The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Advanced Engineering Research Preparation

Matlab II:
Programming in Matlab

Presented by: Prof. Eric Larkins (Coates A46)
Optics and Photonics Group
Department of Electrical and Electronic Engineering

Prepared by: Dr. Steve Greedy
George Green Institute for Electromagnetics Research
Department of Electrical and Electronic Engineering

Overview

- Aims
- Planning your program
- Debugging
- Programming Basics
- The m-files
  - Script files
    - Exercises
  - Function files
    - Exercises

Aims

- This tutorial is intended to cover the basics of the MATLAB programming language.
- Examples included in this lecture together with the exercises will provide you with enough knowledge to write your own program to solve the design problem that will be presented to you.

- Topics to be discussed include the m-files and their types, inline functions, control flow, operators, strings, cell arrays etc.

- Basic knowledge of Matlab is assumed (H14ERP).
- Matlab is available in all the engineering computer labs.

Further Information

- Matlab help
- http://www.mathworks.com
  - Both excellent sources of information and examples
- http://www.mathworks.com/matlabcenteral/fileexchange
  - User contributions & solutions

- Start Matlab! Remember it can take a while to load....

Planning

- Most difficult part of writing a computer program can be working out what steps the program needs to do.
- Knowing the syntax needed to implement the program can be the easy part. To work out the steps, it is necessary to think clearly and logically. This is something we do in other contexts every day. We must learn to transfer those concepts to writing programs!

- Think about the problem before starting to write any code. Decide on the structure and the logic involved.

  - Try writing the problem out in **pseudocode** first.
  - Use **flow diagrams** for complex programs e.g. with loops & condition statements.

- **Pseudocode** is a list of tasks written in 'English' rather than the actual computer language.

e.g.

```
Wake up
Get out of bed
Get dressed
Eat Breakfast
Go to lecture
```

Debugging

- Do not expect programs to work first time!

- Experienced programmers make errors.
  - Syntax errors are usually relatively easy to correct.
  - Errors might not actually be on the lines the computer thinks they are.
  - Some errors can be subtle, such as inconsistent naming of variables. ‘Name’ and ‘name’ are different.

- If you get an error and your program won’t run
  - Test small parts of a large program separately.
  - Try all possible input parameters to see if all cases are allowed for. For example, if a program asks for a number, is that number implicitly assumed to be positive? What happens if a negative number is chosen?
  - Check your results for known cases (with a calculator, if necessary).
  - Include **disp** statements to test values during the program execution.
  - Comment out blocks of code in order to try to identify where the faults lie.

- This is referred to as a top down approach - “the breaking down of a system to gain insight into its compositional sub-systems.”

Programming Basics

- Variables and Operators:
  - ✧ **char** - a single 'ASCII' character, e.g. all keyboard characters
  - ✧ **string** - a list of Char variables, making up a word.
  - ✧ **int** - a 'whole number', eg 1, -2, 25000
  - ✧ **single** - a floating point number with decimal point, eg 1.5, -2.0.
  - ✧ **double** – similar to single but used to represent larger numbers. The default type for Matlab numerical variables.

Programming Basics

- Assignment:
  - ✧ Values assigned to variables using the assignment operator '=', e.g.

```
x = 10;
```

  - ✧ To clear a variable use:

```
clear x;
```

  - ✧ Good practise to clear all variables at start of program using

```
clear all;
```

Programming Basics

- Variable Names:
  - Try to pick something meaningful e.g. `averageMark` or `studentName`
- Rules for Variable Names
  - Begins with a letter
  - No spaces
  - No hyphens
  - Underscores are allowed
  - Names are case-sensitive
  - Cannot be a Matlab ‘keyword’ – type `iskeyword` at the command prompt for a list of keywords.

