Razi Murshed
260516333

ECSE 543: ASSIGNMENT 3

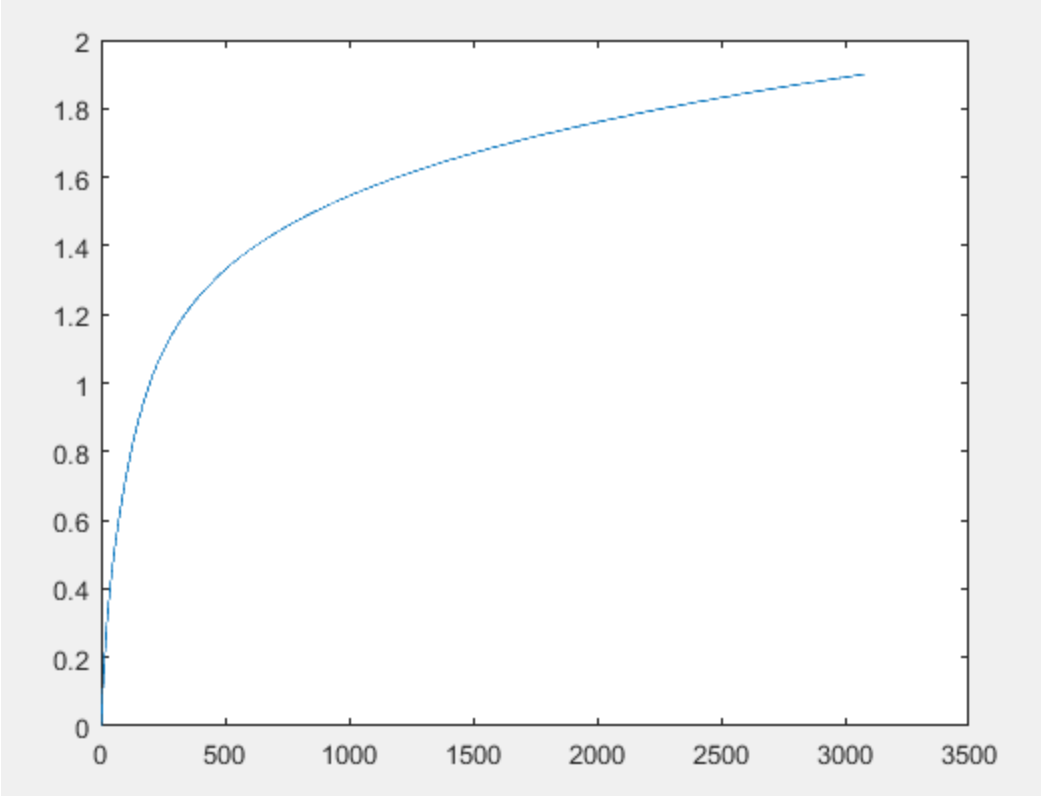
RAZI MURSHED

260516333

QUESTION 1

PART A

The code for the interpolation of this polynomial was done by the function “Lagrange.m”. A plot of the 6 points passed into the function Lagrange is shown below –

