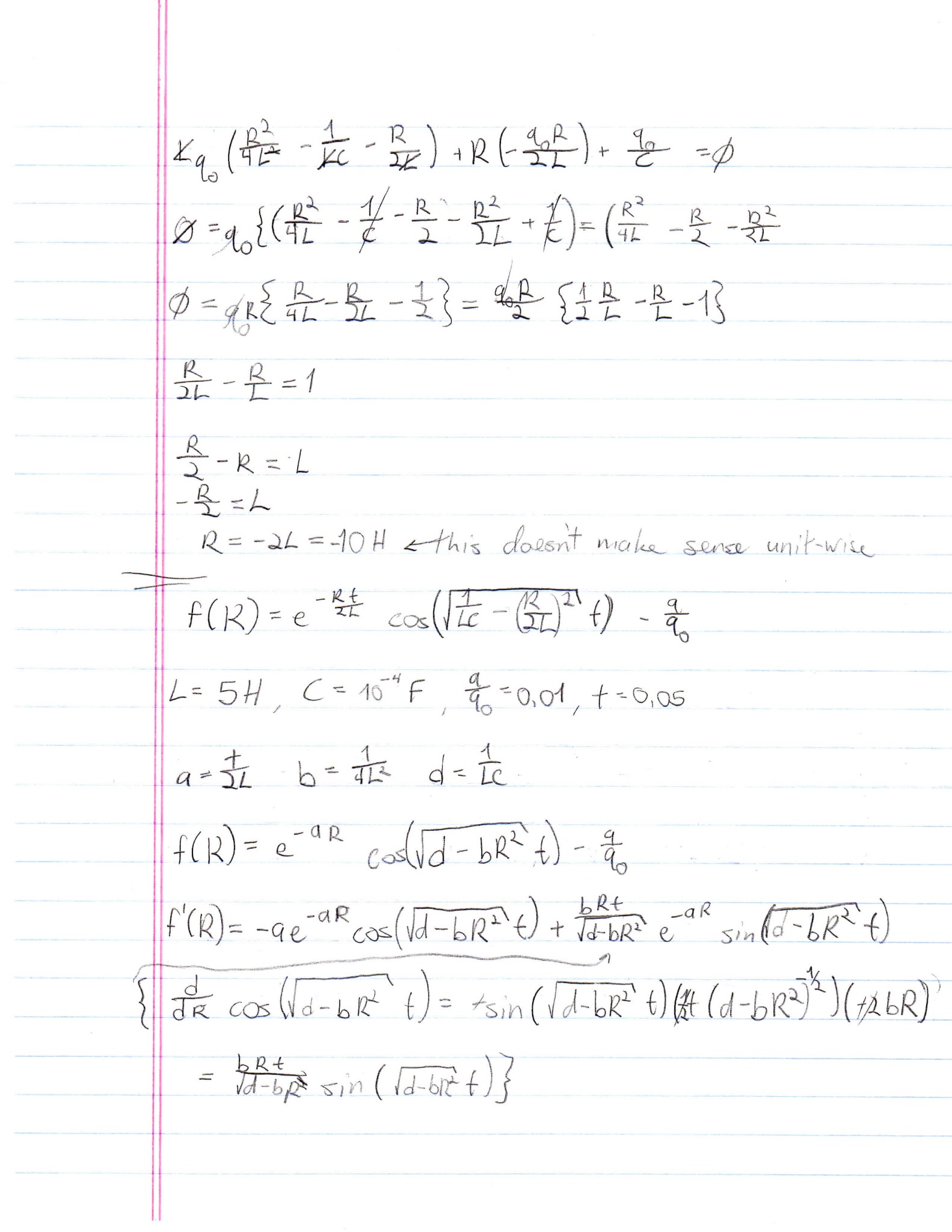