Programming Basics

- Input
  - ✧ input can be used to get information *from* the user e.g.

```
number   = input(`Enter a number’);
```

This would display the message ‘Enter a number’ and assign the value to the variable number

Programming Basics

- Displaying a Variable
  - ✧ Variables can simply be displayed by using its name and not putting a ‘;’ after it. e.g.

```
x = 10;
x
```

Would display the value of x

  - ✧ Sometimes desirable to include a bit more information, can use the command ‘disp’ e.g.

```
disp (‘The value of x is ’);
disp (x);
```

  - ✧ Or more conveniently, by converting the number y to a string using the ‘num2str’ command:

```
disp ([‘The value of x is ‘, num2str(x)]);
```

Programming Basics

- For Loops

  ✧ To repeat a set of statements a **for** loop can be used. e.g.

```
for repeat = 1:3
  disp (‘In a loop!’);
end
```

would repeat the disp instruction 3 times

  ✧ Can also use loops to increment a variable

```
sum = 0;
for value = 1:10
        sum = sum + value;
end
disp ([‘sum of numbers 1 to 10 =‘,num2str(sum)]);
```

Programming Basics

- Conditional Statements
  - ✧ Matlab allows for the following checks:

```
a < b   is a less than b?
a > b   is a greater than b?
c >= 5  is c greater than or equal to 5
d <= r  is d less than or equal to r
a == b  is a equal to b?
a ~= 0  is a not equal to 0
```

  - ✧ Conditional statements are either **true** or **false.**

Programming Basics

- Checks can be used as follows:

```
while condition is true
    Statement
    Statement
end
```

‘while’ implements a loop until the condition becomes false. **Note:** If the condition does not change in the body of the while statement, the loop will continue forever.

```
if condition is true
    Statement
    Statement
End
```

‘if’ implements the once only execution of the statements only if the condition is true.

```
if condition is true
    Statement 1
else
    Statement a
end
```

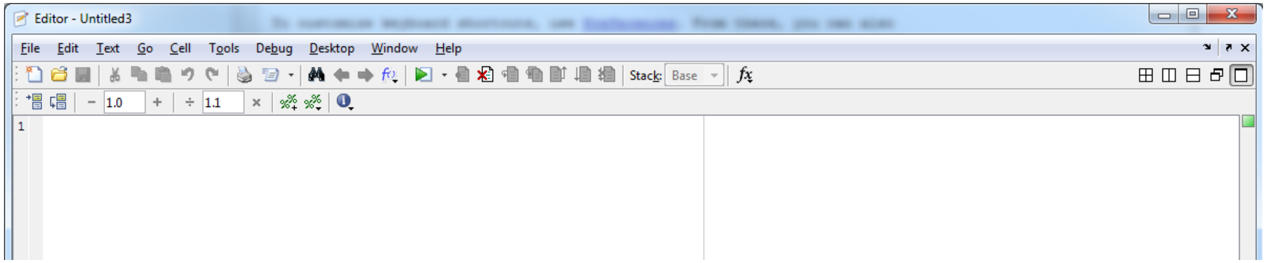
‘if’ implements the once only execution of the statement 1 only if the condition is true.
‘else’ implements the once only execution of the statements a only if the condition is false

Writing a Matlab Program

The m-files

- Files that contain matlab code are called m-files
- Two types of m-files:
  - script files: these do not take input arguments or return output arguments
  - function files: these may take input arguments and return output arguments

- To create an m-file:
  - ✧ Type edit 'name of file' at the command prompt and press return, or
  - ✧ Click File then select New and select the type of file e.g. Script
  - ✧ The Matlab editor/debugger screen will appear



Script Files

A Script File

Scripts are the simplest kind of MATLAB program. They are useful for automating blocks of MATLAB commands, such as computations you have to perform repeatedly from the command line.

