!--GNU Octave was used--!

?!-GNU Octave Copy-To-Clipboard is not working properly so the output may not look as it should be

**Problem 1**

Call the mysumscript

**-Files-**

**mysum.m**

This is the main function

function [ Sum ] = mysum( n )

% j is Result

j = 0;

% For loop from 1 to n

for i = 1:n

j = j + i;

end

% Transfer j to the output variable, Sum

Sum = j;

end

**mysumscript.m**

Call this and you get the mysum(20) and mysum(100)

fprintf('Printing mysum(20): ');

disp(mysum(20));

fprintf('\n');

fprintf('Printing mysum(100): ');

disp(mysum(100));

fprintf('\n');

**Output Section**

>> mysumscript

Printing mysum(20): 210

؀Printing mysum(100): 5050

**Analysis/Results/Discussion Section (ARD Section)**

The main point of doing this problem is to become familiar with matlab or octave syntax and functionalities. More specifically, we are getting used to the for loop as well as the function syntax of Matlab/Octave.

There is no Mathematical concept involved.

**Problem 2**

Call the vectornormscript.m

**Files Section**

**-vectornorm.m**

function [ norm ] = vectornorm( x )

% 2-Norm Function

% (Sum of x^2)^(1/2)

% Initialize Sum with 0

Sum = 0;

% for loop the (Sum of x^2)

for i = x

Sum = Sum + i^2;

end

norm = Sum^(1/2);

end

**-vectornormscript.m**

fprintf('Printing when x = [1, 1, 1]: ');

x = [1,1,1];

disp(vectornorm(x));

fprintf('\n');

fprintf('Printing when x = [1/(2^(1/2)), 0, 1/(2^(1/2))]: ');

x = [1/(2^(1/2)), 0, 1/(2^(1/2))];

disp(vectornorm(x));

fprintf('\n');

fprintf('Printing when x = 0:0.01:1: ');

x = 0:0.01:1;

disp(vectornorm(x));

fprintf('\n');

**Output Section**

>> vectornormscript

ណവPrinting when x = [1, 1, 1]: 1.7321

Printing when x = [1/(2^(1/2)), 0, 1/(2^(1/2))]: 1.00000

Printing when x = 0:0.01:1: 5.8168

**ARD Section**

The relevant mathematical concept is the computer arithmetic. Computer arithmetic cannot have all the decimal values in its system due to limited capacity and thus, most decimals in the right-most are cut off or rounded off. Thus, this causes error but approximation using the rounding or chopping method are used.

For example, the first input of [1,1,1] has an output of 1.7321 but it is longer than that: 1.73205080757..... Here, the output is rounded at the 4th decimal value from the decimal point, resulting in 1.7321

**Problem 3**

SinCosine must be called

**Files Section**

**-SinCosine.m**

#Meshwidth of 0.0001 for each axis

x = -4:0.0001:4;

y = -2:0.0001:2;

#plot the functions

plot(x, sin(x), '-b', "linewidth", 2);

hold on;

plot(x, cos(x), '-r', "linewidth", 2);

#Labeling using xlabel and ylabel

xlabel("x");

ylabel("y");

#plot the y area after the plotting

ylim([-2, 2]);

#legend

legend("cos(x)", "sin(x)");

