d	An example using box_cox(),
demo_reg	demo using most all regression functions

continued on next page

Regression in LeSage Econometrics Toolbox *continued*

program	description
felogit	computes binomial logistic regression with a one-dimensional fixed effect
felogit_demo	demonstrate use of felogit.m
felogit_lik	Compute probabilities and value of log-likelihood
garch_like	log likelihood for garch model
garch_sigt	generate garch model sigmas over time
garch_trans	function to transform garch(1,1) a0,a1,a2 garch parameters
ham_itrans	inverse transform Hamilton model parameters
ham_like	log likelihood function for Hamilton's model
ham_trans	transform Hamilton model parameters
hwhite	computes White's adjusted heteroscedastic
hwhite_d	An example of hwhite(),
ksmooth	Kim's smoothing for Hamilton() model
lad	least absolute deviations regression
lad_d	An example using lad(),
lmtest	computes LM-test for two regressions
lmtest_d	demo using lmtest()
lo_like	evaluate logit log-likelihood
logit	computes Logit Regression
logit_d	An example of logit(),
mlogit	multinomial logistic regression
mlogit_d	An example of mlogit(),
mlogit_lik	Calculates likelihood for multinomial logit regression model.
multilogit	implements multinomial logistic regression
multilogit_demo	demonstrates the use of multilogit.m
multilogit_lik	Computes value of log likelihood function for multinomial logit regression
nwest	computes Newey-West adjusted heteroscedastic-serial
nwest_d	An example using nwest(),
ols	least-squares regression
ols_d	An example using ols(),
ols_g	MCMC estimates for the Bayesian heteroscedastic linear model
ols_gcbma	MC^3 x-matrix specification for homoscedastic OLS model

continued on next page

Regression in LeSage Econometrics Toolbox *continued*

program	description
ols_gcbmad	Demo of ols_gcbma() model comparison function
ols_gd	demo of ols_g()
ols_gv	MCMC estimates for the Bayesian heteroscedastic linear model
ols_gvd	demo of ols_g()
olsar1	computes maximum likelihood ols regression for AR1 errors
olsar1_d	demonstrate olsc, olsar1 routines
olsc	computes Cochrane-Orcutt ols Regression for AR1 errors
olsc_d	demonstrate ols_corc roc
olse	OLS regression returning only residual vector
olsrs	Restricted least-squares estimation
olsrs_d	An example using olsrs(),
olst	ols with t-distributed errors
olst_d	An example using olst(),
panel_d	Demonstrates use of panel data estimation
pfixed	performs Fixed Effects Estimation for Panel Data
phaussman	prints haussman test, use for testing the specification of the fixed or
plt_eqs	plots regression actual vs predicted and residuals for:
plt_gibbs	Plots output from Gibbs sampler regression models
plt_reg	plots regression actual vs predicted and residuals
plt_tvp	Plots output using tvp regression results structures
ppooled	performs Pooled Least Squares for Panel Data(for balanced or unbalanced data)
pr_like	evaluate probit log-likelihood
prandom	performs Random Effects Estimation for Panel Data
probit	computes Probit Regression
probit_d	demo of probit()
probit_g	MCMC sampler for the Bayesian heteroscedastic Probit model
probit_gd	demo of probit_g
prt_bmao	print results from ols_gcbma function
prt_eqs	Prints output from mutliple equation regressions
prt_felogit	Prints output from felogit function
prt_gibbs	Prints output from Gibbs sampler regression models
prt_multilogit	Prints output from multilogit function

