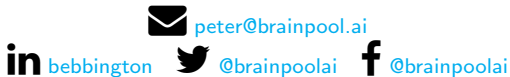


# Matlab for Finance Course

Dr. Peter A. Bebbington

Brainpool AI



March 12, 2023

# REVIEW OF SESSION 1

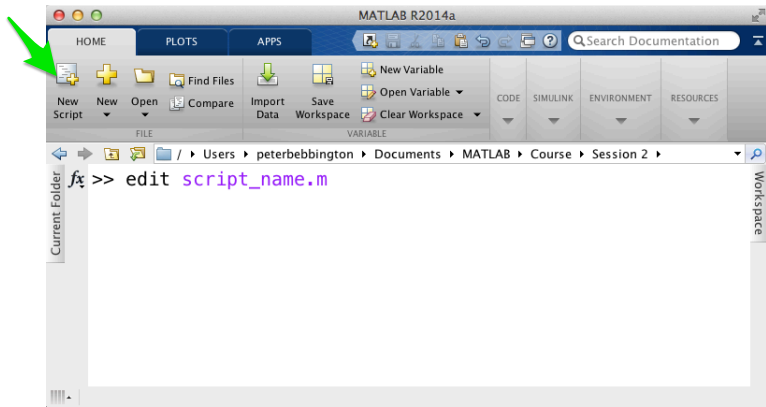
- MATLAB Desktop; Command Window, Command History, Current Directory, Launch Pad, Workspace
- Executing Basics Commands, Initialising and Defining
- Utility Commands
- Arrays, Matrices and Vectors
- Set Functions
- Addressing Array/Matrix Elements
- Solving Linear Equations
- Basic Plots

# 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

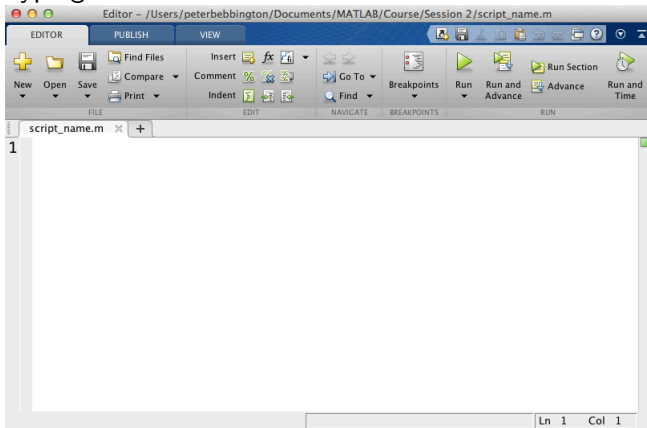
# CREATING SCRIPTS

- 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  $X(r,c)$

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

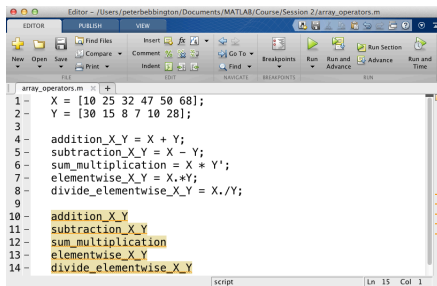
- Hence one can access an element by

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

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

# 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

EDITOR      PUBLISH      VIEW
+ Find Files  Insert  Format  View  Run
New Open Save Compare Comment Indent Go To Breakpoints Run Run and Run and
Print Print Indent Find Find Find Advance Advance Time

FILE EDIT NAVIGATE BREAKPOINTS RUN

array_operators.m
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

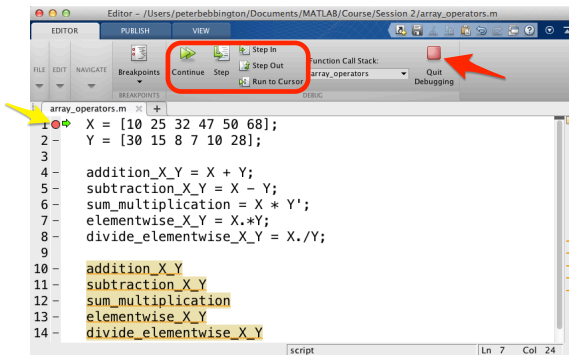
script Ln 15 Col 1
```

- 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.

- 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
```

