**ENS409 Project 5 Report**

**Question 1**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Solution**

Initially, let me start with main.m

clear;

clc;

%% Forward Difference Formula

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

forward\_difference(t, y)

%% Backward Difference Formula

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

backward\_difference(t, y)

%% Three Point Formula

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

three\_point(t, y)

%% Central Difference Formula

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

central\_difference(t, y)

%% Five Point Formula

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

five\_point(t, y)

%% Richard Extrapolation

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

t1 = 0:0.1:2\*pi;

Y1 = 3\*cos(t1\*1000);

t2 = 0:0.05:2\*pi;

Y2 = 3\*cos(t2\*1000);

richard(t, y, t1, Y1, t2,Y2)

%% Trapezoidal Rule

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

trapezoidal(t,y)

%% Simpson's Rule

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

area = simpsons(t,y);

fprintf(" Integral by Simpson's rule is %.5f", area);

fprintf("\n");

%% Composite Simpson's Rule

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

area = composite\_simpsons(t,y);

fprintf(" Integral by Composite Simpson's rule is %.5f", area);

fprintf("\n");

%% Romberg Integration

t = 0:0.2:2\*pi;

y = 3\*cos(t\*1000);

area = romberg(t,y);

fprintf(" Integration by Romberg rule with R(6,6) is %.5f", area);

fprintf("\n");

First, let’s look at the differentiation results. In each function, I followed the methods given in the lecture notes. Firstly, forward difference.

I am also going to show with diff function to show the resemblance because the function value changes dramatically after differentiation, and it does not have the same effect when I use forward difference. It can be also seen when I use diff function.

function forward\_difference(X, Y)

Y\_derivative = -3000\*sin(X\*1000);

second\_derivative = -3000000\*cos(X\*1000);

C = diff(Y)/0.2;

h = 0.2;

estim = [];

for i=1:31

estim(i) = ((Y(i + 1) - Y(i))/h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

subplot(2,2,4)

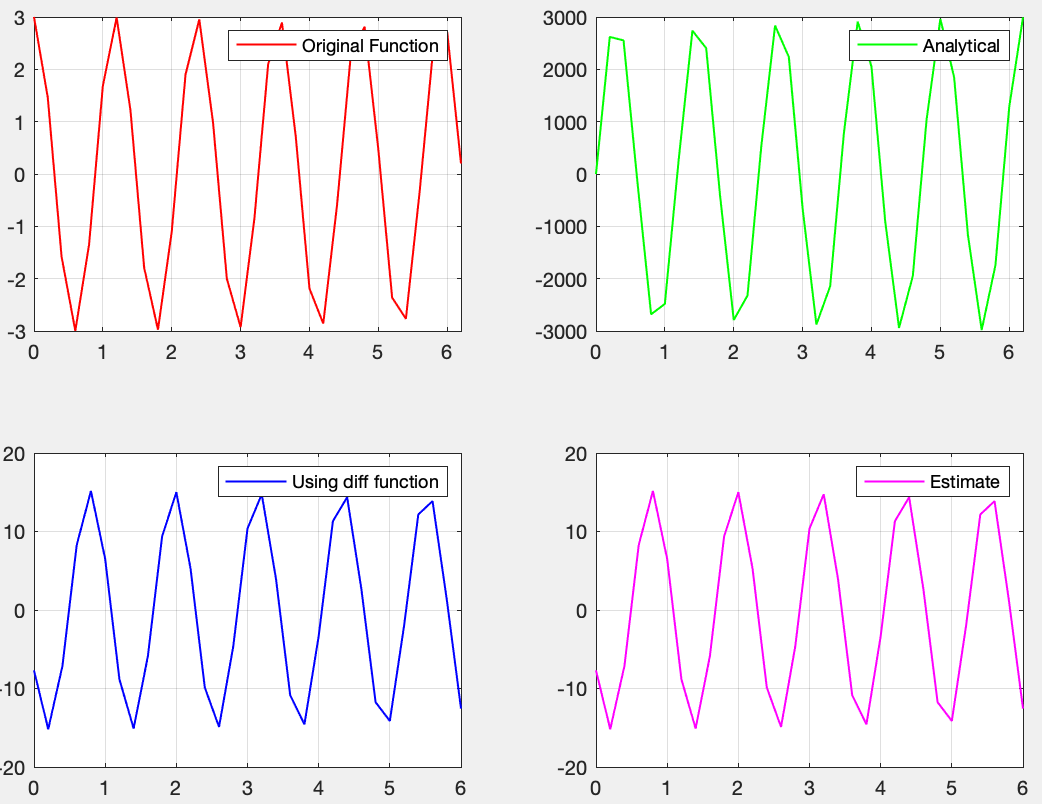
plot(X(1:end -1), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Estimate");

grid on

end

Now, let’s look at the graph.



As you can see we get a very similar result which is given in diff function.

Now, backward difference.

function backward\_difference(X, Y)

Y\_derivative = -3000\*sin(X\*1000);

C = diff(Y)/0.2;

h = -0.2;

estim = [];

for i=1:31

estim(i) = (Y(i) - Y(i + 1))/h;

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

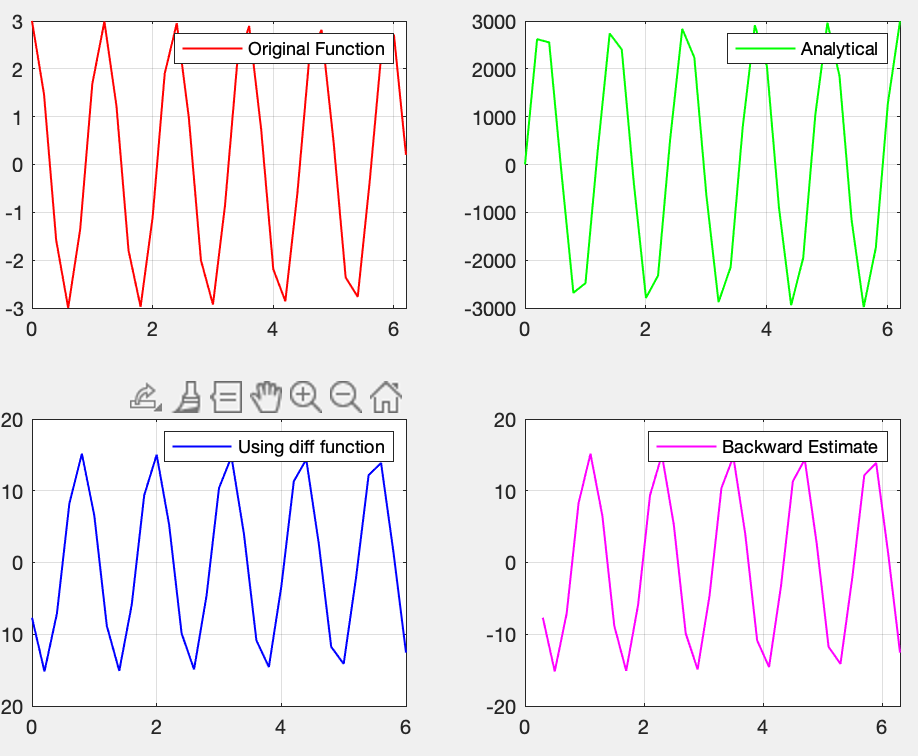
subplot(2,2,4)

plot(X(2:end)- h/2, estim, 'LineWidth', 1, 'color', 'magenta')

legend("Backward Estimate");

grid on

end



Now, let’s look at the three-point formula

function three\_point(X, Y)

Y\_derivative = -3000\*sin(X\*1000);

second\_derivative = -3000000\*sin(X\*1000);

h = 0.2;

estim = [];

for i=1:30

estim(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

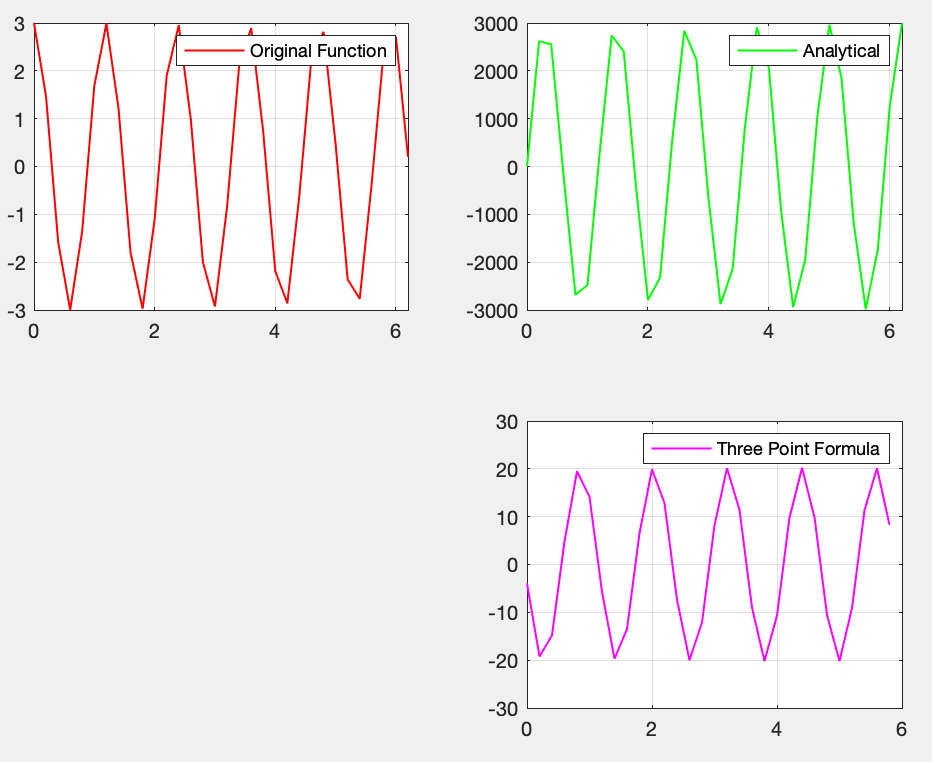
subplot(2,2,4)

plot(X(1:end -2), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Three Point Formula");

grid on

end



Now, let’s continue with central difference.

function central\_difference(X, Y)

Y\_derivative = -3000\*sin(X\*1000);

C = diff(Y)/0.2;

h = 0.2;

estim = [];

for i=1:30

estim(i) = (Y(i + 2) - Y(i))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

subplot(2,2,4)

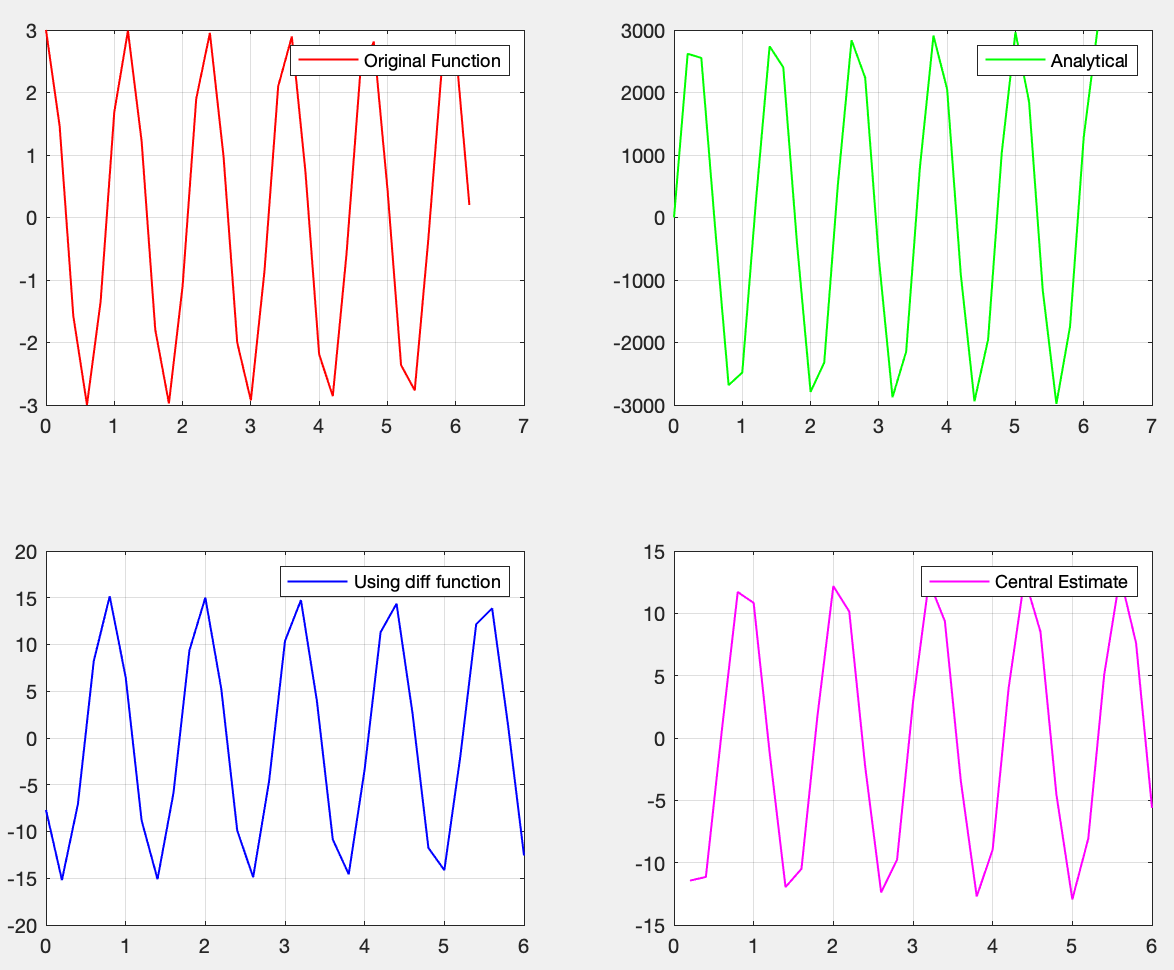
plot(X(2:end -1), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Central Estimate");

grid on

end

Now, let us look at the graph



Now, let’s look at five point

function five\_point(X, Y)

