**BME 313L: Introduction to Numerical Methods in Biomedical Engineering**

**Lab Report**

**Lab \_1: Introduction to MATLAB (Chapters 1-3)**

**Last name, First name: Liu, Vincent**

**EID: VL5649**

**Lab Section: 14035 (Tuesday 9:30-12:30)**

**Problem 1**

The density of freshwater can be computed as a function of temperature with the following cubic equation:

where ρ = density (g/cm3) and TC = temperature (ºC). Use MATLAB to generate a vector of temperatures ranging from 32 ºF to 93.2 ºF using increments of 3.6 ºF. Convert this vector to degrees Celsius and then compute a vector of densities based on the cubic formula. Create a plot of ρ versus TC.

Recall that TC = 5/9(TF - 32). (This problem is from textbook P.45, problem 2.9)

**MATLAB code:**

T\_F = 32:3.6:93.2; %creates a vector of temperatures

T\_C = 5 / 9 \* (T\_F - 32); %Converts the temperature

Density = (5.5289e-8 \* T\_C.^3) - (8.5016e-6 \* T\_C.^2) + (6.5622e-5 \* T\_C) + 0.99987; %Computes the density based on the temperatures

plot(T\_C,Density); %plots the graph

xlabel('Temperature (°C)'); %graph labels

ylabel('Density');

title('Density of Freshwater with Respect to Temperature');

**MATLAB function:**

To approach this problem, first we had to generate a vector of temperatures (in Fahrenheit) for which the different densities of water could be calculated. We could then convert the temperatures from Fahrenheit to Celsius. These temperatures could then be used in the equation given to calculate the density of water for each given temperature. These values could then be plotted.

T\_F = 32:3.6:93.2; %creates a vector of temperatures

This line of code generates a vector ranging from 32 to 93.2, the minimum and maximum temperatures given in Fahrenheit. This vector is stored under the variable ‘T\_F’, shorthand for temperature in Fahrenheit, as to keep it separate from the later temperature in Celsius. By including the 3.6 sandwiched between the colons, we specify that the values should increase incrementally by 3.6 every time. If the increment was not specified, the temperatures would go up by whole integers instead.

T\_C = 5 / 9 \* (T\_F - 32); %Converts the temperature

The next line of code is very straightforward—it converts the temperatures, in the vector we generated, from Celsius to Fahrenheit, and then stores it as the variable ‘T\_C’. To do this, the code subtracts 32 from each of the temperatures in the vector, ‘T\_F’, and then multiplies the value by 5/9. Because 5/9 is just a scalar, we do not need to worry about element wise multiplication.

Density = (5.5289e-8 .\* T\_C.^3) - (8.5016e-6 .\* T\_C.^2) + (6.5622e-5 .\* T\_C) + 0.99987; %Computes the density based on the temperatures

This line creates the variable, Density, which the densities for each temperature are stored under. converts the temperatures that we obtained from the last line, into density, with the equation given. Special care must be given because the list of temperatures, ‘T\_C’, is a vector. Therefore, we must use .^ to perform element wise power operations; however, .\* does not need to be used as all the multiplications in the equation are with scalars.

plot(T\_C,Density); %plots the graph

This line plots the calculated densities on the y axis, against the temperatures in Celsius, on the x axis.

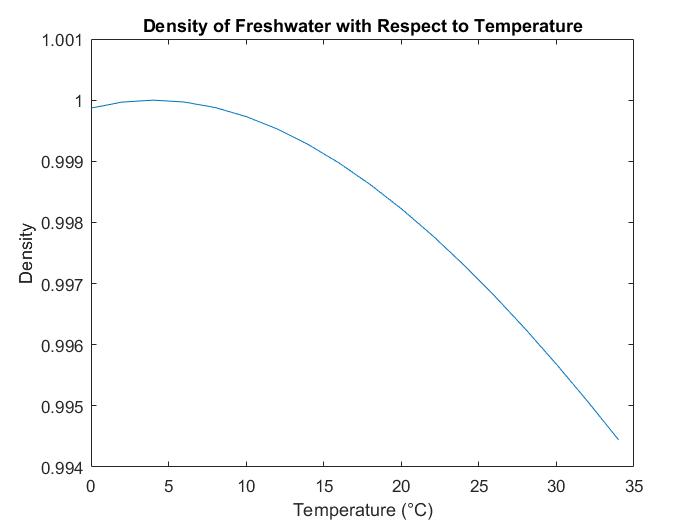
xlabel('Temperature (°C)'); %graph labels

ylabel('Density');

title('Density of Freshwater with Respect to Temperature');

The last 3 lines are used just to clarify what each of the axes are as well as to add a title to the graph.

**Results:**



**Discussion:**

As shown by the results, the density of water increases for a little bit as temperature increases, up until around 5 degrees Celsius or 1 g/mL. Beyond that, for temperatures 6+, the density of water decreases exponentially all the way down to around .9945 g/mL at around 34 degrees Celsius. Overall, the graph shows that the density of water is not always a constant value and because of the constant movement of the molecules, varies around .006 g/mL in 30 degrees.

