MATLAB introductory course

Vitali Alexeev

University of Tasmania, School of Economics and Finance

Contents

About the MATLAB Notes

These notes are intended for students or staff at the University of Tasmania. It focuses in time series econometrics and applications in finance and portfolio theory. All references to internal functions and their syntax assume MATLAB version R2012b. It is also assumed that users have MATLAB with Statistics, Econometrics and Financial Toolboxes installed.

It is assumed that users are familiar with basic programming, i.e. `for` and `while` loops, `if .. else` statements. Some examples include the implementation of these functions and their use can be easily inferred.

Data files and external m-files and packages are located at `www.valexeev.yolasite.com/MATLABnotes.zip`. This file should be extracted into the current MATLAB project folder.

Part I

Introduction

This section is intended as an introduction to MATLAB environment, manipulating the data, import and export data from text and Excel files.

Chapter 1

MATLAB environment

1.1 Basic operations

Bellow are some commonly used commands and operations for MATLAB.

General commands		
`clear`	deletes/resets all variables	
`clear X`	deletes/resets all variable `X`	
;	suppresses output if put at the end of the line or expression	
`diary on`	record input/output to file	
`diary off`	stop recoding	
`tic ...toc`	timing the execution of some code `...`	
`clc`	clears command window	

Basic matrix operations		
+	add	
-	subtract	
*	multiply	`A*B`
/	divide	`A/B` means `A*inv(B)`
^	power/exponent	`A^3` means `A*A*A`
`A'`	transpose of `A`	
`det(A)`	determinant of `A`	
`inv(A)`	inverse of `A`	
`norm(A)`	norm of `A`	
`[v,l]=eig(A)`	eigenvalues and eigenvectors of `A`	
`svd(A)`	singular values	
`chol(A)`	Cholesky factorization	

`rank(A)`	rank of `A`	

Element-by-element operations

`.*`		
`./`		
`.^`	raise each element to a power	

Defining variables

`X=1+2-3`	defines variable `X` as the result of the expression	
`A=[1 2 3; 4 5 6]`	defines 2×3 matrix `A`	
`A=1:6`	`A=[1 2 3 4 5 6]`	
`A=1:2:6`	Starting from 1, every 2nd element until 6, e.g. `A=[1 3 5]`	
`linspace(1,6,50)`	Creates vector with 50 elements, equally spaced from 1 to 6	
`zeros(n,m)`	$n \times m$ matrix filled with 0s	
`ones(n,m)`	$n \times m$ matrix filled with 1s	
`NaN(n,m)`	$n \times m$ matrix filled with `NaN`s (Not-a-Number is a missing value in MATLAB)	also mcodeInf can be used for infinity
`rand(n,m)`	$n \times m$ matrix with random draws from uniform distribution $\sim U(0,1)$	
`randn(n,m)`	$n \times m$ matrix with random draws from normal distribution $\sim N(0,1)$	
`eye(n)`	$n \times n$ identity matrix I ("eye")	
`diag(x)`	matrix whose diagonal is the entries of x (other matrix entries are 0s)	only when x is a vector
`diag(A)`	column vector of the diagonal elements of A	only when A is a matrix
`repmat(x,n,m)`	defines an $n \times m$ matrix in which each element is x	
`whos`	lists all variables currently defined	

Basic functions

`pi`		
`exp(x)`	e^x	
`log(x)`	$\ln x$	also `log2(x)` and `log10(x)`
`sqrt(x)`	$\sqrt{x}$	
`abs(x)`	$\lvert x \rvert$	
`rem(x,y)`	Remainder of $\frac{x}{y}$	
`factorial(x)`	Remainder of $x!$	
`round(x)`	Round	
`ceil(x)`	Round up	

`floor(x)`	Round down	
`fix(x)`	Round towards zero	
`korn(A,B)`	Kronecker tensor product of `A` and `B`	

Relational operators

`x==y`	Equal	
`x<y` or `x>y`	Less (greater) than	
$x \leq y$ or $x \geq y$	Less (greater) than or equal	
$x \neq y$	Not equal	

Logical operators

`x && y`	AND	
`x & y`	element-wise AND	also `and(x,y)`
`x || y`	OR	
`x | y`	element-wise OR	also `or(x,y)`
`xor(x,y)`	eXclusive OR	
`x` and `not(x)`	NOT	
`any(x)`	True is any element is nonzero	
`all(x)`	True is all elements are nonzero	

Set membership

`unique(x)`	returns unique values in x
`union(x,y)`	set union of x and y
`intersect(x,y)`	set intersection of x and y
`setdiff(x,y)`	set difference, returns the values in x that are not in y
`setxor(x,y)`	set exclusion of x and y
`ismember(c,x)`	true for set member

