Tutorial 6 – Week 7

Aims:

Upon successfully completing these tutorial exercises, students should be able to:

- Understand the difference between a function and a script.
- Understand more complicated loops and conditional statements.

Tutorial 6: You have been using inbuilt functions in Matlab for a few weeks now. One of the strengths of functions is that you can re-use the same code with different inputs. In Matlab, you can build your own functions, which is what we will cover today.

1 Understand the difference between a function and a script.

We have focused on writing scripts thus far. Scripts do not require any inputs to be passed; all data needed exists inside the script itself. Functions, on the other hand, require the passing of one or more inputs and may have one or more outputs.

When running a script, it is sufficient to just click the run button. However, because functions require inputs, the run button will not be appropriate. You have already called functions in this subject, perhaps without realising it. One example of a function is sin(). This function has one input and one output. The format for calling this function is:

```
output=sin(input).
```

The general form for calling a function is:

```
[output1,output2,…]=function_name(input1,input2,…);
```

One of the strengths of coding is the creation of new functions. Functions are a powerful method for making code reusable easily. Let's say we want to create a new function that performs Pythagoras' theorem. The inputs to the function will be the non-hypotenuse side lengths of the square angled triangle. The first line of a new function (which we write in an editor) must always be in the form:

```
function [output1, output2,…] = function_name(input1,input2,…)
```

And then the following lines should contain the code that should be executed when the function is called. In this case, we will have two inputs b and c and one output a. We will let the function name be Pyth().

In the editor, write the code (in much the same way you would a script):

```
function [a] = Pyth(b,c)
```

```
    a=sqrt(b^2+c^2);
```

Save the function (the file name should match the function name).

Now to call the function, go to the command window and type:

```
    hypotenuse = Pyth(3,4);
```

If all steps have been followed correctly, the variable hypotenuse should contain the output value 5.

1.1 For you to do:

Create a function that solves a quadratic equation. The first line of your function should be:

```
    function [x1,x2]=quadratic_solver(a,b,c)
```

The inputs are a, b and c which must return all solutions such that $ax^2+bx+c=0$. Use conditional statements for the following cases:

- If a and b are both equal to zero, then no solution exists.
- Otherwise, if a is zero, the problem becomes linear and there is only one solution.
- If solution is complex, assume no solution and inform the user.
- If a double root exists, return same solution twice and inform the user.
- Otherwise, return both solutions.

In any cases where there is no solution, return x1=0 and x2=0 with an error message. If there is only one solution, set both x1 and x2 to this solution.

2 Understand more complicated loops and conditional statements

The applications of **for** loops are enormous. Let us take the example of a factorial calculator. Hopefully from your high school mathematics that you know n!=1*2*3*…*n. Hence, we can write the following function:

```
function output=factorial_calc(n)
output=1;
for i=1:n
    output=output*i;
end
```

2.1 For you to do:

Create a function which is a compound interest calculator. The inputs should be the principal investment, the interest rate (per annum), the number of months invested. Interest should be calculated monthly and added to the principle. The output should be an array containing the balance at each month.

Last tutorial, we looked at **for** loops being used with repetitive code. What will be the output of the following code:

```
    for i=[2,4,1,0,9]

        disp(['The value of i is: ',num2str(i)]);

    end
```

Recall that the code inside the loop will run as many times as there are elements in **i** (5 in this case). Inside the loop, **i** is not equal to the entire array, rather one element of the array each iteration. In this case, **i** will be equal to 2 in the first iteration, 4 in the second and so forth.

It is also possible to have a loop inside a loop. This is known as nesting loops. Consider the following code:

```
for i=1:10
    disp(['i=',num2str(i)]);
    for j=1:5
        disp(['j=',num2str(j)]);
    end
end
```

Predict what you think the output will be. How many times will **i** be displayed? What about **j**? Nested loops can be useful when performing operations with two dimensional arrays. Let’s create a 5x5 random array using the function:

a=rand(5);

Let’s say we want to find out how many elements contain a value 0.8 or greater. We can solve this problem using nested loops and an **if** statement. Create a new script an insert the following code:

```
a=rand(5);
number_found=0;
for i=1:5
    for j=1:5
        if a(i,j) >= 0.8
            number_found=number_found+1;
        end
    end
end
disp(number_found)
```

Notice how each individual element of the matrix a will be compared with 0.8. The variable number_found will be incremented if the condition is TRUE.

2.2 For you to do:

Create a random array of size 10x10. Use nested for loops and if statements to set all elements less than 0.2 to zero. Leave all other elements unchanged.

Another useful looping statement is the **while** loop. **For** loops are useful when it is known exactly how many loops are required. **While** loops are preferred when the loop should be broken when a condition is no longer met. Consider the following example:

```
a=0;
while a<10
    a=a+1;
    disp(a);
end
```

The loop will repeat until the **while** condition a<10 is false. Can you predict the output?

It is possible to create an infinite loop by using a condition that is always TRUE (i.e. always 1). However, it may be desirable to break a loop when some condition is met. The **break** statement can be used to this effect:

```
a=0;
while 1
    a=a+1;
    disp(a);
    if a >= 10
        break
    end
end
```

It is also possible to skip the remainder of the code in a loop and move directly to the next iteration using the **continue** command. Can you predict the output of:

```
a=0;
while 1
    a=a+1;
    if a == 4 || a == 6
        continue
    end
    disp(a);
    if a >= 10
        break
    end
end
```

Continue and **break** can both also be used in **for** loops in the same way.

2.3 For you to do:

Write a script which randomly chooses a number between 1 and 10 which the user has to guess. randi(10) may be used to randomly generate a number between 1 and 10. If the user is wrong, display that the guess is too high or too low and ask for a new guess. If the user is correct, display that the guess is correct and end the script.