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

**Example:**

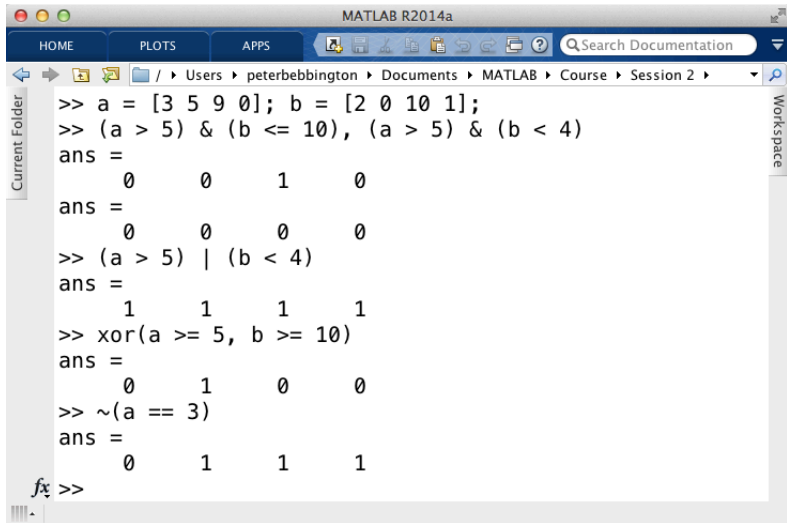
`(x != 0) | | (x < y)`

`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 breadcrumb navigation path: "/ > Users > peterbebbington > Documents > MATLAB > Course > Session 2". The main workspace area contains the following MATLAB code and its output:

```
>> 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 workspace, there is a "Current Folder" pane and a "Workspace" pane. The "Current Folder" pane shows a folder icon and the text "fx". The "Workspace" pane is empty.



# CONDITIONAL INDEXING

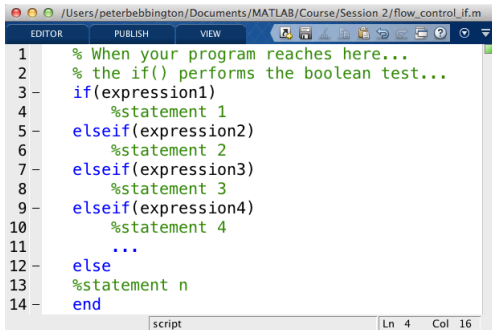
- There is a neat trick for returning the values in an array that satisfy a Boolean condition.
- Place the condition within the index brackets () of the array.

```
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.



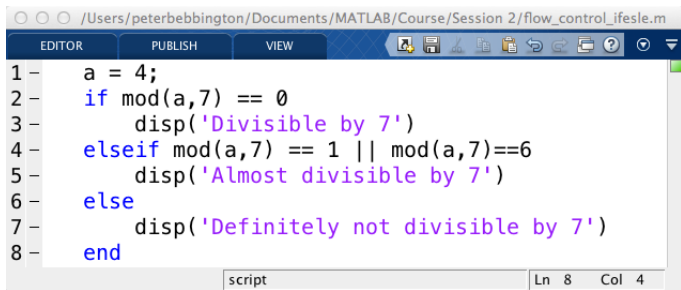
```
1 % When your program reaches here...  
2 % the if() performs the boolean test...  
3 if(expression1)  
4     %statement 1  
5 elseif(expression2)  
6     %statement 2  
7 elseif(expression3)  
8     %statement 3  
9 elseif(expression4)  
10    %statement 4  
11    ...  
12 else  
13    %statement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

The image shows a MATLAB Editor window with a menu bar (EDITOR, PUBLISH, VIEW) and a toolbar. The code is a script file named 'flow\_control\_if.m'. The code structure is as follows:

```
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 'script' and 'Ln 4 Col 16'.

# IF-ELSE EXAMPLE



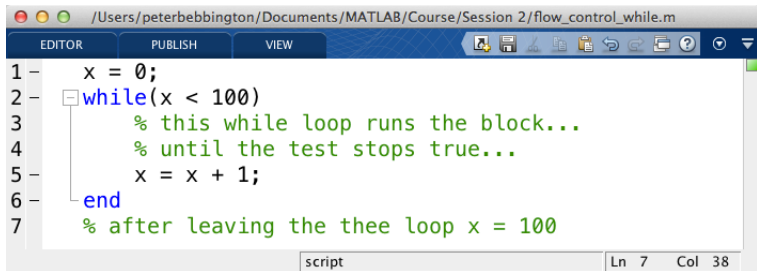
A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow\_control\_ifesle.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active. The script content is as follows:

```
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.

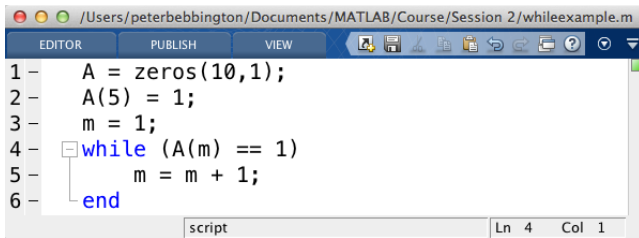


A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow\_control\_while.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with the following code:

```
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 thee loop x = 100
```

The status bar at the bottom indicates the current position is Ln 7, Col 38.

# WHILE EXAMPLE



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

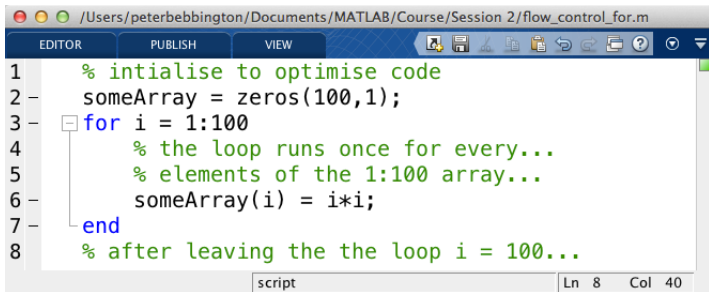
```
1 - A = zeros(10,1);  
2 - A(5) = 1;  
3 - m = 1;  
4 - while (A(m) == 1)  
5 -     m = m + 1;  
6 - end
```

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.

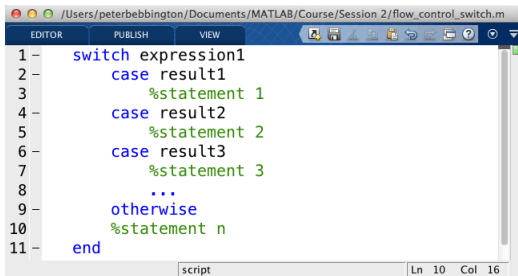
A screenshot of the MATLAB Editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow\_control\_for.m. The window has three tabs: EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with the following code:

```
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 code is color-coded: comments are green, keywords (for, end) are blue, and variables/numbers are black. A small square icon is next to the 'for' keyword on line 3. The status bar at the bottom shows 'script' and '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 script editor window. The title bar shows the file path: /Users/peterbebbington/Documents/MATLAB/Course/Session 2/flow\_control\_switch.m. The editor has three tabs: EDITOR, PUBLISH, and VIEW. The code is as follows:

```
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
```

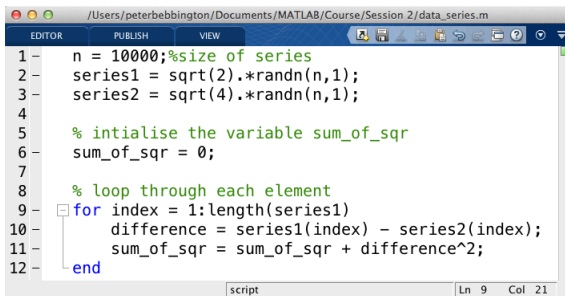
The status bar at the bottom indicates 'script' and 'Ln 10 Col 16'.

# 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
```

The image shows a MATLAB script editor window with the following code:

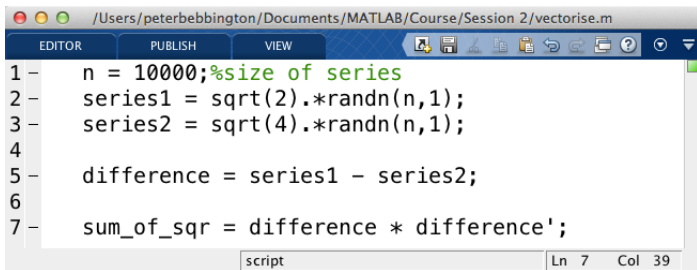
```
1 - n = 10000;%size of series
2 - series1 = sqrt(2).*randn(n,1);
3 - series2 = sqrt(4).*randn(n,1);
4
5 - % intialise the variable sum_of_sqr
6 - sum_of_sqr = 0;
7
8 - % loop through each element
9 - for index = 1:length(series1)
10 -     difference = series1(index) - series2(index);
11 -     sum_of_sqr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

The window title is `/Users/peterbebbington/Documents/MATLAB/Course/Session 2/data_series.m`. The status bar at the bottom indicates `script`, `Ln 9`, and `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 the following code:

```
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';
```

The status bar at the bottom indicates 'script' and 'Ln 7 Col 39'.

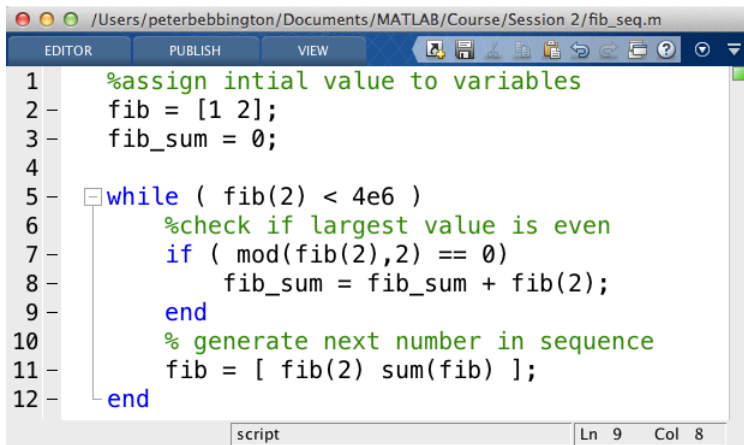
# FIBONACCI EXAMPLE

- 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



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

```
1 %assign intial 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
```

The status bar at the bottom indicates the file is named 'script' and the cursor is at Line 9, Column 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?