From this we learn that the density of water can be modeled by the equation given. We also learned that the density of water is not the constant 1 g/mL that we are often taught, and instead fluctuates between the values of .994 and 1, likely because of the expansion and contraction of molecules when excited. In this problem, we learned how to implement vectors with irregular intervals. Lastly, we also learned how to perform element-wise operations to avoid any errors that MATLAB would have encountered performing vector operations.

**Problem 2**

The butterfly curve is given by the following parametric equations:

Generate values of x and y for values of *t* from 0 to 100 with Δt = 1/16. Construct plots of (**a**) *x* and *y* versus *t* and (**b**) *y* versus *x*. Use subplot to stack these plots vertically and make the plot in (**b**) square. Include titles and axis labels on both plots and a legend for (**a**). For (**a**), employ a dotted line for *y* in order to distinguish it from *x*. (This problem is from textbook P.47, problem 2.22)

**MATLAB code:**

t = 0:1/16:100; %generates values of t

x = sin(t) .\* (exp(cos(t)) - 2 \* cos(4 \* t) - (sin(t/12)).^5); %given parametric equations

y = cos(t) .\* (exp(cos(t)) - 2 \* cos(4 \* t) - (sin(t/12)).^5);

subplot(2,1,1) %sets subplot location

plot(t,x,t,y,'--') %makes graph

xlabel('t values') %graph labels

ylabel('x and y values')

title('Graph of x and y versus t')

legend('x','y')

subplot(2,1,2) %sets 2nd subplot location

plot(x,y) %makes second graph

axis square %makes graph a square

xlabel('x values') %graph labels

ylabel('y values')

title('Graph of y versus x')

**MATLAB function:**

The goal of this problem was to generate a set of t values and calculate x and y using the given equations. To do this, we first created a vector of all the t values with the specified interval between. We then could plug the entire vector and work, element-wise, with the equations given to produce x and y. From there it was a simple matter of plotting graphs of the value and labeling the graphs

t = 0:1/16:100; %generates values of t

This first line generates a vector of values ranging from 0 to 100, in increments of 1/16 or .0625. This vector is stored as the variable t to be used later to calculate x and y.

x = sin(t) .\* (exp(cos(t)) - 2 \* cos(4 \* t) - (sin(t/12)).^5); %given parametric equations

y = cos(t) .\* (exp(cos(t)) - 2 \* cos(4 \* t) - (sin(t/12)).^5);

These 2 lines of code generate values of x and y, using the given parametric equations, for the values of t generated in the line before. For operations, such as cos and sin, they function element-wise on elements in a vector, so no care needs to be taken. When multiplying the vector, t, by a constant, no care needs to be taken either because it is just scalar multiplication. However, for the exponent, .^ must be used so as to only raise each element in the vector to the power, rather than performing matrix multiplication on the entire vector.

subplot(2,1,1) %sets subplot location

This line sets the dimensions for the subplot as well as specifies the location for the plot created in the next line. The first 2 parameters specify the dimensions (a 2x1 block) and the 3rd parameter specifies the 1st location for the graph

plot(t,x,t,y,'--') %makes graph

This line plots the 2 sets of data on the graph. Taken in order of sets, x and y are both plotted against t.

The ‘--‘ line at the end specifies that the graph that it proceeds should be comprised of dotted lines (in this case, y versus t).

xlabel('t values') %graph labels

ylabel('x and y values')

title('Graph of x and y versus t')

legend('x','y')

These lines add labels to the graph making it more understandable. The axes are labeled, a title added, and a legend is added, distinguishing the solid and the dashed graphs by the values graphed against t. The labels in the legend for the function are in the order that they are coded above, with ‘x’ corresponding to the x versus t graph and ‘y’ corresponding to the y versus t graph.

subplot(2,1,2) %sets 2nd subplot location

plot(x,y) %makes second graph

Like the subplot and plot lines used earlier, these lines of code set the position of the 2nd graph in position 2, below that of the 1st graph. In this case, the plot is of the values of x and y, generated by the given parametric equations. Combined, these values create our so called ‘butterfly’ curve.

axis square %makes graph a square

In order to make plot b square, like specified in the problem, we use this line of code. Basically, sets the aspect ratio of the graph to a square without missing any of the plotted data, centering the origin.

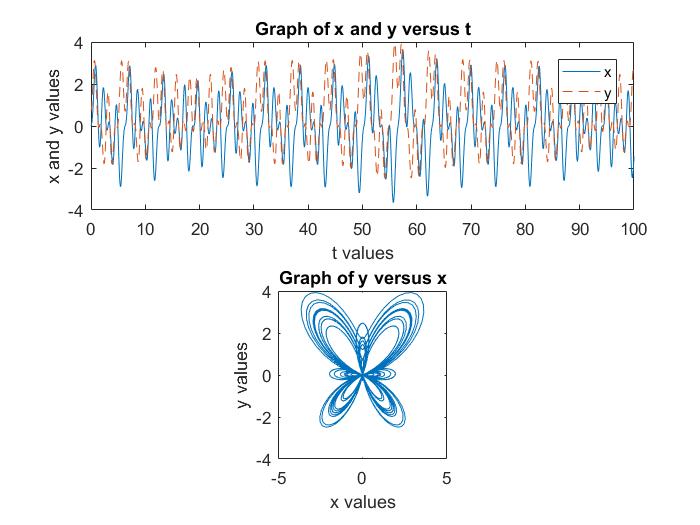
xlabel('x values') %graph labels

ylabel('y values')

title('Graph of y versus x')

These last lines of code clarify the axes of the second graph while titling it, making it easier to understand.

