MATLAB for MAPH 3071 Lab 1

1) Introduction

MATLAB (which stands for MATrix LABoratory) is a software package for scientific and engineering numerical calculations. Among some of its applications are:

a) Finding the roots of polynomials and non-linear equations (e.g. by iteration).
b) x-y and polar plotting.
c) Solution of linear algebraic equations (i.e. matrix manipulation).
d) 3-dimensional graphics.

The best way to learn about MATLAB, as with most things on a computer, is to sit down at a terminal and play with a few examples. This document, and in particular the examples contained here, should be used in conjunction with the notes from course MP3.7 and the problem sheets which will be handed out for that course.

2) Starting a Session

Login to your account and double click on NAL (Novell Application Launcher). Click on Network Applications, then mathematics and Statistics, and finally on MATLAB. You will now see the *MATLAB* icon on the right hand side of the which will bring you into the MATLAB software package.

N.B. : When you are finished using MATLAB you must close your windows session and then type *logout* so that nobody else can use your account.

Now, lets jump straight in and get something done with MATLAB - we will discuss the nuances of the MATLAB language as we go along.

Basic arithmetic

MATLAB uses a straightforward notation for basic arithmetic. The following table summarises the symbols used by MATLAB for calculations

+ addition
- subtraction
* multiplication
/ division
^ exponentiation (i.e. x^2 stands for x squared)

The simplest operation you can perform is to assign a number to a variable, i.e if we want to set x=1 then at the prompt we simply type

>> **x=3;** (-do this now).

To check that this worked type,

» x

We can use the arithmetic operators to change the value of x or create a new variable.

» hello = x^2;
» hello

This second variable called hello is now 9 as you can see. While using MATLAB we frequently create a large number of variables. To get information on what variables there are type;

» who (or type;)
» whos (whos gives more info on variables than who)

Generally in MATLAB we work with a mixture of scalars, vectors and matrices. You have just scalar variables, now create a vector;

» v1 = 0:10;
» v1

The colon here tells MATLAB to create a vector of the number from 1-10. This is a row vector.

It is important to note that MATLAB essentially sees every variable as a matrix or a scalar. The multiplication/addition etc rules for matrices therefore apply. However, if you need to say square every element in a vector special syntax rules must be followed - we will come to these later.

Now, we know enough to look at course material, so clear the work space of MATLAB. You can delete variables individually or together.

» whos

to see what you have, and

» clear x hello v1

or

» clear all

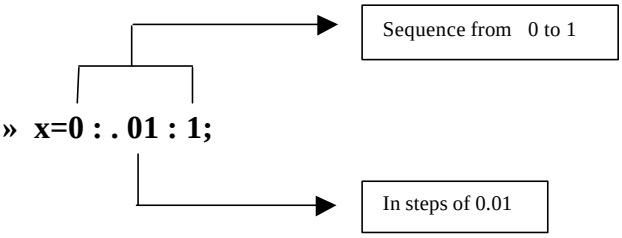
to delete the 3 variables.

Non-linear Equations

1. Plotting

From 1st lecture - problem was to find the solution of $x = e^{-x}$. First lets plot this function is MATLAB across a certain range.

Create a vector x by typing;



MATLAB creates a vector with number from 0 to 1 in steps of .01

To view this vector type

» x

Create a second vector y by typing

» y = exp(-x)

repeat this command (to scroll through previous commands in MATLAB use the up arrow) this time ending the command with a semi-colon.

» y = exp(-x);

Do you see the purpose of the semi-colon ? To run a command but not see the result use the semi-colon. If you want to see the result of your action immediately leave it out.

Now plot the x against y by,

» plot(x,y)

This is helpful, but it is hard to see where the solution is. What we need is a line on the same plot that shows where x=x function intersects the function exp(-x). To plot over the existing plot type;

» hold on

Now all following plot command will be drawn on the existing plot. When you want to turn this off , the command is **hold off**.

» plot(x,x, 'r'); (the 'r' option tells MATLAB to draw this line in red)
» grid; (- this command shows a grid over the plot, use it again to get rid of the grid)

Plot this 3rd line.

» plot(x,x-y, 'k'); (- 'k' is black obviously !)

Plot a horizontal line at y=0;

» plot(x,0,'k');

look at the relationship between these 3 lines - can you see the approximate solution ?

Have a go your self;

***Using the plot command find an approximate solution for sin(x) = x^3 . Hint: Create a vectore X between -1 and 1. To raise each element of this vector to the power of 3, use the following syntax*: y=x.^3** *The full stop between the x and ^ tells MATLAB that you wish to work with each element separately and not raise the vector to the power of 3 - try it without the full stop and see what happens !*

With such a relatively simple problem we can easily get an 'eye ball' estimate of the solution. But this is numerical analysis - we want the solution correct to a number of decimal places. So now we will go on to do some simple iteration problems.

2. Iteration Methods:

Recall that simple iterations takes the form $x_{new} = f(x_{old})$ - Where f(x) is a function of X.. In the case of $x = e^{-x}$ we have got an approximate starting value of 0.6 by plotting the function. So , we can use this as a pretty good starting value.

We want to perform the following calculation a number of times - i.e. until we get convergence.

$x_{(1)} = e^{-x_{(0)}}$
$x_{(2)} = e^{-x_{(1)}}$
$x_{(3)} = e^{-x_{(2)}}$
$x_{(4)} = e^{-x_{(3)}}$
.
.
$x_{(n)} = e^{-x_{(n-1)}}$ - stop at iteration n , where there is only a 'small' difference between n-1 and n.

We want to program this process into MATLAB. The simplest way of doing this is to use the **for** command.

```
» for i=1:4
i
end
```

Notice that MATLAB does not execute the **for** command until the **end** statement is entered. All **for** commands must be finished with an **end** statement. . This is an example of a *loop* (i.e. an operation which is repeated a number of times). The **for** command is also an example of what is termed a *control structure*. We will explore a few other control structures later on.

Here is another example. Run this in MATLAB and try to figure out what is happening.

```
» for i=1:5
x(i)=i^2;
x(i)
end
```

How does the *for* command help us with our iteration problem x(i)=exp(-x(i-1)) ? There are a couple of ways to do the iteration and to display the result on the screen. One such way is the following,

```
» x(1)=0.6
» for i=2:20
x(i)=exp(-x(i-1));
x(i)
end
```

Let's go through this line by line to see that it does in fact reproduce the method we used in the lectures to solve x=exp(-x).

The first line is our initial guess at the solution x(1)=0.6

The *for* command then makes the variable i run from 2 to 20 in unit steps. For every new value of i MATLAB then does two things since there are two commands between *for* and *end*. First of all it performs our actual iteration x(i)=exp(-x(i-1)). For example the first time around we have i=2 so that MATLAB calculates x(2)=exp(-x(1)), the next time around i=3 so that MATLAB now calculates x(3)=exp(-x(2)) and so on until finally i=20 and MATLAB calculates x(20)=exp(-x(19)). The second command between *for* and *end* is just 'x(i)' which tells MATLAB to print out the value of x(i) on the screen so that we can see what is going on and look at the values for x(i) as the computer works throught the iteration.

Have a go yourself;

Using the for command find the <u>positive</u> solution to sin(x) = x^3 .

Assignment:

The same procedure should be tried when solving all of the iteration problems (e.g. the Newton method) on the course.

END