continued on next page

Regression in LeSage Econometrics Toolbox *continued*

program	description
prt_panel	Prints Panel models output
prt_reg	Prints output using regression results structures
prt_swm	Prints output from Switching regression models
prt_tvp	Prints output using tvp() regression results structures
ridge	computes Hoerl-Kennard Ridge Regression
ridge_d	An example using ridge(), bkw()
ridge_d2	An example using ridge(), bkw()
robust	robust regression using iteratively reweighted
robust_d	An example using robust(),
rtrace	Plots ntheta ridge regression estimates
sur	computes seemingly unrelated regression estimates
sur_d	An example using sur(),
switch_em	Switching Regime regression (EM-estimation)
switch_emd	Demo of switch_em
theil	computes Theil-Goldberger mixed estimator
theil_d	An example using theil(),
thsls	computes Three-Stage Least-squares Regression
thsls_d	An example using thsls(),
to_llike	evaluate tobit log-likelihood
to_rlike	evaluate tobit log-likelihood
tobit	computes Tobit Regression
tobit_d	An example using tobit()
tobit_d2	An example using tobit()
tobit_g	MCMC sampler for Bayesian Tobit model
tobit_gd	An example using tobit_g()
tobit_gd2	An example using tobit_g()
tsls	computes Two-Stage Least-squares Regression
tsls_d	An example using tsls(),
tvp	time-varying parameter maximum likelihood estimation
tvp_d	An example using tvp(),
tvp_garch	time-varying parameter estimation with garch(1,1) errors
tvp_garch_like	log likelihood for tvp_garch model
tvp_garchd	An example using tvp_garch(),

continued on next page

Regression in LeSage Econometrics Toolbox *continued*

program	description
tvp_like	returns -log likelihood function for tvp model
tvp_markov	time-varying parameter model with Markov switching error variances
tvp_markov_lik	log-likelihood for Markov-switching TVP model
tvp_markovd	An example using tvp_markov(),
tvp_markovd2	An example using tvp_markov(), and tvp_garch()
tvp_zglike	returns -log likelihood function for tvp model with Zellner's g-prior
waldf	computes Wald F-test for two regressions
waldf_d	demo using waldf()

A.2 Diagnostics

Regression/residual diagnostics from the `diag` directory are included in Table **??**. Included are the usual ARCH, Breusch-Pagan, Q-test, cusum, BKW, recursive residuals etc.

Table A.2: Regression/Residual Diagnostics in LeSage Econometrics Toolbox

Diagnostics and Demonstrations	
arch	computes a test for ARCH(p)
arch_d	demo of arch() test for ARCH(p)
bkw	computes and prints BKW collinearity diagnostics
bkw_d	demo of bkw()
bpagan	Breusch-Pagan heteroscedasticity test
bpagan_d	An example of bpagan(),
cusums	computes cusum-squares test
cusums_d	demo of rec_resid()
dfbeta	computes BKW (influential observation diagnostics)
dfbeta_d	demo of dfbeta(), plt_dfb()
rdiagnose	computes regression diagnostic measures (see RETURNS)
rdiagnose_d	demo of rdiagnose()
plt_cus	plots cusum squared tests
plt_dfb	plots BKW influential observation diagnostics
plt_dff	plots BKW influential observation diagnostics

continued on next page

Regression in LeSage Econometrics Toolbox *continued*

program	description
qstat2	computes the Ljung-Box Q test for AR(p)
qstat2_d	demo of qstat2() Ljunbg-Box Q test
rdiag	residual analysis plots
rdiag_d	demo of rdiagn()
recresid	compute recursive residuals
recresid_d	demo of recresid()
studentize	If x is a vector, subtract its mean and divide
unstudentize	returns reverse studentized vector
unstudentize_d	demonstrate unstudentize function

A.3 Unit Roots and Cointegration

The unit root and co-integration tests and estimation procedures include Dickey-Fuller, Philips-Peron, HEGY seasonal unit roots and Johansen.

Table A.3: Unit Roots and Cointegration in LeSage Econometrics Toolbox

Pograms and Demonstrations	
acf	Estimate the coefficients of the autocorrelation
adf	carry out DF tests on a time-series vector
adf_d	demonstrate the use of adf()
c_sja	find critical values for Johansen maximum eigenvalue statistic
c_sjt	find critical values for Johansen trace statistic
cadf	compute augmented Dickey-Fuller statistic for residuals
cadf_d	demonstrate the use of cadf()
crthegy	return critical values for the hegy statistics
detrend	detrend a matrix y of time-series using regression
hegy	performs the Hylleberg, Engle, Granger and Yoo(1990)
hegy_d	demo program for hegy
johansen	perform Johansen cointegration tests
johansen_d	demonstrate the use of johansen()
phillips	compute Phillips-Perron test of the unit-root hypothesis
phillips_d	demonstrate the use of phillips()

continued on next page

Unit Roots and Cointegration LeSage Econometrics Toolbox *continued*

program	description
prt_coint	Prints output from co-integration tests
ptrend	produce an explanatory variables matrix
rztcrit	return critical values for the Zt statistic used in cadf()
ztcrit	return critical values for the Zt statistic used in adf()

A.4 Vector Autoregression — Classical/Bayesian

Included are the standard var estimation routines, impulse response functions, causality tests and various Bayesian procedures.