#ticks

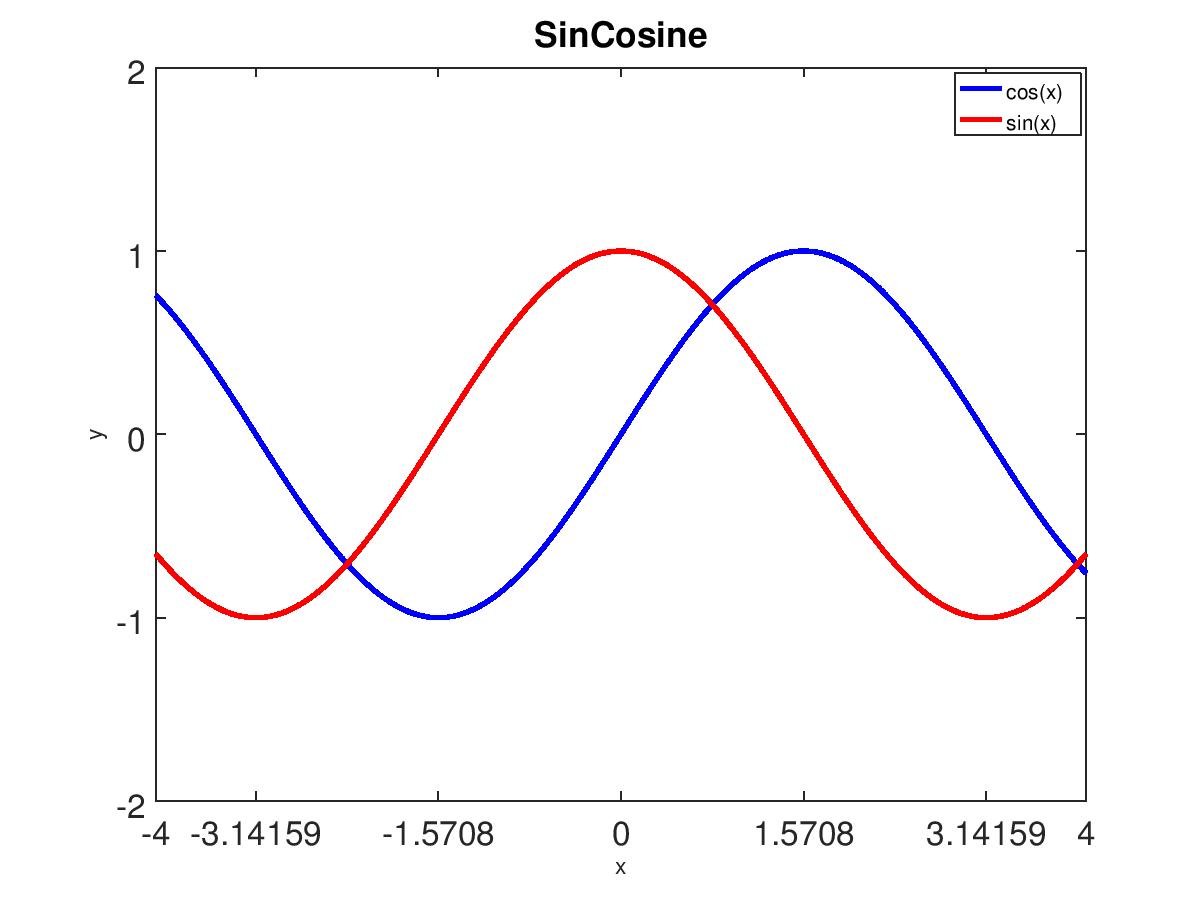
set(gca, 'fontsize', 16);

set(gca, 'xtick', [-4, -pi, -pi/2, 0, pi/2, pi, 4]);

#title

title("SinCosine");

**Output Section**



**ARD Section**

One thing that I definitely learned here is the ordering of the syntax. xlabel and ylabel must be written after the plot() functions in order to show on the graph and this grammar applies to many other functions in the next problem.

This problem was a tutorial or familiarization for the incoming students on plotting graphs, ticks, labels, legends, titles, colors, and many more.

The only mathematical model is the binary machine numbers or computer arithmetic. It is consistent in with the plot graph and values.

**Problem 4**

arctan must be called

**Files Section**

**-arctan.m**

x = -6:0.0001:6;

y = -2:0.0001:2;

plot(x, atan(x), '-b', "linewidth", 2);

ylim([-2, 2]);

#Plot the asymptotes

hold on;

ny = -pi/2;

py = pi/2;

plot([-6, 6], [ny, ny], '--r');

plot([-6, 6], [py, py], '--r');

xlabel("x");

ylabel("y = arctan(x)");