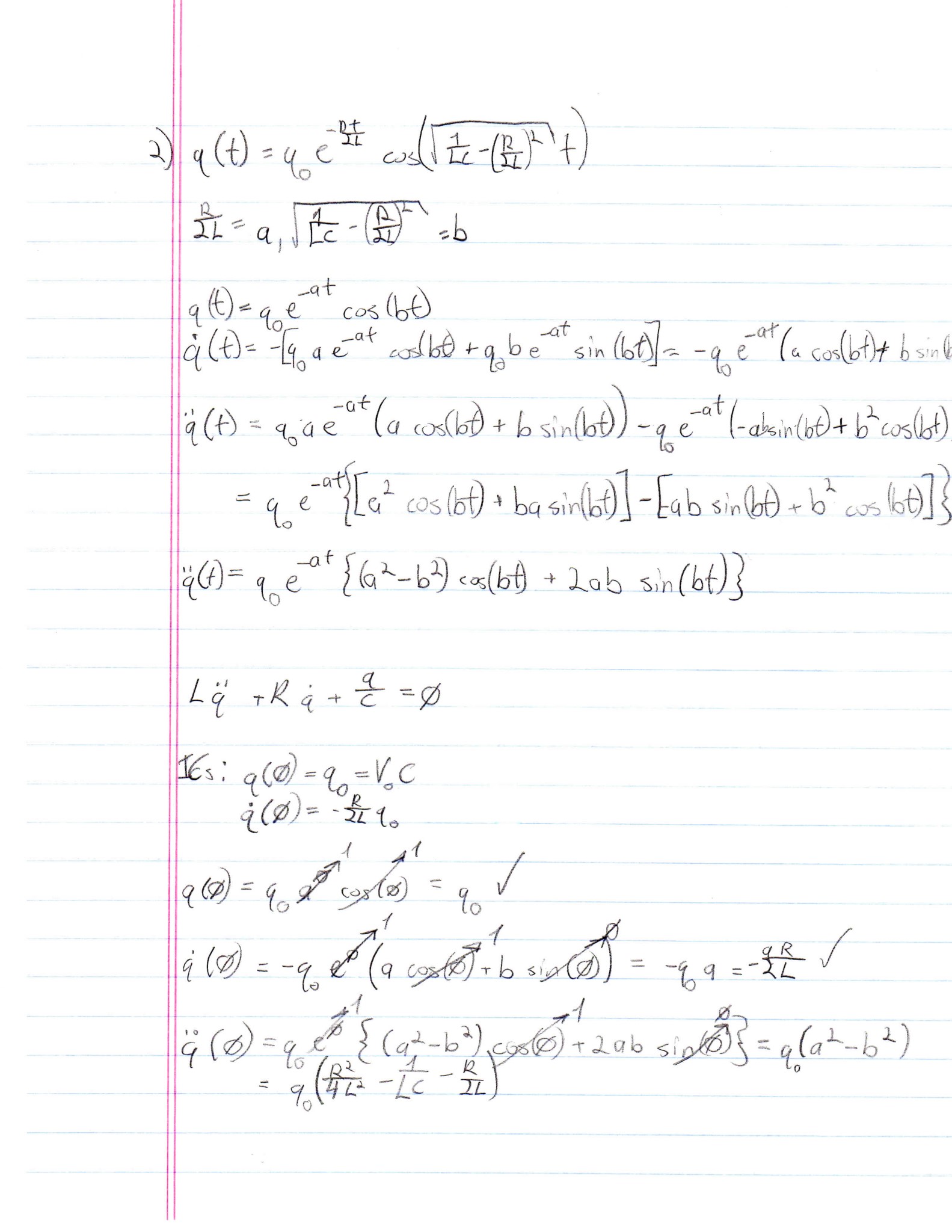
**PAPER WORK**