Conditionals and Loops

```
1 for i=1:10
2 [procedure]
3 end
```

Performs a `for i=a:b` loop by executing the `[procedure]` for values of i from a to b with stepping of 1. Alternatively, you can use custom stepping, by using `for i=a:step:b`. For example, `for i=1:2:10` will consider only odd values of i.

```
1 while [criteria]
2 [procedure]
3 end
```

While `[criteria]` is true, the loop will continue to execute. Example, `f=0; while f<100 f=f+1; end;`

Conditional statements

```
1 if [criteria_1]
2 [procedure_1]
3 elseif ...
      [criteria_2]
4 [procedure_2]
5 elseif ...
      [criteria_3]
6 [procedure_3]
7 else
8 [procedure_4]
9 end
```

```
1 switch [var_name]
2 case ...
      [var_value_1]
3 [procedure_1]
4 case ...
      [var_value_2]
5 [procedure_2]
6 otherwise
7 [procedure_3]
8 end
```

Although can be substituted by `if...else...end` statement, this is easily applied for situations with small number of qualitative variables. For example, if you are writing a code for ADF unit root test, you may have three distinct procedures for a test a) without a constant; b) with constant; c) with constant and trend.

1.2 Organizing your MATLAB script

1.2.1 Directories and Paths

Your folder with MATLAB code may contain a large number of files, including external m-files or packages downloaded from user contributed MATLAB community, data files, saved results, your own functions or modifications of functions of others, etc. It is, therefore, advisable to create sub-folders, e.g. `\Data, \Results, \My Codes, \External Codes`. Adding the search path will allow an easy access to these files.

```
1 clear all
2 addpath('./Data','-begin');          %Add /Data without ...
      subfolders to the search path
3 addpath('./Results','-begin');
```

```
4 addpath(genpath('./External Codes')); %Add /External Codes and ...
      its subfolders to the search path
5 addpath('./My Codes','-begin');
```

Instead you can simply generate and add to path all of the current folder incluing its subfolders.

```
1 addpath(genpath('.'))
```

1.2.2 Commenting

While writing your code in MATLAB, it is a good idea to comment lines or blocks of code. It is often the case when you come back to finish or modify your code a few weeks (months?) later you may not recall exactly what the code does and it may take some time to recollect. Use `%` on a separate line preceeding your code or block of code or after the code on the same line. Use `%%` on a separate line to separate the code script into sections. Comments are also useful for program development and testing - comment out any code that does not need to run. To comment out multiple lines of code, you can use the block comment operators, `%{` and `%}` .

```
1  %% Section 1
2  % In this section we will introduce commenting inside the script
3  % This command assigns a value of 5 to x
4  x=5;
5  %{
6  x=6;
7  x=x+1;
8  %}
9  disp(x); % displays a value of x
10 %% Section 2
11 % In this section we can do something different
```

Chapter 2

Data import

There are a number of file formats that can be imported in MATLAB. This section will only cover two most commonly used formats, namely, comma separated values (CSV) files and Excel files (XLSX).

2.1 Housekeeping

Command `clear all` clears all variables from MATLAB memory.

2.1.1 Variables

String variables

Numeric variables

NaN numbers

2.2 Import data from Excel files

When the data contains mixed variables (text and numeric, e.g. company names in column headings in the first row and numeric stock prices) it is easier to import these types of data from Excel whenever possible. Storing data in XLSX format (Excel 2007 and higher) is preferred to XLS due to row and column limitations in XLS format (e.g. in Excel 2003, the maximum worksheet size is 65536 rows by 256 columns).

In the example below, we will read the file `AUSdata.xlsx` that contains 2 sheets: i) a sheet with daily price quotes for 3170 equities (common stocks, ETFs, ADRs and GDRs) currently and previously listed on the Australian Securities Exchange from 2000 to 2013.

```
datafile='AUSdata.xlsx';
[status,sheets,format] = xlsfinfo(datafile)
numSheets=size(sheets,2);
tic
[num,txt,raw] = xlsread(datafile,sheets{1});
toc
```

Price quotes have been loaded from the first sheet of the Excel file and stored in three variables `num` - containing only numerical values; `txt` - containing only text strings and `raw` - containing all values in the sheet and preserving the original Excel file layout. Variable `txt` is often not needed and may take up memory in the working space especially for large data. You can prevent the variable from being stored by substituting it with ~ :

```
[num,~,raw] = xlsread(datafile,sheets{1});
```

We can define the variables (matrix of price data and vector of stings for dates with stock information from the loaded data as follows.

```
DATES=raw(:,1);                 % cell array of strings ...
    containing dates
P=num;                          % matrix of prices

% Loading equity information
[num,txt,raw] = xlsread(datafile,sheets{2});
INFO=raw;

% Clear useless variables and save the workspace with the remaining
% variables into a .mat file
clear num txt raw format numSheets sheets status
save AUSdata -v7.3
```

2.3 Import data from text files (incomplete)

Function `dlmread` read ASCII-delimited file of numeric data into matrix.

