NATIONAL UNIVERSITY OF SINGAPORE
Department of Mathematics

MA1506 Laboratory 1 (MATLAB)

In this course, we use a highly acclaimed numerical computing software called MATLAB. (Version 6.5, copyright ©1984-2002 The MathWorks Inc.) MATLAB stands for matrix laboratory. The aim of lab 1 is to introduce the basic functions of the MATLAB software, with emphasis on graphing solutions to differential equations. The lab consist of two parts: Part A which is a guided tour of MATLAB and its functions and Part B which is on graphing solutions of differential equations. Students are expected to work independently. Note: This is the MATLAB version of the lab practice, scilab users should refer to the other version.

Part A: Guided Tour of MATLAB

1. Arithmetic and Standard Mathematical Functions

   In its most elementary use, MATLAB is an extremely powerful calculator. For example, type

```
>> (50^2 - 4*10)/3 + 61
```

   Here the symbol ^ stands for exponentiation, i.e. the expression we were evaluating was actually $(50^2 - 4 \times 10) \div 3 + 61$.

   There is a list of mathematical functions that are built into MATLAB. Included are all of the standard mathematical functions.

abs(x)	The absolute value of x, i.e. $\lvert x \rvert$
sqrt(x)	The square root of x, i.e. $\sqrt{x}$
exp(x)	The exponential of x, i.e. e^x
log(x)	The natural logarithm of x, i.e. $\ln x$
log10(x)	The logarithm of x to base 10, i.e. $\log_{10} x$
sin(x)	The sine of x, i.e. $\sin x$
cos(x)	The cosine of x, i.e. $\cos x$
tan(x)	The tangent of x, i.e. $\tan x$
cot(x)	The cotangent of x, i.e. $\cot x$
sec(x)	The secant of x, i.e. $\sec x$
csc(x)	The cosecant of x, i.e. $\csc x$
asin(x)	The inverse sine of x, i.e. $\sin^{-1} x$
sinh(x)	The hyperbolic sine of x, i.e. $\sinh x$
asinh(x)	The inverse hyperbolic sine of x, i.e. $\sinh^{-1} x$

Also included in MATLAB are the inverse trigonometric and hyperbolic analogues of the other trigonometric functions:

acos(x), atan(x), acot(x), asec(x), acsc(x).
cosh(x), tanh(x), coth(x), sech(x), csch(x).
acosh(x), atanh(x), acoth(x), asech(x), acsch(x).

Use the following command to see the list of elementary functions supported by MATLAB:

```
>> help elfun
```

All of these functions can be combined to give complicated expressions. For example,

```
>> sinh(1)*exp(sqrt(2))+1
```

MATLAB also contains the standard constant π and is able to compute in complex numbers, using i (as well as j) to denote $\sqrt{-1}$. Try the following:

```
>> sqrt(-1)
>> sqrt(-3)
>> i^2
>> 16/(8i)
>> pi
>> exp(2*pi*i)
```

2. Expressions and Variables

In general, MATLAB commands are entered as statements in the following form:

variable = *expression*

which assigns the result of *expression* to *variable*. For example,

```
>> x = sinh(1)*exp(sqrt(2))+1
```

Note that if you did not assign a variable for your expression the result of *expression* is assigned to a special variable called `ans` (which stands for *answer*). In your MATLAB programme, you should see a window named Workspace. (Otherwise use the drop-down menu → View → Desktop Layout → Default.) Whenever a new variable is assigned, it will appear here. Double clicking the variable will reveal its current value. You can also change the value in this window called Array Editor.

Do the following series of commands and observe what happens to the variables.

```
>> x = 1 + 10
>> x
>> x + 2
>> ans
>> y = ans + 3
```

3. Useful Tips

   - You can use the up arrow key to recall previously typed commands
   - **clc** will clear your command window
   - **clear** will clear all previously assigned variables. You can clear variables individually by typing **clear** *variable*
   - typing the statement ">> help *topic*" will give information and usage about the specified *topic*. Try typing ">> help clc" or ">> help i".

