

Matlab Course 2017-2018

Dr. Peter A. Bebbington

University College London



peter@brainpool.ai



[bebbington](https://www.linkedin.com/in/bebbington)



[@brainpoolai](https://twitter.com/brainpoolai)



[@brainpoolai](https://www.facebook.com/brainpoolai)

July 19, 2018

REVIEW OF SESSION 1

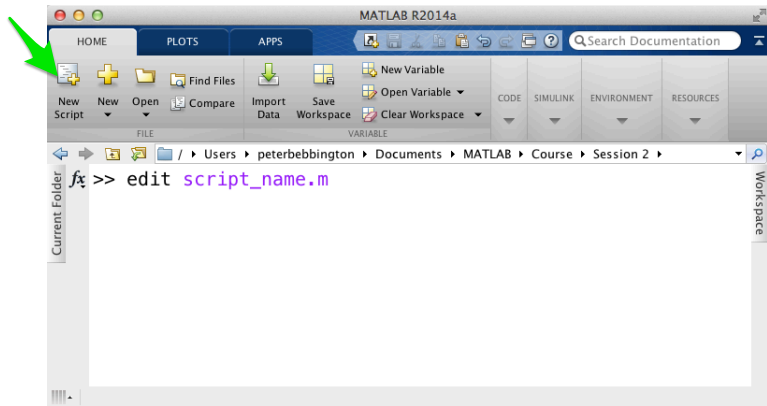
- MATLAB Desktop; Command Window, Command History, Current Directory, Launch Pad, Workspace
- Executing Basics Commands, Initialising and Defining
- Utility Commands
- Arrays, Matrices and Vectors
- Set Functions
- Addressing Array/Matrix Elements
- Solving Linear Equations
- Basic Plots

OBJECTIVE

- Scripts and Editor
- Executing Scripts
- Debugging
- Matrix Inversion
- Boolean Logic and Control Flow
- Conditional Indexing
- Vectorise
- Functions
- Data types
- File Functions
- Import Tool
- `hist_stock_data` Function
- Workspace
- Displaying Financial Data
- Stylized Facts

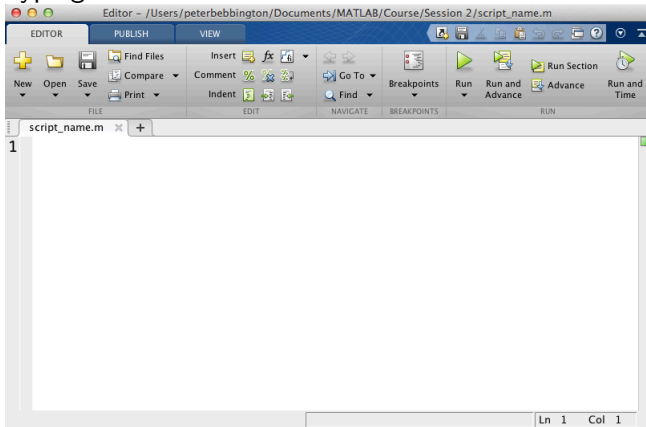
CREATING SCRIPTS

- Whilst the MATLAB console is useful for testing short code segments, you are going to want to use scripts for bigger projects.



EDITING SCRIPTS

- You can write your code in this window, then copy and paste into the console to make it run.
- If you make a mistake, you can fix it in the script, instead of re-typing it in the console window.



ARRAYS REVIEW

- An array is a collection of numbers in a sequence.
- Arrays can be multidimensional, but must always be rectangular.

$$X = [10 \ 25 \ 32 \ 47 \ 50 \ 68]$$

- Think of them as vectors or matrices if it helps.
- We can operate on the whole array at once, or on individual elements by indexing.

$$X(1) = 10$$
$$X(6) = 68$$

- Think of them as vectors or matrices if it helps.
- We can operate on whole array at once, or on individual elements by indexing.

ARRAYS REVIEW CONTD.

- With 2 dimensions, it takes 2 arguments $X(r, c)$

$X = [10 \ 25 \ 32 \ 47 \ 50 \ 68; \ 45 \ 3 \ 98 \ 56 \ 11 \ 22]$

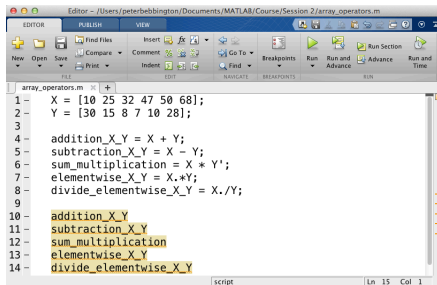
- Hence one can access an element by

$$X(1,2) = 25$$

$$X(2,6) = 22$$

ARRAY OPERATORS

- Create a new script by returning the following command
`edit array_operators.m`
- The following script performs some array operations and prints the values to the Command Window.

