Matlab Course 2018-2019

Dr. Peter A. Bebbington

University College London

peter@brainpool.ai

bebbington @brainpoolai @brainpoolai

July 12, 2019

REVIEW OF SESSION 1

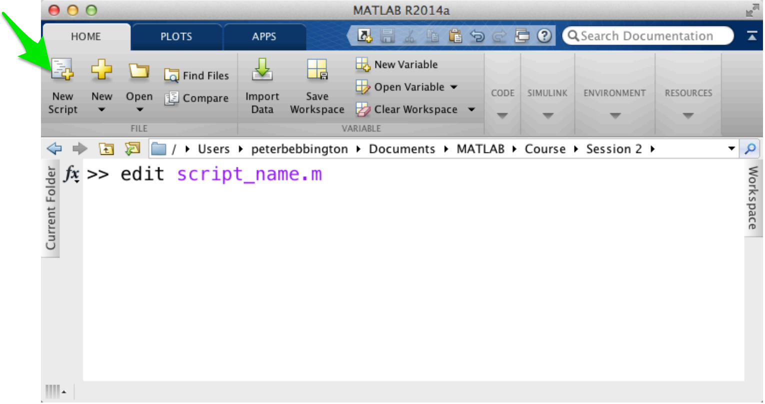
- MATLAB Desktop; Command Window, Command History, Current Directory, Launch Pad, Workspace
- Executing Basics Commands, Initialising and Defining
- Utility Commands
- Arrays, Matrices and Vectors
- Set Functions
- Addressing Array/Matrix Elements
- Solving Linear Equations
- Basic Plots

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

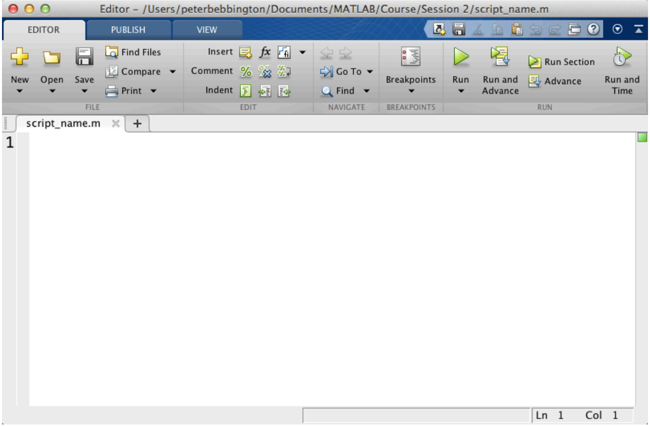
CREATING SCRIPTS

- 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

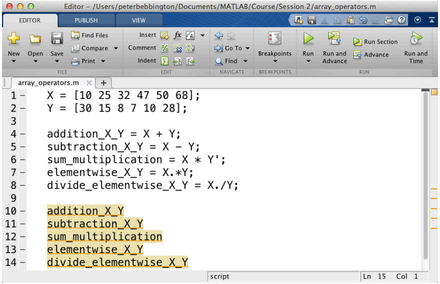
```
X(1,2) = 25
X(2,6) = 22
```

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.

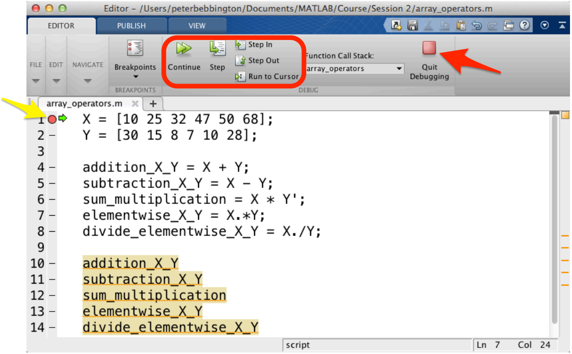


- run the script by returning the following Command in Command Window

```
>> array_operators
```

DEBUGGING

- Debugging is useful to step through your code to see how it works and possible 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 \

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

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

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

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

```
>> 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
>>
```

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.

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_if.m

```matlab
% When your program reaches here...
% the if() performs the boolean test...
if(expression1)
    %statement 1
elseif(expression2)
    %statement 2
elseif(expression3)
    %statement 3
elseif(expression4)
    %statement 4
    ...
else
%statement n
end
```

script | Ln 4 Col 16

IF-ELSE EXAMPLE

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_ifesle.m

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

FLOW CONTROL WHILE

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

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_while.m

```
    x = 0;
while(x < 100)
        % this while loop runs the block...
        % until the test stops true...
        x = x + 1;
end
% after leaving the thee loop x = 100
```

script | Ln 7 Col 38

WHILE EXAMPLE

```
A = zeros(10,1);
A(5) = 1;
m = 1;
while (A(m) == 1)
    m = m + 1;
end
```

- 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.