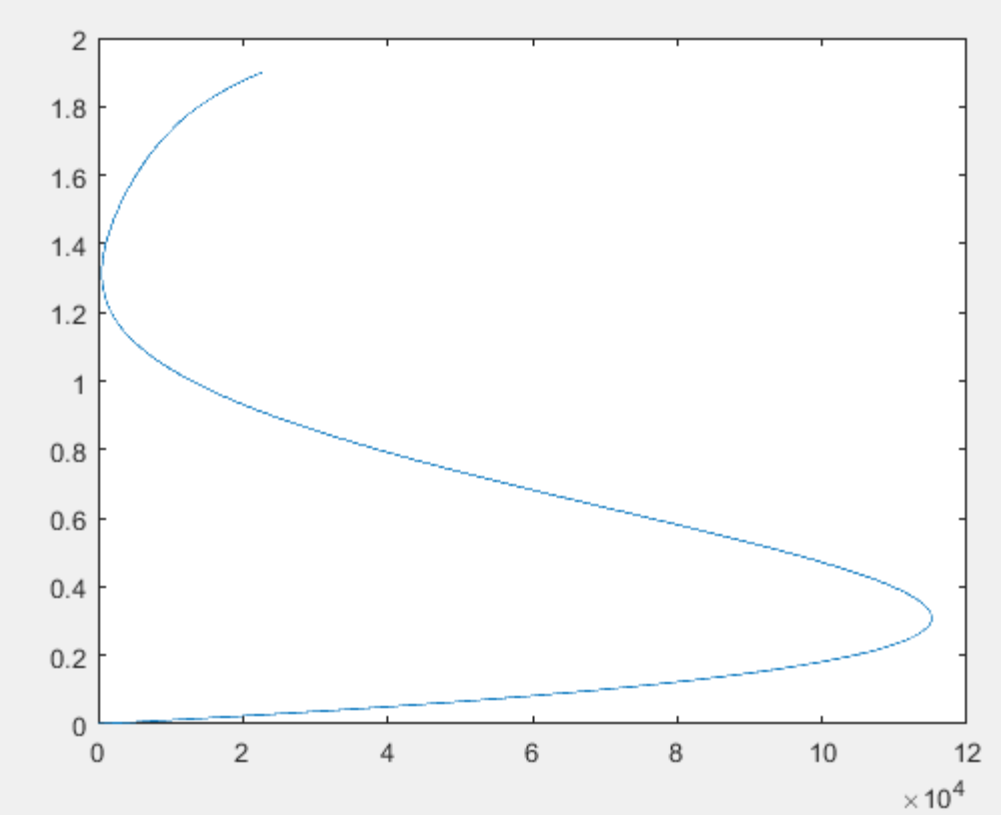


As the curve seems pretty smooth it seems like the result is plausible and is a good representation.

PART B

When the function runs over the points `B = [0, 1.3, 1.4, 1.7, 1.8, 1.9]` we see the following plot –

Razi Murshed
260516333



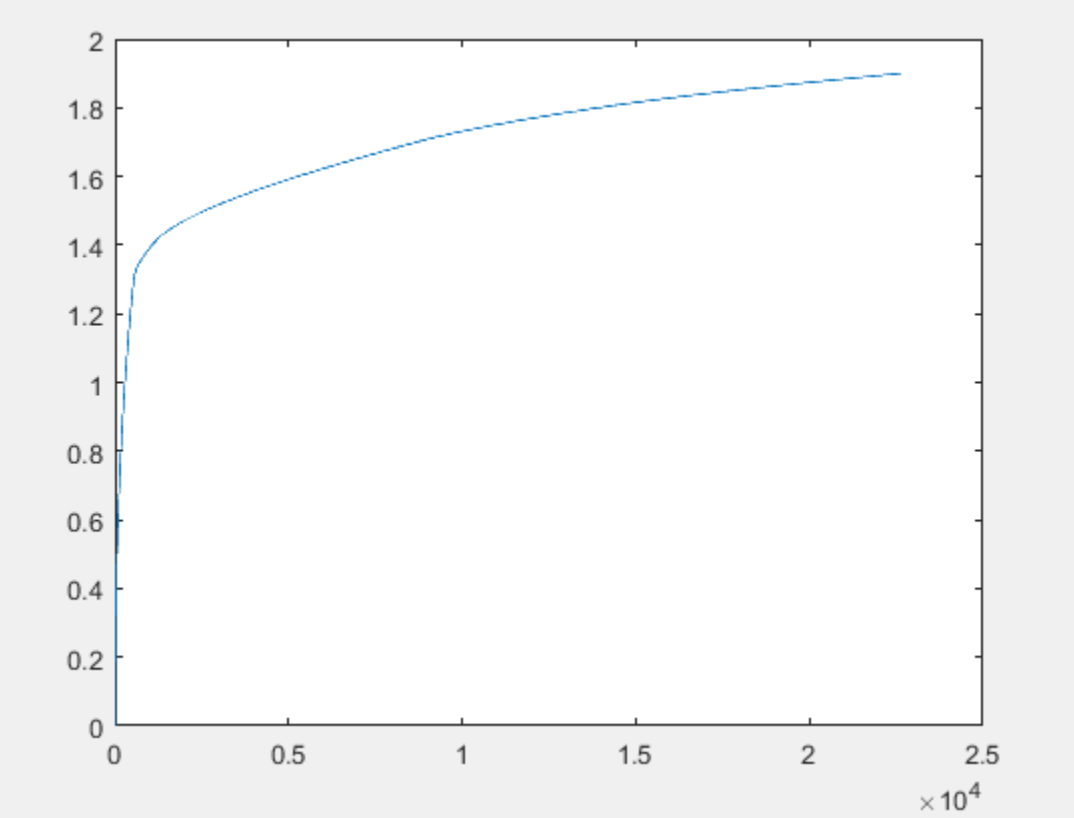
It can be seen that this graph does not look anything like a H versus B graph and therefore not a plausible representation.