Y\_derivative = -3000\*sin(X\*1000);

second\_derivative = -3000000\*sin(X\*1000);

h = 0.2;

estim = [];

for i=1:28

estim(i) = (Y(i)-8\*Y(i+1)+ 8\*Y(i+3)-Y(i+4))/(12\*h);

end

estim2 = [];

for i=1:30

estim2(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

temp = X(3:end -2);

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -2), estim2, 'LineWidth', 1, 'color', 'magenta')

legend("Three Point Formula");

grid on

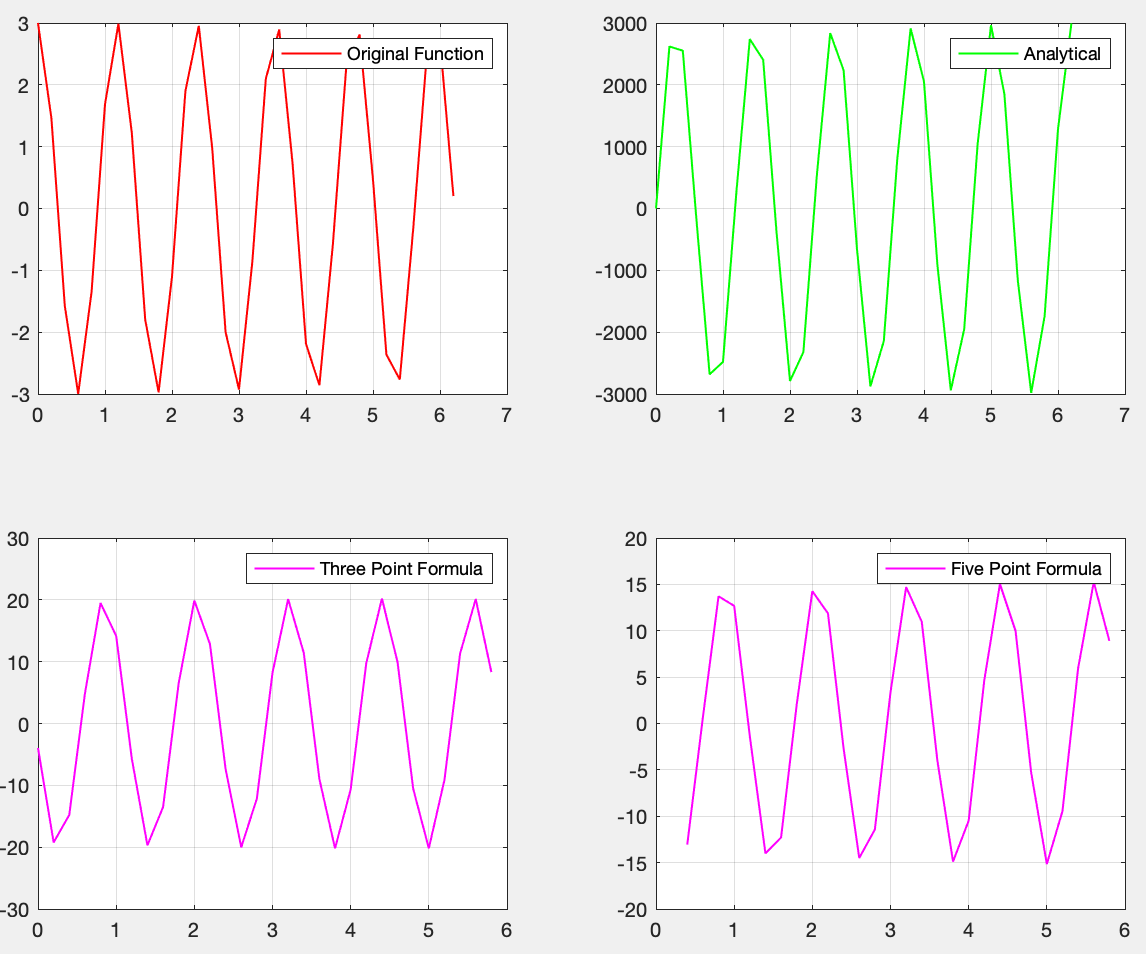
subplot(2,2,4)

plot(X(3:end -2), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Five Point Formula");

grid on

end



Now, let’s look at richardson extrapolation.

function richard(X,Y, X1, Y1, X2, Y2)

Y\_derivative = -3000\*sin(X\*1000);

second\_derivative = -3000000\*sin(X\*1000);

h = 0.2;

h1 = 0.1;

h2 = 0.05;

richardson\_array = [];

for i=1:30

richardson\_array(1,i) = (Y(i + 2) - Y(i))/(2\*h);

end

for i=1:30

an = X1(2\*i);

bn = X1(2\*i+2);

richardson\_array(2,i) = (Y1(2\*i+2) - Y1(2\*i))/(2\*h1);

end

for i=1:30

an = X2(4\*i);

bn = X2(4\*i+2);

richardson\_array(3,i) = (Y2(4\*i+2) - Y2(4\*i))/(2\*h2);

end

N\_array = [];

for i=1:30

N\_array(1,i) = richardson\_array(2,i) + ((richardson\_array(2,i)- richardson\_array(1,i))/3);

end

for i=1:30

N\_array(2,i) = richardson\_array(3,i) + ((richardson\_array(3,i)- richardson\_array(2,i))/3);

end

for i=1:30

N\_array(3,i) = N\_array(2,i) + ((N\_array(2,i)- N\_array(1,i))/15);

end

estim = [];

for i=1:28

estim(i) = (Y(i)-8\*Y(i+1)+ 8\*Y(i+3)-Y(i+4))/(12\*h);

end

estim2 = [];

for i=1:30

estim2(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -2), estim2, 'LineWidth', 1, 'color', 'magenta')

hold on, axis on

plot(X(3:end -2), estim, 'LineWidth', 1, 'color', 'blue')

legend("Three Point Formula", "Five Point Formula");

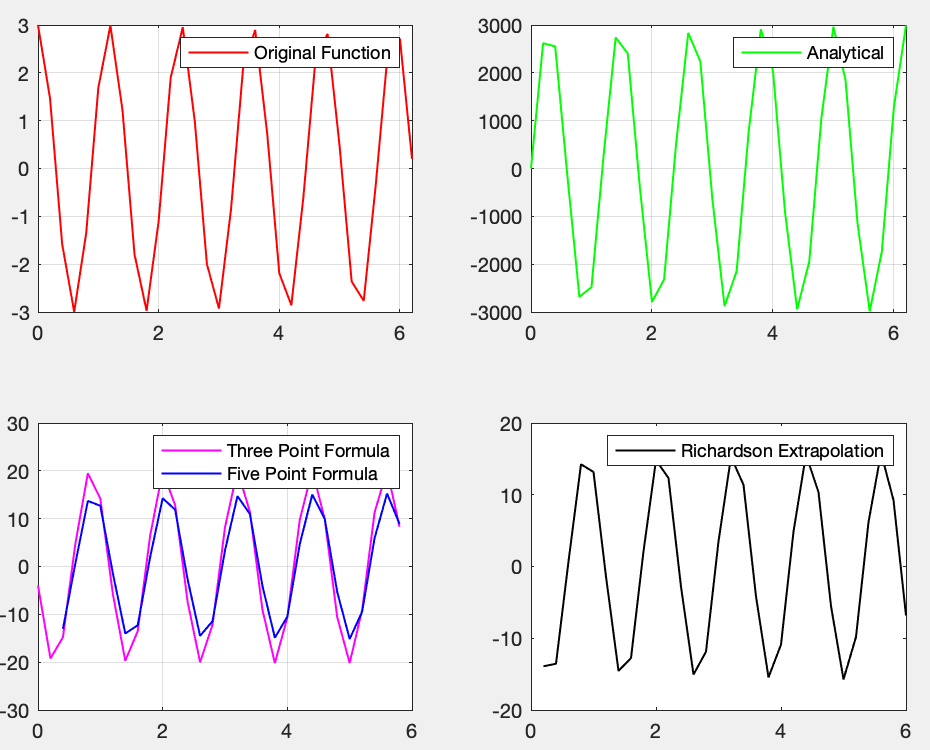
grid on

subplot(2,2,4)

plot(X(2:end -1), N\_array(3,:), 'LineWidth', 1, 'color', 'black')

legend("Richardson Extrapolation");

end



The plot of differentiation methods have been finished.

Now, let’s continue with integration methods.

Initially, let’s start with trapezoidal rule.

function Iout = trapezoidal(X,Y)

h = X(end) - X(1);

traparea = h/2\*(Y(end) + Y(1));

m = (Y(end) - Y(1))/ (X(end) - X(1));

draw\_func = @(x)(m\*x + 3);

draw\_y = draw\_func(X);

Iout = traparea;

figure

subplot(1,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(1,2,2)

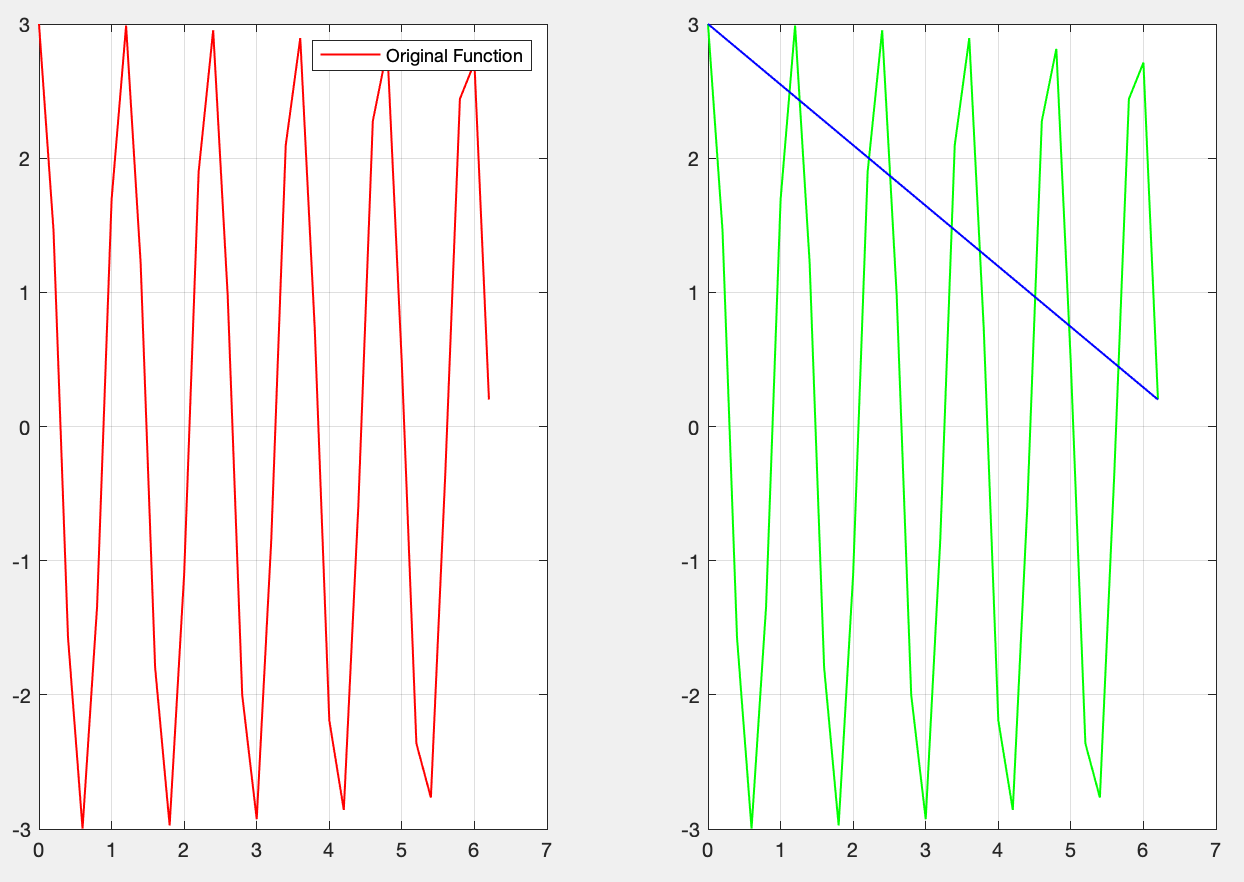
plot(X,Y, 'LineWidth', 1, 'color', 'green')

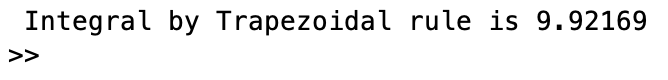
hold on; axis on;

plot(X,draw\_y, 'LineWidth', 1, 'color', 'blue')

grid on

end





As you can see, it is not a good result.

Now, let’s continue with Simpson’s rule

function Iout = simpsons(X,Y)

h = (X(end) - X(1))/2;

mid = 3\*cos((X(1) + h)\*1000);

area = h/3\*(Y(end) + Y(1) + 4\*mid);

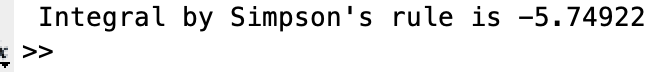
Iout = area;

% m = (Y(end) - Y(1))/ (X(end) - X(1));

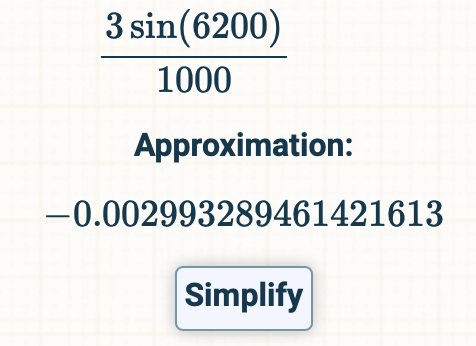
% draw\_func = @(x)(m\*x + 3);

% draw\_y = draw\_func(X);

end



Again, not a very good result. Moreover, the integral result we are loking for is very close to 0.