**Matlab Driver Trace**

------------------------PROBLEM #1------------------------

------------------------RUN N = 30------------------------

x\_a =

Columns 1 through 13

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 14 through 26

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0002 0.0009

Columns 27 through 31

0.0035 0.0145 0.0595 0.2439 1.0000

x\_b =

Columns 1 through 13

0.0395 0.0116 0.0034 0.0010 0.0003 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 14 through 26

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0002 0.0009

Columns 27 through 31

0.0035 0.0145 0.0595 0.2439 1.0000

----------------------RUN N = 37------------------------

x\_a =

Columns 1 through 13

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 14 through 26

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 27 through 38

0.0000 0.0000 0.0000 0.0000 0.0001 0.0002 0.0009 0.0035 0.0145 0.0595 0.2439 1.0000

x\_b =

Columns 1 through 13

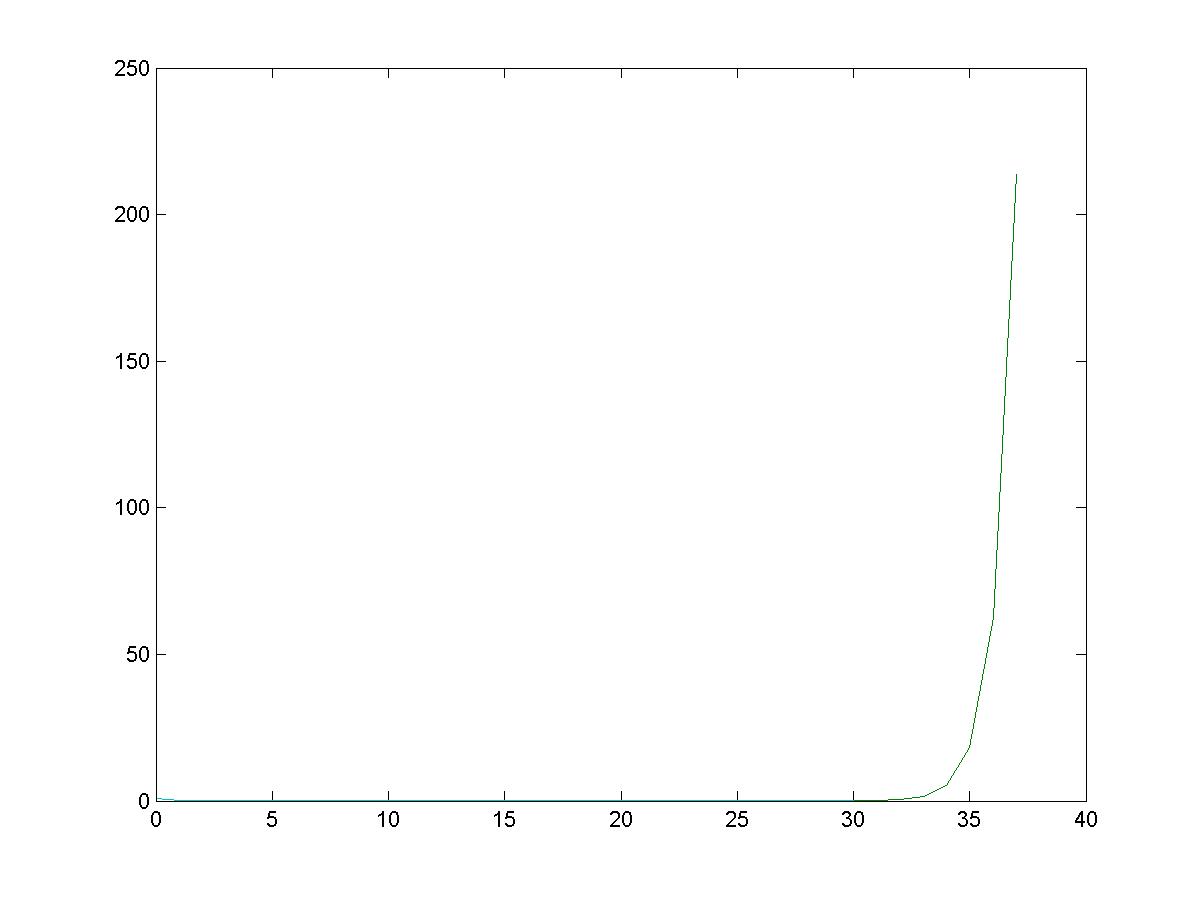
213.8447 62.6260 18.3405 5.3711 1.5730 0.4607 0.1349 0.0395 0.0116 0.0034 0.0010 0.0003 0.0001

Columns 14 through 26

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 27 through 38

0.0000 0.0000 0.0000 0.0000 0.0001 0.0002 0.0009 0.0035 0.0145 0.0595 0.2439 1.0000