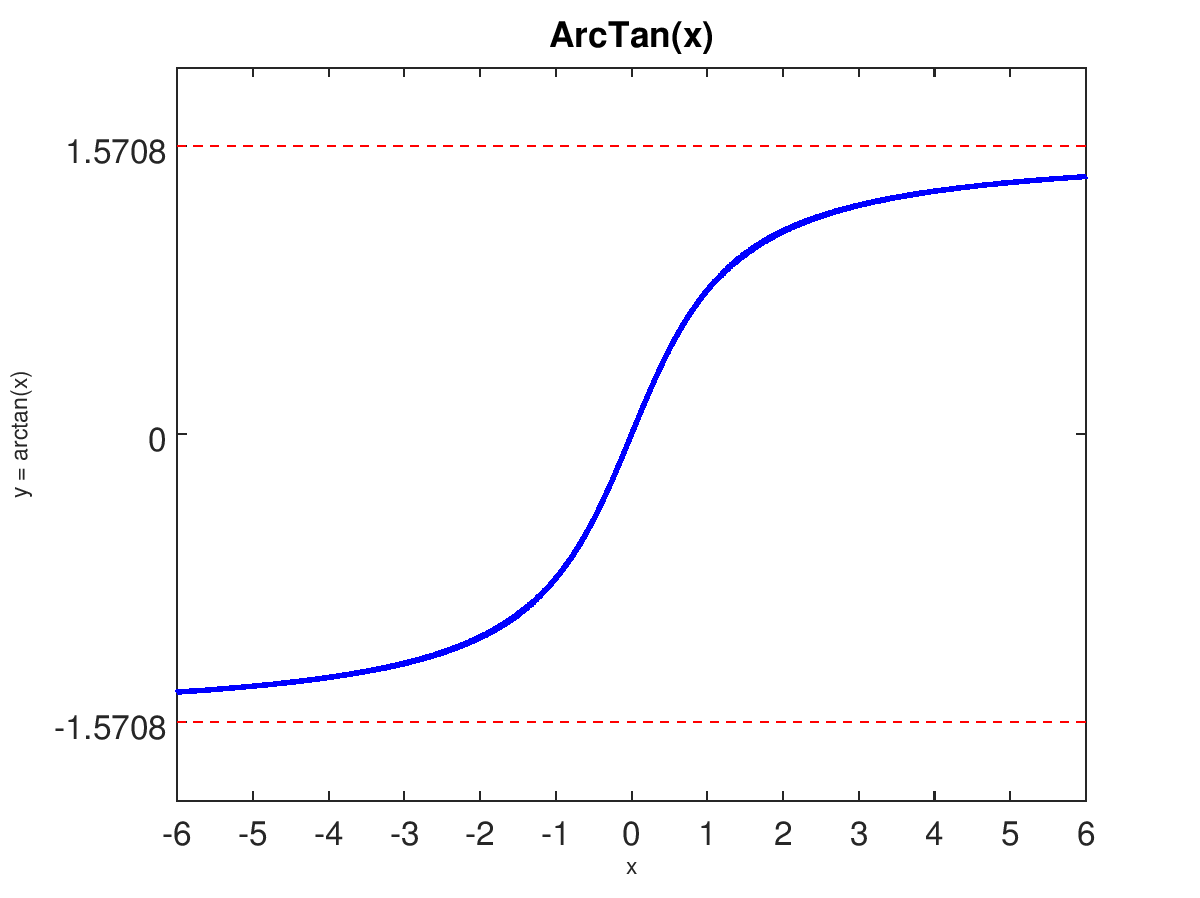
set(gca, 'fontsize', 16);

set(gca, 'xtick', [-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6]);

set(gca, 'ytick', [-pi, -pi/2, 0, pi/2, pi]);

title("ArcTan(x)");

**Output Section**



**ARD Section**

The relevant mathematical concept for this equation is the Rates of Convergence. Although the graph does not show at which x integer the graph is reaching the limit of -1.5708 and +1.5708 but it does indicate that it is approaching that y value.

The arctangent function has a solid blue line with linewidth of 2 and the horizontal asymptotes has a dashed red lines. Other graph modification are x and y ticks.

**Problem 5**

Call…

PlotA - For problem (a) plot

PlotSemiLogX - For semilogx plot

PlotSemiLogY - For semilogy plot

PlotLogLog - For loglog plot

**Files Section**

**-PlotA.m**

x = 1:0.2:2;

y = [1.0139, 0.7959, 0.6249, 0.4906, 0.3851, 0.3023];

#(a)

plot(x, y);

set(gca, 'fontsize', 16);

xlabel('x');

ylabel('y');

title('plot(x, y)');

**-PlotSemiLogX.m**

x = 1:0.2:2;

y = [1.0139, 0.7959, 0.6249, 0.4906, 0.3851, 0.3023];

#(b)

#(semilogx)

semilogx(x,y);

xlim([1, 2]);

set(gca, 'fontsize', 16);

xlabel('x');

ylabel('y');

title('semilogx(x, y)');

set(gca, 'fontsize', 8);

set(gca, 'xtick', 1:0.1:2);

**-PlotSemiLogY.m**

x = 1:0.2:2;

y = [1.0139, 0.7959, 0.6249, 0.4906, 0.3851, 0.3023];

semilogy(x,y);

set(gca, 'fontsize', 16);

xlabel('x');

ylabel('y');

title('semilogy(x, y)');

**-PlotLogLog.m**

x = 1:0.2:2;

y = [1.0139, 0.7959, 0.6249, 0.4906, 0.3851, 0.3023];

loglog(x,y);

xlim([1,2]);

set(gca, 'fontsize', 16);

xlabel('x');

ylabel('y');