Now, let’s continue with Composite Simpson’s rule.

function Iout = composite\_simpsons(X,Y)

h = (X(end) - X(1))/size(X,2) -1;

odd = 0;

even = 0;

for i=1: size(X,2)/2 -1

odd = odd + Y(2\*i);

end

for i=2: size(X,2)/2

even = even + Y(2\*i -1);

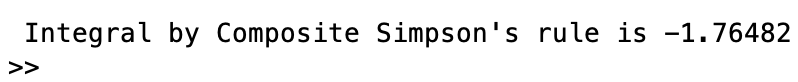
disp(Y(2\*i -1));

end

area = h/3\*(Y(end) + Y(1) + 4\*odd + 2\*even);

Iout = area;

end



Finally, let’s look at the Romberg integration.

function Iout = romberg(X,Y)

h(1) = (X(end) - X(1));

h(2) = (X(end) - X(1))/2;

h(3) = (X(end) - X(1))/4;

h(4) = (X(end) - X(1))/8;

h(5) = (X(end) - X(1))/16;

h(6) = (X(end) - X(1))/32;

Rom = zeros(6,6);

Rom(1,1) = h(1)/2\*((Y(end) + Y(1)));

for i = 2:6

temp = 0;

for j = 1: 2^(i-2)

temp = temp + 3\*cos((i + (2\*j -1)\*h(i))\*1000);

end

Rom(i,1) = 0.5\*(Rom(i-1,1) + h(i-1)\*temp);

end

for i = 2:6

Rom(i,2) = (4\*Rom(i,1) - Rom(i-1,1))/3;

end

for i = 3:6

Rom(i,3) = (16\*Rom(i,2) - Rom(i-1,2))/15;

end

for i = 4:6

Rom(i,4) = (64\*Rom(i,3) - Rom(i-1,3))/63;

end

for i = 5:6

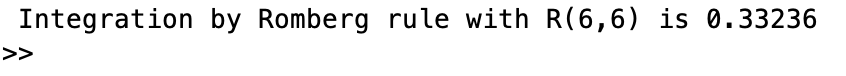
Rom(i,5) = (256\*Rom(i,4) - Rom(i-1,4))/255;

end

Rom(6,6) = (1024\*Rom(6,5) - Rom(5,5))/1023;

Iout = Rom(6,6);

end



The closest result to the real solution that we reached among the integration methods is Romberg. Then, Composite Simpson is following Romberg integration method.

**Question 2**

****

**Solution**

In order to implement the approximation with different step sizes, I used a general approach and then called the respective functions. Here, you will see the code q2\_main.m

clear;

clc;

%% Forward Difference Formula

h1 = 10^(-1);

h2 = 10^(-5);

h3 = 10^(-15);

x1 = [1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3];

y3 = x3.\*sin(x3);

q2forward\_difference(x1, y1)

%% Backward Difference Formula

h1 = 10^(-1);

h2 = 10^(-5);

h3 = 10^(-15);

x1 = [1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3];

y3 = x3.\*sin(x3);

q2backward\_difference(x1, y1)

%% Three Point Formula

h1 = 10^(-1);

h2 = 10^(-5);

h3 = 10^(-15);

x1 = [1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3];

y3 = x3.\*sin(x3);

q2three\_point(x1, y1)

%% Central Difference Formula

h1 = 10^(-1);

h2 = 10^(-5);

h3 = 10^(-15);

x1 = [1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3];

y3 = x3.\*sin(x3);

q2central\_difference(x1, y1)

%% Five Point Formula

h1 = 10^(-1);

h2 = 10^(-5);

h3 = 10^(-15);

x1 = [1.2-3\*h1 1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1 1.2+3\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-3\*h2 1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2 1.2+3\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-3\*h3 1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3 1.2+3\*h3];

y3 = x3.\*sin(x3);

q2five\_point(x1, y1)

%% Richard Extrapolation

h1 = 10^(-1);

h2 = h1/2;

h3 = h2/2;

x1 = [1.2-3\*h1 1.2-2\*h1 1.2-h1 1.2 1.2+h1 1.2+2\*h1 1.2+3\*h1];

y1 = x1.\*sin(x1);

x2 = [1.2-3\*h2 1.2-2\*h2 1.2-h2 1.2 1.2+h2 1.2+2\*h2 1.2+3\*h2];

y2 = x2.\*sin(x2);

x3 = [1.2-3\*h3 1.2-2\*h3 1.2-h3 1.2 1.2+h3 1.2+2\*h3 1.2+3\*h3];

y3 = x3.\*sin(x3);

q2richard(x1, y1, x2, y2,x3, y3)

The reason I have used arrays for finding the approximation for x = 1.2 is that to use diff we need an array.

Firstly, let’s have a look at forward difference.

function q2forward\_difference(X, Y)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

C = diff(Y)/h;

estim = [];

for i=1:size(X,2)-1

estim(i) = ((Y(i + 1) - Y(i))/h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

subplot(2,2,4)

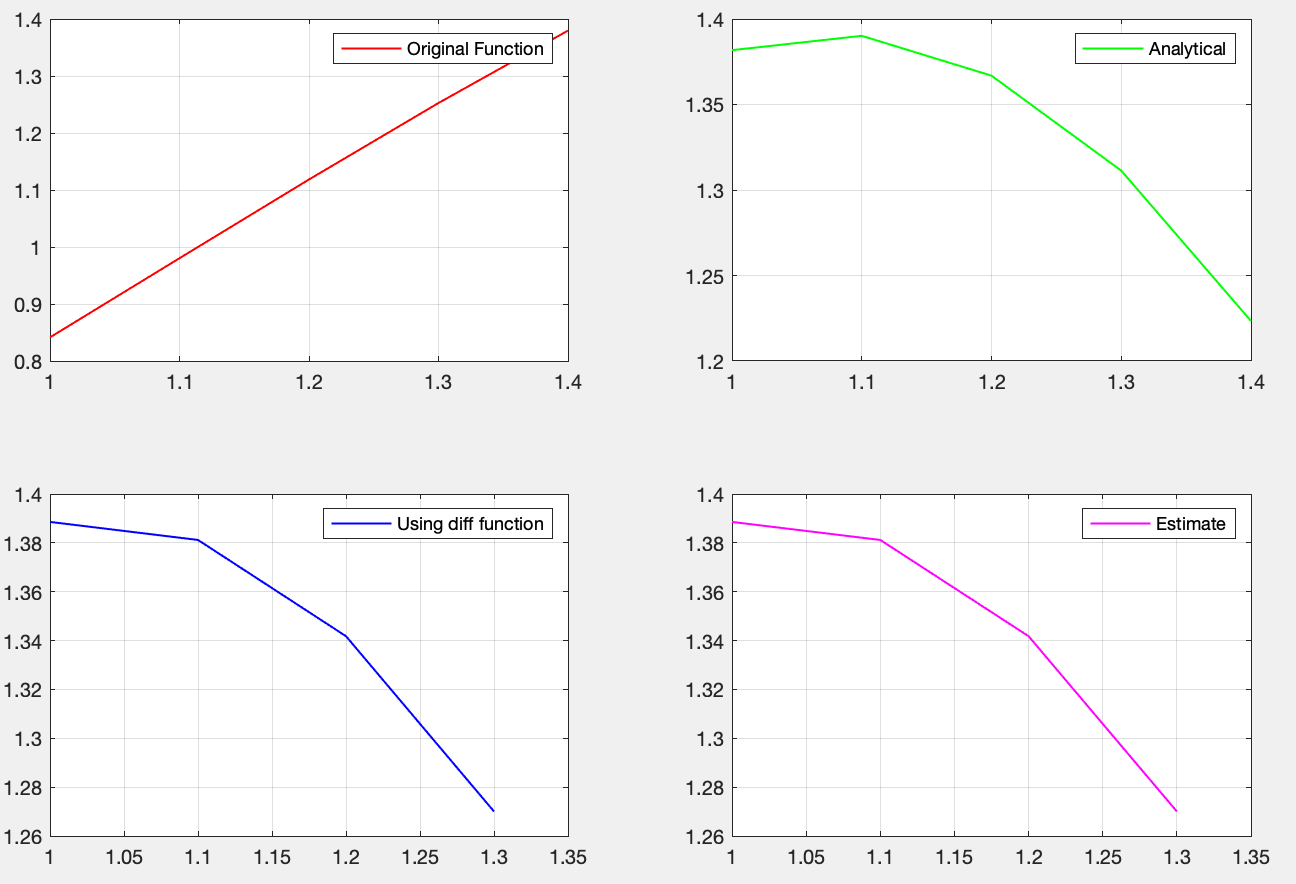
plot(X(1:end -1), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Estimate");

grid on

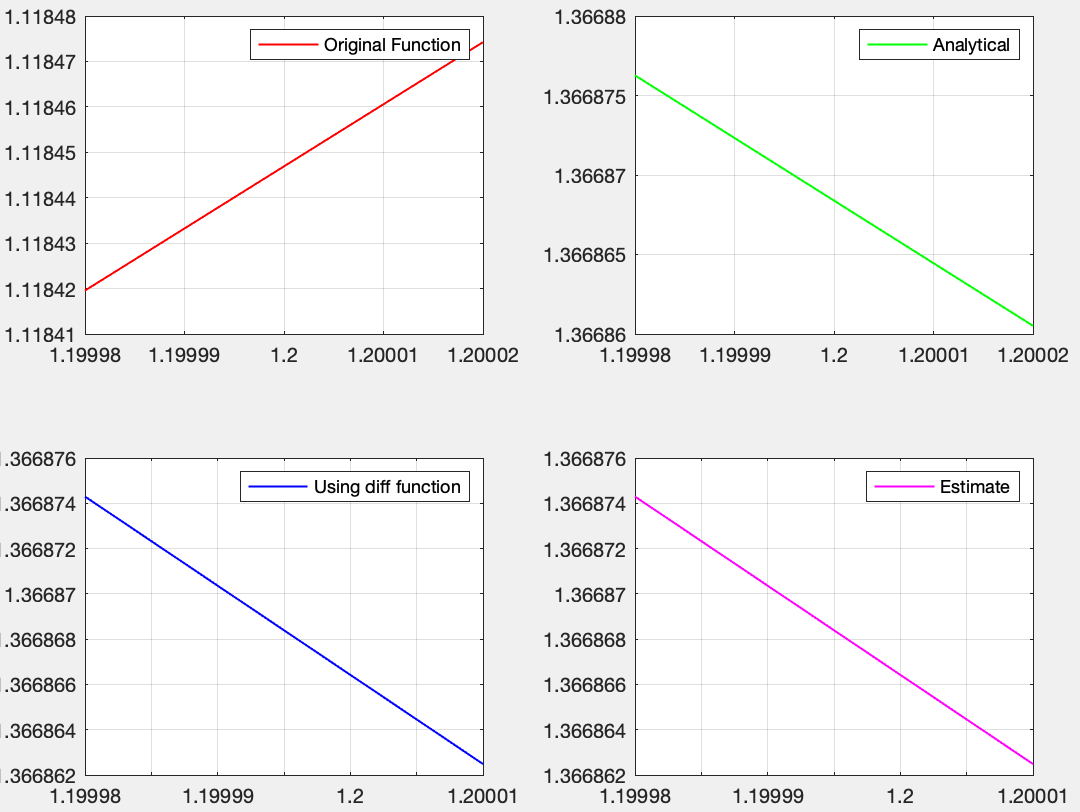
end

Now, let’s plot some graphs. The following is when n = 1



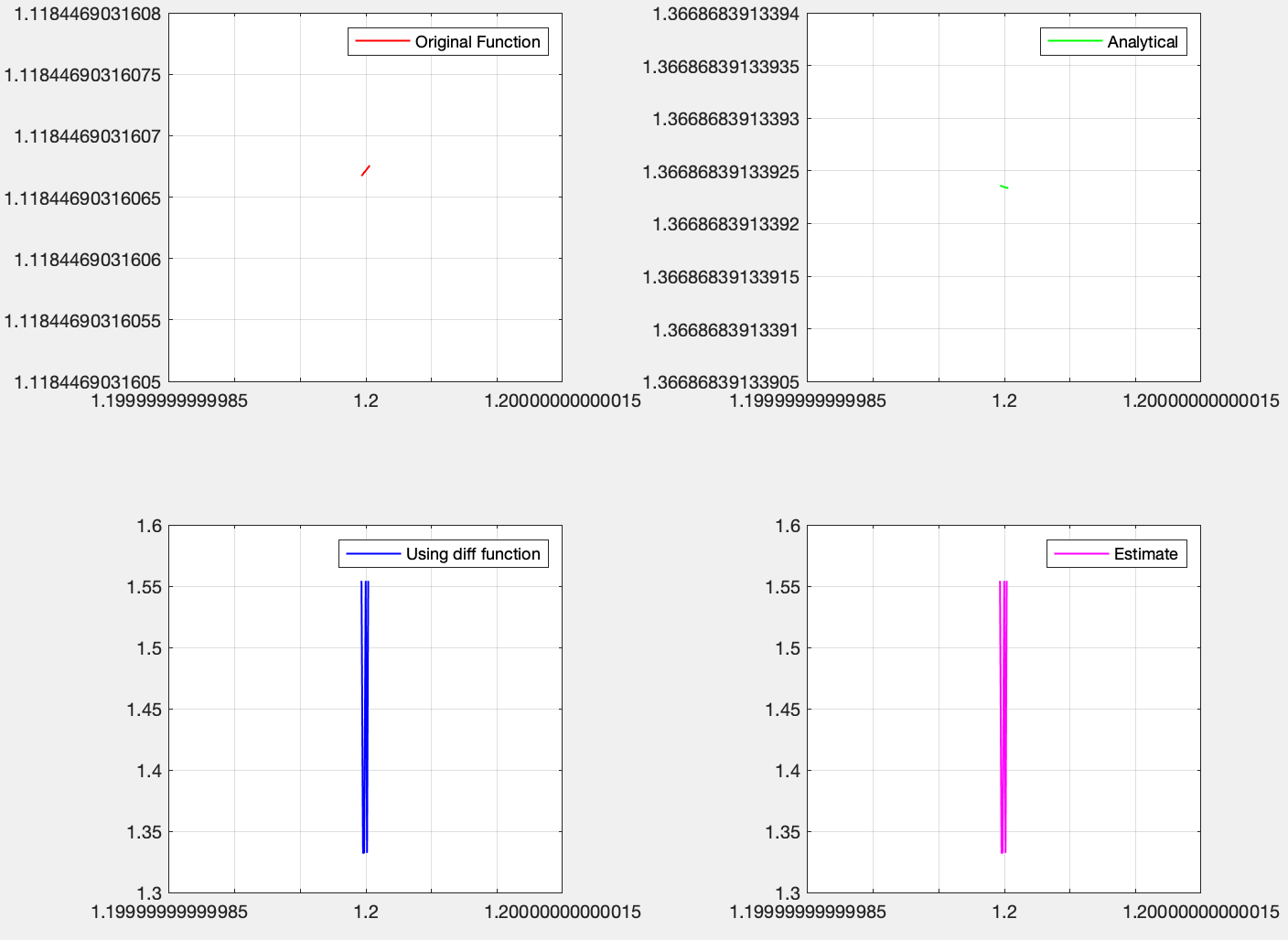
As you can see in the graphs, the value at x = 1.2 is very close in diff and estimate graphs.