PART C

If we happen to know both function values and first derivative values at a set of data points, then piecewise cubic Hermite interpolation can reproduce those data. But since we are not given the derivative values, we need to define the slopes somehow. However, we would have to do this for two types of points, the edge points such as the start and end points and also for the internal points. For the internal points we calculate this by finding the differences in the x-axis and creating a delta by dividing the differences in the x points with the difference in the y-points. We can then use this delta along with the differences in the x-axis to form the slopes. For the end points we can find the slopes by interpolating using the deltas between the first and second points and the second last and last points using their deltas. The codes for cubic hermite interpolation can be found in 'cubicHermiteInterpolation.m' and 'calculateSlopeInternal.m'.

For B `= [0, 1.3, 1.4, 1.7, 1.8, 1.9]` we get the following –

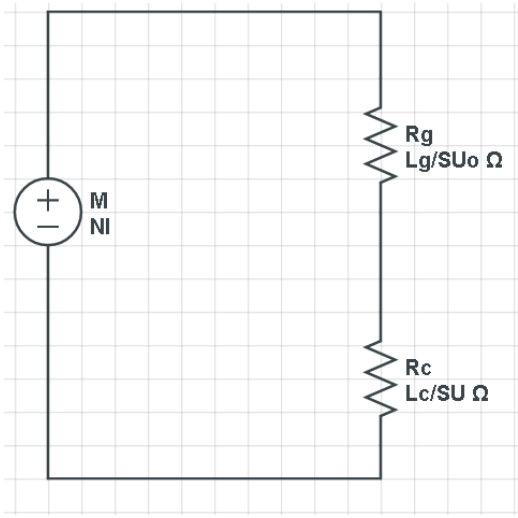
Razi Murshed
260516333



We can see that it produces a better curve for points that are widely separated which the Lagrange fails to do.

PART D

For this part we come up with the following circuit -



Substituting given values we find the equation –

$$(3.98 \times 10^7)\varphi + 0.3\varphi H(\varphi) - 8000 = 0$$

Where $H(\varphi)$ is found by doing a linear piecewise interpolation of the values of B and H given to us. The function to do this interpolation is written in "piecelin.m".

PART E

The two functions that does the solving are "netwtonRhapson.m" and "newtonRhapsonDer.m". These are tested in "MagneticCircuit.m". The equation mentioned above was then used to solve the equations using the Newton-Raphson method. We found x to be $1.0701 \times 10^{-5} \frac{\text{Wb}}{m^2}$ after 3 iterations.

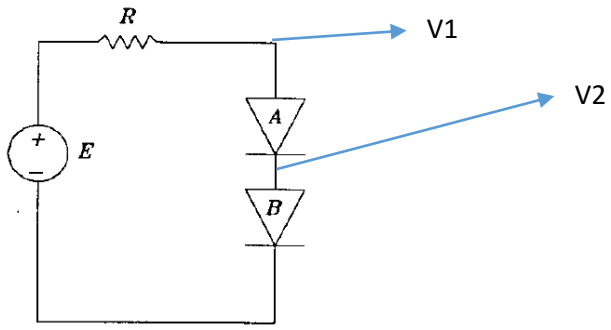
PART F

This part was written in "successiveSubstitution.m". We solved the same equation using successive substitution and found the result to be $1.0701 \times 10^{-5} \frac{\text{Wb}}{m^2}$ as well. With our function it converged without having to make further adjustments.