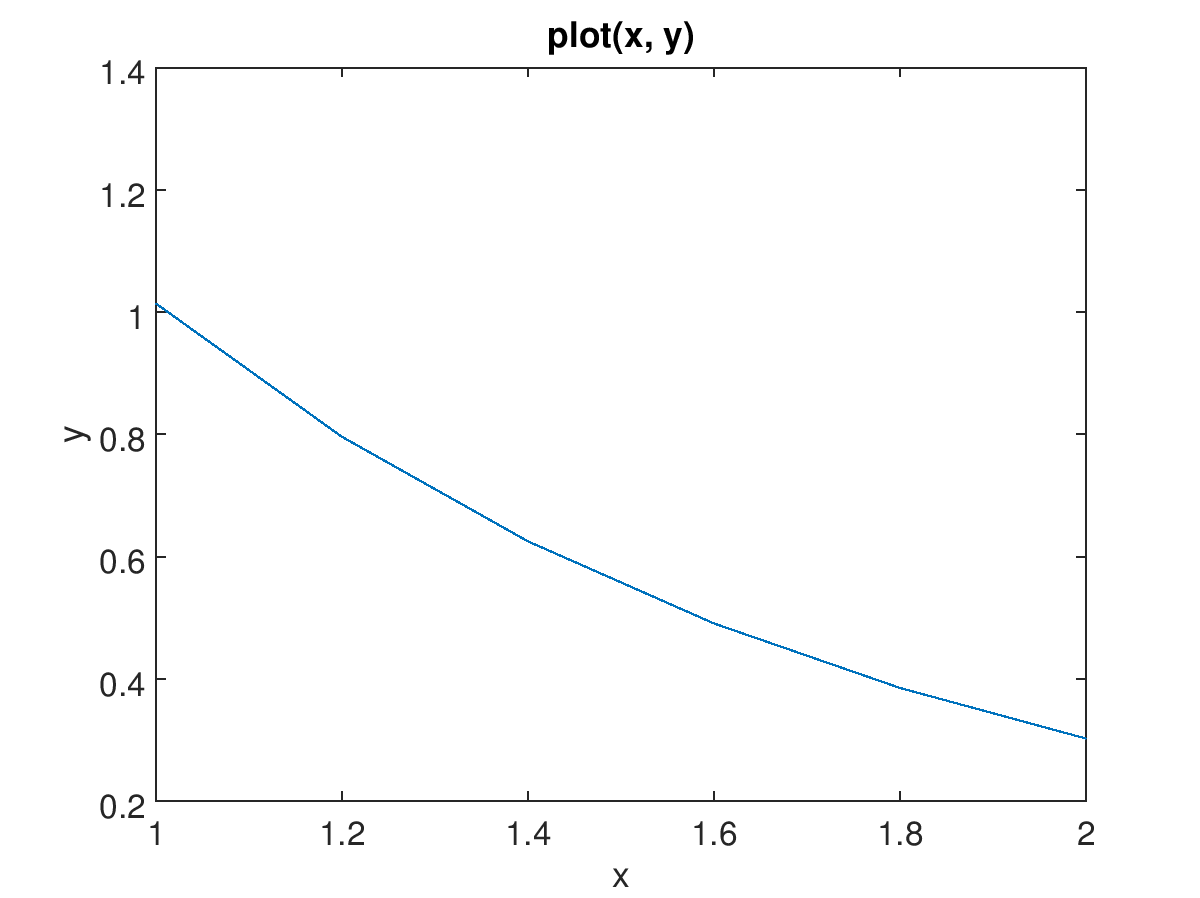
title('loglog(x, y)');

set(gca, 'fontsize', 8);