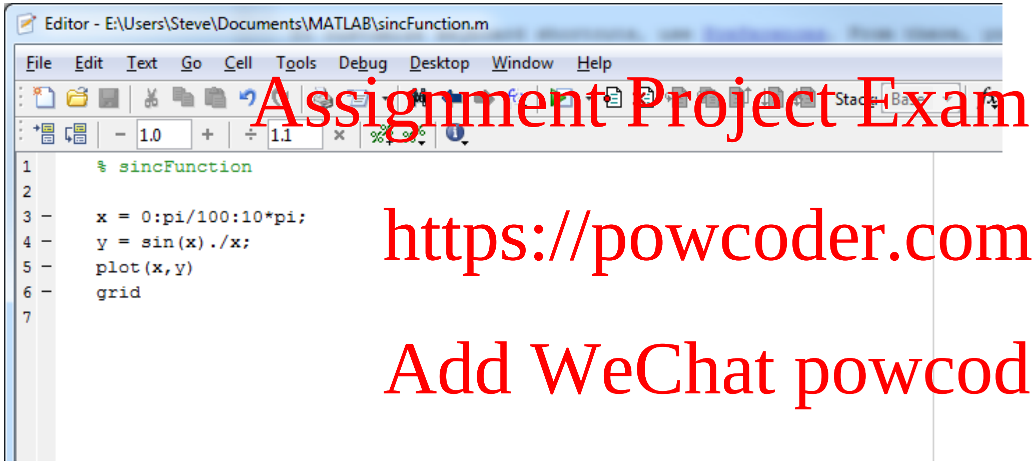
Scripts can operate on existing data in the workspace, or they can create new data on which to operate.

Although scripts do not return output arguments, any variables that they create remain in the workspace, so you can use them in further computations. In addition, scripts can produce graphical output using commands like plot.

http://www.mathworks.com/help/techdoc/ref/script.html

A Script File

- Example of an m-file script:



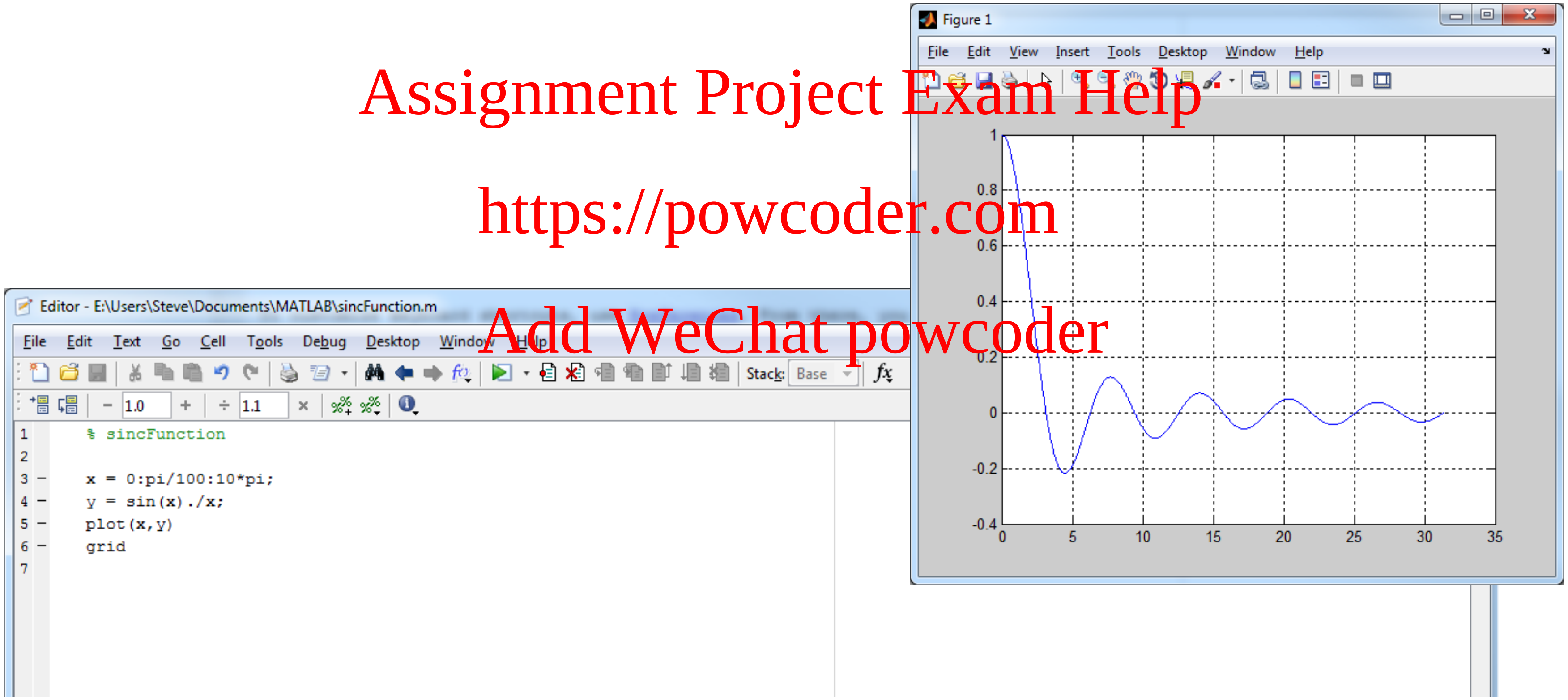
Editor - E:\Users\Steve\Documents\MATLAB\sincFunction.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
Stack: Base

```
% sincFunction

x = 0:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

- To run the script click the run button ▶
- or, type the name of the script file at the Matlab command prompt and press return

A Script File

- Example of an m-file script:



Editor - E:\Users\Steve\Documents\MATLAB\sincFunction.m