The following is when n = 5



As you can see when we decrease the step size we get a more precise result.

The following is when n = 15



When n = 15, we got some sort of an awkward plot for the estimation.

Now, let’s look at the backward difference.

function q2backward\_difference(X, Y)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

C = diff(Y)/h;

estim = [];

for i=1:size(X,2)-1

estim(i) = (Y(i) - Y(i + 1))/-h;

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end-1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

subplot(2,2,4)

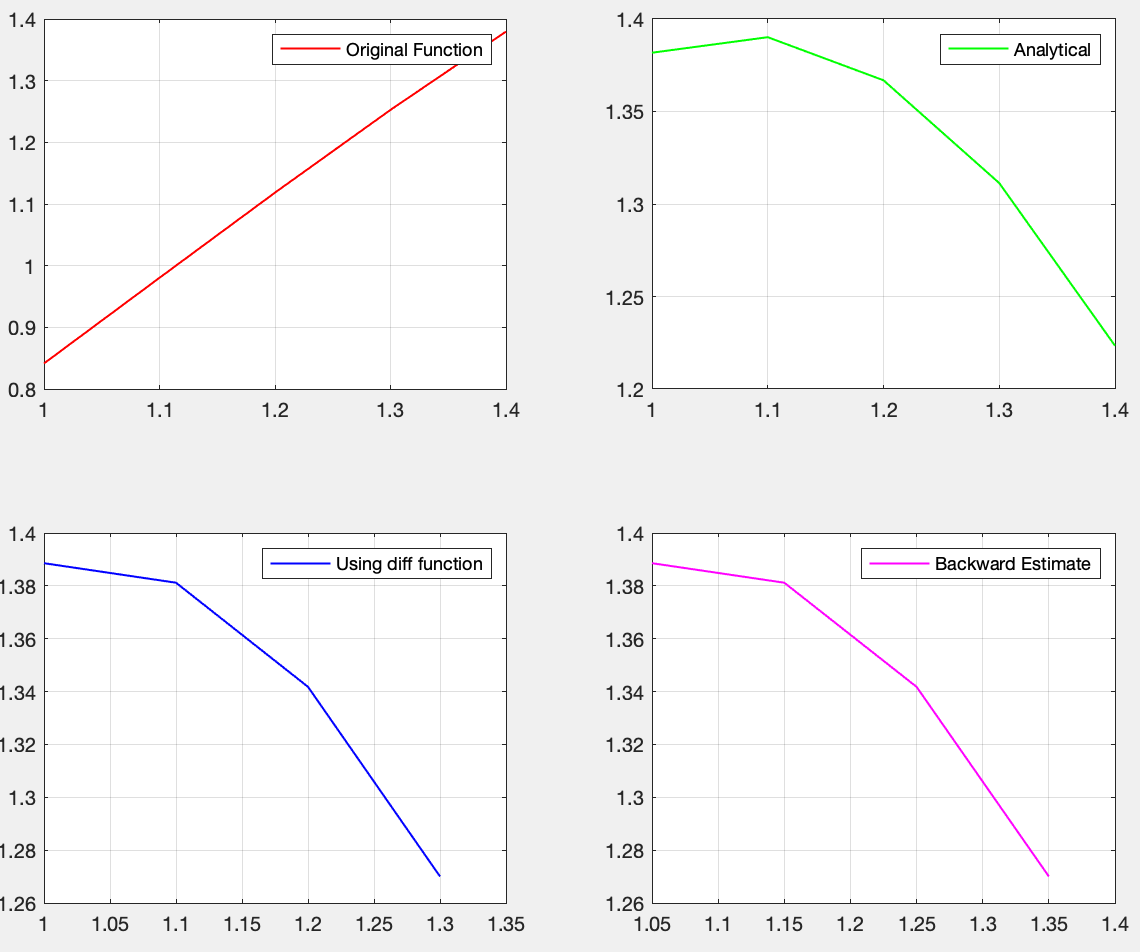
plot(X(2:end)- h/2, estim, 'LineWidth', 1, 'color', 'magenta')

legend("Backward Estimate");

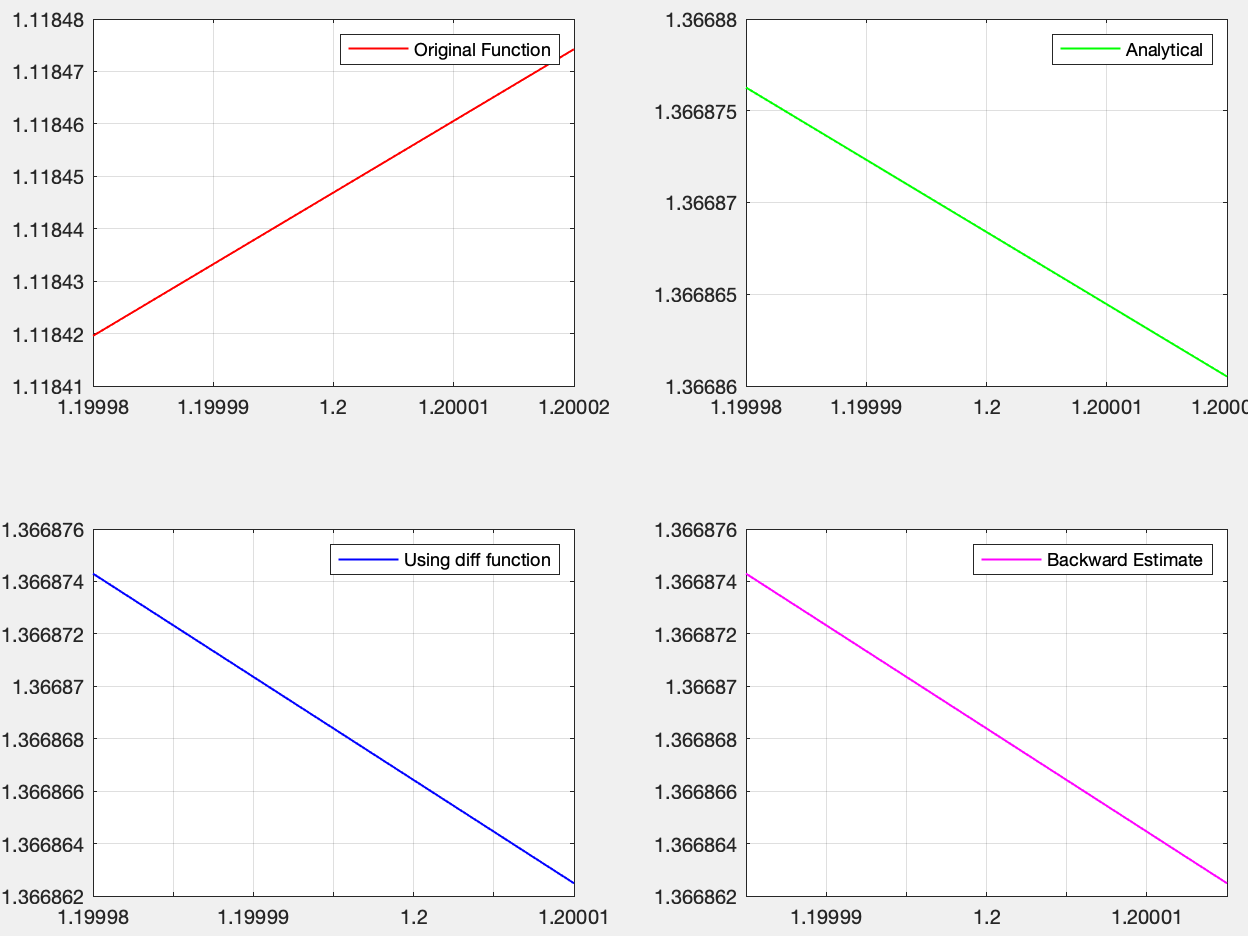
grid on

end

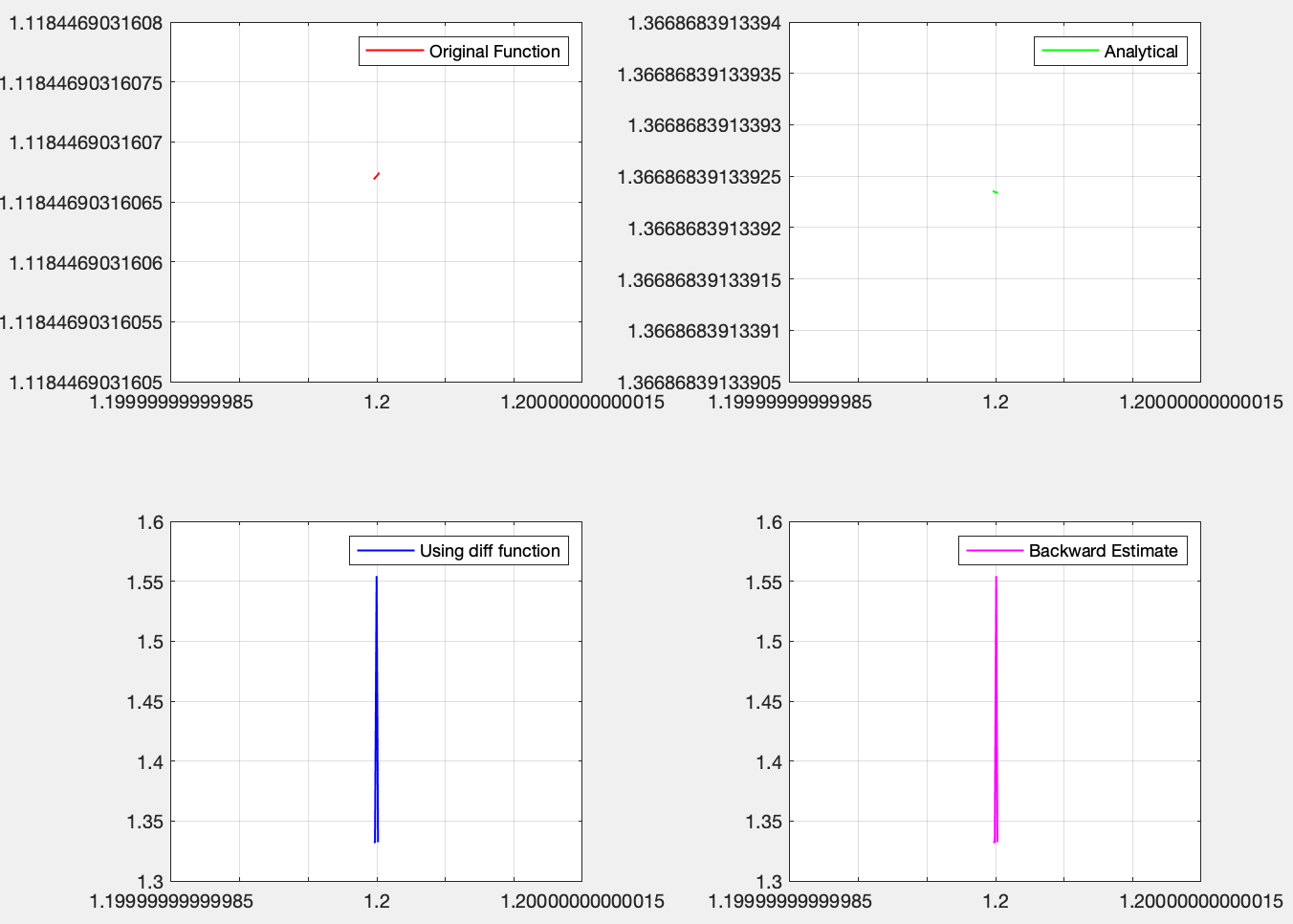
Now, let’s plot some graphs. First when n = 1



Now n = 5



Now n = 15



I believe the reason why we get an awkward plot is that we have a very small array and the values inside of it are very very close to each other.