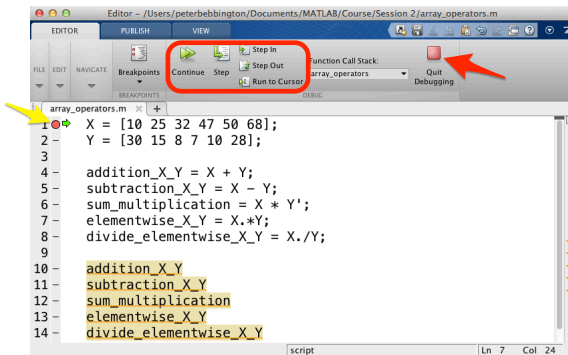


```
Editor - /Users/peterbebbington/Documents/MATLAB/Course/Session 2/array_operators.m
PUBLISH VIEW
+ Find Files Insert
New Open Save Compare Comment
Print Indent Go To
BREAKPOINTS
Run Run and Advance Run and Time
array_operators.m x +
1 X = [10 25 32 47 50 68];
2 Y = [30 15 8 7 10 28];
3
4 addition_X_Y = X + Y;
5 subtraction_X_Y = X - Y;
6 sum_multiplication = X * Y';
7 elementwise_X_Y = X.*Y;
8 divide_elementwise_X_Y = X./Y;
9
10 addition_X_Y
11 subtraction_X_Y
12 sum_multiplication
13 elementwise_X_Y
14 divide_elementwise_X_Y
script Ln 15 Col 1
```

- run the script by returning the following Command in Command Window
`>> array_operators`

DEBUGGING

- Debugging is useful to step through your code to see how it works and possibly find issues.
- Set break points (red dot) by clicking line number (yellow arrow), run the code to the break point with the F5 or run button, step through the code with buttons in red rectangle and click quit debugging button to stop (red arrow).



ARRAY OPERATORS INVERSION

- Solve (for x) system of $\bar{\bar{A}}\bar{x} = \bar{\bar{B}}$ using \backslash

$$x=A\backslash B \Rightarrow \bar{x} = \bar{\bar{A}}^{-1}\bar{\bar{B}}$$

- Solve (for x) system of $\bar{x}\bar{\bar{A}} = \bar{\bar{B}}$ using $/$

$$x=A/B \Rightarrow \bar{x} = \bar{\bar{B}}\bar{\bar{A}}^{-1}$$

BOOLEAN LOGIC

- At some point you will need your software to make decisions.
- These are made using simple Boolean logic tests. A Boolean test assumes that only two answers are possible; true (1) or false (0).

< less than

> greater than

<= less than or equal to

>= greater than or equal to

== equal to

~= not equal to

& AND

&& AND short-circuit

| OR

|| OR short-circuit

TRUTH TABLE

A	B	A&B	A B	xor(A,B)	~A
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

SHORT-CIRCUIT

The statement shown here performs an AND of two logical terms, A and B:

`A && B`

- If A equals zero, then the entire expression will evaluate to logical 0 (false), regardless of the value of B. Under these circumstances, there is no need to evaluate B because the result is already known. In this case, MATLAB short-circuits the statement by evaluating only the first term.
- A similar case is when you OR two terms and the first term is true. Again, regardless of the value of B, the statement will evaluate to true. There is no need to evaluate the second term, and MATLAB does not do so.

BOOLEAN LOGIC AND

- The `&` and `&&` operators compare two Booleans, and returns a true value only if both Booleans are true.

Example:

```
(x != 0) && (x < y)
```

```
x = 5, y=6    : 1
```

```
x = 5, y=2    : 0
```

BOOLEAN LOGIC OR

- The `|` and `||` operators compare two Booleans, and return true if either or both Booleans are true.

Example:

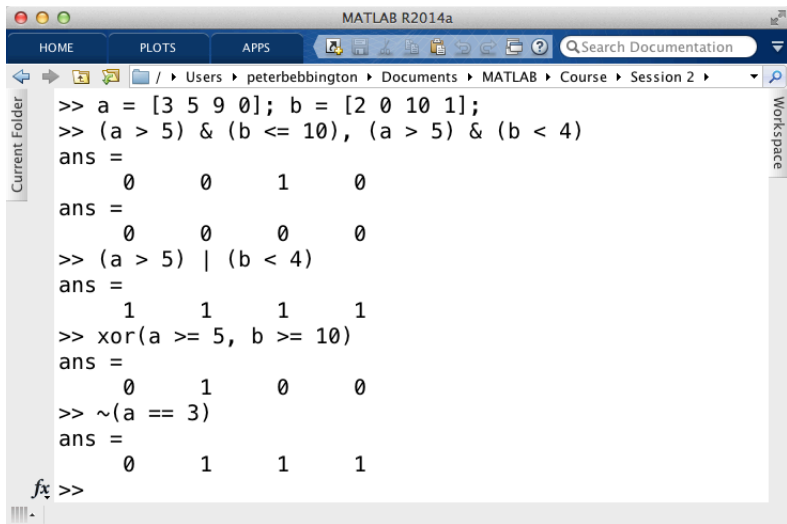
`(x != 0) || (x < y)`

`x = 5, y=6 : 1`

`x = 5, y=2 : 1`

`x = 0, y=0 : 0`

BOOLEAN LOGIC EXAMPLE



The image shows a screenshot of the MATLAB R2014a software interface. The title bar at the top reads "MATLAB R2014a". Below it is a menu bar with "HOME", "PLOTS", and "APPS". To the right of the menu bar is a search bar labeled "Search Documentation". Below the menu bar is a toolbar with various icons. The main window displays a script with the following code:

```
>> a = [3 5 9 0]; b = [2 0 10 1];  
>> (a > 5) & (b <= 10), (a > 5) & (b < 4)  
ans =  
     0     0     1     0  
ans =  
     0     0     0     0  
>> (a > 5) | (b < 4)  
ans =  
     1     1     1     1  
>> xor(a >= 5, b >= 10)  
ans =  
     0     1     0     0  
>> ~(a == 3)  
ans =  
     0     1     1     1
```

At the bottom left of the script area, there is a small icon of a folder with the letter "f" next to it, followed by ">>".

