IMAT5119: Matlab Laboratory 1 [1]

Preface – The Matlab Package and the Fuzzy Logic Toolbox

Matlab is a highly sophisticated, widely used (in industry and universities) package that is very flexible and has a number of tool boxes. The Fuzzy Logic Toolbox (FLT) is a set of tools available in Matlab for carrying out fuzzy logic.

The FLT has a GUI for developing fuzzy logic systems but so you understand what is going on we will use the command line interface and programming language for our purposes. You will gain exposure to the GUI later.

You will frequently be asked to enter unfamiliar Matlab code into the command window. Many of these commands will be useful to you — you should try your best to understand how they work and what they do before moving on.

Learning Outcomes

After this lab you will be able to use Matlab to do simple matrix calculations and you will be able to save data to a file and load matrices from file. You will know some of the simpler capabilities of the Matlab package and will be familiar with the Matlab help function. You will know how to draw simple plots using Matlab.

Tasks for Lab 1

Double click the Matlab icon to get Matlab up and running. We will make much use of the Matlab program and its FLT (and its general functions also).

You can read the help later — for now we just want to use the command window and get Matlab to do some calculations for us.

All the instructions below for some time refer to the command window. Just type at the $>>$ prompt.

We can enter a matrix as follows and Matlab will respond.

```
>> a = [1 2; 1 2]
```

Now enter

```
>> b = [1 3; 1 4]
```

We want to check that Matlab knows how to multiply and add matrices correctly — enter the following commands and check the answers.

[1] Sarah Greenfield (acknowledging Bob John, Paco Chiclana and Jenny Carter), 17/10/20

```
>> a + b
>> a * b
```

Let’s have a new matrix **c**

```
>> c = [1 2 3; 3 4 5]
```

We can’t always multiply matrices if they are incompatible in size. Let’s try to multiply **c** * **a**:

```
>> c * a
```

We can get **a** * **c** though:

```
>> a * c
```

If we want we can use algebra with the matrices:

```
>> a^2
```

(So this multiplies the matrix by itself using matrix multiplication.)

The usual interpretation of powers is for matrix multiplication. There is also an array multiplication which does usual arithmetic on the array values — this has ’.’ in front of operators.
Try

```
>> a.*a
```

(This squares each value in the matrix.)

Also try

```
>> a.^3
```

Make sure you know what the difference between .* and * is; it is extremely important.

If I want a name for the answer I can set

```
>> y = a * c
```

and now I can use **y** in a calculation

```
>> y + [1 2 3; 1 2 5]
```

We will need to know about transpose matrices:

```
>> y'
```

So the transpose flips rows and columns. This is useful in situations where the parameters to a Matlab function require a column vector when our data is a row vector etc.

If we want to add an additional column to an existing matrix we can do this:

```
>> z = [y' [1; 2; 3]]
```

Matlab is a programming language so assignment is possible in the usual way:

```
>> a = a + b
```

The matrix **a** is not a big matrix — but it might be and we might want to save it to a file.

```
>> save data.txt a -ascii
```

Look at this file to see what you have got.

We can load files as well — be careful with the quote marks. They are essential.

```
>> z = load('data.txt')
```

There are also commands which read and write delimited text if you want other than space delimiters.

```
>> dlmwrite('data2.txt',a,':')
```

Colon delimited files are used on some unix systems — it is more usual to use commas or tabs on windows systems.

There is a similar command *dlmread* which you can find out about using the Matlab *help*.

```
>> help dlmread
```

Matlab help will give you information about any command — just type *help* followed by the command. Try

```
>> help help
```

If you want to keep track of what you are doing try the *diary* command. (Use the help to find out about it.)

If you want to keep all your working you can save the workspace as a .mat file — again see help on *save* for the details.

Entering data by hand can be tedious so Matlab allows creation of uniformly spaced data points.

```
>> a = 0:0.1:pi
```

I might want a hundred data points to use for something — I don't want them to display and fill the screen.

```
>> b = 0:0.01:1;
```

The effect of the semicolon at the end of a command line input is to suppress the output display.

Matlab comes with an invaluable built in graphics capability. See the help on *plot* for the various variants of plotting. For example, see the effect of the following:

```
>> c = a.^2
>> plot(a,c)
>> plot(a,c,'+:')
>> xlabel('The x axis')
>> ylabel('The y axis')
>> title('Power of 2 (squaring)')
```