QUESTION 2

PART A

From the following circuit we determine voltages at the two nodes by following the conventions mentioned in the diagram –



We find the two equations of F to be –

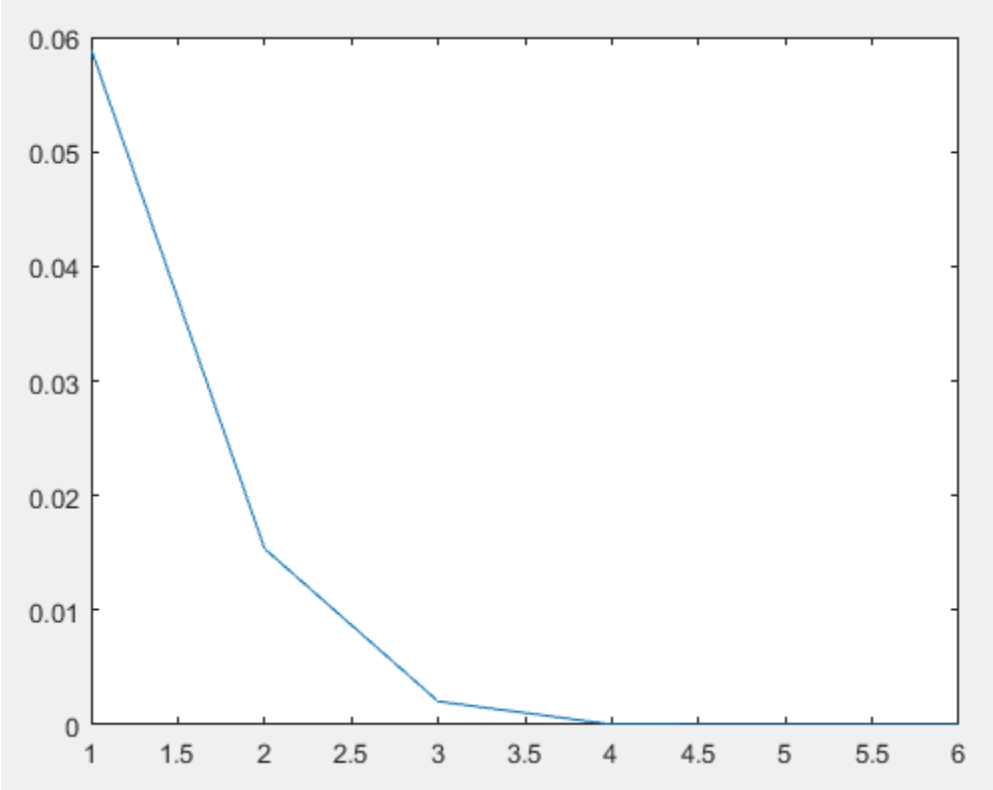
$$F_1 = v_1 - E + RI_{sA}\left(e^{\frac{v_1 - v_2}{25m}} - 1\right) = 0$$

$$F_2 = 0 = I_{sA}\left(e^{\frac{v_1 - v_2}{25m}} - 1\right) - I_{sB}\left(e^{\frac{v_2}{25m}} - 1\right)$$

PART B

We modelled the circuit in "ElectricCircuit.m" and the solver was written in the file "newRapCircuit.m". It took us 5 iterations to get to our selected error margin $\varepsilon_k = 10^{-6}$. The final voltages were found to be V = [0.183430837132538, 0.0832765631282870]. The following shows a plot of number of iterations against difference and we can see that the convergence is actually quadratic.