CONDITIONAL INDEXING

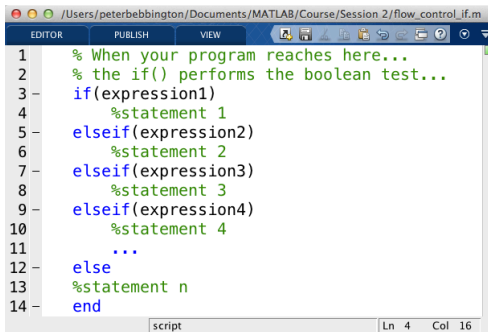
- There is a neat trick for returning the values in an array that satisfy a Boolean condition.
- Place the condition within the index brackets () of the array.

```
someArray[someArray > 10]
```

- This will return all the values in `someArray` that have a value greater than 10.

FLOW CONTROL IF-ELSE

- Not only do you want your software to be able to make decisions, you want it to act on them.
- Flow control (or Branching) allows you to tell the program how to react to given situations.
- The **if** statement checks to see if an argument is true, and only runs its block of code if it is.

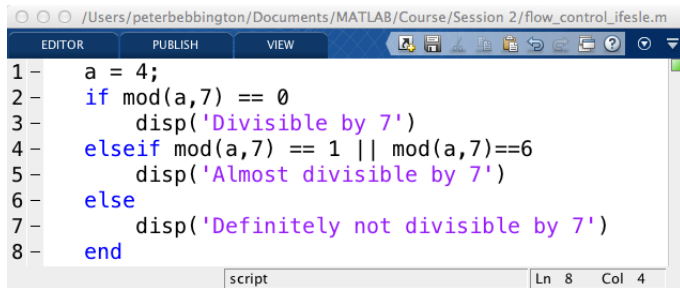


A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_if.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with the following code:

```
1 % When your program reaches here...
2 % the if() performs the boolean test...
3 if(expression1)
4     %statement 1
5 elseif(expression2)
6     %statement 2
7 elseif(expression3)
8     %statement 3
9 elseif(expression4)
10    %statement 4
11    ...
12 else
13    %statement n
14 end
```

The status bar at the bottom indicates the current position: script | Ln 4 Col 16.

IF-ELSE EXAMPLE



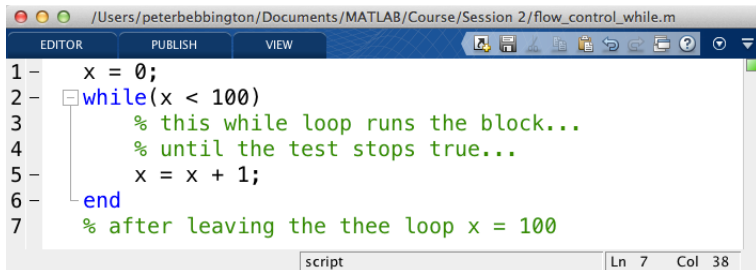
The image shows a MATLAB Editor window with the following content:

```
1 - a = 4;  
2 - if mod(a,7) == 0  
3 -     disp('Divisible by 7')  
4 - elseif mod(a,7) == 1 || mod(a,7)==6  
5 -     disp('Almost divisible by 7')  
6 - else  
7 -     disp('Definitely not divisible by 7')  
8 - end
```

The status bar at the bottom indicates the file is named 'script' and the cursor is at Line 8, Column 4.

FLOW CONTROL WHILE

- The **while** statement repeats (loops) its code block until the Boolean test condition stops being true.

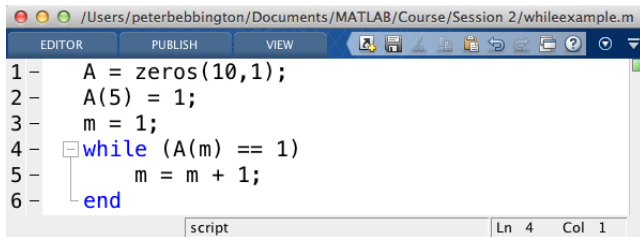


A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_while.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active. The script content is as follows:

```
1 - x = 0;
2 - while(x < 100)
3     % this while loop runs the block...
4     % until the test stops true...
5     x = x + 1;
6 - end
7     % after leaving the the loop x = 100
```

The status bar at the bottom indicates 'script' and 'Ln 7 Col 38'.

WHILE EXAMPLE



The image shows a MATLAB Editor window with the following code:

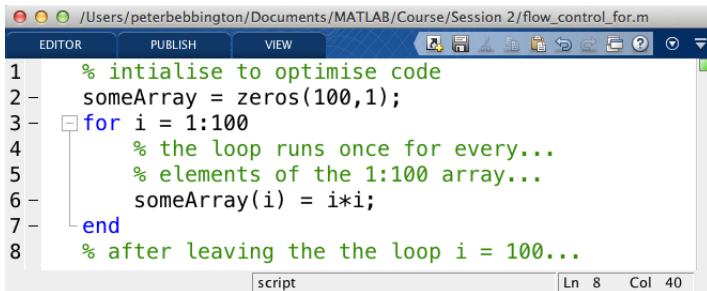
```
1 - A = zeros(10,1);  
2 - A(5) = 1;  
3 - m = 1;  
4 - while (A(m) == 1)  
5 -     m = m + 1;  
6 - end
```

The window title is `/Users/peterbebbington/Documents/MATLAB/Course/Session 2/whileexample.m`. The editor has tabs for EDITOR, PUBLISH, and VIEW. The status bar at the bottom indicates "script" and "Ln 4 Col 1".