**Results:**



**Discussion:**

As shown by the results of the graph, the graphs generated by x and y are very similar, a result of them differing only by a sin(t) and a cos(t). The values of x and y fluctuate between -4 and 4, in a seemingly random pattern which are used to create the butterfly curve in the second graph. The important take away from this problem is that sometimes, what may seem to be random patterns can form intricate designs.

From this problem, we learned that what we might overlook as random can turn out to be something very specific. Another take away from this problem is the ability to plot multiple figures to clarify and make sense of data. Furthermore, we developed our ability to clarify figures so that the can be better understood by others.

**Problem 3**

An amount of money P is invested in an account where interest is compounded at the end of the period. The future worth F yielded at an interest rate i after n periods may be determined from the following formula:

Write an M-file that will calculate the future worth of an investment for each year from 1 through n. The input to the function should include the initial investment *P*, the interest rate *i* (as a decimal), and the number of years *n* for which the future worth is to be calculated. The output should consist of a table with headings and columns for *n* and *F*. Run the program for *P* = $100,000, *i* = 0.05, and *n* = 10 years. (This problem is from textbook P.83, problem 3.2)

**MATLAB code:**

**Function:**

function output = InvestWorth\_VL(P,i,n) %sets function name

Year = (1:n)'; %creates a vector of years

Money = P .\*(1 + i) .^Year; %calculates money for a given year

output = table(Year,Money); %creates table

end %closes function

**Main script:**

P = 10^5; %variables

i = 0.05;

n = 10;

InvestWorth\_VL(P,i,n) %calls investworth function

**MATLAB function:**

The purpose of this code is to output a table from a function called from another file. The variables used in the function are defined in the main script, so all the function has to do is manipulate the variables and create a table.

function output = InvestWorth\_VL(P,i,n) %sets function name

This first line of code creates a function with the intended output as the variable ‘output’, titled as ‘InvestWorth\_VL(P,i,n)’. This is important because first it already outlines what the function Is supposed to do (outputs ‘output’) and it also sets a name for which we can call the function with, later.

Year = (1:n)'; %creates a vector of years

This line of code generates a vector from 1 to n of whole integers and stores it as the variable “Year”. An apostrophe was added at the end so as to orient the vector, vertically, because we will need to use it in a table later.

Money = P .\*(1 + i) .^Year; %calculates money for a given year

This line of code is the equation that calculates the money, with interest, for any given year. The equation takes the variables defined in the main script and outputs the amount of money. “.^” must be used to prevent errors from working with the ‘Year’ vector. The output should also be a 10x1 vector like the Year vector.

output = table(Year,Money); %creates table

This line of code creates the table that is outputted by the function. As the last functional line of code in the function and the intended output, the variable used is also delineated as part of the first line of code. The code puts the vector ‘Year’ in the first column and the vector ‘Money’ in the second column. As the ‘Money’ vector was generated element-wise with values from the ‘Year’ vector, the amounts should line up with the year.

end %closes function

This last line of code in the ‘investworth‘ program is pretty straightforward— it closes the function so that nothing else after it would be interpreted as part of the function..

P = 10^5; %variables

i = 0.05;

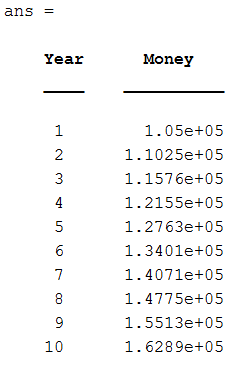
n = 10;

These first 3 lines of the main script outline the parameters to be used by the InvestWorth function.

InvestWorth\_VL(P,i,n) %calls investworth function

This line of code calls the ‘InvestWorth\_VL(P,i,n)’ function and outputs it. In this case, the output of the function is the table of values.

**Results:**



**Discussion:**

As shown by the results, after 10 years at a .05 investment rate and an initial investment of $100,000, the user would have a total of $162,890. The program also displays the amount of money the user would have for any given year leading up to 10, in an easy to read and intuitive fashion. As expected, the total amount of money increases as the duration increases.

From this problem, we learned to implement a function from another file with predefined variables. This could have been taken a step further by prompting the user for an input using input(‘prompt’). We also learned how to manipulate vectors in order to work with them in different orientations to fit the needs of the problem. Lastly, we learned how to display sets of data in different ways (a table in this case, instead of a plot) for an output that is easier to understand.