```
% sincFunction

x = 0:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
5
10
15
20
25
30
35

A Script File

- Analysis of the example file:

```
% sincFunction


x = pi/100:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

The percentage sign % denotes that what follows is a comment.

Comments are ignored by Matlab and are added to improve readability of the code.

Commenting code in general is good practise.

A Script File

- Analysis of the example file:

```
% sincFunction

x = pi/100:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

Next, two arrays **x** and **y** are created.

Array **x** holds the 1001 evenly spaced numbers in the interval 0 to 10π.

Array **y** holds the values of the sinc function, sin(x)/x, at each of these points.

Note the dot operator before division operator indicating component wise division. Also note that the semicolon follows both commands to suppress the contents being displayed on the screen.

A Script File

- Analysis of the example file:

```
% sincFunction


x = pi/100:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

Create the graph using the values in vectors **x** and **y** and add a grid to the graph.

A Script File

Summary

- A script file is a collection of Matlab commands, saved to a file.
- These commands execute in sequence when the script file is run.
- Script files cannot take argument as input and cannot pass arguments as output.

Exercises

Task 1

- Create a script to calculate the sum and mean of a set of 5 numbers
- Pseudo code:

```
Acquire the set of numbers
Add them all together to calculate the sum
Count how many numbers
Divide the sum by the count to calculate the average
Tell the program user the values of the sum and mean
```

Task 1 - Solution

- Aquire a set of numbers:

```
a = [ 2 4 6 8 10];
```

- Calculate the sum:

```
sum = a(1) + a(2) + a(3) + a(4) + a(5);
```

- Calculate the mean:

```
mean = sum/5;
```

- Display results:

```
sum
mean
```

The m-file:

```
% calculate sum and mean of a()

a = [ 2 4 6 8 10];
sum = a(1)+a(2)+a(3)+a(4)+a(5);
mean = sum/5;
sum
mean
```

Task 2

- Again create a script to calculate the sum and mean of a set of 5 numbers, but ask the user to input the numbers

- Pseudo code:

```
Ask for each number
Add them all together to calculate the sum
Count how many numbers
Divide the sum by the count to calculate the average
Tell the program user the values of the sum and mean
```

Task 2 - Solution

- Ask the user for the numbers:

```
a(1) = input('please enter the first number   ');
                  .
                  .
a(5) = input('please enter the next number   ');
```

- Calculate the sum:

```
sum = a(1) + a(2) + a(3) + a(4) + a(5);
```

- Calculate the mean:

```
mean = sum/5;
```

- Display results:

```
sum
mean
```

The m-file:

```
% calculate sum and mean of a()

a(1) = input('please enter the first number   ');
a(2) = input('please enter the next number   ');
a(3) = input('please enter the next number   ');
a(4) = input('please enter the next number   ');
a(5) = input('please enter the next number   ');

sum = a(1)+a(2)+a(3)+a(4)+a(5);
mean = sum/5;
sum
mean
```

Task 3

- Again create a script to calculate the sum and mean of a set of 5 numbers and the user to input the numbers, this time using a **for** loop
- Pseudo code:

```
Ask for the numbers
Loop 5 times
        Please enter a number
end
Add them all together to calculate the sum
Count how many numbers
Divide the sum by the count to calculate the average
Tell the program user the values of the sum and mean
```

Task 3 - Solution

- Ask the user for the numbers:

```
a(1) = input('please enter the first number    ');
                    .
                    .
a(5) = input('please enter the next number    ');
```

- Calculate the sum:

```
sum = a(1) + a(2) + a(3) + a(4) + a(5);
```

- Calculate the mean:

```
mean = sum/5;
```

- Display results:

```
sum
mean
```

The m-file:

```
% calculate sum and mean of a()

a(1)=input('please enter the first number    ');

for i=2:5
   a(i)=input('please enter the next number    ');
End

sum = a(1)+a(2)+a(3)+a(4)+a(5);
mean = sum/5;
sum
mean
```

Task 4

- Modify the script in task 3 to allow the user to input the number of values required
- Pseudo code:

```
input the count (number of entries must be greater than 2)
input the first number
Set sum = first number, we will calculate sum as the numbers are
   entered
input the first number
loop from 2 to count
   Input the next number
   Add the number entered to sum
end
average =sum/count
tell the user the values of sum and count
```

Task 4 - Solution

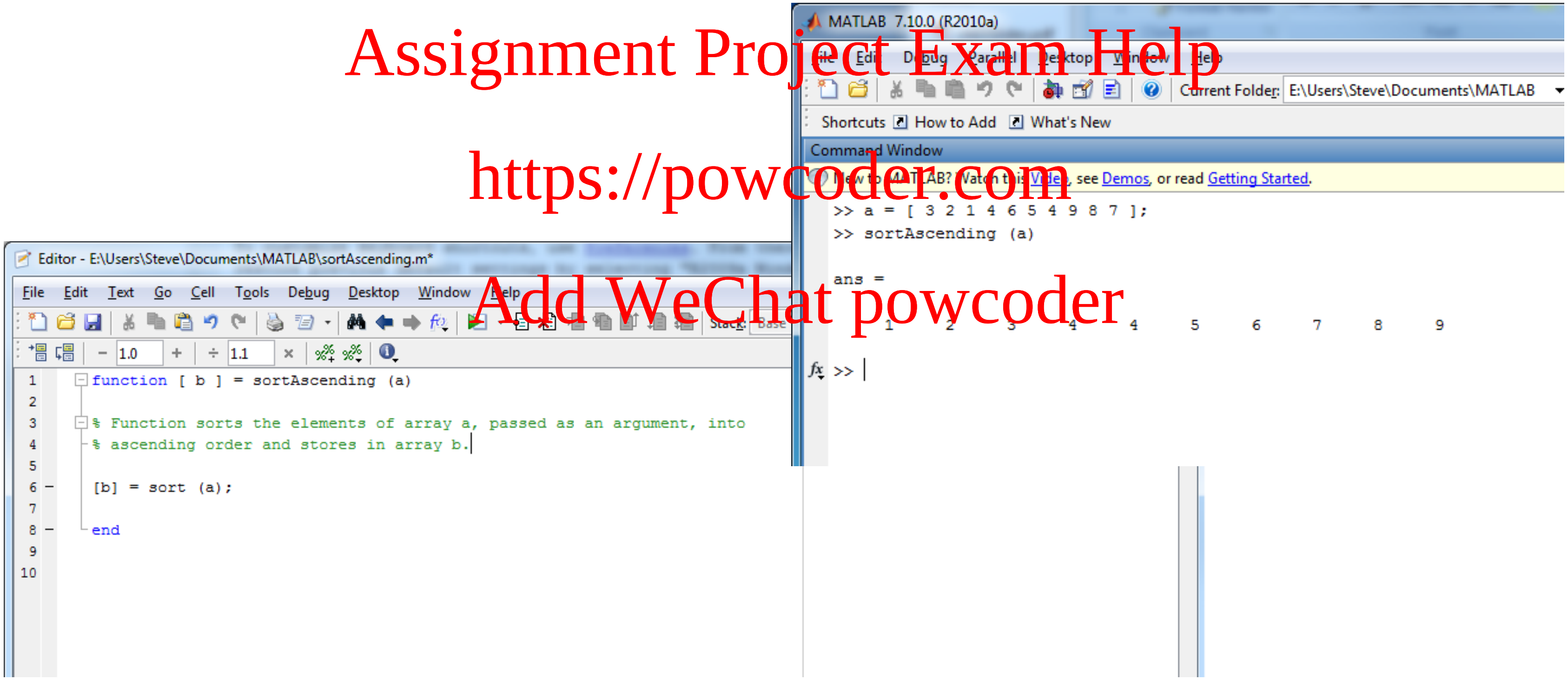
The m-file:

```
% calculate sum and mean of a()

count = input('please enter the number of values   ');

a(1)=input('please enter the first number    ');
sum = a(1);

for i=2:count
   a(i)=input('please enter the next number    ');
   sum  = sum +a(i);
end

mean = sum/count;
sum
mean
```

Function Files

A Function File

- Example of a function file:


Editor - E:\Users\Steve\Documents\MATLAB\sortAscending.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
MATLAB 7.10.0 (R2010a)
File Edit Debug Parallel Desktop Window Help
Current Folder: E:\Users\Steve\Documents\MATLAB
Shortcuts How to Add What's New
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
function [ b ] = sortAscending (a)

% Function sorts the elements of array a, passed as an argument, into
% ascending order and stores in array b.

[b] = sort (a);

end
```

```
>> a = [ 3 2 1 4 6 5 4 9 8 7 ];
>> sortAscending (a)

ans =

     1     2     3     4     4     5     6     7     8     9

>> 
```

A Function File

- Analysis of the function file:

```
function [ b ] = sortAscending (a)

% Function sorts the elements of a
% argument, into ascending order

[b] = sort (a);

end
```

Function definition. Starts with the function keyword.

The first parameter defines the output arguments, in this case an array **b**. Then the function name, **sortAscending**, followed by the input arguments, an array **a**.

A Function File

- Analysis of the function file:

Comments, for a function it is useful to describe the input and output arguments and their type.

```
function [ b ] = sortAscending (a)

% Function sorts the elements of array a, passed as an
% argument, into ascending order and stores in array b.

[b] = sort (a);

end
```

A Function File

- Analysis of the function file:

```
function [ b ] = sortAscending (a)

% Function sorts the elements of array a, passed as an
% argument, into ascending order and stores in array b.

[b] = sort (a);

end
```

Call the Matlab **sort** function to sort the elements of **a** into ascending order and store in a vector **b**

Exercises

Task 1

- Write a program to perform a bubble sort of a vector of numbers into ascending order. A bubble sort works through a sequence of numbers, in pairs, swapping them if they are out of sequence.
- Pseudo code - The swap operation, swap a(i) with a(a+1):

```
Save a(i) to temp
Set a(i) equal to a(i+1)
Set a(i+1) to temp
```

Matlab to swap a(i) with a(i+1)

```
temp = a(i);
a(i) = a(i+1);
a(i+1) = temp;
```

Task 1

- Pseudo code – Inner Loop

```
Set flag to zero to indicate no swaps yet
Loop number of items – 1
    if next number is smaller than current number
        swap numbers
        set flag to 1 to indicate a swap was required
    end
end
```

- Pseudo code – Inner Loop

```
Set flag to one to ensure a pass of outer loop
While flag
    Inner loop
End
Print sorted vector
```

Task 2

A. Re-write the program so that it now sorts the numbers into descending order.

B. Make this an option available to the user and pass it as an argument for the sort function