- What is the value of `m` after while statement has finished?

FLOW CONTROL FOR

- The **for** statement takes an array as input, and loops its code block once for each element in the given array.
- A loop variable takes the value of the array element in each iteration, allowing you to do something with it.



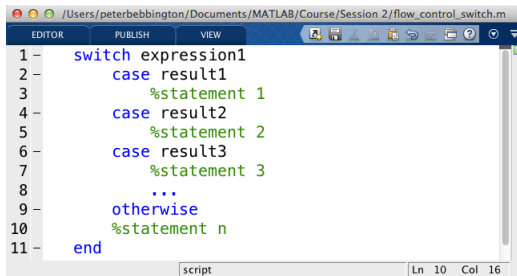
A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_for.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with the following code:

```
1 % initialise to optimise code
2 someArray = zeros(100,1);
3 for i = 1:100
4     % the loop runs once for every...
5     % elements of the 1:100 array...
6     someArray(i) = i*i;
7 end
8 % after leaving the the loop i = 100...
```

The status bar at the bottom indicates the current position: Ln 8 Col 40.

FLOW CONTROL SWITCH

- The **switch** statement takes an 'switch' variable as input, and activates the code block who's 'case' variable matches the switch.
- This could be achieved with an if loop with lots of **elseif** conditions. However, when there are allot of code blocks the switch is much faster.

A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_switch.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a MATLAB script. The script is a switch statement. Line 1: switch expression1. Line 2: case result1. Line 3: %statement 1. Line 4: case result2. Line 5: %statement 2. Line 6: case result3. Line 7: %statement 3. Line 8: ... (three dots). Line 9: otherwise. Line 10: %statement n. Line 11: end. The status bar at the bottom shows 'script' and 'Ln 10 Col 16'.

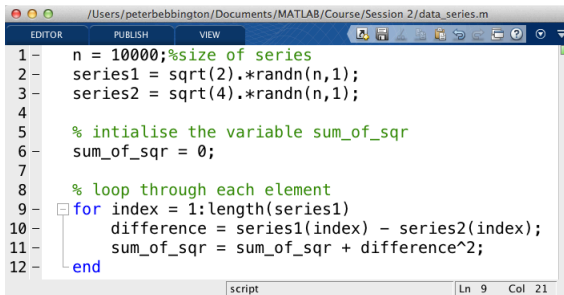
```
1 - switch expression1
2 -     case result1
3 -         %statement 1
4 -     case result2
5 -         %statement 2
6 -     case result3
7 -         %statement 3
8 -         ...
9 -     otherwise
10 -         %statement n
11 - end
```

DATA SERIES

- Find the sum of squares between two data series.
- The two series are stored in row-vector arrays;

Series 1 & Series 2

- Each has the same number of data points, which means the arrays have the same dimensions...

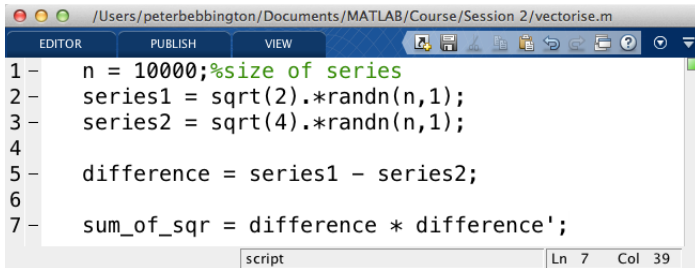


```
1 - n = 10000;%size of series
2 - series1 = sqrt(2).*randn(n,1);
3 - series2 = sqrt(4).*randn(n,1);
4
5 - % intialise the variable sum_of_sqr
6 - sum_of_sqr = 0;
7
8 - % loop through each element
9 - for index = 1:length(series1)
10 -     difference = series1(index) - series2(index);
11 -     sum_of_sqr = sum_of_sqr + difference^2;
12 - end
```

script Ln 9 Col 21

VECTORISE

- Not only is it less code, but this kind of 'vectorised' computing runs much faster than using loops.

A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/vectorise.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with 7 lines of code. The code is: 1 - n = 10000;%size of series, 2 - series1 = sqrt(2).*randn(n,1);, 3 - series2 = sqrt(4).*randn(n,1);, 4 - (blank line), 5 - difference = series1 - series2;; 6 - (blank line), 7 - sum_of_sqr = difference * difference';. The status bar at the bottom indicates 'script' and 'Ln 7 Col 39'.

```
1 - n = 10000;%size of series
2 - series1 = sqrt(2).*randn(n,1);
3 - series2 = sqrt(4).*randn(n,1);
4
5 - difference = series1 - series2;
6
7 - sum_of_sqr = difference * difference';
```