4. Precision

   All numeric computations in MATLAB are performed with about 16 decimal digits of precision. The format of the displayed output can be controlled by the following commands:

format short	fixed point, 5 digits (default setting)
format long	fixed point, 15 digits
format short e	scientific notation, 5 digits
format long e	scientific notation, 15 digits
format rat	approximate fractions

   Type the following:

```
>> format short e
>> 235.556
>> format long
>> pi
>> format short
>> pi
```

5. Plotting

   Suppose we want to plot the following graph, $y = \sin^2(x)$. The most straightforward way is to pick several values of x, $x_1 = 0, x_2 = 1$, etc. Plot the points $\big(x_i, \sin^2(x_i)\big)$, and join these points. It is clear that this method will only work well when we choose a large number of points which are close together. MATLAB allows us to very efficiently replicate this process. Type the following:

```
>> x = 0: 0.2 : 1 ;
>> y = sin(x).*sin(x);
>> plot(x,y)
```

This does not look like the graph that we want because we used a bad choice of points. Type the following:

```
>> x = 0: 0.01 : 6.28 ;
>> y = sin(x).*sin(x);
>> plot(x,y)
```

This graph looks much better because we used 629 points to plot our graph. Let us dissect these commands line by line.

```
>> x = 0: 0.01 : 6.28 ;
```

We are declaring x as an array (or a row vector) containing the numbers from 0 to 6.28, in increments of 0.01. So x contains 629 values. Notice that the size of x in the Workspace window now says 1×629. Double click on x to see all the individual values. The semicolon ';' asks MATLAB to execute the command but suppresses the output. Type the same line again, this time without the semicolon and see what happens.

You can also work with individual elements in your array. For example,

```
>> x(10) * 2
```

will get the value of the 10-th element of your array x and multiply it by 2. The second line was

```
>> y = sin(x).* sin(x);
```

This command computes $\sin(x) \times \sin(x)$ and saves the answer into an array y. Notice that we used .* instead of the usual * to denote array multiplication, i.e. multiplying 629 values to 629 values. The '.' reminds MATLAB to do the right thing and operate element by element in order. Other examples are:

```
>> z = 2 + x.*exp(-x.^2);
```

Pay attention to where we need to add the '.' and note that the mathematical functions like $\sin(x)$ are smart enough to compute element by element. The third line

```
>> plot(x,y)
```

simply plots the graph of y against x.

$\boxed{\textit{Practice}}$

1. Refer to Example 4 in Chapter 1 of your lecture notes. The solution was

$$v(t) = 4.87\,\frac{1 + 0.345e^{-4.02t}}{1 - 0.345e^{-4.02t}}.$$

We shall plot a smooth graph of this solution.

```
>> t= 0:0.01:10;
>> v= 4.87*(1 + 0.345*exp(-4.02*t))./(1 - 0.345*exp(-4.02*t));
>> plot(t,v)
```

Note that we need to use ./ when dividing an array by another array.

2. Refer to Example 11 in Chapter 1 of your lecture notes. Use MATLAB to plot the curve of T/U (which we denote by y) with the parameters in the lecture given in the lecture notes. Note that the unit of time used is in years and we set the range of t to 1 million years.

```
>> ku = log(2)/245000
>> kt = log(2)/75000
>> t= 0: 10000 : 10^6;
>> y = ku/(kt-ku) * (1 - exp((ku-kt)*t));
>> plot(t,y)
```

Observe that this exponential graph flattens considerably after 500,000 years and the dating will not be accurate. Now, if T/U is 0.35, how can we calculate the age of the coral sample?

The quickest way is to 'eye-ball' the graph and estimate it as somewhere between 200,000 to 300,000 years. Another way is to solve the equation analytically to get the answer 2.4586×10^5 years. Be careful not to read too much into the accuracy of this number since our inputs were only correct up to three significant digits.

A third way is to use the data we have already computed. We ask MATLAB to find the values of T/U that is nearest to 0.35. We can do this by

```
>> [value index]=min(abs( y-0.35));
>> t(index)
```

We are asking for the minimum value of $|T/U - 0.35|$. The *min* command returns the value of the minimum element in the array and saves it in the variable called *value*. At the same time, it saves the index (position) of this minimum element in the variable called *index*. t(index) then returns the time when this minimum occurs. Recall that we only use 101 points at intervals of 10000, so the answer will only be accurate up to that.

Part B: Graphing First and Second Order Differential Equations

1. Stable Solutions of First Order Differential Equations

   Suppose we want to plot several graphs on the same diagram, we can use the command "hold on" to superimpose any new graphs on the current one. The command "hold off" turns off this feature, i.e. any new plot will overwrite the current graph. Several other commands are available for enhancing your plots. For example, you can use the following commands to label the x and y axis and to title your graph.

```
>> xlabel('t')
>> ylabel('y')
>> title('Graph of y = t^2')
```

   You can also use the command

```
>> axis([ xmin xmax ymin ymax])
```

   to set the maximum and minimum values of your x and y axis. Lastly, use

```
>> plot(x,y,'c')
```

   to plot in different colours by replacing 'c' with
   b for blue; **g** for green; **r** for red; **y** for yellow; and **k** for black.