Razi Murshed
260516333



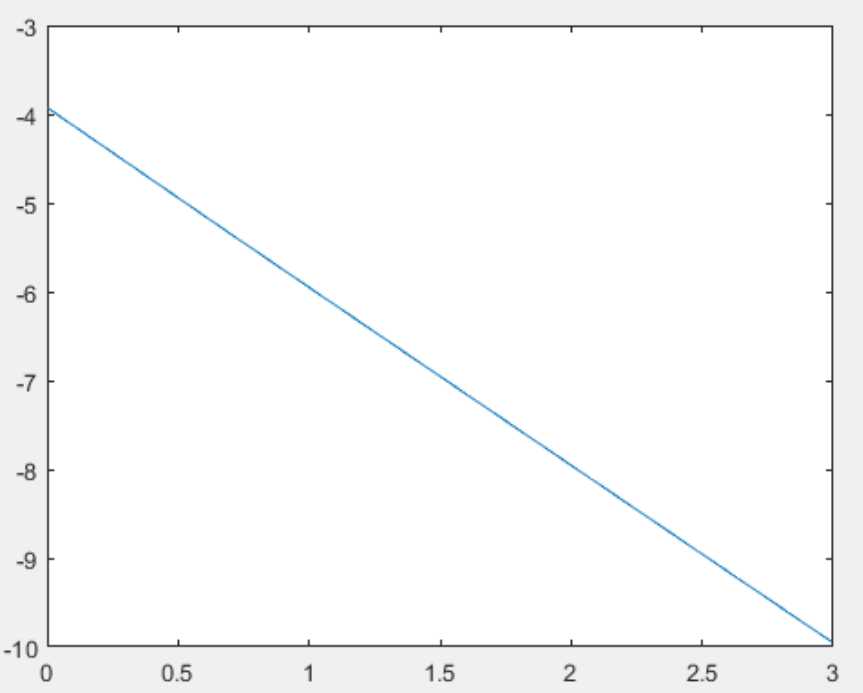
QUESTION 3

The test code for this question is written in “integrationTest.m”. The function to carry out integration was written in “integration.m” (modified for different inputs).

PART A

This function basically computes a running sum of the values of the function at the given number of sections and returns these values at midpoints multiplied by the length of the section. The error versus the number of segments is plotted below –

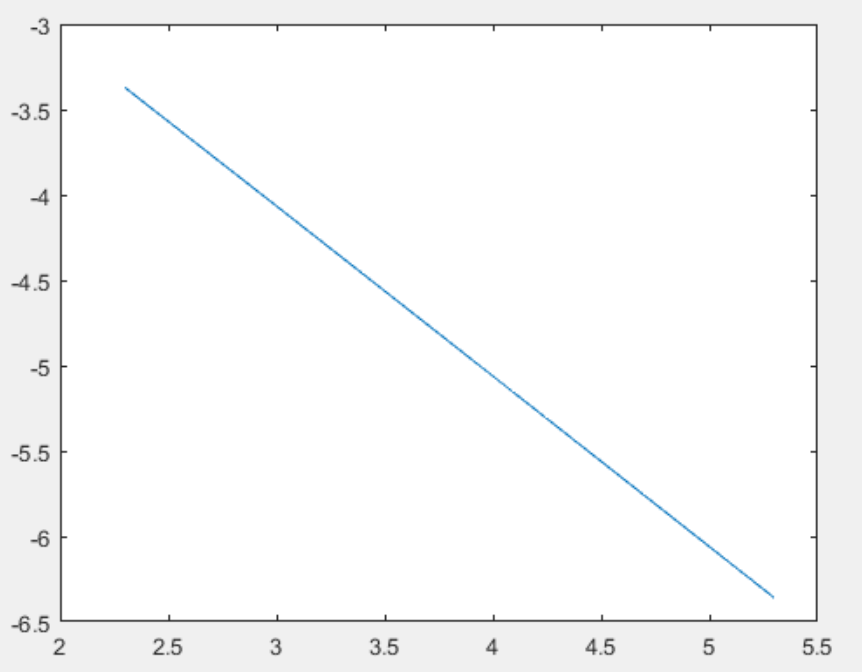
Razi Murshed
260516333



We can see that the $\log_{10}$Error versus $\log_{10}$N produces a linear graph. This means that increasing the number of segments does not reduce the absolute error of this computation.