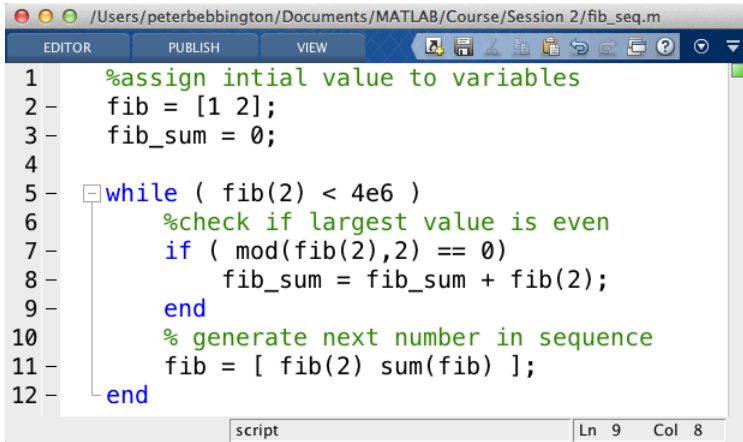
FIBONACCI EXAMPLE

- Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

- By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

SOLUTION



```
1      %assign initial value to variables
2      fib = [1 2];
3      fib_sum = 0;
4
5      while ( fib(2) < 4e6 )
6          %check if largest value is even
7          if ( mod(fib(2),2) == 0)
8              fib_sum = fib_sum + fib(2);
9          end
10         % generate next number in sequence
11         fib = [ fib(2) sum(fib) ];
12     end
```

script Ln 9 Col 8

EXERCISERS

- Write scripts to perform the following tasks:
 - ① Figure out how many terms in the sum $1+2+3+\dots$ it requires for the sum to exceed one million.
 - ② Write a program to calculate the sum $1+2+3+\dots+300$. Display the total after every 20 terms by using an if statement to check if the current number of terms is a multiple of 20.
 - ③ Write a simulator to determine the result of flipping of a coin 1000 times using a for loop, and the `rand()` command to generate a `uniform[0,1]` random number. (type '`help rand`' in console). How would you change the probability of getting heads/tails in your model? Can this have a vectorised solution?
 - ④ By modifying your simulation from (3), generate a random walk (starting from a value of 1) based on the result of each coin flip. If the coin lands on heads multiply the previous step in the walk by 1.1 to get the new step. If it lands on tails then multiply by 0.9 instead. Keep track of the result from each step. Can this have a vectorised solution?

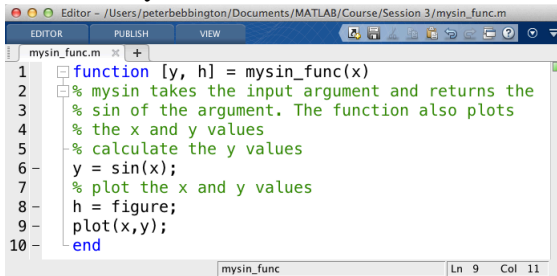
FUNCTIONS

```
function[out1, out2, ...] = filename(arg1, arg2, ...)
% help comment
statements
```

variables inside a function are local and cannot be accessed from the command prompt or another function

Example:

- `>> edit mysin_func.m`

A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 3/mysin_func.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing the code for mysin_func.m. The code is as follows:

```
1 function [y, h] = mysin_func(x)
2 % mysin takes the input argument and returns the
3 % sin of the argument. The function also plots
4 % the x and y values
5 % calculate the y values
6 y = sin(x);
7 % plot the x and y values
8 h = figure;
9 plot(x,y);
10 end
```

The status bar at the bottom shows 'mysin_func' and 'Ln 9 Col 11'.

- `>> [y, h] = mysin_func(0:pi/50:2*pi)`

DATA TYPES

Most financial data that will be imported into Matlab will come in three main forms

- .csv: Comma Separated Values
- .tsv: Tab Separated Values
- .txt: Text data in some formate

Other types of data that will be imported include .xls, .xml, .mat (can be loaded using `load data.mat` command) It is important to understand the organisation of different data types in order to understand the memory requirements for data

FILE FUNCTIONS

Command	Meaning
<code>fopen(filename)</code>	Open a file
<code>fclose(fid)</code>	Close a file
<code>fread(fid)</code>	Read binary data
<code>fwrite(fid,A,precision)</code>	Write binary data
<code>fprintf(fid,A,precision)</code>	Write formatted data
<code>fscanf(fid,format)</code>	Read formatted data
<code>sprintf(format,A)</code>	Write to a string
<code>sscanf(s,format)</code>	Read string
<code>ferror(fid)</code>	Query about errors
<code>feof(fid)</code>	Test for end of file
<code>fseek(fid,offset,origin)</code>	Set the file position indicator

IMPORT TOOL

- Simply drag and drop a “.csv” file to the command window of Matlab to import data

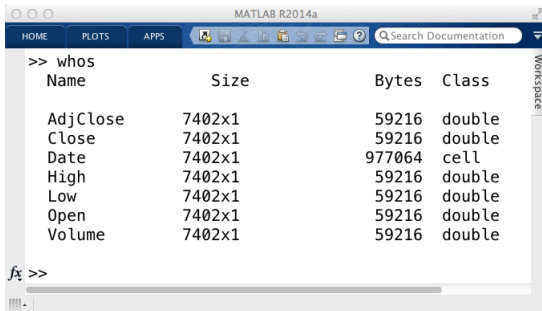
The screenshot shows the MATLAB Import Tool window. The 'Import' tab is active, and the 'VIEW' button is highlighted. The 'Column delimiters' are set to 'Comma'. The 'Import Selection' button is highlighted with a green arrow. The data preview shows a table with 8 columns: Date, Open, High, Low, Close, Volume, and Adj Close. The first row of data is highlighted with a yellow arrow. A pink arrow points to the 'Date' column header, and a yellow arrow points to the 'Adj Close' column header.