**PROBLEM 1 GRAPH**

The cause of the deviation of the iterative method in problem 1 is due to the loss of accuracy from roundoff error.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Problem 1 is printed on the screen above. And graphs have been

made as jpgs in the folder this was run from for 30 and 37

iterations. The output for problem 2 has been put into a file

output.txt in the folder this was run as well.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OUTPUT.TXT**

------------------------PROBLEM #2------------------------

---------------------BISECTION METHOD---------------------

Iteration: 53

Error: 0

Root: 328.1514

Function Value at Root: -8.6736e-18

Bisection Method Time: 0.019184

----------------------NEWTONS METHOD----------------------

Iteration: 5

Error: 0

Root: 328.1514

Function Value at Root: -8.6736e-18

Newton Method Time: 0.0068939

------------------------SEC METHOD------------------------

Iteration: 6

Error: 8.6736e-18

Root: 328.1514

Function Value at Root: -8.6736e-18

Secant Method Time: 0.0068939

------------------------ F-ZERO ------------------------

F-Zero Method Time: 0.0068939

**Matlab Functions**

**PROBLEM 1 FUNCTION**

function [x\_n, x\_n\_1, x\_n\_2, x\_n\_a, x\_n\_b] = xn(iterations, filename)

if(nargin < 2)

filename = 'xn';

end

x\_n = 1;

x\_n\_1 = 1/4.1;

a = 15/4.1;

b = 14/16.81;

x\_a = [];

x\_b = [];

iter = [];

if(iterations >= 0)

while(iterations >= 0)

if(iterations >= 2)

while(iterations >=2)

x\_n\_2 = a\*x\_n\_1 - b\*x\_n;

x\_n = x\_n\_1;

x\_n\_1 = x\_n\_2;

x\_n\_b = x\_n\_2;

x\_n\_a = 1/power(4.1, iterations);

iter(end+1) = iterations;

x\_a(end+1) = x\_n\_a;

x\_b = [x\_n\_b, x\_b];

iterations = iterations-1;

end

elseif(iterations == 1)

x\_n\_b = 1/4.1;

x\_n\_a = 1/power(4.1, iterations);

iter(end+1) = iterations;

x\_a(end+1) = x\_n\_a;

x\_b(end+1) = x\_n\_b;

iterations = iterations - 1;

elseif(iterations == 0)

x\_n\_b = 1;

x\_n\_a = 1/power(4.1, iterations);

iter(end+1) = iterations;

x\_a(end+1) = x\_n\_a;

x\_b(end+1) = x\_n\_b;

iterations = iterations - 1;

end

end

x\_a

x\_b

xn\_plot = plot(iter, x\_a);

hold all;

xn\_plot = plot(iter, x\_b);

file = strcat(filename, '.jpg');

saveas(xn\_plot, file);

elseif (iterations < 0)

error('N must not be negative');

end

**BISECTION METHOD**

function[iter, root, root\_value, err] = bisection(a, b, fun, max\_iter, max\_error, outputfile)

%Initalize values for tests

iter = 0;

err = 999;

root = 999;

output = fopen(outputfile, 'a');

%Test to see if a nd b have the same values or if a > b

% If a > b, swap a and b

if( a == b )

fprintf('Please specify valid endpoints.');

error('Endpoints are equal. Cannot evaluate.');

elseif( a > b )

c = b;

a = b;

b = c;

clear c;

end

%Test to see if interval is valid.

fa = fun(a);

fb = fun(b);

test = fa \* fb;

if ( test >= 0 )

fprintf('Given interval is not suitable for calculating the root.');

error('Value of fa \* fb is greater than or equal to zero.');

end

%Run first iteration of the bisection method manually for while loop

%to work

c\_old = (a+b)/2;

fc = fun(c\_old);

if(fc < fb)

b = c\_old;

elseif (fc > fa)

a = c\_old;

end

%Iteration of bisection method

while(iter < max\_iter && err > max\_error && fc ~= 0)

iter = iter + 1;

c\_curr = (a+b)/2;

err = abs((c\_curr - c\_old)/(c\_curr));

fa = fun(a);

fb = fun(b);

fc = fun(c\_curr);