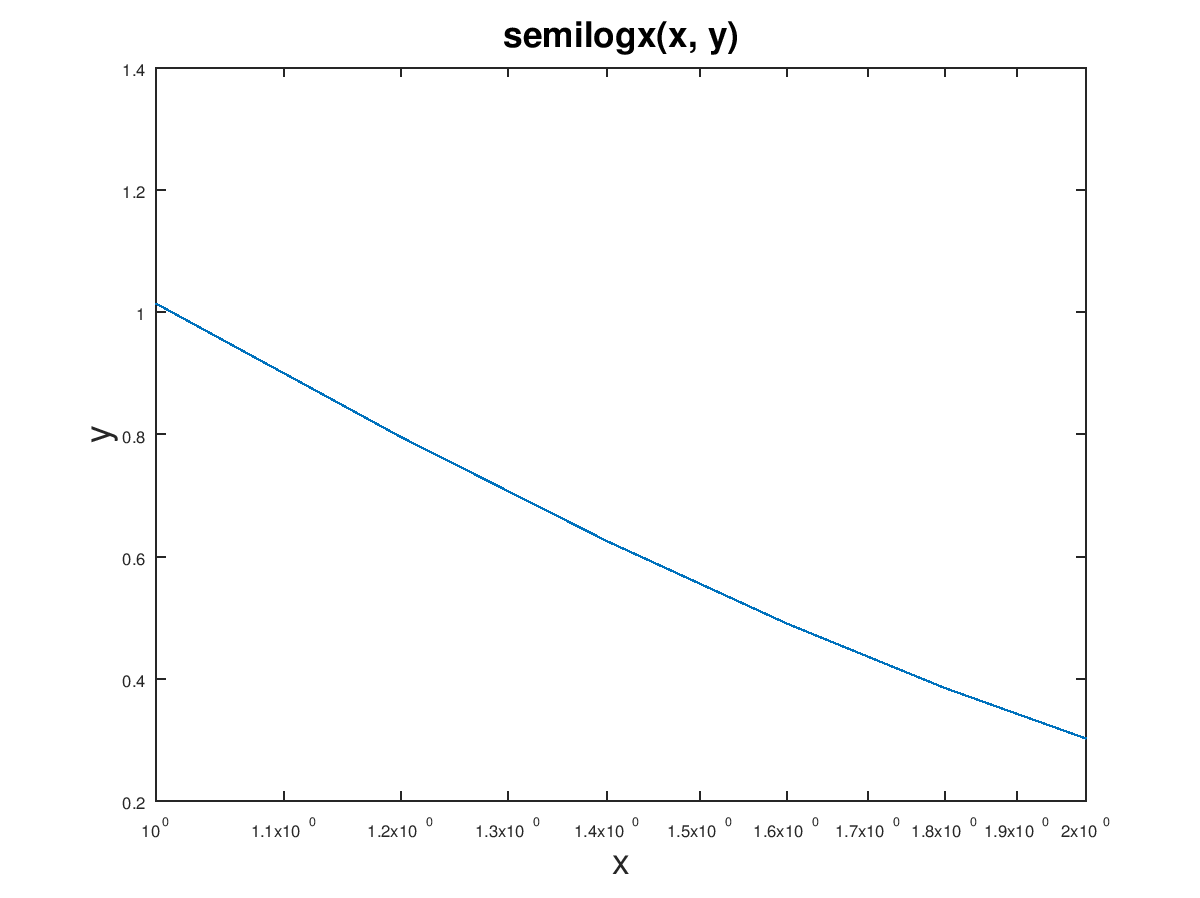
set(gca, 'xtick', 1:0.1:2);