Table A.4: Vector Autoregression in LeSage Econometrics Toolbox

Pograms and Demonstrations	
ar	autoregressive model estimation
arf	autoregressive model forecasts
arf_d	demo of autoregressive model forecasts
becm	performs Bayesian error correction model estimation
becm_d	demonstrate the use of becm
becm_g	Gibbs sampling estimates for Bayesian error correction
becm_gd	An example of using becm_g(),
becmf	estimates a Bayesian error correction model of order n
becmf_d	demonstrate the use of becmf
becmf_g	Gibbs sampling forecasts for Bayesian error
becmf_gd	An example of using becmf_g(),
bvar	Performs a Bayesian vector autoregression of order n
bvar_d	An example of using bvar(),
bvar_g	Gibbs sampling estimates for Bayesian vector
bvar_gd	An example of using bvar_g(),
bvarf	Estimates a Bayesian vector autoregression of order n
bvarf_d	An example of using bvarf(),
bvarf_g	Gibbs sampling forecasts for Bayesian vector
bvarf_gd	An example of using bvarf_g(),
ecm	performs error correction model estimation

continued on next page

Vector Autoregression LeSage Econometrics Toolbox *continued*

program	description
ecm_d	demonstrate the use of ecm()
ecmf	estimates an error correction model of order n
ecmf_d	demonstrate the use of ecmf
irf	Calculates Impulse Response Function for VAR
irf_d	An example of using irf
irf_d2	An example of using irf
lrratio	performs likelihood ratio test for var model
lrratio_d	demonstrate the use of lrratio()
pftest	prints VAR model ftests
pftest_d	An example of using pftest
pgranger	prints VAR model Granger-causality results
plt_var	plots VAR model actual vs predicted and residuals
plt_varg	Plots Gibbs sampled VAR model results
prt_var	Prints vector autoregressive models output
prt_varg	Prints vector autoregression output
recm	performs Bayesian error correction model estimation
recm_d	demonstrate the use of recm
recm_g	Gibbs sampling estimates for Bayesian error correction
recm_gd	An example of using recm_g function
recmf	Estimates a Bayesian error correction model of order n
recmf_d	An example of using recmf(),
recmf_g	Gibbs sampling forecasts for Bayesian error correction
recmf_gd	An example of using recmf_g function
rvar	Estimates a Bayesian vector autoregressive model
rvar_d	An example of using rvar() function
rvar_g	Gibbs estimates for a Bayesian vector autoregressive
rvar_gd	An example of using rvar_g function
rvarb	Estimates a Bayesian vector autoregressive model
rvarf	Estimates a Bayesian autoregressive model of order n
rvarf_d	An example of using rvarf(),
rvarf_g	Gibbs forecasts for a Bayesian vector autoregressive
rvarf_gd	An example of using rvarf_g(),
svar	svar verifies the identification conditions for a given structural form

continued on next page

Vector Autoregression LeSage Econometrics Toolbox *continued*

program	description
svar_d	An example of using svar
vare	performs vector autogressive estimation
var_d	An example of using var, pgranger, prt_var,plt_var
varf	estimates a vector autoregression of order n
varf_d	An example of using varf()

A.5 Markov chain Monte Carlo (MCMC)

Table A.5: MCMC in LeSage Econometrics Toolbox

Pograms and Demonstrations	
apm	computes Geweke's chi-squared test for two sets of MCMC sample draws
apm_d	demo of apm()
coda	MCMC convergence diagnostics, modeled after Splus coda
coda_d	demo of coda()
momentg	computes Gewke's convergence diagnostics NSE and RNE
momentg_d	demo of momentg()
prt_coda	Prints output from Gibbs sampler coda diagnostics
raftery	MATLAB version of Gibbsit by Raftery and Lewis (1991)
raftery_d	demo of raftery()

A.6 Time Series Aggregation/Disaggregation

This is the best collection of interpolation disaggregation and aggregation routines that I have encountered.