Now, let’s continue with 3-point method.

function q2three\_point(X, Y)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

C = diff(Y)/h;

estim = [];

for i=1:3

estim(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,4)

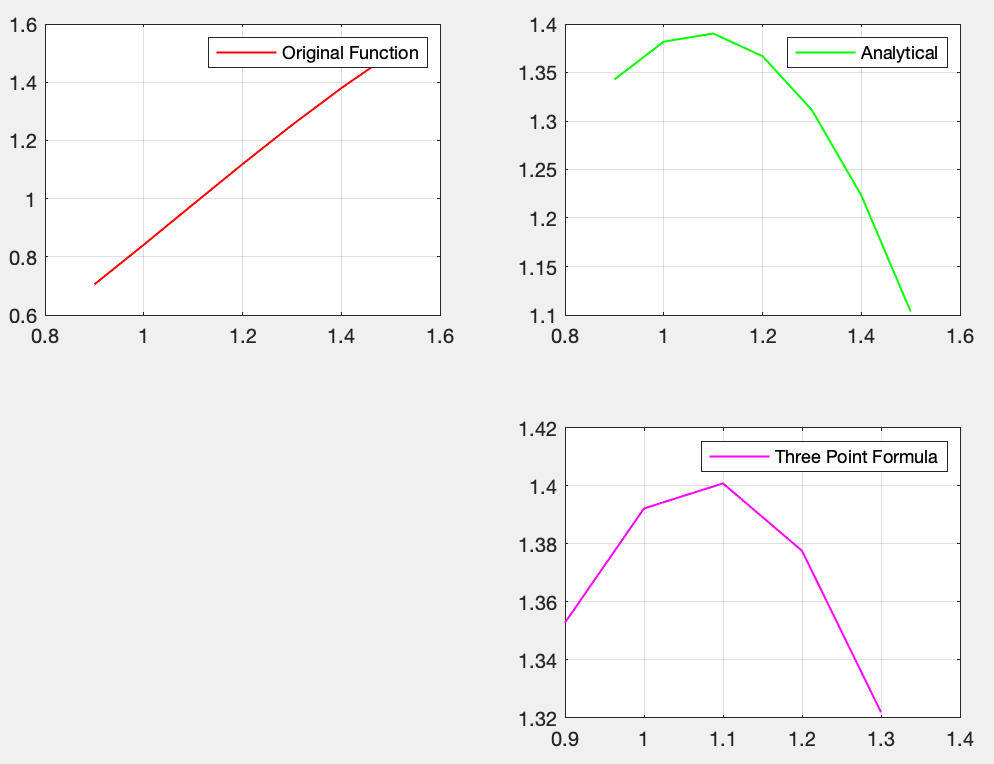
plot(X(1:end -2), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Three Point Formula");

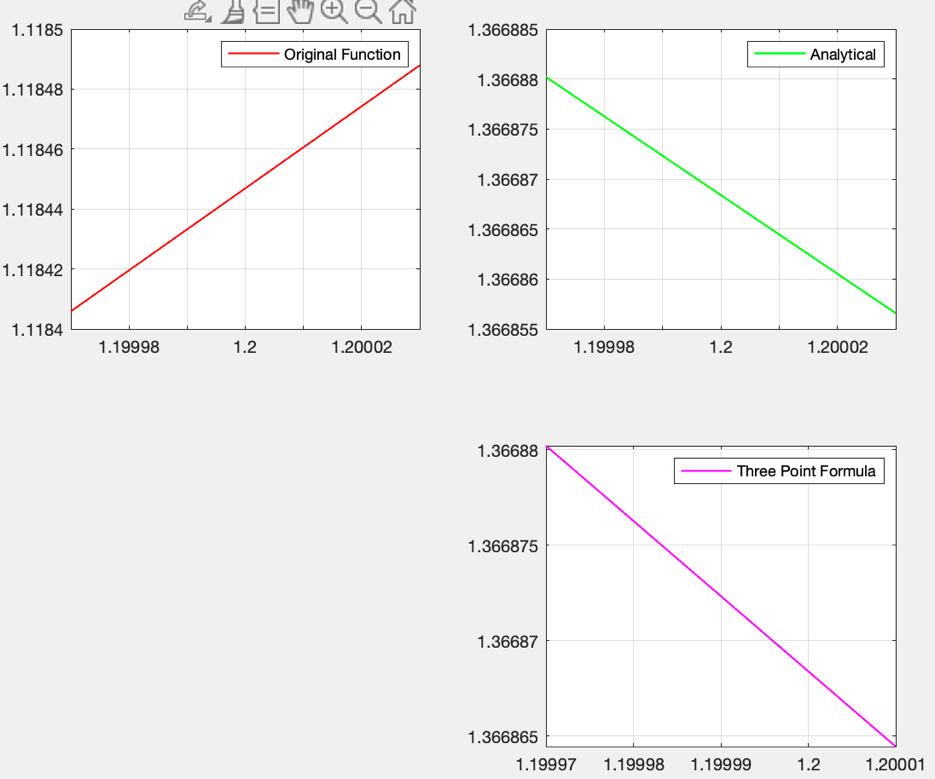
grid on

end

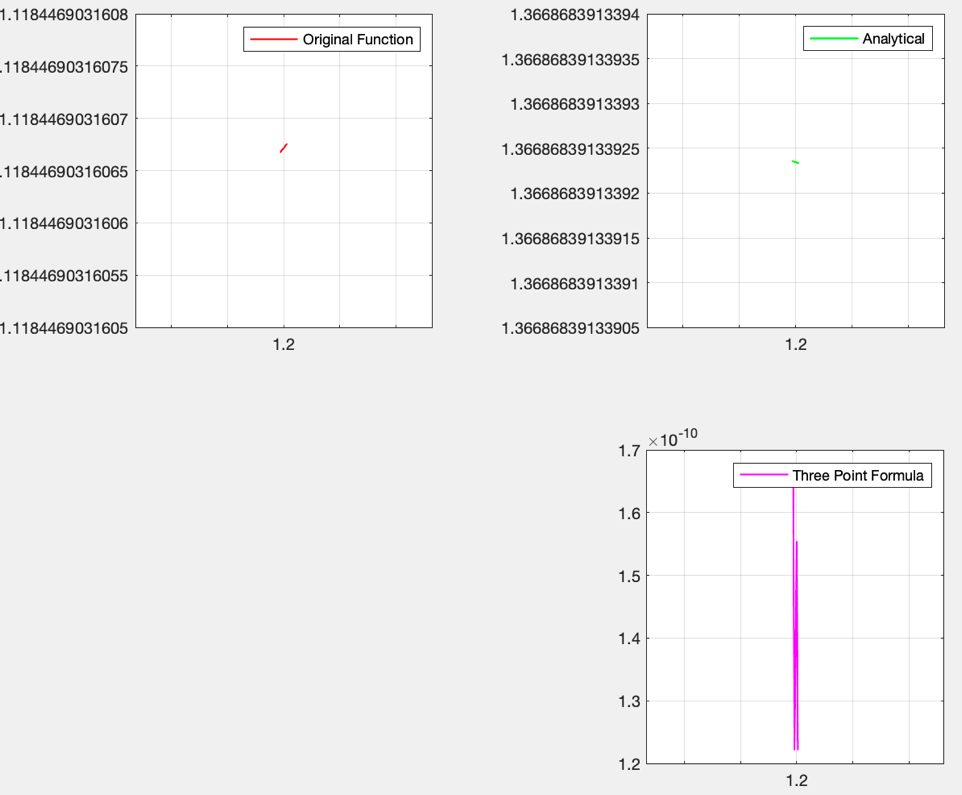
Let’s continue with n = 1



Now, continue with n = 5



Lastly, n = 15



Now, I continue with the central difference method.

function q2central\_difference(X, Y)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

C = diff(Y)/h;

estim = [];

for i=1:size(X,2)-2

estim(i) = (Y(i + 2) - Y(i))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -1),C, 'LineWidth', 1, 'color', 'blue')

legend("Using diff function");

grid on

subplot(2,2,4)

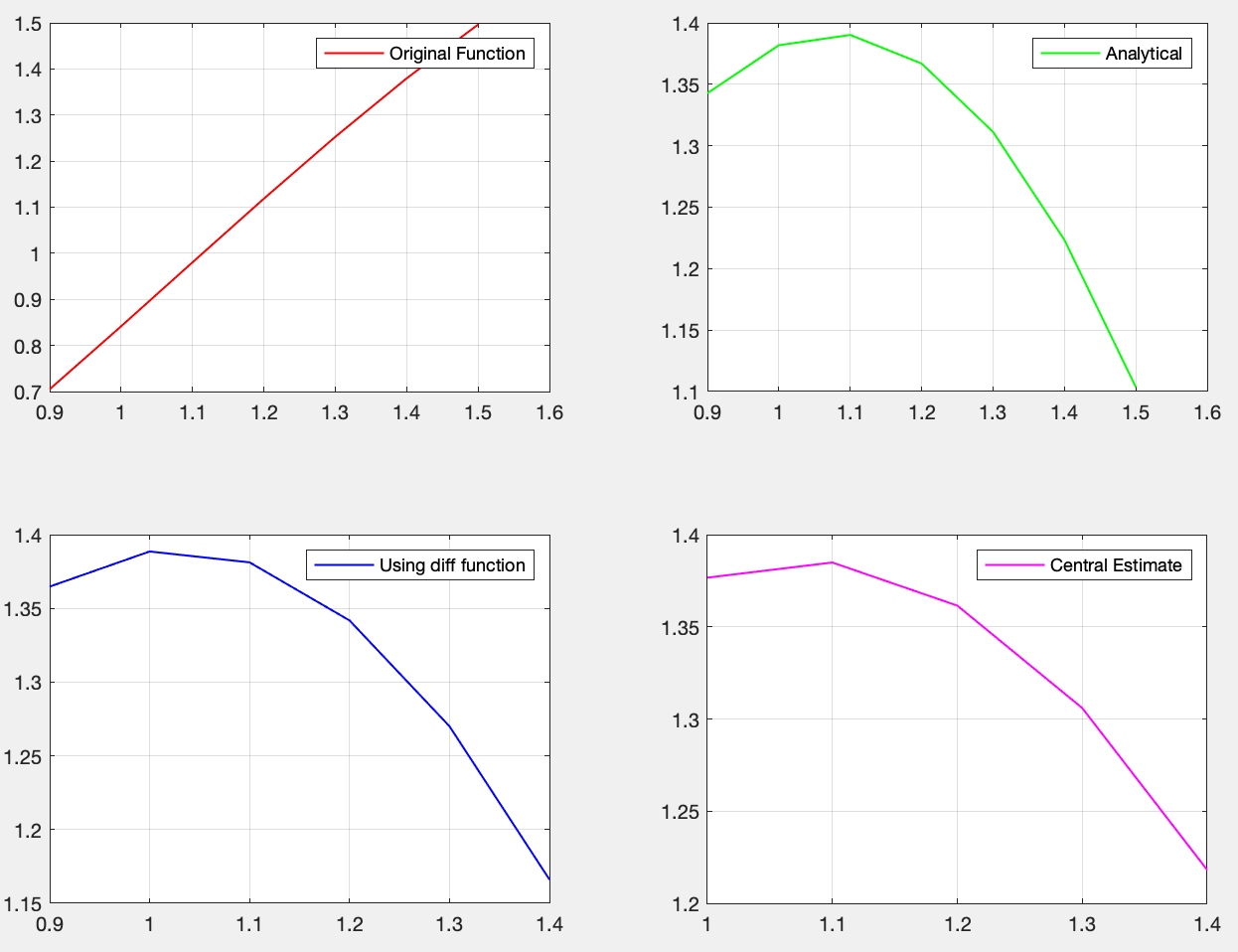
plot(X(2:end -1), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Central Estimate");

grid on

end

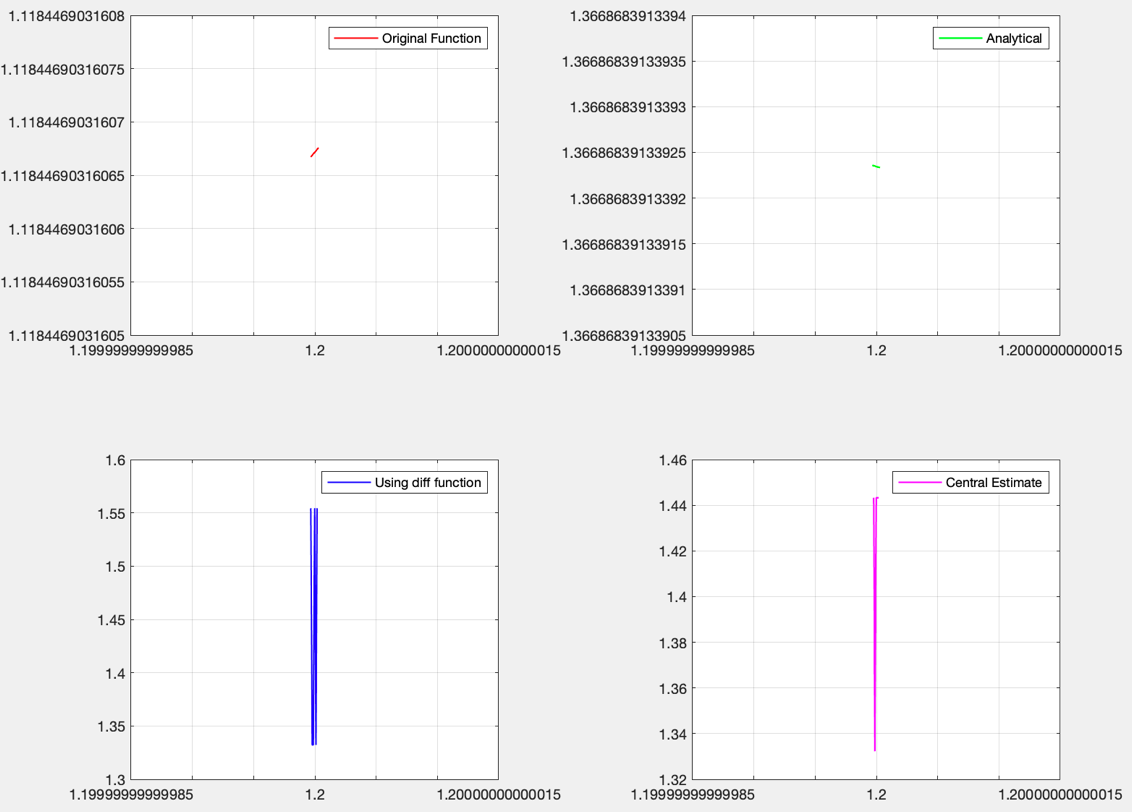
Firstly, n = 1



Secondly, n = 5



Finally, n = 15



Now, I continue with five point method.

function q2five\_point(X, Y)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