PART B

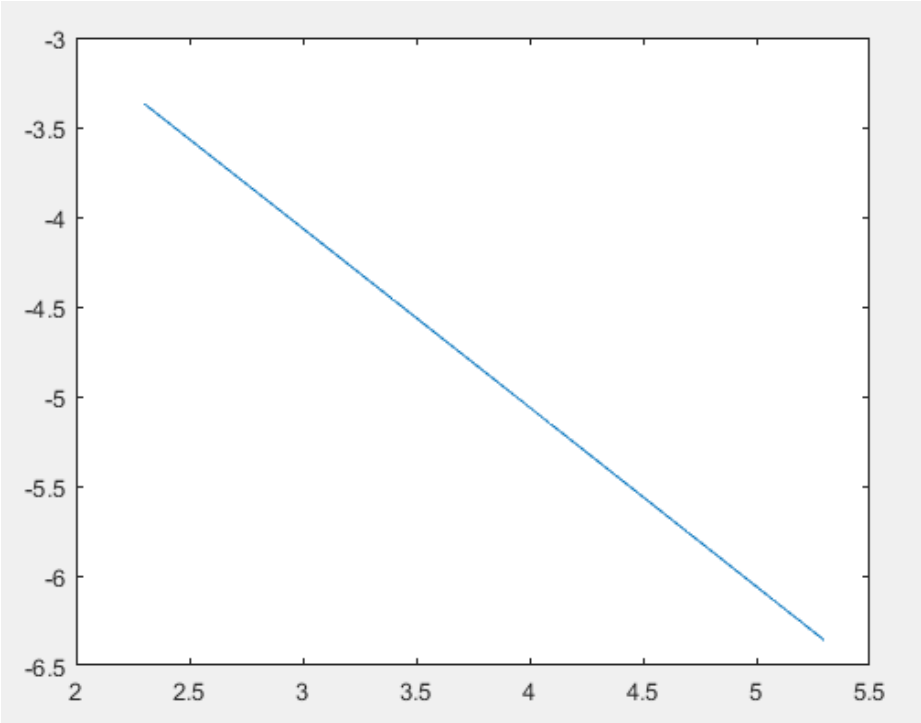
When repeated for $f(x) = ln(x)$, we see the same linear pattern between $\log_{10}$Error versus $\log_{10}$N as expected–



Razi Murshed
260516333

PART C

We carry out the same steps as part B and end up with the same plot in the case of $f(x) = \ln(0.2|\sin(x)|)$.



We can see even though the equation is much more complicated than before the $\log_{10}$Error versus $\log_{10}$N remains the same as before.

PART D

We tested by creating uneven segments and tried to do integration on $f(x) = ln(x)$. The segment width was gradually increased since the function is trickier to integrate near values close to zero. We found out that with 10 uneven segments we get an error of 0.0197 as opposed to 0.0342 which shows the uneven segments to clearly provide superior results. However, for $f(x) = \ln(0.2|sin(x)|)$ we see that the error for uneven 10 segments is 0.0764 whereas for even segments its 0.0344. One could argue that uneven segments tend to work better for simpler functions but the performance of using even intervals remains relatively the same always whereas for uneven segments we get a higher error margin with a more complicated function.

APPENDIX

POLYNOMIAL.M

```
X = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0];
Y = [0.0, 14.7, 36.5, 71.7, 121.4, 197.4];
xx = 0:.0001:1.9;
%P1 = Lagrange(X,Y);
```

```
figure;
plot(Lagrange(X,Y,xx),xx);

P2 = cubicHermiteInterpolation(X,Y,xx);
figure;
plot(cubicHermiteInterpolation(X,Y,xx),xx);

X= [0, 1.3, 1.4, 1.7, 1.8, 1.9];
Y = [0.0, 540.6, 1062.8, 8687.4, 13924.3, 22650.2];
%P3 = Lagrange(X,Y);
figure;
plot(Lagrange(X,Y,xx),xx);

P4 = cubicHermiteInterpolation(X,Y,xx);
figure;
plot(cubicHermiteInterpolation(X,Y,xx),xx);
```

LAGRANGE.M

```
function v = Lagrange(x,y,u)
n = length(x);
v = zeros(size(u));
for k = 1:n
w = ones(size(u));
for j = [1:k-1 k+1:n]
w = (u-x(j))./(x(k)-x(j)).*w;
end
v = v + w*y(k);
end
```

CUBICHERMITEINTERPOLATION.M

```
function v = cubicHermiteInterpolation(x,y,u)
h = diff(x);
delta = diff(y)./h;
d = calculateSlopeInternal(h,delta);
% Piecewise polynomial coefficients
n = length(x);
c = (3*delta - 2*d(1:n-1) - d(2:n))./h;
b = (d(1:n-1) - 2*delta + d(2:n))./h.^2;
% Find subinterval indices k so that x(k) <= u < x(k+1)
k = ones(size(u));
for j = 2:n-1
k(x(j) <= u) = j;
end
% Evaluate interpolant
s = u - x(k);
v = y(k) + s.*(d(k) + s.*(c(k) + s.*b(k)));
```

