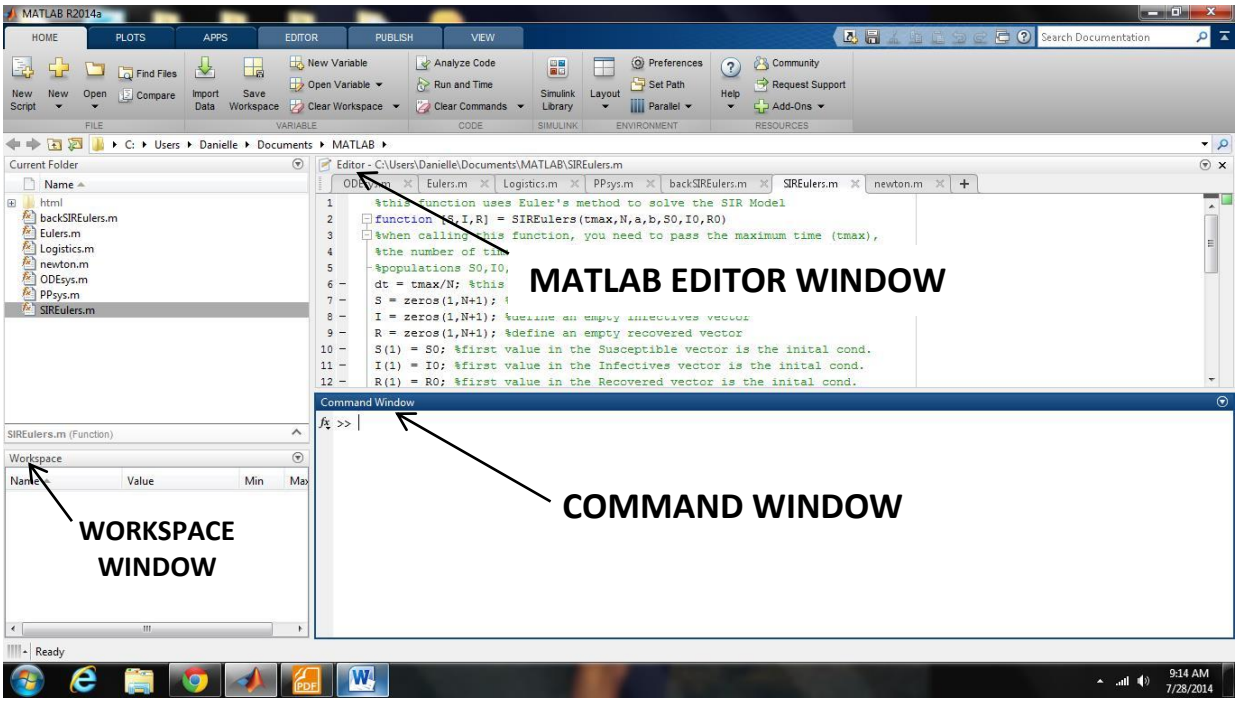
## Section I: Getting Started With Matlab

1. Where is MATLAB available? MATLAB is available on computers in the Undergraduate Computer Lab (UCL) located in RLM 7.122. You may also purchase a student version of MATLAB in the Campus Computer Store located in the FAC.

2. If you choose to use the computers in the UCL, you will need an account. If you are taking a Math course, you're entitled to have a class account. Please go to our undergraduate computer lab (RLM 7.122) and signup for an account yourself.

3. Once you have an account and have logged onto a computer, go to the lower left corner of the screen to access the programs available. One of the choices will be Science and Math. It is in this folder that you will find MATLAB. When you start MATLAB, the following screen will appear (or a somewhat similar screen):




4. There are three components to this main window in Matlab; the *Editor Window*, the *Command Window*, and the *Workspace Window*.
   a. **Editor Window**. Many of the assignments in this class will require the use of executable m-files. You can download these m-files from

      http://www.ma.utexas.edu/users/sadun/F14/M408R/Matlab.html

      You will want to save these m-files to a convenient location (perhaps in the document folder, create a new folder named MATLAB and save them in this folder). Once you have saved the m-file to your computer, you will want to launch MATLAB and open the m-file (you can use the "Open" icon on the MATLAB toolbar). The m-file should now

be open in the Editor Window. If you need to edit an m-file, it will be necessary to use the Editor Window.

b. **Command Window**. Once you have opened your m-file in the Editor Window, you can now call and execute that m-file while in the Command Window. Directions for using specific m-files are included in the m-file section of this document.

c. **Workspace Window**. This window shows the user what data structures are available for use. It also gives the dimensions of these data structures. Users can manipulate these variables when working in the Command Window. For example, plots can be created using the appropriate variables.