C = diff(Y)/h;

estim = [];

for i=1:size(X,2)-4

estim(i) = (Y(i)-8\*Y(i+1)+ 8\*Y(i+3)-Y(i+4))/(12\*h);

end

estim2 = [];

for i=1:size(X,2)-2

estim2(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -2), estim2, 'LineWidth', 1, 'color', 'magenta')

legend("Three Point Formula");

grid on

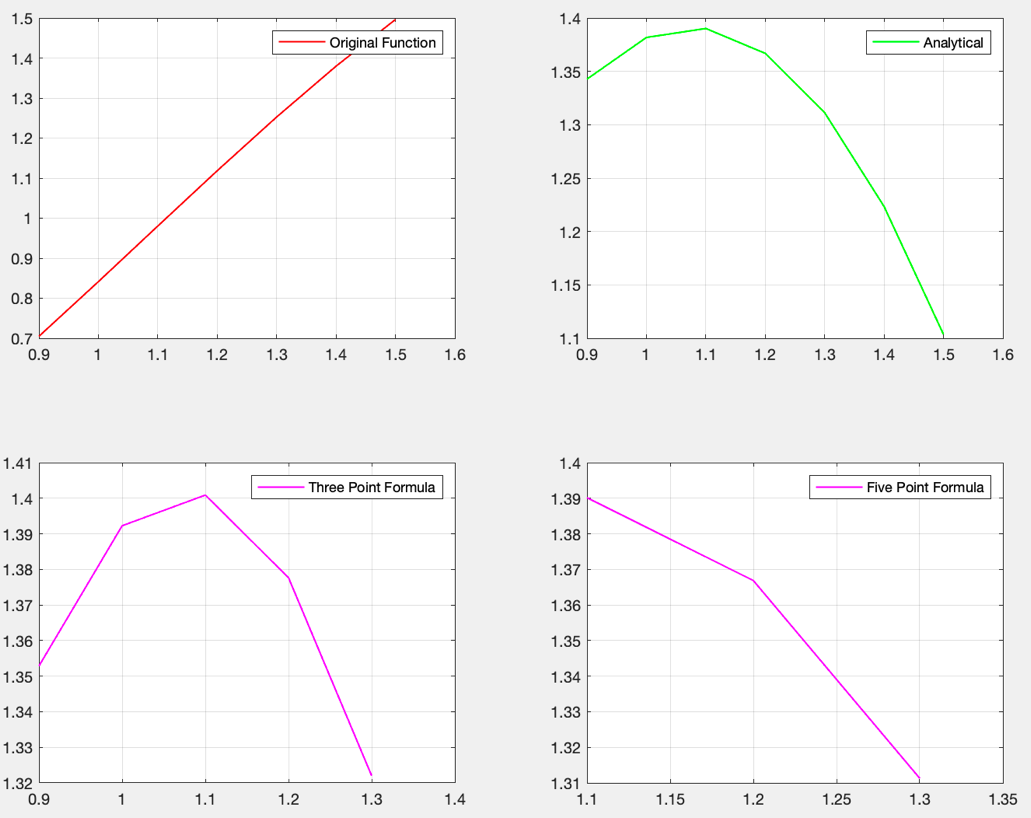
subplot(2,2,4)

plot(X(3:end -2), estim, 'LineWidth', 1, 'color', 'magenta')

legend("Five Point Formula");

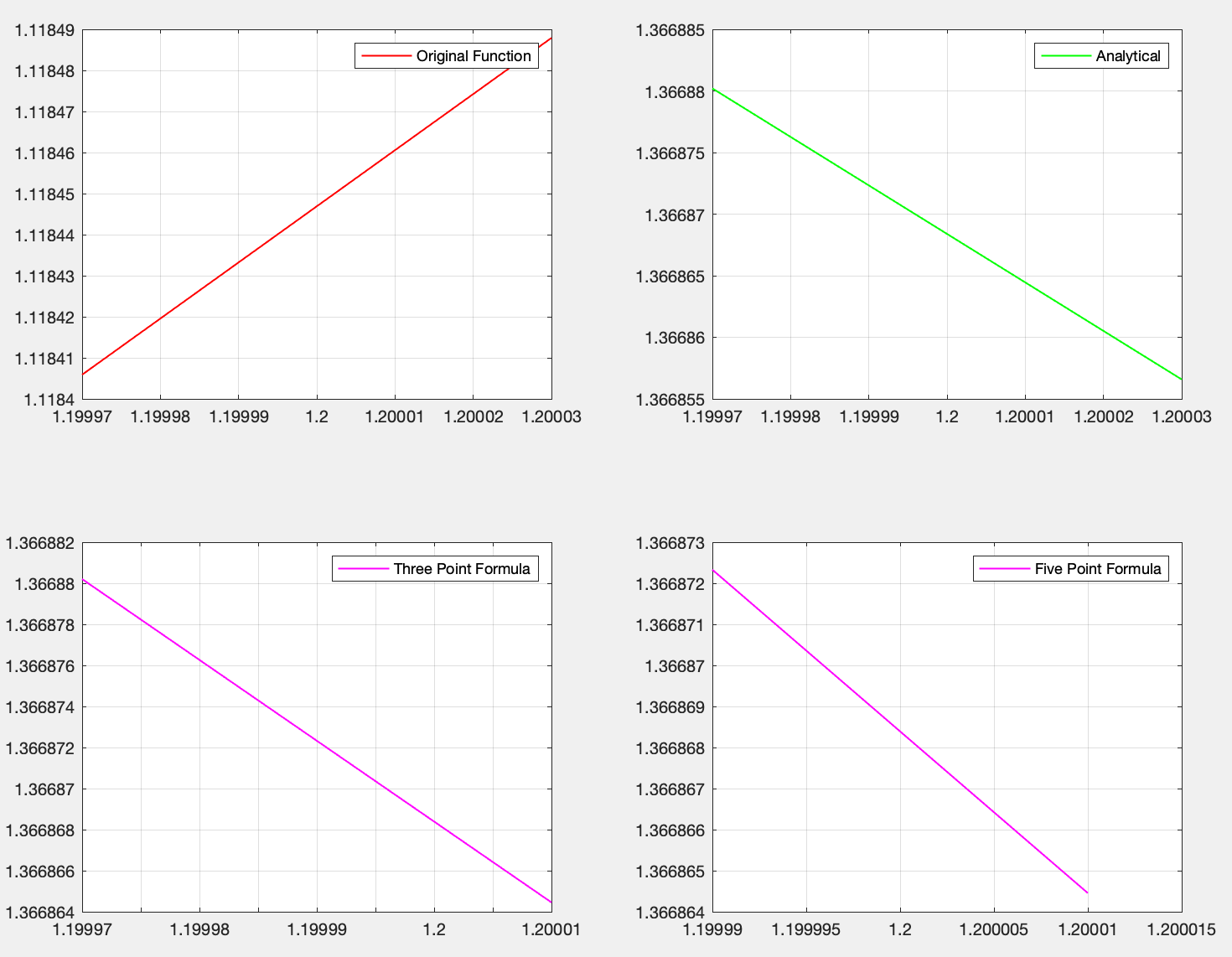
grid on

end



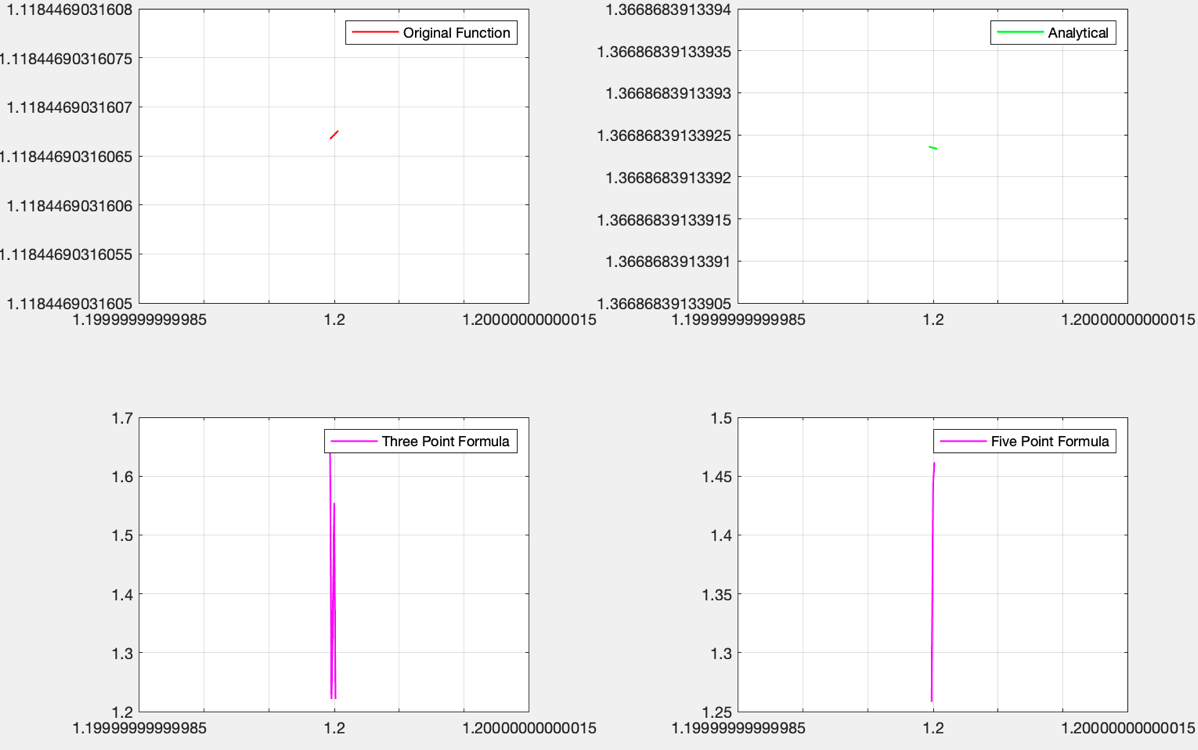
This plot represents when n = 1

Now, I will continue with when n = 5



Be careful about plots, the point we are focusing on is when x = 1.2

Now, look at when n = 15



Lastly, let’s look at Richardson extrapolation.

function q2richard(X,Y, X1, Y1, X2, Y2)

Y\_derivative = sin(X) + X.\*cos(X);

h = 10^(-1);

h1 = h/2;

h2 = h1/2;

richardson\_array = [];

for i=1:5

richardson\_array(1,i) = (Y(i + 2) - Y(i))/(2\*h);

end

for i=1:5

richardson\_array(2,i) = (Y1(i+2) - Y1(i))/(2\*h1);

end

for i=1:5

richardson\_array(3,i) = (Y2(i+2) - Y2(i))/(2\*h2);

end

N\_array = [];

for i=1:5

N\_array(1,i) = richardson\_array(2,i) + ((richardson\_array(2,i)- richardson\_array(1,i))/3);

end

for i=1:5

N\_array(2,i) = richardson\_array(3,i) + ((richardson\_array(3,i)- richardson\_array(2,i))/3);

end

for i=1:5

N\_array(3,i) = N\_array(2,i) + ((N\_array(2,i)- N\_array(1,i))/15);

end

estim = [];

for i=1:3

estim(i) = (Y(i)-8\*Y(i+1)+ 8\*Y(i+3)-Y(i+4))/(12\*h);

end

estim2 = [];

for i=1:5

estim2(i) = (-3\*Y(i) + 4\*Y(i+1) - Y(i+2))/(2\*h);

end

figure

subplot(2,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(2,2,2)

plot(X,Y\_derivative, 'LineWidth', 1, 'color', 'green')

legend("Analytical");

grid on

subplot(2,2,3)

plot(X(1:end -2), estim2, 'LineWidth', 1, 'color', 'magenta')

hold on, axis on

plot(X(3:end -2), estim, 'LineWidth', 1, 'color', 'blue')

legend("Three Point Formula", "Five Point Formula");

grid on

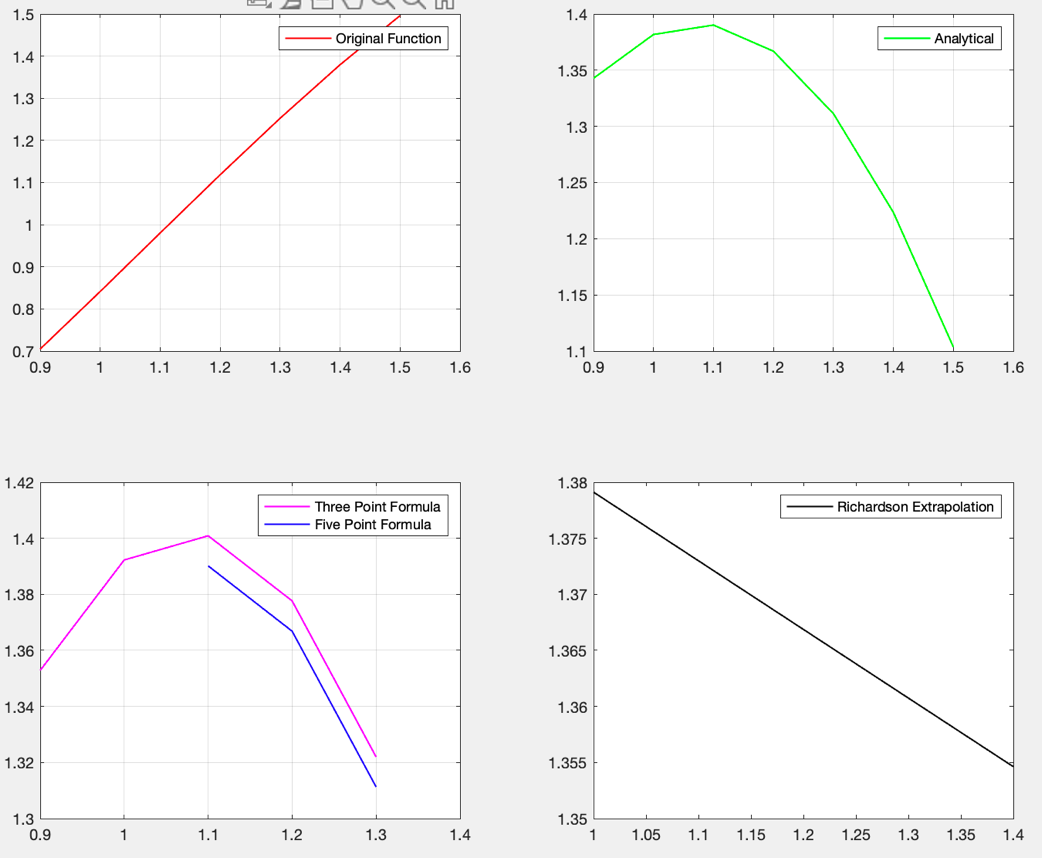
subplot(2,2,4)

plot(X(2:end -1), N\_array(3,:), 'LineWidth', 1, 'color', 'black')