**Output Section**

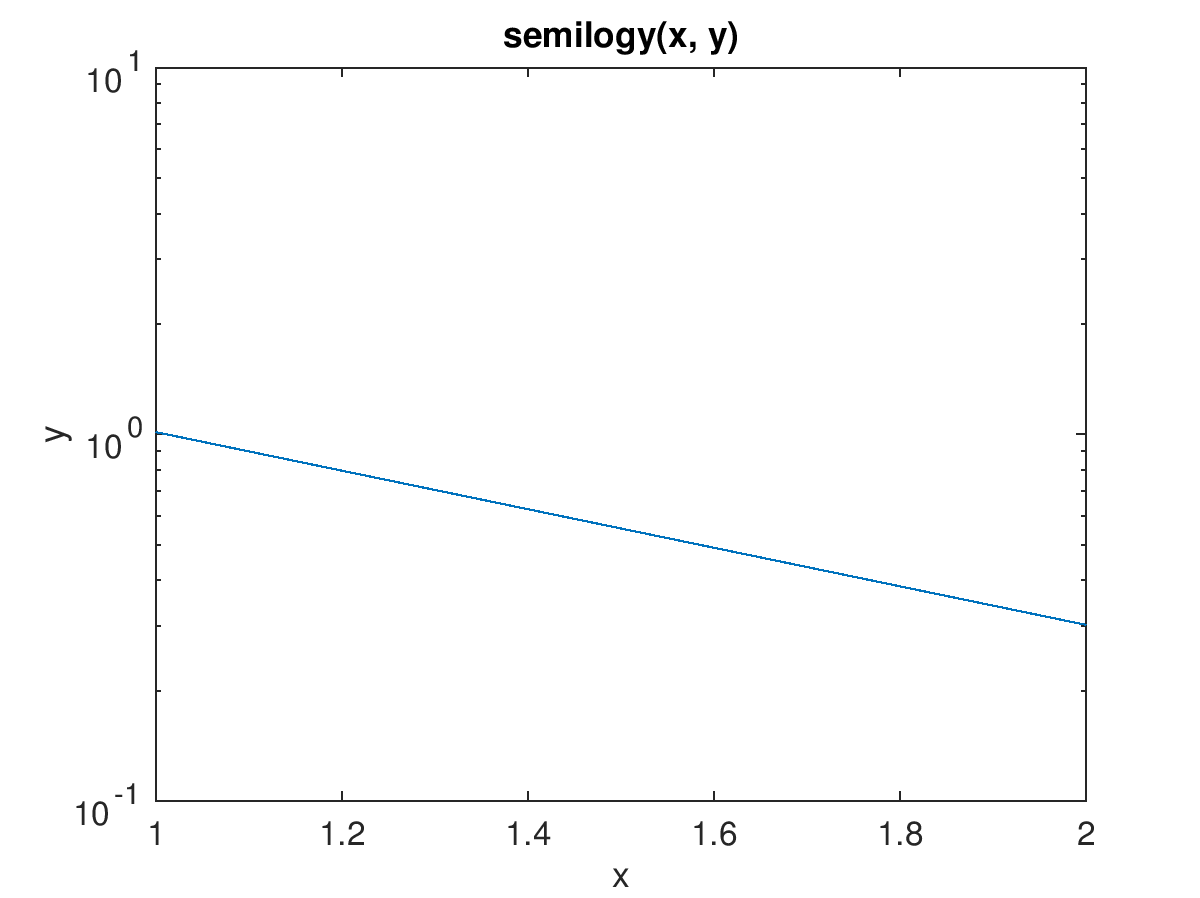
**-PlotA.m**



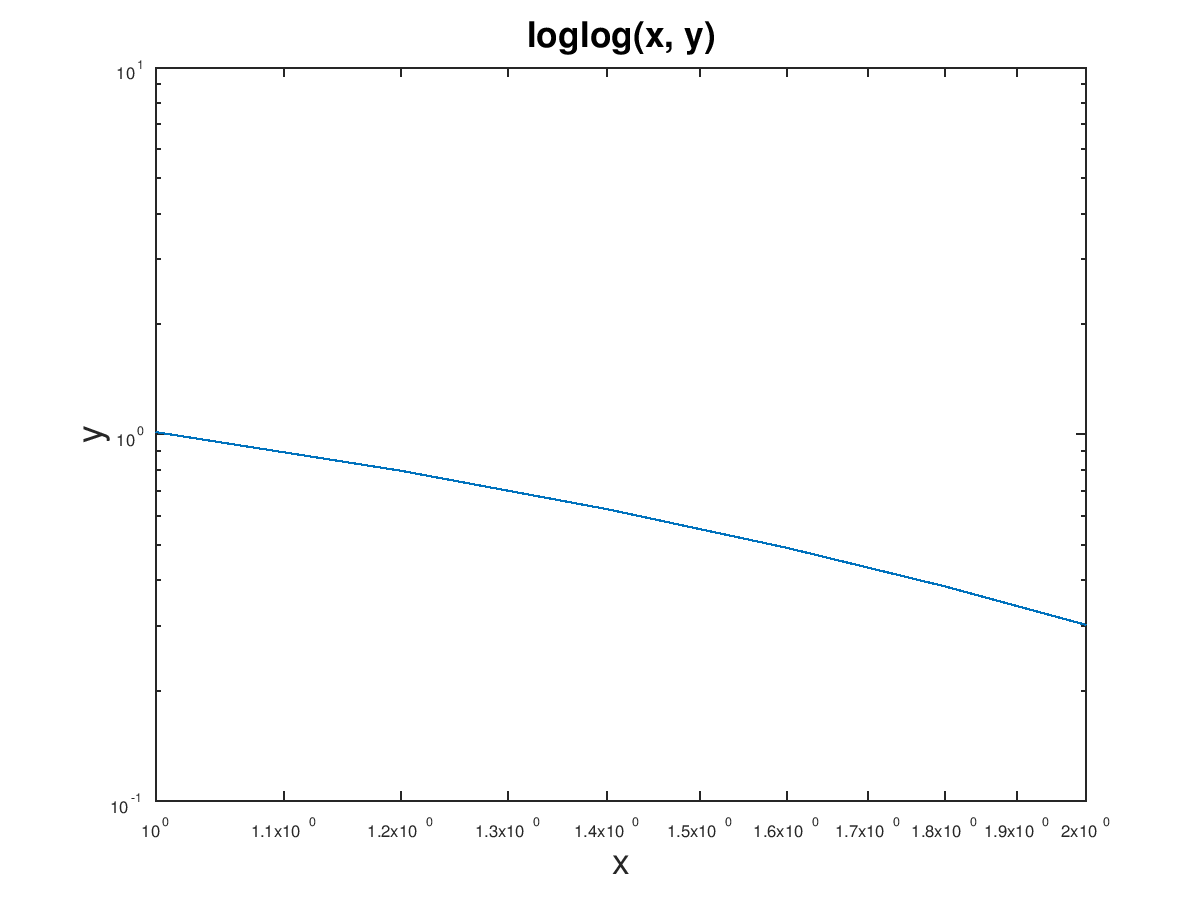
**-PlotSemiLogX.m**



**-PlotSemiLogY.m**



**-PlotLogLog.m**



**-ARD Section**

The main focus on these plots were the labeling and limiting the axis. For part c, I believe that we were suppose to use polyfit() but I never managed to figure out what to do as it doesn’t describe much. Originally, asking for help was the only option but I was unable to get enough time.

**Problem 6**

Call…

sine\_Taylor(Terms, X) for the First Term Numbers and the Sum

Error(Terms) for the Array of Absolute Error

Plot for the Plotting of the Error values that includes regular plot, semilogy plot, semilogx plot, loglog plot

**Files Section**

**-sine\_Taylor.m**

function [Output] = sine\_Taylor (n, x)

# positive integer n that are the first n terms

# number x

# outputs the first n terms

#Because we know that this is the MacLauren Expansion, we do not need to

# worry about the (x- x.)^k

#Initialize

Output = 0;

# Array of the output of sin(0). Can be indexed by x(#)

# # 1, 2, 3, 4

y = [0, 1, 0, -1];

# Array index which can be increased and reset for the next iteration

index = 1;

# i is the iteration variable

for i = 1:n

#Factorial f. prod(1:0) is 1

f = prod(1:i-1);

Output = Output + y(index)/f\*(x.^i);

disp(["Term ", num2str(i), ": ", num2str(Output)])

index = index + 1;

if(index > 4)

index = 1;

endif

endfor

endfunction

**-ST.m**

#Simplified version of sine\_Taylor without the output for every term except the last term

function [Output] = ST (n, x)

# positive integer n that are the first n terms

# number x

# outputs the first n terms

#Because we know that this is the MacLauren Expansion, we do not need to