## Section II: MATLAB Basics

1. MATLAB uses the following characters for the basic arithmetic characters; '+' for addition, '-' for subtraction, '*' for multiplication, '/' for division, and '^' for exponentiation. For example, to perform the operation $3x + x^3$ in MATLAB, you would enter in the Command Window:

```
3*x + x^3
```

MATLAB observes the basic rules for the order of operations performed.

2. In MATLAB, the semi-colon ';' plays an important role. If you place a semi-colon at the end of a MATLAB statement, it suppresses the output from appearing on the screen. We use a simple example to illustrate the purpose of a semi-colon; compare what occurs when you perform the following sequence of commands in the Command Window, first without a semi-colon and then with a semi-colon (note what is showing in the Workspace Window as you perform these operations).

```
x = 2
y = 4
x + y
```

It is often useful to clear the current variables from the workspace. The command

```
clear all
```

removes all the variables currently in your Workspace Window from memory. If you would like a "clean screen" in your Command Window, (that is, remove all the previous input/output that is showing in the Command Window) you simply enter

```
clc
```

to clear the Command Window.

3. MATLAB is particularly efficient when working with vectors (similar to arrays or lists of numbers). Suppose you want to create a vector containing your exam scores in a particular class. If your exam scores are 89, 77, 94, and 86, then the following MATLAB command would create a row vector containing these scores (you may also use a comma between entries if you would prefer);

```
s = [89 77 94 86]
```

You have now created a row vector containing your exam scores (observe what is showing in your Workspace Window). If you would prefer to have this information as a column vector, you would simply amend your earlier command to include an apostrophe;

```
s = [89 77 94 86]'
```

You may also create a column vector by using a semi-colon between the entries;

```
s = [89;77;94;86]
```

## Section III: The Basics of Graphing

1. **Plotting Data Vectors**: When you have two data vectors of the same size, you can create a plot of the ordered pairs that result from pairing the corresponding elements in each data vector. For example, suppose you have the following set of ordered pairs representing time (seconds) and speed (mph) respectively; (0,0), (2,5), (4,8), (7,12), (10, 15), (11,15), (14,16), (15,16). To create a plot of this data using an asterisk to represent the points, we could enter the following sequence of commands in the Command Window:

```
T = [0 2 4 7 10 11 14 15];
S = [0 5 8 12 15 15 16 16];
plot(T,S,'*')
```

The first command creates a vector, T, which contains the times when the measurements occurred. The second command creates a vector, S, which contains the measured speeds. The third command creates a plot of the order pairs (T, S) where the horizontal axis represents T and the vertical axis represents S. Furthermore, it uses an asterisk for the point marker (you are free to use any type of marker). We next use a sequence of commands that creates the appropriate labels for the plot.

```
xlabel('Seconds');
ylabel('mph');
title('Time vs. Speed Plot');
```

The first command labels the horizontal axis "Seconds," the next command labels the vertical axis "mph", and the third command provides a title for the plot. Finally, we may want to use a red line (you could use any color) to connect the points on our existing plot. To do this we use the following commands:

```
hold on
plot(T,S,'r')
```

This first command, "hold on", tells MATLAB to keep everything on our previous plot. Without this command, MATLAB would create an entirely new plot. The second command will create a line plot connecting the data points with a red line. If we would not have used '`r`', MATLAB would have defaulted to using blue for the line color.

There a many other options available. The MATLAB Help mode provides information on all options available.

2. **Graphing an equation using "ezplot"**: There are several ways that we can graph an equation in MATLAB. We will discuss two of these options. We begin with discussing "inline" functions and using the "ezplot" option. Suppose that we would like to create a graph of $y = \sin(2x) + 3$. We could do this using the following sequence of commands:

```
y = inline('sin(2*x) +3');
ezplot(y,[0,4*pi]);
```

The first command creates what MATLAB refers to as an "inline" equation. This treats the function as a string of characters. While it does have its advantages (easy to create), there are disadvantages (limited options for manipulating graph and the function). The second command is used to plot an inline function. Included in the ezplot command is the interval that we want to use for *x*.