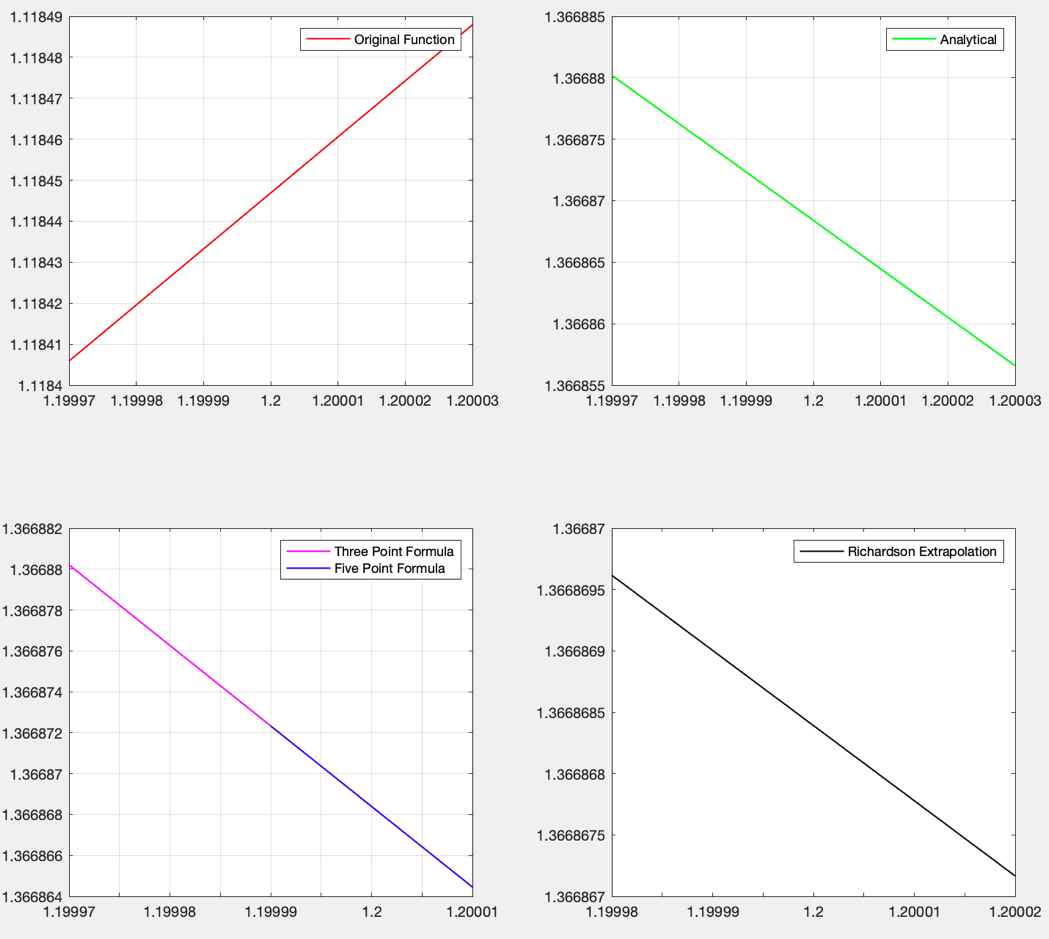
legend("Richardson Extrapolation");

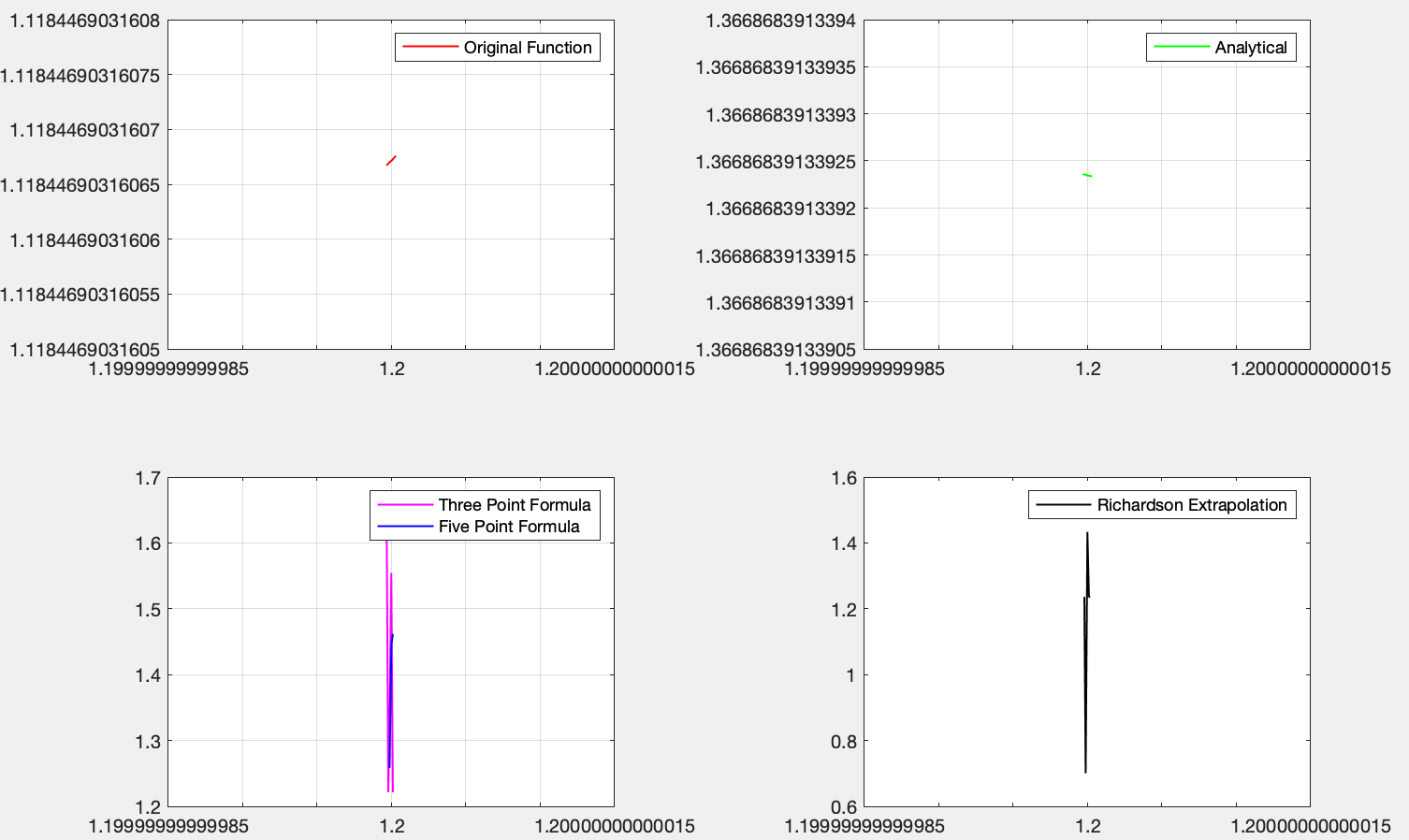
end

Let’s take a look at when n = 1



Now, continue with n = 5



Lastly, let’s look at n = 15

I believe, when I look at the values, if we compare the difference methods, central difference method gives us a better result. When we compare 3-point method and 5 point method, I can say that since we use more terms we get a better result in 5 point method. Richardson extrapolation also gave us a better result. However, since we get the results from the Richardson extrapolation and results are close to each other even though we decrease the step size. I believe I would choose between central difference, 5-point method and Richardson extrapolation. Since the implementation of central difference is very fast, I would probably use it in general. However, five-point method and Richardson are very useful and gave us very accurate results.

**Question 3**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Solution**

Initally the results we will receive from Trapezoidal, and Simpson’s will be based on initial and the last value we have within the range of the domain and output. Thus, we will not have a chance to improve them. However, we can focus on composite and Romberg functions in detail.

Integral value of the given function is 2.4689.

Firstly, let me share the q3main.m

clear;

clc;

%% Trapezoidal Rule

X = 0:0.2:5;

y = (20\*(sin(X).^4) + exp(X))./(30 - 12\*cos(X)+ exp(X));

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

q = integral(func,0,5);

area = q3trapezoidal(X,y);

error = abs(q-area);

fprintf(" Integral by trapezodial rule is %.5f", area);

fprintf("\n");

fprintf(" Absolute error is %.5f", error);

fprintf("\n");

if(error < 0.1)

fprintf(" Absolute error is less than 10^(-1)");

fprintf("\n");

end

if(error < 0.001)

fprintf(" Absolute error is less than 10^(-3)");

fprintf("\n");

end

if(error < 0.000001)

fprintf(" Absolute error is less than 10^(-6)");

fprintf("\n");

end

%% Simpson's Rule

X = 0:0.2:5;

y = (20\*(sin(X).^4) + exp(X))./(30 - 12\*cos(X)+ exp(X));

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

q = integral(func,0,5);

area = q3simpsons(X,y,q);

error = abs(q-area);

fprintf(" Integral by Simpson's rule is %.5f", area);

fprintf("\n");

fprintf(" Absolute error is %.5f", error);

fprintf("\n");

if(error < 0.1)

fprintf(" Absolute error is less than 10^(-1)");

fprintf("\n");

end

if(error < 0.001)

fprintf(" Absolute error is less than 10^(-3)");

fprintf("\n");

end

if(error < 0.000001)

fprintf(" Absolute error is less than 10^(-6)");

fprintf("\n");

end

%% Composite Simpson's Rule

X = 0:0.01:5;

y = (20\*(sin(X).^4) + exp(X))./(30 - 12\*cos(X)+ exp(X));

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

q = integral(func,0,5);

area = q3composite\_simpsons(X,y,q);

error = abs(q-area);

fprintf(" Integral by Composite Simpson's rule is %.5f", area);

fprintf("\n");

fprintf(" Absolute error is %.5f", error);

fprintf("\n");

if(error < 0.1)

fprintf(" Absolute error is less than 10^(-1)");

fprintf("\n");

end

if(error < 0.001)

fprintf(" Absolute error is less than 10^(-3)");

fprintf("\n");

end

if(error < 0.000001)

fprintf(" Absolute error is less than 10^(-6)");

fprintf("\n");

end

%% Romberg Integration

X = 0:0.2:5;

y = (20\*(sin(X).^4) + exp(X))./(30 - 12\*cos(X)+ exp(X));

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

q = integral(func,0,5);

area = q3romberg(X,y,q);

error = abs(q-area);

fprintf(" Integration by Romberg rule with R(6,6) is %.5f", area);

fprintf("\n");

fprintf(" Absolute error is %.5f", error);

fprintf("\n");

if(error < 0.1)

fprintf(" Absolute error is less than 10^(-1)");

fprintf("\n");

end

if(error < 0.001)

fprintf(" Absolute error is less than 10^(-3)");

fprintf("\n");

end

if(error < 0.000001)

fprintf(" Absolute error is less than 10^(-6)");

fprintf("\n");

end

Now, let’s continue with Trapezoidal rule

function Iout= q3trapezoidal(X,Y)

h = X(end) - X(1);

traparea = h/2\*(Y(end) + Y(1));

m = (Y(end) - Y(1))/ (X(end) - X(1));

draw\_func = @(x)(m\*x + 0.0526);

draw\_y = draw\_func(X);

Iout = traparea;

figure

subplot(1,2,1)

plot(X,Y, 'LineWidth', 1, 'color', 'red')

legend("Original Function");

grid on

subplot(1,2,2)

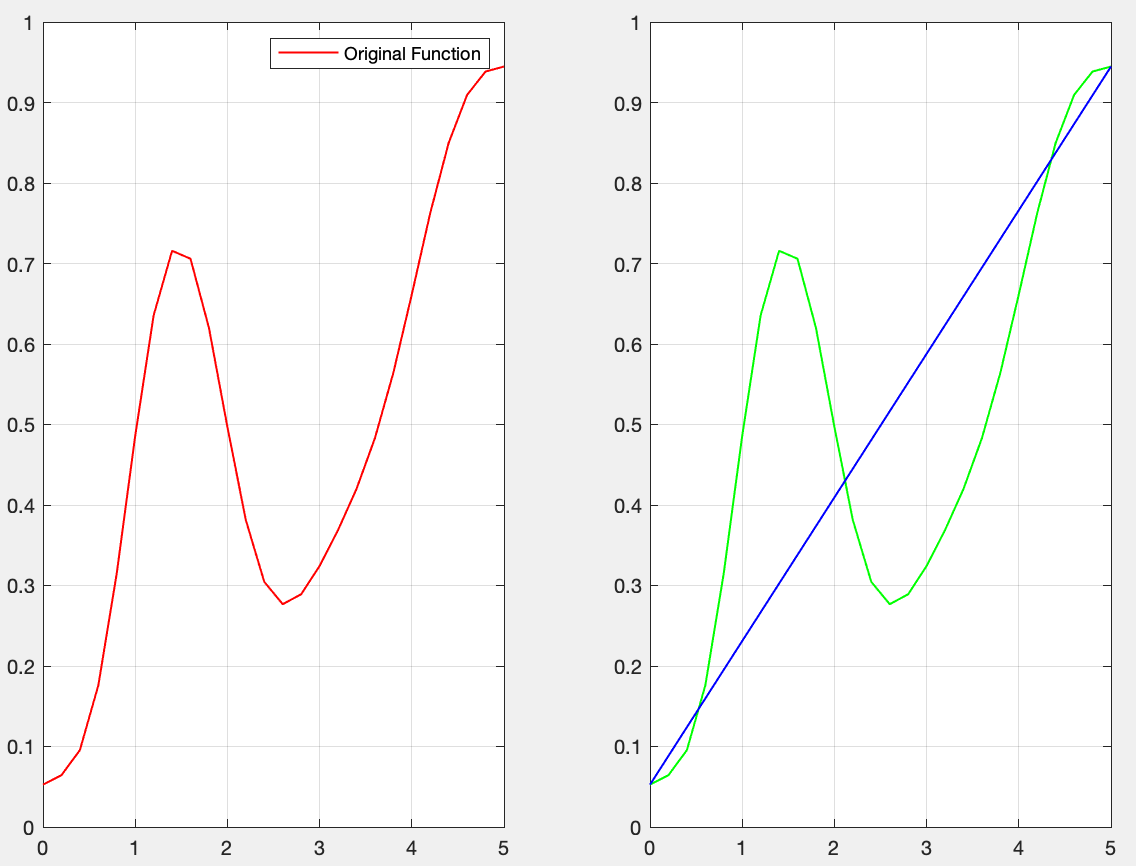
plot(X,Y, 'LineWidth', 1, 'color', 'green')

hold on; axis on;

plot(X,draw\_y, 'LineWidth', 1, 'color', 'blue')

grid on

end



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Now, let’s continue with Simpson’s rule

function Iout = q3simpsons(X,Y, real)

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

h = (X(end) - X(1))/2;

mid = func(X(1) + h);

area = h/3\*(Y(end) + Y(1) + 4\*mid);

Iout = area;

% m = (Y(end) - Y(1))/ (X(end) - X(1));

% draw\_func = @(x)(m\*x + 3);

% draw\_y = draw\_func(X);

end

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

The result is worse than the trapezoidal if we compare the absolute errors.