CALCULATESLOPEINTERNAL.M

```
function d = calculateSlopeInternal(h,delta)
n = length(h)+1;
d = zeros(size(h));
k = find(sign(delta(1:n-2)).*sign(delta(2:n-1))>0)+1;
```

```
w1 = 2*h(k)+h(k-1);
w2 = h(k)+2*h(k-1);
d(k) = (w1+w2)./(w1./delta(k-1) + w2./delta(k));
% Slopes at endpoints
d(1) = calculateSlopeEnd(h(1),h(2),delta(1),delta(2));
d(n) = calculateSlopeEnd(h(n-1),h(n-2),delta(n-1),delta(n-2));

function d = calculateSlopeEnd(h1,h2,del1,del2)
d = ((2*h1+h2)*del1 - h1*del2)/(h1+h2);
if sign(d) ~= sign(del1)
d = 0;
elseif (sign(del1)~=sign(del2))&&(abs(d)>abs(3*del1))
d = 3*del1;
end
```

MAGNETICCIRCUIT.M

```
SUo = (1*10^(-2))*(1*10^(-2))*4*pi*(10^-7);
Nturn = 800;
I = 10;
M = Nturn*I;
La = 0.5*(10^(-2));
Lc = 30*(10^(-2));
Rg = La/SUo;
tolerance = 0.000001;
iterations = 0;
x = 0;
xlist = [];
while (abs(newtonRhapson(x,Rg)/newtonRhapson(0,Rg)) > tolerance)
    iterations = iterations + 1;
    x = x - newtonRhapson(x, Rg)/newtonRhapsonDer(x,Rg);
    xlist = [xlist, x];
end
xlist = xlist';
ilist = 1 : iterations;
ilist = ilist';
%plot(ilist, xlist);
xSub = successiveSubstitution(1e-6, 1e-6);
```

NEWTONRHAPSON.M

```
function f = newtonRhapson(flux, Rg)

B = flux/(1/(100)^2);
X = [0.0,0.2,0.4,0.6,0.8,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9];
Y = [0.0, 14.7, 36.5, 71.7, 121.4, 197.4,
256.2,348.7,540.6,1062.8,2318.0,4781.9,8687.4,13924.3,22650.2];

xx = 0:.01:1.9;
P1 = piecelin(X,Y,xx);
% index = find(xx==flux);
% P1(1);
% Y_point = P1(index);

f = Rg*flux + 0.3*polyval(P1,B) - 8000;
```

```
%f = Rg*flux + 0.3*polyval(P1,flux) - 8000;
```

NEWTONRHAPSONDER.M

```
function f = newtonRhapsonDer(flux, Rg)
B = flux/(1/(100)^2);
X = [-0.2,0.0,0.2,0.4,0.6,0.8,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0];
Y = [-14.7,0.0, 14.7, 36.5, 71.7, 121.4, 197.4,
256.2,348.7,540.6,1062.8,2318.0,4781.9,8687.4,13924.3,22650.2, 36574.6];
xx = 0:.01:1.9;
P1 = piecelin(X,Y,xx);
Ydiff = [];
for i = 2: length(xx)-1
    Ydiff = [Ydiff,(P1(i+1) - P1(i-1))/(xx(i+1) - xx(i-1))];
end
Ydiff = [Ydiff(1), Ydiff];
Ydiff = [Ydiff,Ydiff(length(xx)-1)];
%Ydiff = [Ydiff, Ydiff(length(xx)-1)];
% Ydiff = polyder(P1);
% xx = 0:.01:1.88;
plot(Ydiff,xx);
%plot(xx,P1,xx(index),Y_point,'o');
f = Rg + (0.3*polyval(Ydiff, B))/(1/100^2);
```

SUCCESIVESUBSTITUTION.M

```
function f = successiveSubstitution(x, tolerance)
SUo = (1*10^(-2))*(1*10^(-2))*4*pi*(10^-7);
La = 0.5*(10^(-2));
Rg = La/SUo;
i = 0;
while (abs(newtonRhapson(x,Rg)/newtonRhapson(0,Rg)) > tolerance)
    i = i + 1;
    x = fSubstitution(x);
end
f = x;

function v = fSubstitution(flux)
B = flux/(1/(100)^2);
X = [0.0,0.2,0.4,0.6,0.8,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9];
Y = [0.0, 14.7, 36.5, 71.7, 121.4, 197.4,
256.2,348.7,540.6,1062.8,2318.0,4781.9,8687.4,13924.3,22650.2];

xx = 0:.01:1.9;
P1 = piecelin(X,Y,xx);
v =  8000/(39.78873577e6 + 0.3*polyval(P1,B)/flux);
```

