

Basic MatLab

MatLab is a powerful software created by MathWorks, which is used extensively in mathematics, engineering, and the sciences. It has powerful numerical and graphic capabilities, so is an invaluable tool for students to learn. This sheet is an introduction to the software for **Math 337**. Basic commands are introduced, and several examples related to class are shown, including a couple of functions and graphics commands.

MatLab stands for Matrix Laboratory, so we begin with some basics applied to matrices. The matrices, A and B , are entered into MatLab in the following manner:

```
A = [1 0 -2 3
     -1 2 0 5
      6 0 1 4
      3 -1 -3 0];
```

```
B = [2 1 0 4;-1 -3 5 1;6 0 0 -2;1 -2 -1 5];
```

The matrices in MatLab are stored in the variables A and B . One enters the rows starting with a `[`, then numbers are entered either separated by a space or comma. New rows to the matrix are produced either with with a `;` separating the rows or using **enter** to move to the next line. The matrix ends when `]` is entered. If this is followed by a `;`, then MatLab does not output the matrix. However, if **enter** is used, MatLab displays the matrix.

Normal arithmetic operations are easily performed, such as $A + B$, $A - B$, or raising a power A^3 , in the last case typing `A^3`. Below shows the matrix multiplication, AB , and the inverse, A^{-1} . Note that if no variable is specified, then MatLab automatically stores the answer in the temporary variable *ans*.

```
A*B
ans =
    -7    -5    -3    23
     1   -17     5    23
    22    -2    -4    42
   -11     6    -5    17
```

```
inv(A)
ans =
   -0.3919    0.1486    0.1081    0.2973
  -1.5811    1.0135   -0.0811    1.0270
    0.1351   -0.1892    0.1351   -0.3784
    0.5541   -0.1757    0.0541   -0.3514
```

MatLab can easily solve the linear equation $Ax = b$. We note that the transpose of a matrix is given by b' in MatLab. Below we show two ways to solve this equation.

```
b = [2 0 1 5];
```

```
x = linsolve(A,b')
x =
```

```

0.8108
1.8919
-1.4865
-0.5946
inv(A)*b'
ans =
0.8108
1.8919
-1.4865
-0.5946

```

Similarly, MatLab can readily find eigenvalues and eigenvectors.

```

eig(A)
ans =
2.3090 + 4.9965i
2.3090 - 4.9965i
-1.9021
1.2842

```

```

[v,d]=eig(A)
v =
Columns 1 through 3
-0.1627 - 0.3412i  -0.1627 + 0.3412i   0.4095
-0.3486 - 0.0541i  -0.3486 + 0.0541i   0.7464
-0.7459           -0.7459           -0.1567
0.0000 - 0.4199i   0.0000 + 0.4199i  -0.5006
Column 4
0.0735
0.9698
-0.1967
-0.1242
d =
Columns 1 through 3
2.3090 + 4.9965i      0      0
0      2.3090 - 4.9965i  0
0      0      -1.9021
0      0      0
Column 4
0
0
0
1.2842

```

We note that all of the previous scripts were produced in MatLab and stored in a text file (test.txt). This is readily done by the MatLab commands:

```

diary('test.txt')
diary on
MatLab entries here

```

diary off

The file *test.txt* will contain all the commands entered and the MatLab output in a text file, which can be easily read in any text editor.

At this time we turn specifically to our murder problem. From the main lectures we saw that we needed to find the values for the heat coefficient, k , and the time of death, t_d , which solved the equations

$$22 + 8e^{-k} = 28 \quad \text{and} \quad 22 + 8e^{-kt_d} = 37.$$

The first equation is equivalent to finding the zero of $f(k) = 22 + 8e^{-k} - 28$. MatLab has an easy way to enter an inline function, and the software has the special function *fzero*, which can be used to numerically find zeroes of a function. This special function *fzero* has the form *fzero*(f , $x0$), where f is the function of some variable, and $x0$ is an initial guess where the zero might be. The sequence of commands below begins with obtaining more digits, then it creates the necessary functions and solves them.

```
format long

f = @(k) 22+8*exp(-k)-28;

k = fzero(f,0.3)
k =
    0.287682072451781

ft = @(t) 22+8*exp(-k*t)-37;

td = fzero(ft,-5)
td =
   -2.185081100344673
```

These values agree well with the ones found in the lecture notes.