	A	B	C	D	E	F	G
	Date	Open	High	Low	Close	Volume	Adj Close
1	Date	Open	High	Low	Close	Volume	Adj Close
2	14/01/2...	538.22	546.73	537.66	546.39	11877200	546.39
3	13/01/2...	529.91	542.5	529.88	535.73	13517600	535.73
4	10/01/2...	539.83	540.8	531.11	532.94	10892000	532.94
5	09/01/2...	546.8	546.86	535.35	536.52	9969600	536.52
6	08/01/2...	538.81	545.56	538.69	543.46	9233200	543.46
7	07/01/2...	544.32	545.96	537.92	540.04	11328900	540.04
8	06/01/2...	537.45	546.8	533.6	543.93	14736100	543.93
9	03/01/2...	552.86	553.7	540.43	540.98	14016700	540.98
10	02/01/2...	555.68	557.03	552.02	553.13	8381600	553.13
11	31/12/2...	554.17	561.28	554	561.02	7967300	561.02
12	30/12/2...	557.46	560.09	552.32	554.52	9044500	554.52
13	27/12/2...	563.82	564.41	559.5	560.09	8056500	560.09
14	26/12/2...	568.1	569.5	563.38	563.9	7272500	563.9
15	24/12/2...	569.89	571.88	566.03	567.67	5984100	567.67
16	23/12/2...	568	570.72	562.76	570.09	17870600	570.09
17	20/12/2...	545.43	551.61	544.82	549.02	15548100	549.02

- You can edit; data type (pink arrow), data field name (yellow arrow) and import data (green arrow)

WORKSPACE

- Now that we have the data in Matlab we can create a workspace



The image shows a screenshot of the MATLAB R2014a workspace window. The window has a title bar with 'MATLAB R2014a' and a toolbar with icons for HOME, PLOTS, APPS, and a search bar labeled 'Search Documentation'. The main area displays the output of the 'whos' command, which lists variables in the workspace. The variables are AdjClose, Close, Date, High, Low, Open, and Volume. Each variable is shown with its Name, Size, Bytes, and Class. The classes for all variables are 'double' except for 'Date', which is 'cell'. The 'Date' variable has a size of 7402x1 and 977064 bytes, while the others have a size of 7402x1 and 59216 bytes.

```
>> whos
```

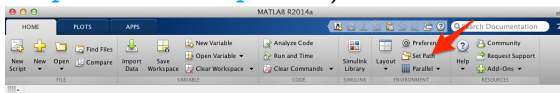
Name	Size	Bytes	Class
AdjClose	7402x1	59216	double
Close	7402x1	59216	double
Date	7402x1	977064	cell
High	7402x1	59216	double
Low	7402x1	59216	double
Open	7402x1	59216	double
Volume	7402x1	59216	double

```
fx >>
```

- `>> clear`

get_yahoo_stockdata3

- Go to following url: get_yahoo_stockdata3.m
- Download get_yahoo_stockdata3.m and put in either the folder which is the current working folder or in the “MATLAB” folder (if the second you will need to add the path of “MATLAB” in the set path option or function [addpath\('folderpath'\)](#))



- One can see that using the `get_yahoo_stockdata3` function data can be retrieved very easily by calling in the following form:

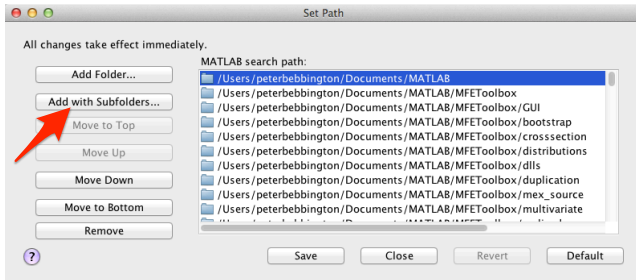
```
get_yahoo_stockdata3({'ticker1','ticker2',...},  
    'StartDate','EndDate')
```
- Lets try an example!

ECONOMETRICS TOOLBOXES

- Matlab has its own Econometrics toolbox which rich functionality
- There are also third party toolboxes that can be installed which can help for summer project if it involves time series (most will)
- The two toolboxes I use are MFEToolbox and jplv7 which can be found on <http://www.kevinsheppard.com> and <http://www.spatial-econometrics.com> respectively

INSTALLING TOOLBOXES

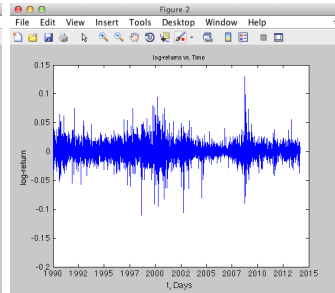
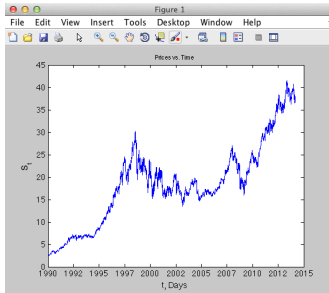
- As we did for `hist_stock_data` put the toolboxes in a folder sensible such as the Matlab folder in “My Documents” or Documents.



- Click “Add with Subfolders...” (red arrow) and Locate the two toolboxes and save.