3. **Graphing an equation using "plot"**: In most instances, we prefer to use the plot command when possible as MATLAB is operates most efficiently when working with vectors. The following sequence of commands would be used to graph $y = \sin(2x) + 3$ on the interval $[0, 4\pi]$ using the plot command:

```
x = linspace(0, 4*pi, 100);
for i = 1:100
   y(i) = sin(2*x(i)) + 3;
end
plot(x,y)
```

This first line in the above sequence divides the interval $[0, 4\pi]$ into 100 partition points and places these points in a vector named **x**. To see what is contained in the vector **x**, simply enter `x` in the command window. Note that the entries in the vector **x** uses the indices from 1 to 100. For the above example, $x(1) = 0$, $x(2) = 4\pi/100$, $x(3) = 2(4\pi/100)$, …, $x(100) = 4\pi$. The next three lines are a loop that will be completed 100 times. Each time the loop is executed it creates an entry in the vector **y**, denoted by y(i). The entry y(i) contains the function value when it is evaluated at x(i). Finally, the final line simply plots the vector **x** versus the vector **y**.

## Section IV: MATLAB m-files

In this section, we include specific directions for using the m-files necessary for this course. An m-file is a simple text file where you can place a sequence of MATLAB commands. When the file is executed, MATLAB reads the commands and executes them exactly as it would if you had typed each command sequentially at the MATLAB Command Window. All m-file names must end with the extension '.m' (e.g. test.m). If you are confused on how to use a particular m-file, please read the comments that are included in the m-file (they are in green and follow percent signs). The comment lines contain information that helps explain how to use the m-file. Below are detailed instructions for the first three m-files that you will encounter in the course.

**WaterDensity.m.** The first m-file that you will use in this course is called WaterDensity.m. It contains information needed to complete Worksheet 2. You will want to download this m-file from

http://www.ma.utexas.edu/users/sadun/F14/M408R/Matlab.html

and save it to your MATLAB folder. Once you have done this, open the file in your MATLAB editor. You will see the following;

```
%This function creates a plot of Density vs. Temperature as well as a
%table containing the data vectors
function [Density, TD] = WaterDensity
%to run this program, you simply need to enter into the Command Window
%WaterDensity;
%If you would like to return the Density vector and Temperature vector
%you would want to enter
%[D,T]=WaterDensity;
%into the Command Window.  You can cut and paste the above commands
Density = [999.841, 999.847, 999.854, 999.86, 999.866, 999.872, 999.878,...
    999.884, 999.889,999.895, 999.9, 999.905, 999.909, 999.914, 999.918,...
    999.923, 999.927, 999.93, 999.934, 999.938, 999.941, 999.944,...
    999.947, 999.95, 999.953, 999.955, 999.958, 999.96, 999.962, 999.964,...
    999.965, 999.967, 999.968, 999.969, 999.97, 999.971, 999.972, 999.972,...
    999.973, 999.973, 999.973, 999.973, 999.973, 999.972, 999.972,
999.972,...
    999.97, 999.969, 999.968, 999.966, 999.965, 999.963, 999.961, 999.959,...
    999.957, 999.955, 999.952, 999.95, 999.947, 999.944, 999.941, 999.938,...
    999.935, 999.931, 999.927, 999.924, 999.92, 999.916, 999.911, 999.907,...
    999.902, 999.898, 999.893, 999.888, 999.883, 999.877, 999.872,
999.866,...
    999.861, 999.855, 999.849, 999.843, 999.837, 999.83, 999.824, 999.817,...
    999.81, 999.803, 999.796, 999.789, 999.781, 999.774, 999.766, 999.758,...
    999.751, 999.742, 999.734, 999.726, 999.717, 999.709, 999.7];
TD = linspace(0,10,101);%temperature vector
plot(TD,Density)
xlabel('Temperature (Celsius)'),ylabel('Density (kg/m^3)')
table(TD',Density','VariableNames',{'Temperature','Density'})
```