Table A.6: Unit Roots and Cointegration in LeSage Econometrics Toolbox

Pograms and Demonstrations	
aggreg	Generate a temporal aggregation matrix
aggreg_test	Test of temporal aggregation

continued on next page

Time Series Aggregation/Disaggregation in LeSage Econometrics Toolbox *continued*

program	description
aggreg_v	Generate a temporal aggregation vector
bal	Proportional adjustment of y to a given total z
bal_d	Demo of bal()
bfl	Temporal disaggregation using the Boot-Feibes-Lisman method
bfl_d	demo of bfl()
calt	Phi-weights of ARIMA model in matrix form
chowlin	Temporal disaggregation using the Chow-Lin method
chowlin_d	demo of chowlin()
chowlin_fix	Temporal disaggregation using the Chow-Lin method
conta	determine number of non-f elements in polynomial
denton	Multivariate temporal disaggregation with transversal constraint
denton_d	Demo of denton()
denton_uni	Temporal disaggregation using the Denton method
denton_uni_d	Demo of denton_uni()
denton_uni_prop	Temporal disaggregation: Denton method, proportional variant
denton_uni_prop_d	Demo of denton_uni_prop()
desvec	Creates a matrix unstacking a vector
dif	Generate the difference operator in matrix form
difonzo	Multivariate temporal disaggregation with transversal constraint
difonzo_d	Demo of difonzo()
fernandez	Temporal disaggregation using the Fernandez method
fernandez_d	demo of fernandez()
guerrero	ARIMA-based temporal disaggregation: Guerrero method
guerrero_d	demo of guerrero()
inter_xls	Interface via Excel Link for univariate temporal disaggregation
inter_xls_d	demo of inter_xls()
litterman	Temporal disaggregation using the Litterman method
litterman_d	demo of litterman()
litterman_fix	Temporal disaggregation using the Litterman method
minter_xls	Interface via Excel Link for multivariate temporal disaggregation
movingsum	Accumulates h consecutive periods of a vector nx1
mtasa	Compute the year-on-year rate of growth of a vector time series
mtd_plot	Graphic output of multivariate temporal disaggregation methods

continued on next page

Time Series Aggregation/Disaggregation in LeSage Econometrics Toolbox *continued*

program	description
mtd_print	Save output of multivariate temporal disaggregation methods
numpar	determine the number of non-zero values of ARIMA model
rossi	Multivariate temporal disaggregation with transversal constraint
rossi_d	Demo of rossi()
ssc	Temporal disaggregation using the dynamic Chow-Lin method
ssc_d	demo of ssc()
ssc_fix	Temporal disaggregation using the dynamic Chow-Lin method
sw	Temporal disaggregation using the Stram-Wei method.
sw_d	demo of sw()
tasa	Compute the year-on-year rate of growth
td_plot	Generate graphic output of temporal disaggregation methods
td_print	Generate output of temporal disaggregation methods
td_print_g	Generate output of temporal disaggregation methods.
tduni_plot	Generate graphic output of the BFL or Denton
tduni_print	Save output of BFL or Denton temporal disaggregation methods
temporal_agg	Temporal aggregation of a time series
vdp	Balancing by means of van der Ploeg method
vdp_d	Demo of vdp()

A.7 Optimization Programs

The optimization function `fminsearch` in recent version of MATLAB employed as in Chapter 9.