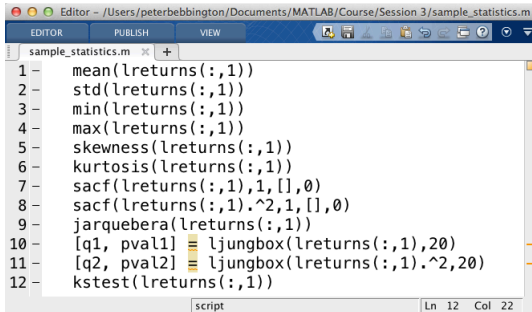
FINANCIAL SERIES

- A good to start when analysing financial time series is to simple plot the price against time quantises such as price, log-returns, volume, etc...



SAMPLE STATISTICS

- Sample statistics will give you an idea about the statistical behaviour of such as the moments and underlying distribution

A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 3/sample_statistics.m. The window has three tabs: EDITOR, PUBLISH, and VIEW, with EDITOR selected. Below the tabs is a toolbar with various icons. The main editor area shows a script named sample_statistics.m with 12 lines of MATLAB code. The code calculates various statistical measures for a variable 'lreturns'. The status bar at the bottom indicates 'script' and 'Ln 12 Col 22'.

```
1 - mean(lreturns(:,1))
2 - std(lreturns(:,1))
3 - min(lreturns(:,1))
4 - max(lreturns(:,1))
5 - skewness(lreturns(:,1))
6 - kurtosis(lreturns(:,1))
7 - sacf(lreturns(:,1),1,[],0)
8 - sacf(lreturns(:,1).^2,1,[],0)
9 - jarquebera(lreturns(:,1))
10 - [q1, pval1] == ljungbox(lreturns(:,1),20)
11 - [q2, pval2] == ljungbox(lreturns(:,1).^2,20)
12 - kstest(lreturns(:,1))
```

- Any Gaussian distributed random variable can be normalized:

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

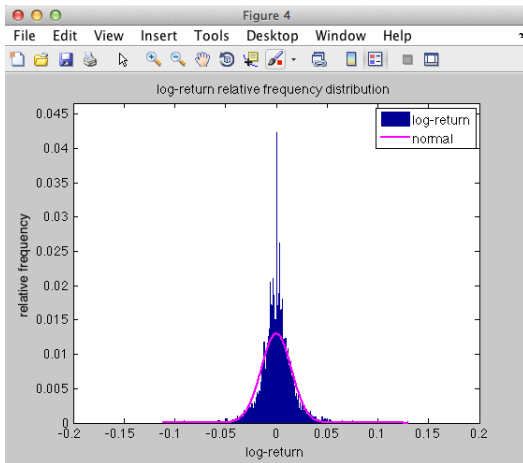
$$Z = \frac{X - \mu}{\sigma}$$

$$X = \sigma Z + \mu$$

- Analysis of return time series is better in this form for comparison between different time series such as a portfolio

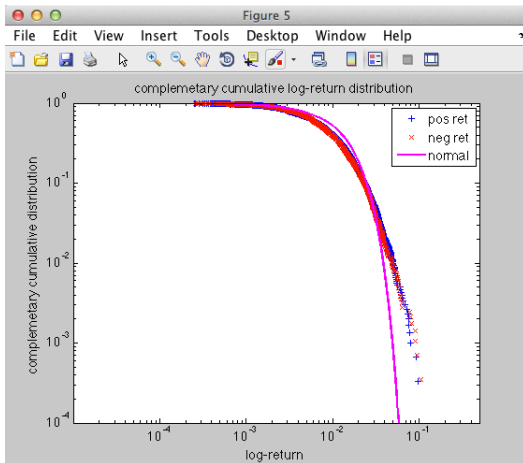
COMPARSION WITH A GUASSIAN

- Here we make a comparison of the empirical histogram against a parametrized normal distribution

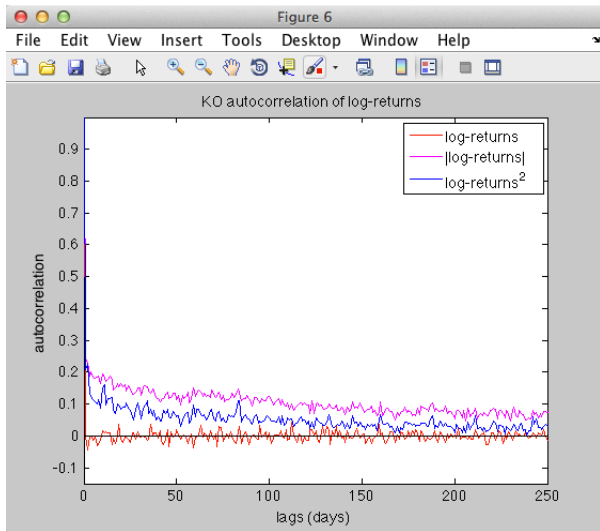


COMPLEMENTARY CUMULATIVE DISTRIBUTION

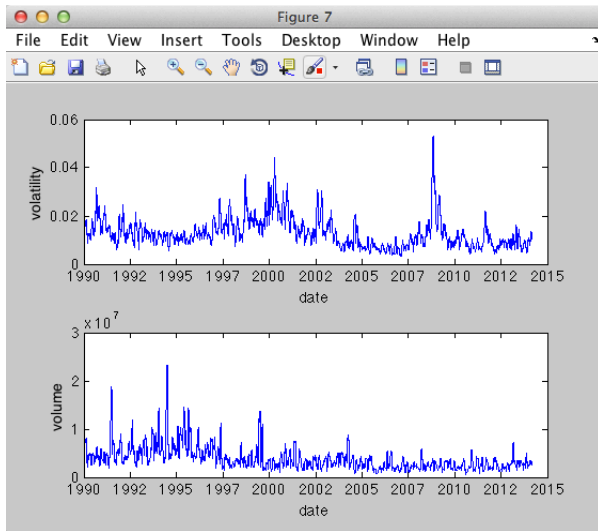
- We see in this log-log plot the empirical time series differs from the tails of a normal distribution