%Test to determine which interval value should move towards the root

if(fc > 0)

b = c\_curr;

elseif (fc < 0)

a = c\_curr;

end

c\_old = c\_curr;

end

%Set values for output

root = c\_old;

root\_value = fc;

%Print all relevant information

fprintf(output, 'Iteration: %s\n', num2str(iter));

fprintf(output, 'Error: %s\n', num2str(err));

fprintf(output, 'Root: %s\n', num2str(root));

fprintf(output, 'Function Value at Root: %s\n', num2str(root\_value));

fclose('all');

**NEWTON-RAPHSON METHOD**

function[iter, root, root\_value, err] = nr\_method(x\_n, fun, fprime, max\_iter, max\_error, outputfile)

%Initalize values for tests

iter = 0;

err = 999;

fxn = fun(x\_n);

output = fopen(outputfile, 'a');

%Check to see if f' is too close to zero

if(fprime(x\_n) < 10^-12)

error('F-Prime is too close to zero!');

end

%Check to see if given starting point is the root.

if(fxn == 0)

error('Given x\_0 is the root!');

end

%Newton-Raphson Method Iteration

while(iter < max\_iter && err > max\_error && fxn ~= 0)

x\_n\_1 = x\_n - (fun(x\_n)/fprime(x\_n));

err = abs((x\_n\_1 - x\_n)/(x\_n));

x\_n = x\_n\_1;

iter = iter+1;

fxn = fun(x\_n);

end

%Set root and root function value.

root = x\_n;

root\_value = fxn;

%Print all relevant information

fprintf(output, 'Iteration: %s\n', num2str(iter));

fprintf(output, 'Error: %s\n', num2str(err));

fprintf(output, 'Root: %s\n', num2str(root));

fprintf(output, 'Function Value at Root: %s\n', num2str(root\_value));

fclose('all');

**SECANT METHOD**

function[iter, root, root\_value, err] = secant\_method(x\_nm\_1, fun, max\_iter, max\_error, outputfile)

%Initalize values for tests

iter = 0;

err = 999;

fxn = fun(x\_nm\_1);

x\_n = x\_nm\_1 + 10^-8;

output = fopen(outputfile, 'a');

%Check to see if given starting point is the root.

if(fxn == 0)

error('Given x\_0 is the root!');

end

%Secant Method Iteration

while(iter < max\_iter && err > max\_error && fxn ~= 0)

%Using x\_n+1 = x\_n - f(x\_n)/Q(x\_n-1, x\_n),

%where Q = [f(x\_n-1) - f(x\_n)]/[x\_n-1 - x\_n]:

%diff 1 and 2 are the numerator and denominator of Q, respectively,

%and denominator is the value of Q.

diff1 = fun(x\_nm\_1) - fun(x\_n);

diff2 = (x\_nm\_1 - x\_n);

denominator = diff1/diff2;

x\_np\_1 = x\_n - (fun(x\_n) / denominator);

fxn = fun(x\_np\_1);

err = abs(fxn);

x\_nm\_1 = x\_n;

x\_n = x\_np\_1;

iter = iter + 1;

end

%Set root and root function value.

root = x\_n;

root\_value = fun(x\_n);

%Print all relevant information

fprintf(output, 'Iteration: %s\n', num2str(iter));

fprintf(output, 'Error: %s\n', num2str(err));

fprintf(output, 'Root: %s\n', num2str(root));

fprintf(output, 'Function Value at Root: %s\n', num2str(root\_value));

fclose('all');

**FUNCTION**

function [y] = fun(R)

L = 5;

C = 10^-4;

t = 0.05;

qq0 = 0.01;

sq = sqrt([1/(L\*C)] - [R/(2\*L)]^2);

y = exp(-[R\*t]/[2\*L]) \* cos(sq \* t) - qq0;

**F-PRIME**

function [y] = fprime(R)

L = 5;

C = 10^-4;

t = 0.05;

sq = sqrt([1/(L\*C)] - [R/(2\*L)]^2);

e = exp(-[(t\*R)/(2\*L)]);

y = -[t/(2\*L)] \* e \* cos(sq \* t) + [(R\*t)/(4\*(L^2)\*sq)] \* e \* sin(sq\*t);