As pointed out in lecture, many differential equations cannot be solved exactly. MatLab has powerful routines for numerically solving a differential equation. Here we use the example from our murder problem:

$$\frac{dT(t)}{dt} = -k(T(t) - T_e), \quad T(0) = 30, \quad (1)$$

where $t = 0$ corresponds to 8:30 AM, $T_e = 22^\circ\text{C}$ is the room temperature, and k is the heat coefficient found above using the body temperature at $t = 1$. Rather than working with line commands in the *Command Window* of MatLab, it is valuable to learn to create *M-files* for both functions and scripts.

Before creating a new function or script file, it is a **very good** idea to point the Command Window to the directory where you are going to store your MatLab files. This is done in the window below greyed area for operations. We begin by creating a function, which is done by clicking on the *New* tab and selecting *Function*. In this case, we want to enter the equation for the right hand side of the differential equation (1) and below we see the M-file (stored *murder.m*).

```
function yp = murder(t,y)

% Model for Newton's Law of Cooling
% Parameters for the model
```

```

k = log(4/3); Te = 22;
yp = -k*(y - Te); % This is the RHS of the DE
end

```

Note that MatLab uses % beginning any comments that are not executed.

The next step is to use MatLab's numerical ODE solver to create the solution of the differential equation from the time of death, t_d , found earlier for about 12 hours after the murder. Below we show the solving of the model and plotting of the graph.

```

[t1,y1] = ode23(@murder,[0,td],30);
[t2,y2] = ode23(@murder,[0,12],30);
plot(t1,y1,'b-',t2,y2,'b-');grid;

```

This produces a blue solution from the time of death until 12 hours after $t = 0$. This is quick way to produce a graph, but we want to illustrate the power of MatLab to produce a very good looking, professional graph.

Below we present an M-file, *Script*, which executes a series of commands to produce a good graph. There are comments interior to the program to help understand what is happening, and clearly one could explore other options to improve the graph.

```

clear                % Clear previous definitions
figure(1)           % Assign figure number
clf                 % Clear previous figures
hold off            % Start with fresh graph

mytitle = 'Body Temperature'; % Title
xlabel = '$t$'; % X-label
ylabel = '$T(t)$'; % Y-label

f = @(k) 22+8*exp(-k)-28; % Function for finding heat coefficient
k = fzero(f,0.3); % Find the heat coefficient
ft = @(t) 22+8*exp(-k*t)-37; % Function for finding the time of death
td = fzero(ft,-5); % Find the time of death

[t1,y1] = ode23(@murder,[0,td],30); % Simulate the heat equation
[t2,y2] = ode23(@murder,[0,12],30); % Simulate the heat equation

plot(t1,y1,'k-', 'LineWidth',1.5); % Plot from death to t = 0
hold on % Plots Multiple graphs
plot(t2,y2,'k-', 'LineWidth',1.5); % Plot from t = 0 to 12
plot([-9,td],[37,37],'r-', 'LineWidth',1.5); % Plot from t = -9 to td
plot([-9,12],[22,22],'b:', 'LineWidth',1.5); % Plot room temperature
plot([td,td],[20,40],'k:', 'LineWidth',1.5); % Plot time of murder
grid % Adds Gridlines
text(-2.7,28,'Time of death, $t_d$', 'rot',90,'FontSize',14,...
'FontName','Times New Roman','interpreter','latex');
text(-4.8,22.6,'Room Temperature', 'color','blue','FontSize',14,...
'FontName','Times New Roman');
axis([-9 12 20 40]); % Defines limits of graph

```

```

fontlabs = 'Times New Roman'; % Font type used in labels
xlabel(xlab,'FontSize',14,'FontName',fontlabs,'interpreter','latex');
    % x-Label size and font
ylabel(ylab,'FontSize',14,'FontName',fontlabs,'interpreter','latex');
    % y-Label size and font
title(mytitle,'FontSize',16,'FontName','Times New Roman','interpreter','latex');
    % Title size/font
set(gca,'FontSize',12); % Axis tick font size

print -depsc body_temp.eps % Create figure as EPS file
print -djpeg body_temp.jpg % Create figure as JPEG file

```

Note: All graphing scripts should have the first **4 commands** to start with a clean **Workspace**, clear old and label a new figure, and refresh the figure window. The last lines starting with **fontlabs** give the user a lot of control on the appearance of writing on the graphs, so are very useful to have readily available for any graph. Finally, the last two lines are necessary to have the graph readily available for use in a document (EPS for \LaTeX and JPG for Word). In subsequent graphing script these heading and trailing lines of code will not appear in these write-ups to avoid repetition.

The product of MatLab script entitled *murder_plot*, which is available from my webpage, is