Table A.7: Unit Roots and Cointegration in LeSage Econometrics Toolbox

Pograms and Demonstrations	
Optimization functions	
banana	Banana function for testing optimization
banana_d	Demonstrate optimization functions
dfp_min	DFP minimization routine to minimize func
dfp_mind	Demonstrate dfp_min optimization function

continued on next page

Time Series Aggregation/Disaggregation in LeSage Econometrics Toolbox *continued*

program	description
frpr_min	Fletcher,Reeves,Polak,Ribiere minimization routine to minimize func
frpr_mind	Demonstrate frpr_min optimization function
hessian	Computes finite difference Hessian
maxlik	minimize a log likelihood function
optim1_d	An example using dfp_min, frpr_min, pow_min, maxlik
optim2_d	An example using fmin function
optim3_d	An example of optimization
pow_min	Powell minimization routine to minimize func
pow_mind	Demonstrate pow_min optimization function
to_like1	evaluate tobit log-likelihood
to_like2	evaluate tobit log-likelihood
to_liked	evaluate tobit log-likelihood
tvp_beta	generate tvp model betas and forecast error variance
tvp_like1	returns -log likelihood function for tvp model
tvp_like2	returns log likelihood function for tvp model

A.8 Plots and Graphs

This section contains functions that produce graphs of interest in econometrics.

Table A.8: Plots and Graphs in LeSage Econometrics Toolbox

Pograms and Demonstrations	
histo	Plot a histogram
k_pdf	Plots a univariate kernel density
pairs	Pairwise scatter plots of the columns of x
pairs_d	demo of pairs (pairwise scatterplots)
plt_turns	plot turning points in a time series
plt_turnsd	demo of fturns()
pltdens	Draw a nonparametric density estimate.
pltdens_d	demo of pltdens (non-parameter density plot)
spyc	colorful version of matlab spy.c function

continued on next page

Plots and Graphs in LeSage Econometrics Toolbox *continued*

program	description
tsplot	time-series plot with dates and labels
tsplot_d	demo of tsplot (time-series plot)

A.9 Statistical Functions

This functions listed in table A.8 calculate density functions, distribution functions, quantiles and simulate random samples for beta, binomial, chisquared, F, gamma, Hypergeometric, lognormal, logistic, normal, left- and right-truncated normal, multivariate normal, Poison, t and Wishart distributions. This is essential material if you do not have access to the statistics toolbox in MATLAB.

Table A.9: Statistical Functions in LeSage Econometrics Toolbox

Pograms and Demonstrations	
beta_cdf	cdf of the beta distribution
beta_d	demo of beta distribution functions
beta_inv	inverse of the cdf (quantile) of the beta(a,b) distribution
beta_pdf	pdf of the beta(a,b) distribution
beta_rnd	random draws from the beta(a,b) distribution
bincoef	generate binomial coefficients
bingen	generate binomial probability
bino_cdf	cdf at x of the binomial(n,p) distribution
bino_d	demo of binomial distribution functions
bino_pdf	pdf at x of the binomial(n,p) distribution
bino_rnd	random sampling from a binomial distribution
chis_cdf	returns the cdf at x of the chisquared(n) distribution
chis_d	demo of chis-squared distribution functions
chis_inv	returns the inverse (quantile) at x of the chisq(n) distribution
chis_pdf	returns the pdf at x of the chisquared(n) distribution
chis_prb	computes the chi-squared probability function
chis_rnd	generates random chi-squared deviates
com_size	makes a,b scalars equal to constant matrices size(x)
demo_distr	demo all distribution functions

continued on next page

Statistical Functions in LeSage Econometrics Toolbox *continued*

program	description
fdis_cdf	returns cdf at x of the F(a,b) distribution
fdis_d	demo of F-distribution functions
fdis_inv	returns inverse (quantile) at x of the F(a,b) distribution
fdis_pdf	returns pdf at x of the F(a,b) distribution
fdis_prb	computes f-distribution probabilities
fdis_rnd	returns random draws from the F(a,b) distribution
gamm_cdf	returns the cdf at x of the gamma(a) distribution
gamm_d	demo of gamma distribution functions
gamm_inv	returns the inverse of the cdf at p of the gamma(a) distribution
gamm_pdf	returns the pdf at x of the gamma(a) distribution
gamm_rnd	a matrix of random draws from the gamma distribution
hypg_cdf	hypergeometric cdf function
hypg_d	demo of Hypergeometric distribution functions
hypg_inv	hypergeometric inverse (quantile) function
hypg_pdf	hypergeometric pdf function
hypg_rnd	hypergeometric random draws
is_scalar	determines if argument x is scalar
logn_cdf	cdf of the lognormal distribution
logn_d	demo of log-normal distribution functions
logn_inv	inverse cdf (quantile) of the lognormal distribution
logn_pdf	pdf of the lognormal distribution
logn_rnd	random draws from the lognormal distribution
logt_cdf	cdf of the logistic distribution
logt_d	demo of logistic distribution functions
logt_inv	inv of the logistic distribution
logt_pdf	pdf of the logistic distribution at x
logt_rnd	random draws from the logistic distribution
norm_cdf	computes the cumulative normal distribution
norm_crnd	random numbers from a contaminated normal distribution
norm_inv	computes the quantile (inverse of the CDF)
norm_pdf	computes the normal probability density function
norm_prb	computes normal probability given
norm_prbd	demo of norm_prb function