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_for.m

```matlab
% intialise to optimise code
someArray = zeros(100,1);
for i = 1:100
    % the loop runs once for every...
    % elements of the 1:100 array...
    someArray(i) = i*i;
end
% after leaving the the loop i = 100...
```

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.

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow_control_switch.m

```
switch expression1
    case result1
        %statement 1
    case result2
        %statement 2
    case result3
        %statement 3
        ...
    otherwise
    %statement n
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...

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/data_series.m

```
n = 10000;%size of series
series1 = sqrt(2).*randn(n,1);
series2 = sqrt(4).*randn(n,1);

% intialise the variable sum_of_sqr
sum_of_sqr = 0;

% loop through each element
for index = 1:length(series1)
    difference = series1(index) - series2(index);
    sum_of_sqr = sum_of_sqr + difference^2;
end
```

VECTORISE

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

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/vectorise.m

EDITOR PUBLISH VIEW

```
n = 10000;%size of series
series1 = sqrt(2).*randn(n,1);
series2 = sqrt(4).*randn(n,1);

difference = series1 - series2;

sum_of_sqr = difference * difference';
```

script Ln 7 Col 39

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

/Users/peterbebbington/Documents/MATLAB/Course/Session 2/fib_seq.m

```matlab
%assign intial value to variables
fib = [1 2];
fib_sum = 0;

while ( fib(2) < 4e6 )
    %check if largest value is even
    if ( mod(fib(2),2) == 0)
        fib_sum = fib_sum + fib(2);
    end
    % generate next number in sequence
    fib = [ fib(2) sum(fib) ];
end
```

EXERCISERS

- Write scripts to perform the following tasks:

1. Figure out how many terms in the sum `1+2+3+...` it requires for the sum to exceed one million.
2. Write a program to calculate the sum `1+2+3+...+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?

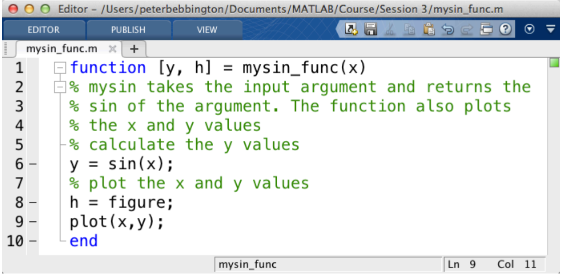
FUNCTIONS

```
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`



- `>> [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

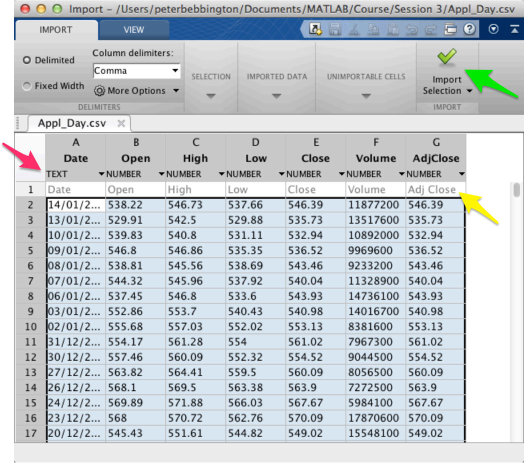
Other types of data that will be imported include .xls, .xml, .mat (can be loaded using `load data.mat` command) It is important to understand the organisation of different data types in order to understand the memory requirements for data

FILE FUNCTIONS

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

IMPORT TOOL

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

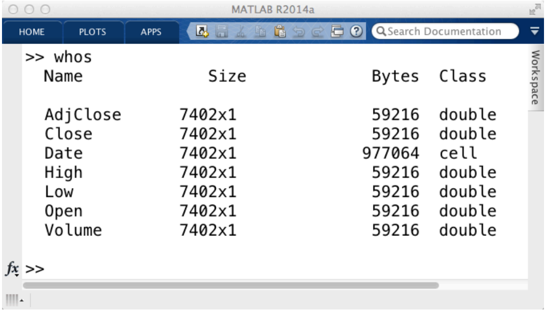


	A	B	C	D	E	F	G
	Date	Open	High	Low	Close	Volume	AdjClose
	TEXT	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER
1	Date	Open	High	Low	Close	Volume	Adj Close
2	14/01/2...	538.22	546.73	537.66	546.39	11877200	546.39
3	13/01/2...	529.91	542.5	529.88	535.73	13517600	535.73
4	10/01/2...	539.83	540.8	531.11	532.94	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

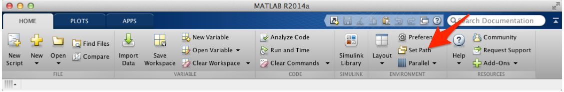
- Now that we have the data in Matlab we can create a workspace