Any text following a percent sign (%) is a comment and provides insight about the m-file. The first executable line of the m-file contains the word "function". The function line of an m-file provides the user with insight on how to use a particular m-file. For example, we see that this

particular m-file is named WaterDensity. The portion in the brackets (`[Density, TD]`) lets the user know that if one chooses to, they can return the vectors Density and TD to the Workspace Window. In order to execute the m-file and return the two vectors, one would enter

```
[Density, TD] = WaterDensity;
```

in the Command Window. If you don't use a semi-colon at the end of this command, MATLAB will show the two vectors Density and TD, and you really don't want this. By returning the vectors Density and TD you now have the option of working with them in the Command Window. If you do not want to return the two vectors, you can simply enter

```
WaterDensity;
```

in the Command Window (for Worksheet 2 it is not necessary to return the two vectors). If you further look at the entries in this m-file, you will recognize all of the commands used (they were talked about above). Again, an m-file is simply a sequence of MATLAB commands that can be executed repeatedly without having to reenter the commands.

**SIR Model m-file**. Another m-file that you will use in this course is called SIREulers.m. It provides a numerical approximation (using Euler's method) for the solution of the SIR model. As before, you will first need to download the file SIREulers.m from

http://www.ma.utexas.edu/users/sadun/F14/M408R/Matlab.html

and save it into your MATLAB folder. Once this is done, you simply go to the Open icon and select open SIREulers.m. The m-file should now appear in your Editor Window and is now ready to use. The entire SIREuler's m-file is included below. We've included line numbering for easy reference.

```
(1)  %this function uses Euler's method to solve the SIR Model
(2)  function [S,I,R,t] = SIREulers(tmax,N,a,b,S0,I0,R0)
(3)  %when calling this function, you need to pass the maximum time (tmax),
(4)  %the number of time steps (N), the parameters a and b, and the initial
(5)  %populations S0,I0, and R0.
(6)  dt = tmax/N; %this determines the size of the time step h
(7)  S = zeros(1,N+1); %define an empty susceptible vector
(8)  I = zeros(1,N+1); %define an empty infectives vector
(9)  R = zeros(1,N+1); %define an empty recovered vector
(10) S(1) = S0; %first value in the Susceptible vector is the inital cond.
(11) I(1) = I0; %first value in the Infectives vector is the inital cond.
(12) R(1) = R0; %first value in the Recovered vector is the inital cond.
(13) for n = 1:N %this is the iterative loop for Euler's Method
(14)    S(n+1) = S(n) - a*S(n)*I(n)*dt;%D.E. for Susceptibles
(15)    I(n+1) = I(n) + (a*S(n)*I(n) - b*I(n))*dt;%D.E. for Infectives
(16)    R(n+1) = R(n) + b*I(n)*dt;%D.E. for Recovered
(17) end %end the for loop
(18) t = linspace(0,tmax,N+1); %this is used only for the plotting
(19) h1 = figure;%provide a handle for first figure (Susceptibles)
(20) h2 = figure;%provide a handle for second figure (Infectives)
(21) h3 = figure;%provide a handle for third figure (Recovered)
```

```
(22) figure(h1);
(23) plot(t,S)
(24) ylabel('Susceptibles'), xlabel('Time'), title('Euler''s Method Example')
(25) figure(h2);
(26) plot(t,I)
(27) ylabel('Infectives'), xlabel('Time'), title('Euler''s Method Example')
```

**General Notes and Comments**

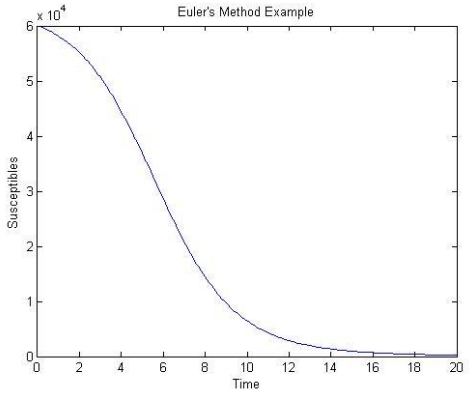
1. Anything following a percent sign (%) is a comment and is included for documentation of the program.