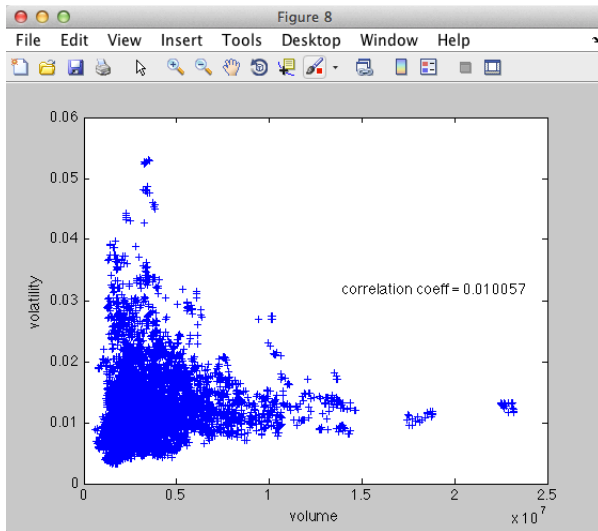
AUTOCORRELOGRAM



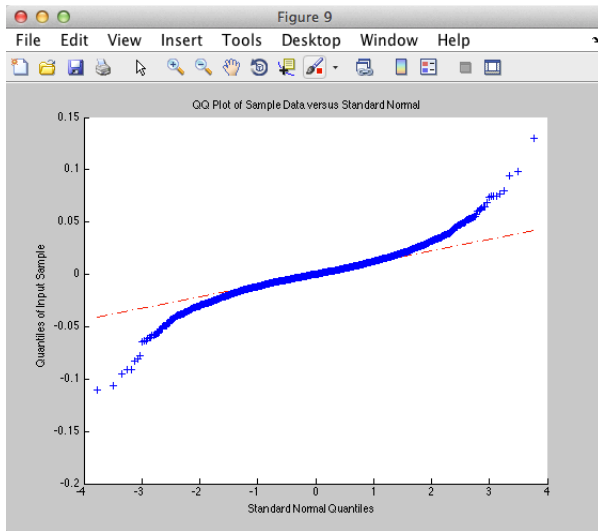
VOLATILITY



VOLATILITY Vs. VOLUME



QQPLOT



- Given some confidence level $\alpha \in (0, 1)$ Value-at-Risk is defined as the following quantile:

$$\text{VaR}_\alpha \triangleq \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\}$$

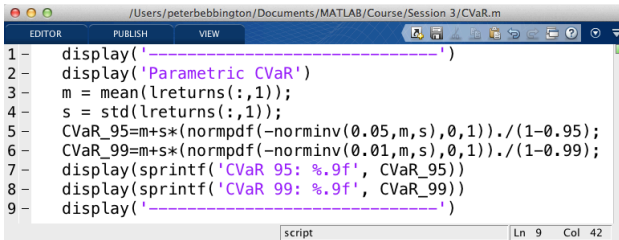
- Value-at-Risk is estimated parametrically assuming a normal distribution in Matlab with the following function
`ValueAtRisk = portvrisk(PortReturn, PortRisk, RiskThreshold, PortValue)`
- Note that one should calculate this quantity empirically to compare to the parametric estimation.

PARAMETRIC CVaR

- For a loss (L) with $\mathbb{E}[|L|] < \infty$ and distribution for F_L , the expected shortfall at the confidence level $\alpha \in (0, 1)$ is defined

$$\text{ES}_\alpha = \frac{1}{1 - \alpha} \int_\alpha^1 \text{VaR}_u(F_L) du$$

- If the loss distribution is assumed to be a normal, one can calculate the parametric CVaR as the follows

A screenshot of a MATLAB script window titled '/Users/peterbebbington/Documents/MATLAB/Course/Session 3/CVaR.m'. The window has tabs for EDITOR, PUBLISH, and VIEW. The script contains 9 lines of code that calculate the parametric CVaR for a normal distribution. It uses the mean and standard deviation of the returns to compute the 95% and 99% VaR values, and then calculates the corresponding CVaR values using the normal probability density function (normpdf) and the inverse normal cumulative distribution function (norminv). The results are displayed using sprintf and display functions.

```
1 - display('-----')
2 - display('Parametric CVaR')
3 - m = mean(lreturns(:,1));
4 - s = std(lreturns(:,1));
5 - CVaR_95=m+s*(normpdf(-norminv(0.05,m,s),0,1))./(1-0.95);
6 - CVaR_99=m+s*(normpdf(-norminv(0.01,m,s),0,1))./(1-0.99);
7 - display(sprintf('CVaR 95: %.9f', CVaR_95))
8 - display(sprintf('CVaR 99: %.9f', CVaR_99))
9 - display('-----')
```

script Ln 9 Col 42