# worry about the (x- x.)^k

#Initialize

Output = 0;

# Array of the output of sin(0). Can be indexed by x(#)

# # 1, 2, 3, 4

y = [0, 1, 0, -1];

# Array index which can be increased and reset for the next iteration

index = 1;

# i is the iteration variable

for i = 1:n

#Factorial f. prod(1:0) is 1

f = prod(1:i-1);

Output = Output + y(index)/f\*(x.^i);

index = index + 1;

if(index > 4)

index = 1;

endif

endfor

endfunction

**-Error.m**

function [absoluteError] = Error (Terms)

#Initialize Array

N = 1:10;

for i = 1:Terms

N(i) = abs(sin(1) - ST(i, 1));

endfor

absoluteError = N;

endfunction

**-Plot.m**

Array = Error(10);

#10 for the Terms

x = 1:1:10;

subplot(2,2,1);

plot(x, Array);

xlabel('x');

ylabel('y');

title('plot(x, y)');

subplot(2,2,2);

semilogx(x, Array);

xlabel('x');

ylabel('y');

title('semilogx(x, y)');

#The plot that will give the linear error decay is semilogy

#points

n = [5:8];

m = [Array(5),Array(6),Array(7),Array(8)];

subplot(2,2,3);

semilogy(x, Array);

xlabel('x');

ylabel('y');

title('semilogy(x, y)');

p = polyfit(n, m, 1);

m1 = polyval(p, n);

m1 = 10.^m1;

hold on;

plot(n, m1);

legend("semilogy(x,y)","Error", "location", "southwest");

subplot(2,2,4);

loglog(x, Array);

xlabel('x');

ylabel('y');

title('loglog(x, y)');

**Output Section**

**sine\_Taylor.m** **when Terms = 10, X = 1: sine\_Taylor(10,1)**

>> sine\_Taylor(10,1)

Term 1: 0

Term 2: 1

Term 3: 1

Term 4: 0.83333

Term 5: 0.83333

Term 6: 0.84167

Term 7: 0.84167

Term 8: 0.84147

Term 9: 0.84147

Term 10: 0.84147

ans = 0.84147

**Error.m when Terms = 10: Error(10)**

>> Error(10)

ans =

Columns 1 through 7:

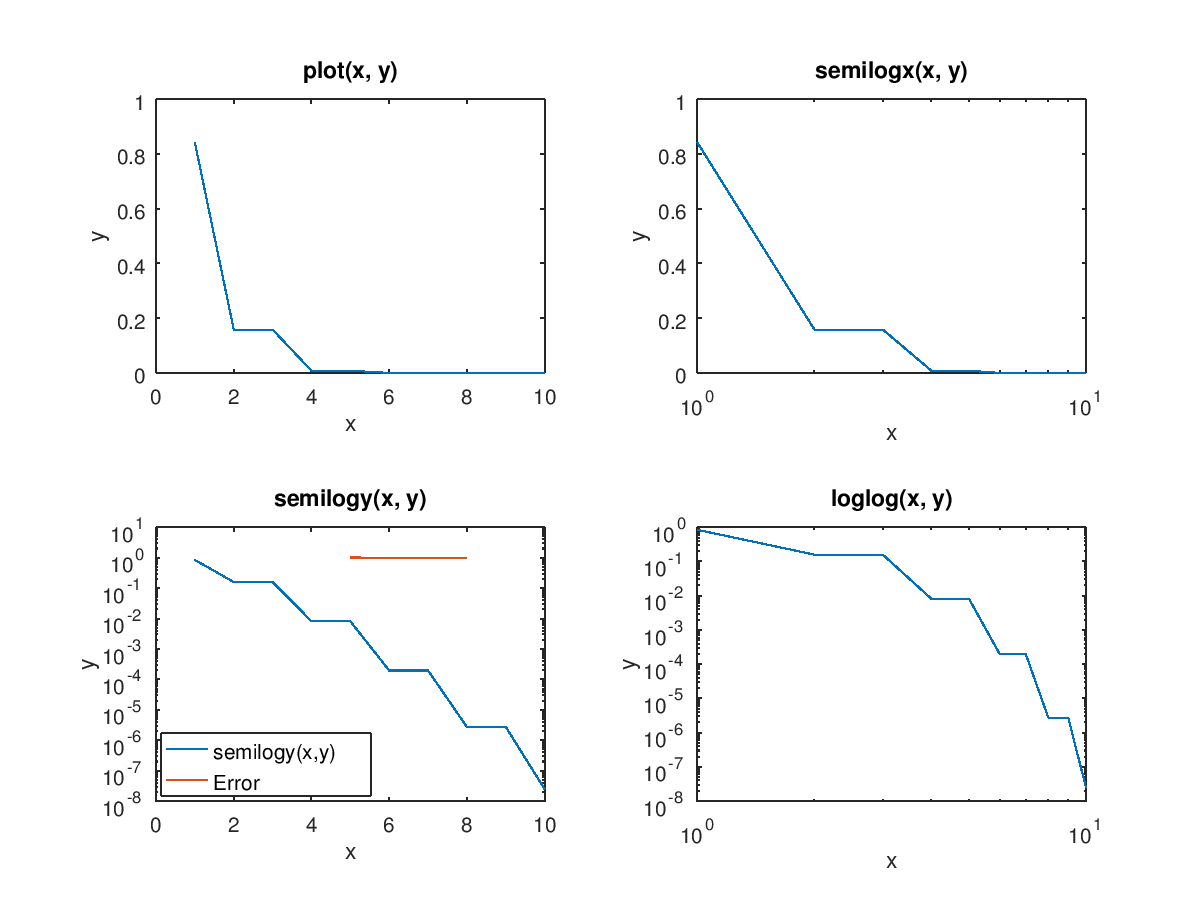
8.4147e-001 1.5853e-001 1.5853e-001 8.1377e-003 8.1377e-003 1.9568e-004

1.9568e-004

Columns 8 through 10:

2.7308e-006 2.7308e-006 2.4892e-008

**Plot.m**



**ARD Section**

(d)The plot that gives a linear error decay is semilogy(x,y). However, I was unable to produce the line that represents the linear error decay.

p= polyfit(n, m, 1); <-This is suppose to output the coefficient in x and the b in slope form.

m1 = polyval(p,n); <- This is suppose to output the values of y in a linear form, not semilogy form.

Thus, the results of y in m1 should be 10^(m1) giving the result in terms of semilogy(x,y) graph in linear form because…

Y = 10^(X \* SLOPE + B) is the converted values

However, the code that I built did not follow this logic as evidently by the Error linear line around 10^0.

(e)

(f) The points n = 9, 10 in my plots were of extremely low decimal number; They were below 10^-8 which means that the error is very minimal. The more iteration, the better approximation it became.

The mathematical methods involved in this was the Taylor Polynomial as well as the Error Measurement and graph. It is consistent in that the absolute error was eventually reduced. Not only the above theories, there is also the algorithms of the Taylor Expansion of sin(x). It required conversion of the the functions into the Octave and programming language. The mathematical theory is the Mean Value Theorem that resides in the error formula.