continued on next page

Statistical Functions in LeSage Econometrics Toolbox *continued*

program	description
norm_rnd	random multivariate random vector based on
normc_d	demo of contaminated normal random numbers
normlt_d	demo of left-truncated normal random numbers
normlt_inv	compute inverse of cdf for left-truncated normal
normlt_rnd	compute random draws from a left-truncated normal
normrt_d	demo of right-truncated normal random numbers
normrt_inv	compute inverse of cdf for right-truncated normal
normrt_rnd	compute random draws from a right-truncated normal
normt_d	demo of truncated normal random numbers
normt_inv	compute inverse of cdf for truncated normal
normt_rnd	random draws from a normal truncated to (left,right) interval
pois_cdf	computes the cumulative distribution function at x
pois_d	demo of poisson distribution functions
pois_inv	computes the quantile (inverse of the cdf) at x
pois_pdf	computes the probability density function at x
pois_rnd	generate random draws from the possion distribution
quantile	compute empirical quantile (percentile).
stdn_cdf	computes the standard normal cumulative
stdn_d	demo of standard normal distribution functions
stdn_inv	computes the quantile (inverse of the CDF)
stdn_pdf	computes the standard normal probability density
tdis_cdf	returns cdf at x of the t(n) distribution
tdis_d	demo of Student t-distribution functions
tdis_inv	returns the inverse (quantile) at x of the t(n) distribution
tdis_pdf	returns the pdf at x of the t(n) distribution
tdis_prb	calculates t-probabilities for elements in x-vector
tdis_rnd	returns random draws from the t(n) distribution
trunc_d	demo of truncated normal draws
unif_d	demo of uniform distribution functions
unif_rnd	returns a uniform random number between a,b
wish_d	demo of random draws from a Wishart distribution
wish_rnd	generate random wishart matrix

A.10 Utilities

This is a collection of functions that an economist might find useful.

Table A.10: Utilities in LeSage Econometrics Toolbox

Pograms and Demonstrations	
accumulate	accumulates column elements of a matrix x
blockdiag	Construct a block-diagonal matrix with the inputs on the diagonals.
cal	create a time-series calendar structure variable that
cal_d	An example of using cal()
ccorr1	converts matrix to correlation form with unit normal scaling.
ccorr2	converts matrix to correlation form with unit length scaling.
cols	return columns in a matrix x
crlag	circular lag function
cumprodc	compute cumulative product of each column
cumsumc	compute cumulative sum of each column
delif	select values of x for which cond is false
diagrv	replaces main diagonal of a square matrix
dmult	computes the product of diag(A) and B
find_big	finds rows where at least one element is > number
find_bigd	An example of using find_big()
findnear	finds element in the input matrix (or vector) with
fturns	finds turning points in a time-series
fturns_d	demo of fturns()
growthr	converts the matrix x to annual growth rates
ical	finds observation number associated with a year,period
ical_d	An example of using ical()
indexcat	Extract indices for y being equal to val if val is a scaler
indicator	converts the matrix x to indicator variables
invccorr	converts matrix to correlation form with
invpd	A dummy function to mimic Gauss invpd
invpd_d	An example of using invpd()
kernel_n	normal kernel density estimate
lag	creates a matrix or vector of lagged values
levels	produces a variable vector of factor levels