- `>> 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')`)



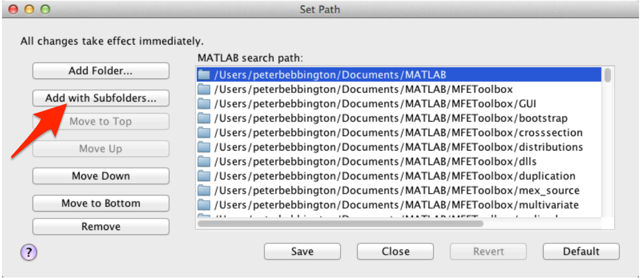
- One can see that using the get_yahoo_stockdata3 function data can be retrieved very easily by calling in the following form: `get_yahoo_stockdata3({'ticker1','ticker2',...}, 'StartDate','EndDate')`
- Lets try an example!

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 andhttp://www.spatial-econometrics.com respectively

INSTALLING TOOLBOXES

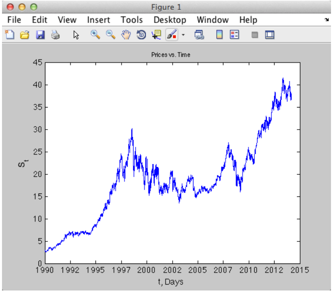
- As we did for `hist_stock_data` put the toolboxes in a folder sensible such as the Matlab folder in "My Documents" or Documents.

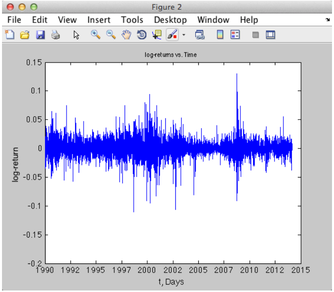


- Click "Add with Subfolders..." (red arrow) and Locate the two toolboxes and save.

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

```
mean(lreturns(:,1))
std(lreturns(:,1))
min(lreturns(:,1))
max(lreturns(:,1))
skewness(lreturns(:,1))
kurtosis(lreturns(:,1))
sacf(lreturns(:,1),1,[],0)
sacf(lreturns(:,1).^2,1,[],0)
jarquebera(lreturns(:,1))
[q1, pval1] = ljungbox(lreturns(:,1),20)
[q2, pval2] = ljungbox(lreturns(:,1).^2,20)
kstest(lreturns(:,1))
```

STANDISATION

- Any Gaussian distributed random variable can be normalized:

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

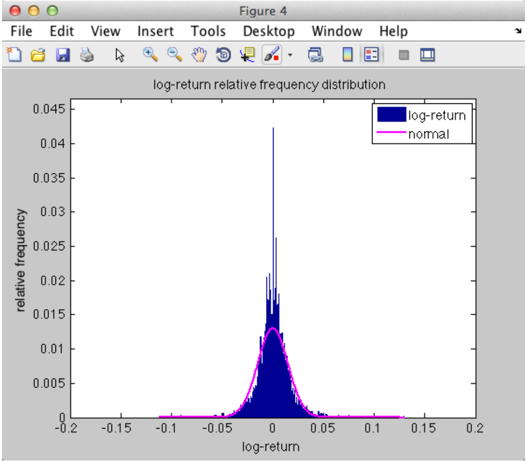
$$Z = \frac{X - \mu}{\sigma}$$

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

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

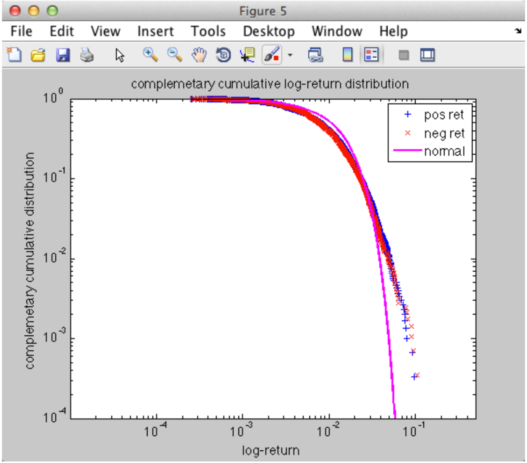
COMPARSION WITH A GUASSIAN

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

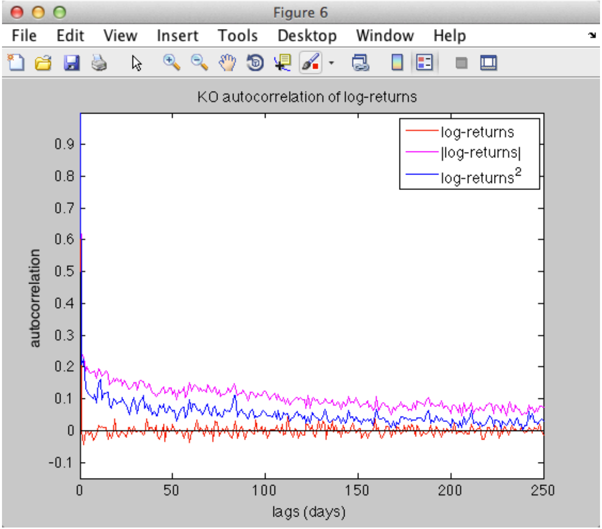


COMPLEMENTARY CUMULATIVE DISTRIBUTION

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



AUTOCORRELOGRAM



VOLATILITY

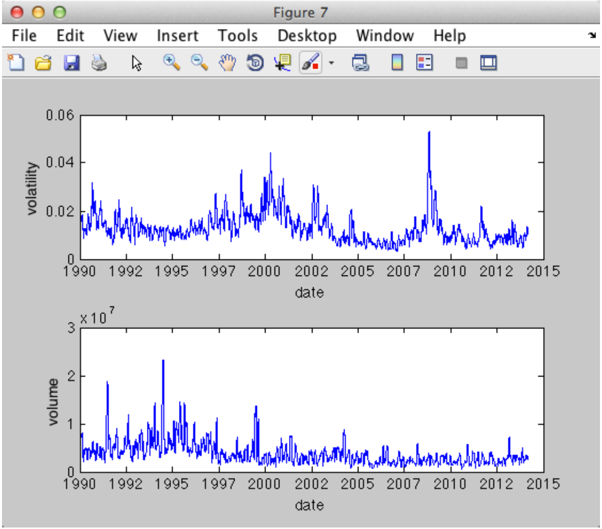
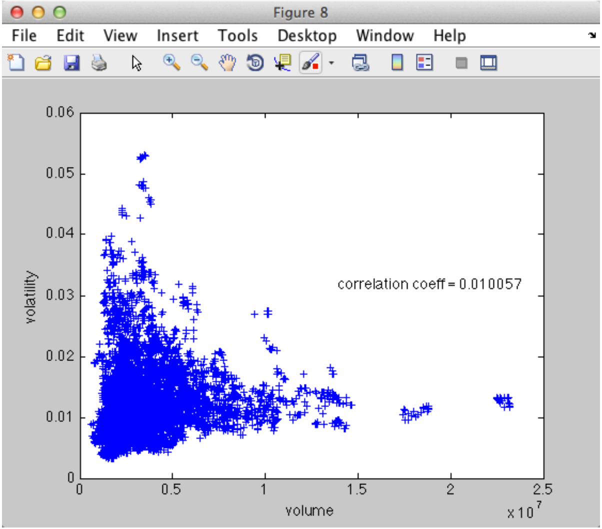


Figure 7
File Edit View Insert Tools Desktop Window Help
0.06
0.04
0.02
0
volatility
1990 1992 1995 1997 2000 2002 2005 2007 2010 2012 2015
date
x 10^7
3
2
1
0
volume
1990 1992 1995 1997 2000 2002 2005 2007 2010 2012 2015
date

VOLATILITY Vs. VOLUME



QQPLOT

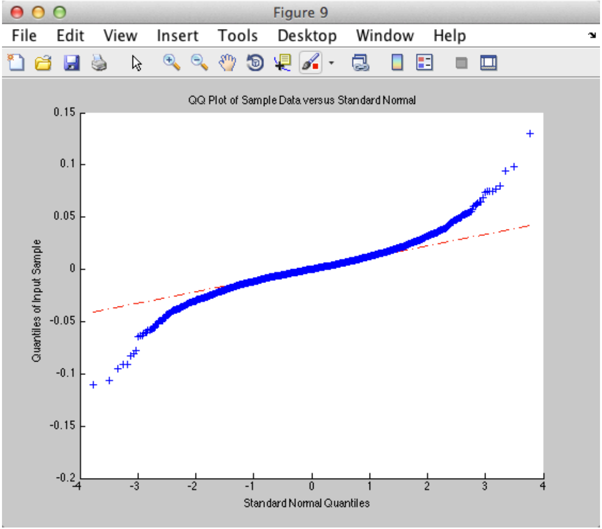


Figure 9
File Edit View Insert Tools Desktop Window Help
QQ Plot of Sample Data versus Standard Normal
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-4
-3
-2
-1
0
1
2
3
4
Quantiles of Input Sample
Standard Normal Quantiles

VAR PARAMETRIC

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

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

- 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

$$\text{ES}_\alpha = \frac{1}{1-\alpha}\int_\alpha^1 \mathrm{VaR}_u(F_L)\,\mathrm{d}u$$

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

/Users/peterbebbington/Documents/MATLAB/Course/Session 3/CVaR.m

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