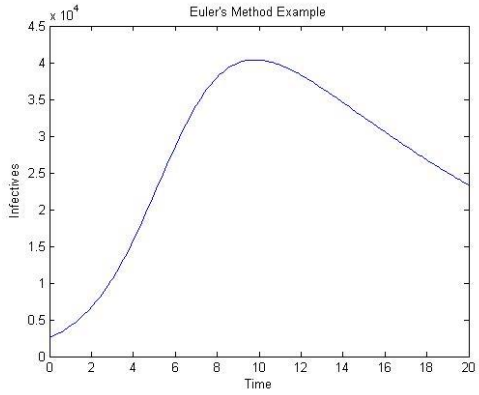
2. Line (2) contains an essential line for all m-files. It contains the name of the function, the necessary input parameters (`tmax,N,a,b,S0,I0,R0` are all needed input parameters) , as well as the variables that may be returned to the Command workspace (the S, I, R, and t). Once an m-file is saved to the current MATLAB directory, you will be able to use the m-file in your Command Window.

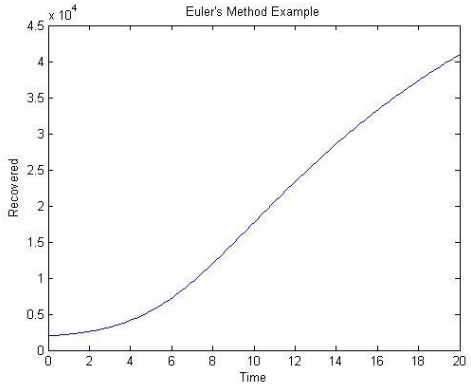
3. To execute/run the m-file, you have two options: Option 1 is to simply specify the needed input parameters and execute the m-file. The needed input parameters will always be contained in the parenthesis that follow the name of the function. In this m-file, we see `SIREulers(tmax,N,a,b,S0,I0,R0)`. Thus, we must specify values for the parameters `tmax`, `N`, `a`, `b`, `S0`, `I0`, and `R0` in order to execute the m-file. This can be done by entering the following in the Command Window;

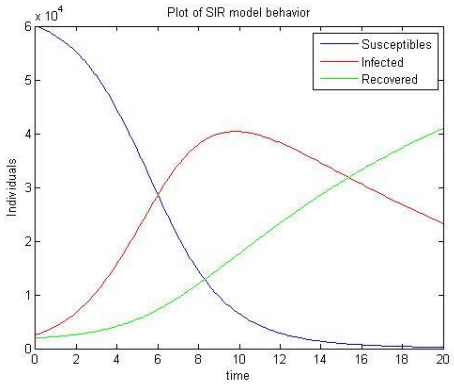
```
SIREulers(20, 200, .00001, 1/14, 60000, 2500, 2000);
```

This command executes SIREulers.m and passes the following parameter values to the m-file; `tmax = 20, N = 200, a = .00001, b = 1/14, S0 = 60000, I0 = 2500, R0 = 2000`. It should return four plots; one for the number of susceptibles (S), one for the number infected (I), one for the number of recovered (R), and one that contains all three plots on a single set of axes.






Option 2 (a better option) for executing SIREulers.m is to allow the m-file to return the data vectors for **S**, **I**, **R** and **t** to the Workspace memory. This allows us to further work with the returned vectors to create additional plots, tables, or perform data analysis. To return the vectors **S**, **I**, **R**, and **t** to the Workspace memory we amend our previous command to include `[S,I,R,t]` (feel free to use other variable names);

```
[S,I,R,t] = SIREulers(20,200,.00001,1/14,60000,2500,2000);
```

When you execute this command, it will still return the previous plots, but you will now see the vectors **S**, **I**, **R,** and **t** in the Matlab Workspace Window. You can now manipulate these vectors in the Command Window using Matlab commands.

**Eulers.m m-file**. In the learning module for Section 2.1, you are asked to use the m-file appropriately named Eulers.m. This m-file requires that you pass the following values to the m-file: the time you want to end the simulation at (`tmax`), the number of time steps that you want to use (`N`), the growth parameter (`r`), the harvest rate (`h`), and the initial amount (`y0`). In order to execute the m-file, you will want to enter the following command in the Command Window:

```
y = Eulers(tmax,N,r,h,y0);
```

where you have specified numerical values for `tmax, N, r, h,` and `y0.` In all of the m-files where Euler's method is used, remember that the step size can be computed by `tmax/N` (that is, $\Delta t$ = `tmax/N`). When computing the step size in this manner, we are assuming that the interval we are working over is [0, `tmax`]. By choosing to enter "`y =`" in our call to the m-file, MATLAB will return the solution vector **y**.