   For black and white plots, we can use

```
>> plot(x,y,'-'); plot(x,y,'-.'); plot(x,y,'--')
```

   for solid lines, dash-dotted lines and dashed lines respectively. More options can be found with 'help plot'.

   $\boxed{\textit{Practice}}$ Consider the first order IVP

$$\frac{dy}{dt} - 2y = -3e^{-t}, \qquad y(0) = y_0.$$

   Verify that the solution is given by

$$y = e^{-t} + (y_0 - 1)e^{2t}.$$

   Let us graph this solution using three different initial values $y_0 = 0.97,\ 1,\ 1.03$.

```
>> t= -3:0.01:3;
>> y1=exp(-t) + (0.97 - 1)*exp(2*t);
>> plot(t,y1,'b')
>> hold on
>> y2=exp(-t) + (1 - 1)*exp(2*t);
>> plot(t,y2,'g')
>> y3=exp(-t) + (1.03 - 1)*exp(2*t);
>> plot(t,y3,'r')
>> axis([0 3 -3 7])
>> hold off
```

Notice that this IVP is very sensitive to small changes in the initial value. A difference of 0.03 in y_0 results in a big difference for y as t gets large. This d.e. is **unstable**.

An example of a stable solution is

$$y = e^{-t} + (y_0 - 1)e^{-2t}.$$

Again we graph with different initial values $y_0 = 0.97$, 1, 1.03.

```
>> t= -3:0.01:3;
>> y1=exp(-t) + (0.97 - 1)*exp(-2*t);
>> plot(t,y1,'b')
>> hold on
>> y2=exp(-t) + (1 - 1)*exp(-2*t);
>> plot(t,y2,'g')
>> y3=exp(-t) + (1.03 - 1)*exp(-2*t);
>> plot(t,y3,'r')
>> axis([-3 3 -3 20])
>> hold off
```

Note that t is usually taken as time and hence we should ignore the part of the graph where $t \leq 0$. Then the three solutions are virtually identical lines.

2. <u>Forced Undamped Harmonic Oscillators</u> (Requires knowledge of Chapter 2.)

   Recall that such a system has a beat if the natural frequency is close to the forcing frequency. An example of such a solution is

$$x(t) = 2\sin\left(\frac{t}{2}\right)\sin\left(\frac{23t}{2}\right).$$

   We can graph this solution as follows:

```
>> t= linspace(0,4*pi,1000);
>> x= 2*sin(t/2).*sin(23*t/2);
>> plot(t,x)
>> xlabel('t')
>> ylabel('x(t)')
>> title('Beats')
>> hold on
>> y= 2*sin(t/2);
>> plot(t,y,'r')
```

Note: the *linspace* command produces 1000 equally spaced points between 0 and 4π.

Exercise 1

Complete the following questions. You are **not** required to submit your solutions.

1. Find the value of $e^{0.5}$, correct to 13 decimal places.
2. Evaluate $\sin^2(23.195) + \sqrt{\tanh(0.12)}$, correct to 13 decimal places.
3. Plot the function
$$y(t) = e^{-t/2}\cos(2t), \qquad 0 \le t \le 10.$$
4. Plot these three functions on the same graph:
$$\sinh(t), \cosh(t), \tanh(t) \qquad -2 \le t \le 2.$$
5. Which of the following IVP is stable? (i.e. the solutions converge for small changes in the initial value.)
   (i) $\dfrac{dy}{dx} - y = -4e^{-x}, \qquad y(0) = 2.$
   (ii) $\dfrac{dy}{dx} - 2y = -6e^{-x}, \qquad y(0) = 2.$
   (iii) $\dfrac{dy}{dx} + 2y = 2e^{-x}, \qquad y(0) = 2.$
6. Find the solution for the following nonhomogeneous d.e.
$$x'' + 2x' + 2x = 2\cos(t), \qquad x(0) = x'(0) = 0.$$
Using different colours, plot a graph containing the three curves $x(t)$, the homogenous solution $x_h(t)$ and the particular solution $x_p(t)$. Use t between 0 to 10 for your horizontal range.

—The End—