`M = dlmread('AUSdata.csv', delimiter, ',', R, C)` reads data whose upper left corner is at row R and column C in the file. Values R and C are zero-based, so that $R = 0$, $C = 0$ specifies the first value in the file. Normally, to read the CSV file you would omit R and C, but if the file is large, you may want to read the file by parts (say every 10,000 rows) and save the output after every read.

2.4 Writing data to files (incomplete)

```
1  [N M]=size(Prices.data)
2  CVrange=21:stp:201;
3  dlmwrite(outfilecrHS, [s str2num(sectorid{s}) ...
       str2num(industryid{s}) mean(r) std(r) str2num(group1{s}) ...
       str2num(group2{s}) str2num(group3{s}) str2num(group4{s}) ...
       str2num(group5{s}) str2num(group6{s}) str2num(group7{s}) ...
       str2num(group8{s}) str2num(group9{s}) crHS], '-append');
```

Chapter 3

Plotting data

3.1 Build-in functions

3.2 Custom functions

Chapter 4

Financial Time Series objects

4.1 Merging `fints`

4.2 Data frequency

4.3 Plotting `fints` objects

Part II

Time Series models

Chapter 5

ARMA, GARCH models

```
% Demean the data before analysis
dm=mean(r);
r=r-dm;

% Pick the best MA(k)-EGARCH(p,q) model corresponding to the ...
    lowest BIC and store the optimul numbers of coefficients in ...
    $$\hat{k},\hat{p},\hat{q}$$
EGARCHout=zeros(nmax,5);
f=0;
ii=0;jj=0;pp=0;qq=0;BICopt=1000000.00;
for i=0:imax
for j=0:jmax
for p=0:pmax
for
q=1:qmax
model = garchset('R', i, 'M', j, 'P', p, 'Q', q, ...
    'VarianceModel', ModelVarType, 'Distribution', Dist, ...
    'Display', 'off'); ...
    [Coeff,Errors,LLF,Innovations,Sigmas,Summary]=garchfit(model,r);
garchdisp(Coeff, Errors);
[AIC,BIC]=aicbic(LLF,garchcount(Coeff),n);
f=f+1;
EGARCHout(f,1:5)=[i;j;p;q;BIC];
if (BIC<BICopt)
BICopt=BIC;ii=i;jj=j;pp=p;qq=q;
end;
end;
end;
end;
end;

%% Estimate the parameters of MA(k)-EGARCH(p,q) model
model = garchset('R', ii, 'M', jj, 'P', pp, 'Q', qq, ...
    'VarianceModel', ModelVarType, 'Distribution', Dist, ...
    'Display', 'off'); ...
    [Coeff,Errors,LLF,Innovations,Sigmas,Summary]=garchfit(model,r);
%garchdisp(Coeff, Errors);
```

```
30 [AIC,BIC]=aicbic(LLF,garchcount(Coeff),n);
31
32 % Store standarized residuals in $$z_1$$
33 z1 = Innovations./Sigmas; % Standardize residuals
34
35 % Simulate the return series with the coefficients from optimal ...
       MA(k)—EGARCH(p,q) model but with the residuals boostrapped ...
       from $$z_1$$
36 for i=1:B
37 [e1, s1, y1] = garchsim(Coeff, n, 1, randsample(z1,n,true));
38 Psim=ret2price((y1+dm),P0);
39 end;
```

Chapter 6

VAR models

Part III

Portfolio Construction

Chapter 7

Portfolio construction

7.1 Efficient portfolio

7.1.1 Expected Returns

7.1.2 Variance-Covariance

Using Ledoit and Wolf [2004]

7.2 Rolling frontier

Chapter 8

Performance measures

Using Ledoit and Wolf [2008]

Bibliography

Bibliography

Oliver Ledoit and Michael Wolf. Robust performance hypothesis testing with the sharpe ratio. *Journal of Empirical Finance*, 15(5):850 – 859, 2008.

Olivier Ledoit and Michael Wolf. Honey, i shrunk the sample covariance matrix. *Journal of Portfolio Management*, 30(4):110–119, 2004.

Appendix A

Using LaTeX in MATLAB

`latex(S)` returns the LaTeX representation of the symbolic expression S.

A.1 Examples

The statements

```
1  syms x
2  f = taylor(log(1+x));
3  latex(f)
```

return

```
1  ans = \frac{x^5}{5} - \frac{x^4}{4} + \frac{x^3}{3} - ...
       \frac{x^2}{2} + x
```

You can use the latex command to annotate graphs:

```
1  syms x
2  f = taylor(log(1+x));
3  ezplot(f)
4  hold on
5  title(['$' latex(f) '$'],'interpreter','latex')
6  hold off
```

Index