PIECELIN.M

```
function v = piecelin(x,y,u)
%PIECELIN Piecewise linear interpolation.
% v = piecelin(x,y,u) finds piecewise linear L(x)
% with L(x(j)) = y(j) and returns v(k) = L(u(k)).
% First divided difference
delta = diff(y)./diff(x);
% Find subinterval indices k so that x(k) <= u <x(k+1)
```

```
n = length(x);
k = ones(size(u));
for j = 2:n-1
k(x(j) <= u) = j;
end
% Evaluate interpolant
s = u - x(k);
v = y(k) + s.*delta(k);
```

ELECTRICCIRCUIT.M

```
E = 0.2;
R = 512.0;
Isa = 0.0000006;
Isb = 0.0000012;
V1 = 0.0;
V2 = 0.0;
k = 0;
Vlist = [];
f1list = [];
f2list = [];
V = newRapCircuit(E,R,V1,V2,Isa,Isb,k);
Vlist = [Vlist, V];
f1 = V(1,1) - E + R * Isa *(exp((V(1,1) - V(2,1))/0.025) - 1.0);
f1list = [f1list, f1];
while (f1 > 10^(-6))
    k = k + 1;
    V = newRapCircuit(E,R,V(1,1),V(2,1),Isa,Isb,k);
    Vlist = [Vlist, V];
    f1 = V(1,1) - E + R * Isa *(exp((V(1,1) - V(2,1))/0.025) - 1.0);
    f2 = Isa * ((exp((V(1,1) - V(2,1)) / 0.025) - 1.0)) - Isb * (exp(V(2,1) /
0.025) - 1.0) ;
    f1list = [f1list, f1];
    f2list = [f2list, f1];
end
f1 = V(1,1) - E + R * Isa * (exp((V(1,1) - V(2,1)) / 0.025) - 1.0);
f2 = Isa * ((exp((V(1,1) - V(2,1)) / 0.025) - 1.0)) - Isb * (exp(V(2,1) /
0.025) - 1.0) ;
f1list = [f1list, f1];
f2list = [f2list, f1];
Vlist = Vlist';
f1list = f1list';
f2list = f2list';
klist = 1:k+2;
klist = klist';
figure
plot(klist,f1list);
```

INTEGRATIONTEST.M

```
error = [];
N = [];
x = 0:0.01:1;
%plot(log(0.2*abs(sin(x))));
fun = @(x) log(0.2*abs(sin(x)));
q = integral(fun,0,1);
for i = 10:10:200
```

```
    N = [N, i];
    %temp = abs(integration(0.0 , 1.0, i)-(-cos(1.0)-(-cos(0.0))));
    %temp = abs(integration(0.0 , 1.0, i)-(-1));
    temp = abs(integration(0.0 , 1.0, i)-q);
    error = [error, temp];
end
```

INTEGRATION.M

```
function f =  integration(a, b, n)
segLen = (b-a)/n;
runningSum = 0;

for i =1:n
    %runningSum = runningSum + sin(segLen*((i-1)+0.5));
    %runningSum = runningSum + log(segLen*((i-1)+0.5));
    runningSum = runningSum + log(0.2*abs(sin(segLen*((i-1)+0.5))));
end
f  = runningSum * segLen;
```

INTEGRATEUNEVEN.M

```
b = 1.0;
a = 0.0;
relativeWidths = [1,2,4,8,16,32,64,128,256, 512];
scale = (b-a) / sum(relativeWidths);
widths = [];
for i = 1:length(relativeWidths)
    temp = (relativeWidths(i)*scale);
    widths = [widths, temp];
end

runningWidth = 0;
runningSum = 0;
for i = 1: length(widths)
    %runningSum = runningSum + log((widths(i)/2) + runningWidth)*widths(i);
    runningSum = runningSum + log(0.2*abs((widths(i)/2) +
runningWidth))*widths(i);
    runningWidth = runningWidth + widths(i);
end

f =  runningSum;
%errorUneven = abs(runningSum-(-1));
errorUneven = abs(runningSum-(q));
```