continued on next page

Utilities in LeSage Econometrics Toolbox *continued*

program	description
lprint	print an (nobs x nvar) matrix in LaTeX table format
lprint_d	demo of lprint()
lprintf	Prints a matrix of data with a criteria-based symbol next
lprintf_d	demo of lprintf()
make_contents	makes pretty contents.m files for the Econometrics Toolbox
matadd	performs matrix addition even if matrices
matdiv	performs matrix division even if matrices
matmul	performs matrix multiplication even if matrices
matsub	performs matrix subtraction even if matrices
mlag	generates a matrix of n lags from a matrix (or vector)
mprint	print an (nobs x nvar) matrix in formatted form
mprint3	Pretty-prints a set of matrices together by stacking the
mprint3_d	An example of using mprint3
mprint_d	demo of mprint()
mth2qtr	converts monthly time-series to quarterly averages
nclag	Generates a matrix of lags from a matrix containing
plt	Plots results structures returned by most functions
prodc	compute product of each column
prt	Prints results structures returned by most functions
recserar	computes a vector of autoregressive recursive series
recsercp	computes a recursive series involving products
roundoff	Rounds a number(vector) to a specified number of decimal places
rows	return rows in a matrix x
sacf	find sample autocorrelation coefficients
sacf_d	demo of sacf()
sdiff	generates a vector or matrix of lags
sdummy	creates a matrix of seasonal dummy variables
selif	select values of x for which cond is true
seqa	produce a sequence of values
seqm	produce a sequence of values
shist	spline-smoothed plot of a histogram
spacf	find sample partial autocorrelation coefficients
spacf_d	demo of spacf()

continued on next page

Utilities in LeSage Econometrics Toolbox *continued*

program	description
stdc	standard deviation of each column
sumc	compute sum of each column
tally	calculate frequencies of distinct levels in x
tdiff	produce matrix differences
trimc	return a matrix (or vector) x stripped of the specified columns.
trimr	return a matrix (or vector) x stripped of the specified rows.
tsdate	produce a time-series date string for an observation
tsdate_d	demonstrate tsdate functions
tsprint	print time-series matrix or vector with dates and column labels
tsprint_d	Examples of using tsprint()
unsort	takes a sorted vector (or matrix) and sort index as input
unsort_d	demo of unsort()
util_d	demonstrate some of the utility functions
vec	creates a column vector by stacking columns of x
vech	creates a column vector by stacking columns of x
vecr	creates a column vector by stacking rows of x
vprob	returns val = (1/sqrt(2*pi*he))*exp(-0.5*ev*ev/he)
xdiagonal	spreads an nxk observation matrix x out on
yvector	repeats an nx1 vector y n times to form

APPENDIX B

Data Sets

Data files used in this book are embedded in the book. To extract a data file when you are reading the book using Adobe Acrobat, right-click on the (red) file name below and choose "**Save Embedded File to Disk. . .**" (or in older versions of Adobe Acrobat, "**Extract File. . .** "). You can also double-click on the file to open it immediately. If you're unable to access the attached file, or you observe miscellaneous strange behaviour, your pdf viewer might not be capable of handling file attachments properly. You should check the file for viruses an other mall-ware before using the file.

See Section 5 of (Pakin, 2011) for details of some pdf viewer problems that may occur.

1. `g10xrate.xls`. This file `g10xrate.xls` contains daily observations on the US Dollar exchange rates of the G10 countries[1]. The ten exchange rate series are in the ten columns columns. Each row then gives one observation of each series There are 6237 observations of each exchange rate. The name of each series is given in the first row at the head of each series. This construction is typical of many such files

[1]There are 11 G10 countries - Belgium, Canada, France, Germany, Italy, Japan, the Netherlands, Sweden, Switzerland, the United Kingdom and the United States. When the eleventh country joined the original name was retained.

that are encountered in economics.

2. NationalIncomeReal.csv. This file Quarterly Irish National Income Data and is used in chapter 5. It contains two columns, year and NI (National Income at constant Prices) downloaded from `www.cso.ie` on 13 October 2014.

3. tobacco.mat. This dataset iis in native MATLAB format. It is analysed in Verbeek (2008, 2012). The data set contains the following variables

   bluecol: dummy, 1 if head is blue collar worker (1)
   whitecol: dummy, 1 if head is white collar worker (1)
   flanders: dummy, 1 if living in Flanders (2)
   walloon: dummy, 1 if living in Wallonie (2)
   nkids: number of children $>$ 2 years old
   nkids2: number of children $<=$ 2 years old
   nadults: number of adults in household
   lnx: log total expenditures
   share2: budgetshare tobacco
   share1: budgetshare alcohol
   nadlnx nadults $\times$ lnx
   agelnx: age $\times$ lnx
   age: age in brackets (0—4)
   d1: dummy, 1 if share1$>$0
   d2: dummy, 1 if share2$>$0
   w1: budgetshare alcohol ,if $>$0, missing otherwise
   w2: budgetshare tobacco ,if $>$0, missing otherwise
   lnx2: lnx squared
   age2: age squared

Bibliography

Aniţţa, S., V. Arnăutu, and V. Capasso (2011). *An Introduction to Optimal Control Problems in Life Sciences and Economics*. Birkhauser. 2

Campbell, S. L., J.-P. Chancelier, and R. Nikoukhah (2006). *Modeling and Simulation in Scilab/Scicos*. Springer. 139

Cerrato, M. (2012). *The Mathematics of Derivatives Securities with Applications in MATLAB*. The Wiley Finance Series. Wiley. 2

Cottrell, A. and R. J. Lucchetti (2014, January). *Gretl Users Guide*. Department of Economics, Wake Forest University and Department of Economics, Wake Forest University. 135

Creel, M. (2014). Econometrics. http://econpapers.repec.org/paper/aubautbar/575.03.htm. 2, 138

Frain, J. C. (2010). An introduction to matlab for econometrics. Trinity Economics Papers tep0110, Trinity College Dublin, Department of Economics. i, 4, 136

Green, W. H. (2000). *Econometric Analysis* (fourth ed.). Prentice Hall. 95, 96

Green, W. H. (2012). *Econometric Analysis*. Prentice Hall. 96

Higham, D. J. and N. J. Higham (2005). *MATLAB Guide*. Society for Industrial and Applied Mathematics. 2

Huynh, H. T., V. S. Lai, and I. Soumare (2008). *Stochastic Simulation and Applications in Finance with MATLAB Programs.* Wiley. 2

Kendrick, D. A., P. Mercado, and H. M. Amman (2006). *Computational Economics.* Princeton University Press. 2

Kienitz, J. and D. Wetterau (2012). *Financial Modelling: Theory, Implementation and Practice with MATLAB Source.* The Wiley Finance Series. Wiley. 2

Kleiber, C. and A. Zeileis (2008). *Applied Econometrics with R.* Springer. 140

LeSage, J. P. (1999, October). Applied econometrics using MATLAB. 2

Lim, G. and P. McNelis (2008). *Computational Macroeconomics for the Open Economy.* MIT Press. 2

Ljungqvist, L. and T. J. Sargent (2004). *Recursive Macroeconomic Theory* (Second edition ed.). Princeton University Press. 2

Marimon, R. and A. Scott (Eds.) (1999). *Computational Methods for the Study of Dynamic Economies.* Oxford University Press. 2

Miranda, M. J. and P. L. Fackler (2002). *Applied Computational Economics and Finance.* Princeton University Press. 2

Pakin, S. (2011, March). *The attachfile package.* LaTeX package documentation. 159

Paolella, M. S. (2006). *Fundamental Probability: A Computational Approach.* Wiley. 2

Paolella, M. S. (2007). *Intermediate Probability: A Computational Approach.* Wiley. 2

Pratap, R. (2006). *Getting Started with MATLAB 7 : A Quick Introduction for Scientists and Engineers.* Oxford University Press. 2

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical Recipes: The Art of Scientific Computing* (Third ed.). Cambridge University Press. 130

R Core Team (2014). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. 75, 140

Shone, R. (2002). *Economic Dynamics* (Second ed.). Cambridge. 65

Verbeek, M. (2008). *A Guide to modern Econometrics* (Third ed.). Wiley. 136, 160

Verbeek, M. (2012). *A Guide to modern Econometrics* (Fourth ed.). Wiley. 160