Now, let’s continue with composite simpson.

function Iout = q3composite\_simpsons(X,Y, real)

h = (X(end) - X(1))/(size(X,2)-1);

odd = 0;

even = 0;

for i=1: size(X,2)/2 -1

odd = odd + Y(2\*i);

end

for i=2: size(X,2)/2

even = even + Y(2\*i -1);

%disp(Y(2\*i -1));

end

area = h/3\*(Y(end) + Y(1) + 4\*odd + 2\*even);

Iout = area;

end

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

If we assign X as X = 0:0.0000001:5;

Then we can get the following

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Now, let’s continue with Romberg.

function Iout = q3romberg(X,Y, real)

func = @(x)(20\*sin(x).^4 + exp(x))./(30 - 12\*cos(x)+ exp(x));

h(1) = (X(end) - X(1));

h(2) = (X(end) - X(1))/2;

h(3) = (X(end) - X(1))/4;

h(4) = (X(end) - X(1))/8;

h(5) = (X(end) - X(1))/16;

h(6) = (X(end) - X(1))/32;

h(7) = (X(end) - X(1))/64;

h(8) = (X(end) - X(1))/128;

Rom = zeros(8,8);

Rom(1,1) = h(1)/2\*((Y(end) + Y(1)));

for i = 2:8

temp = 0;

for j = 1: 2^(i-2)

temp = temp + func((X(1) + (2\*j -1)\*h(i)));

end

Rom(i,1) = 0.5\*(Rom(i-1,1) + h(i-1)\*temp);

end

for i = 2:8

Rom(i,2) = (4\*Rom(i,1) - Rom(i-1,1))/3;

end

for i = 3:8

Rom(i,3) = (16\*Rom(i,2) - Rom(i-1,2))/15;

end

for i = 4:8

Rom(i,4) = (64\*Rom(i,3) - Rom(i-1,3))/63;

end

for i = 5:8

Rom(i,5) = (256\*Rom(i,4) - Rom(i-1,4))/255;

end

for i = 6:8

Rom(i,6) = (1024\*Rom(i,5) - Rom(i-1,5))/1023;

end

for i = 7:8

Rom(i,7) = (4096\*Rom(i,6) - Rom(i-1,6))/4095;

end

Rom(8,8) = (16384\*Rom(8,7) - Rom(8,8))/16383;

Iout = Rom(8,8);

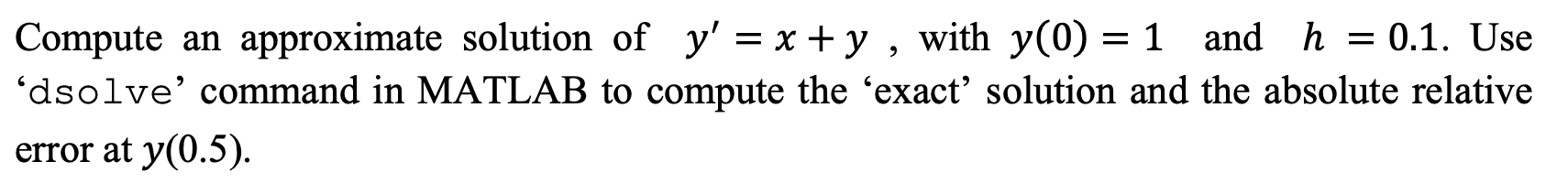
End

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

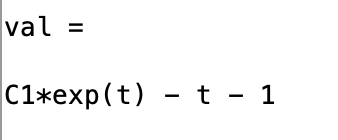
By increasing the Romberg size, it is possible that we can achieve error less than 10^(-6)

**Question 4**



**Solution**

Thus, I first started with dsolve command so that I can find the exact solution of the problem. When I used the dsolve to find the equation, I get following result.

**** Since I know the y(0) = 1, I find that C1 is 2

Then, I continue with Euler Method, Midpoint Method and Runge-Kutta method consecutively.

For implementing each method, I followed the lecture notes.

Firstly, let’s look at Euler’s method

%% Euler's Method

x\_real = 0:0.1:0.5;

y0 = 1;

syms y(t) a

eqn = diff(y,t) == t+y;

S = dsolve(eqn);

real\_func = @(x)(2\*exp(x) - x - 1);

y = real\_func(x\_real);

y\_estimate = euler\_method(x\_real, y0,y);

error = abs(y - y\_estimate)./abs(y);

fprintf("Absolute relative error at 0.5 is %.8f", error(end));

fprintf("\n");

a = 5;

It is the part of euler’s method in part2main.m. Here, I am finding the actual function by using dsolve. Then, by using y0, I found the unknown coefficient. Then, I calculate the result of the real values. Then, I sen the X array to the euler\_method function.

function Iout = euler\_method(X,Y, real)

est\_func = @(x,y)(x + y);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

y\_est(i) = y\_est(i-1) + (temp)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

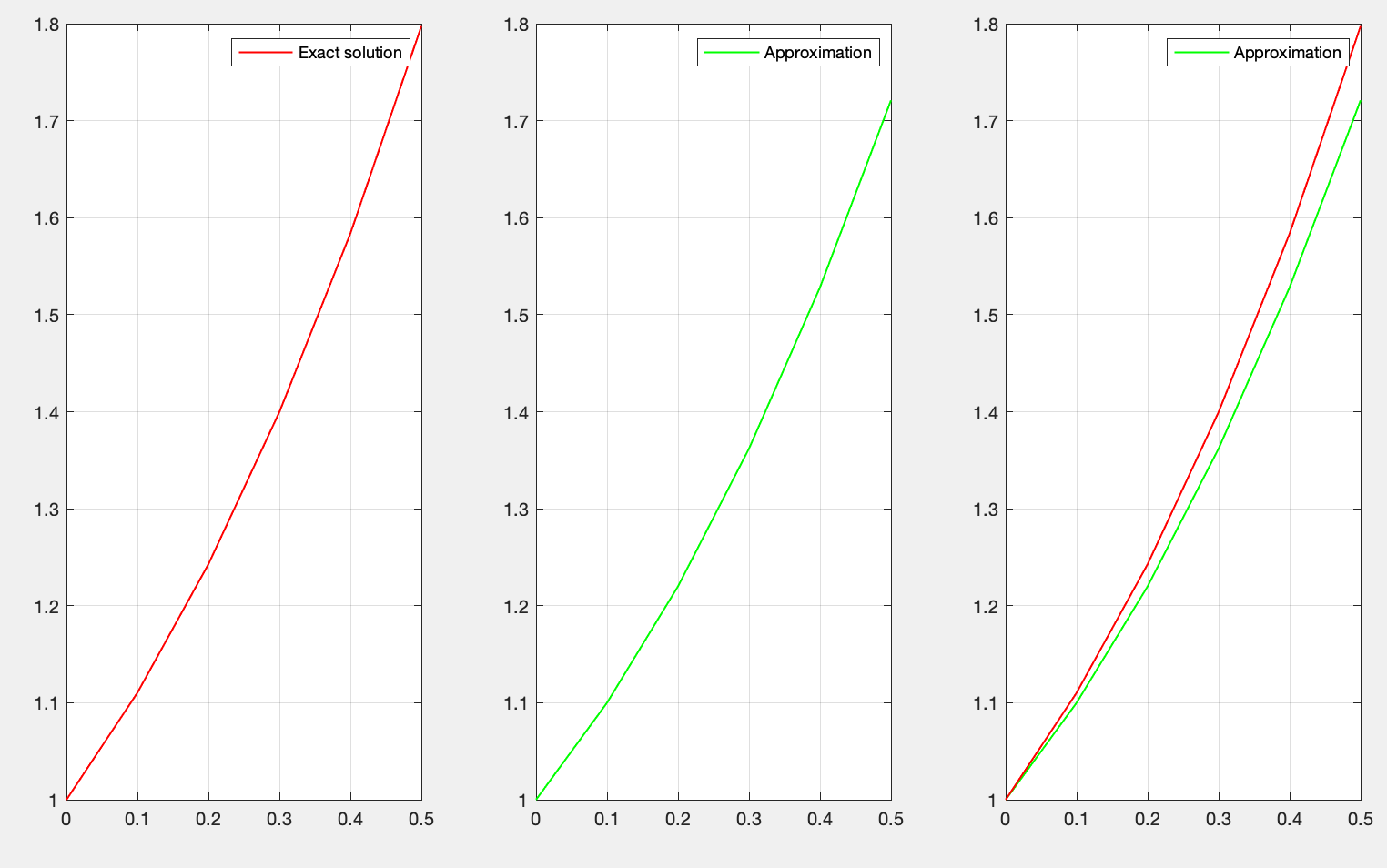
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

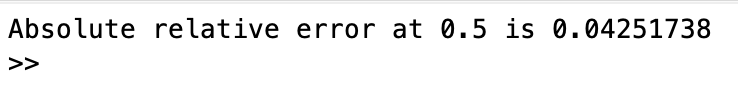
grid on

end

Here, I followed the basic knowledge we have learned in the lecture to implement the euler method.



When we sketch the result of the approximation and exact solution which we found by using dsolve, we get the following graphs. Now, let’s look at the absolute relative error at 0.5.



Now, let’s continue with the midpoint method.

%% Midpoint Method

x\_real = 0:0.1:0.5;

y0 = 1;

syms y(t) a

eqn = diff(y,t) == t+y;

S = dsolve(eqn);

real\_func = @(x)(2\*exp(x) - x - 1);

y = real\_func(x\_real);

y\_estimate = midpoint\_method(x\_real, y0,y);

error = abs(y - y\_estimate)./abs(y);

fprintf("Absolute relative error at 0.5 is %.8f", error(end));

fprintf("\n");

Here, I am finding the actual function by using dsolve. Then, by using y0, I found the unknown coefficient. Then, I calculate the result of the real values. Then, I sen the X array to the midpoint\_method function

function Iout = midpoint\_method(X, Y, real)

est\_func = @(x,y)(x + y);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

y\_est(i) = y\_est(i-1) + est\_func(X(i-1) + h/2, y\_est(i-1) + h/2\*temp)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

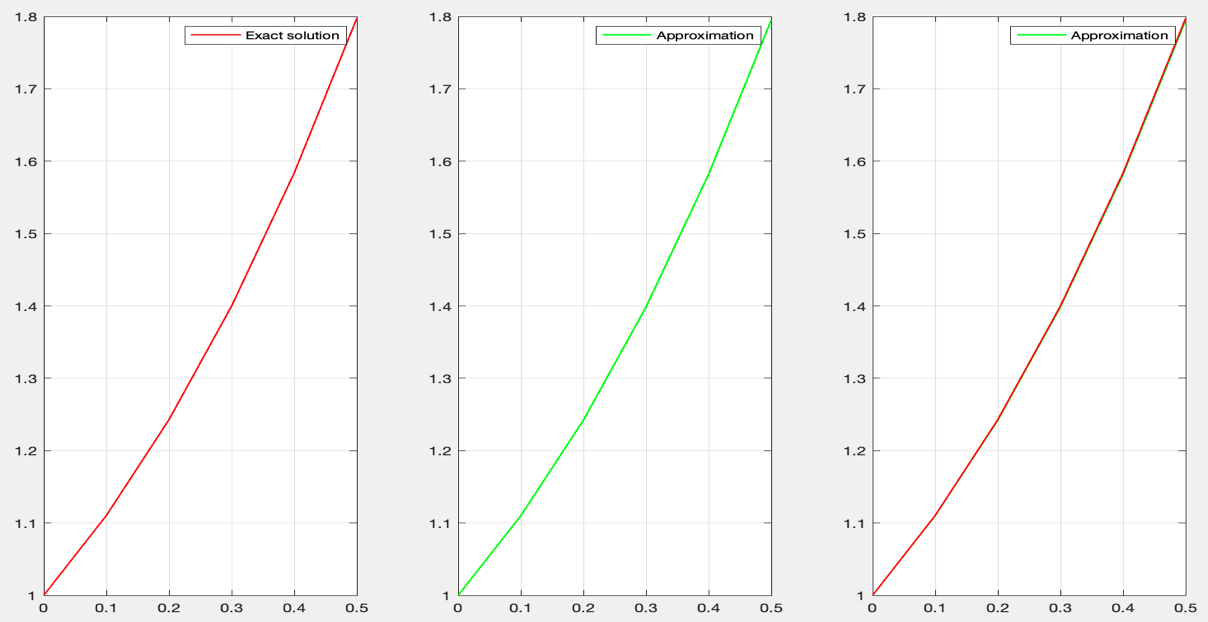
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

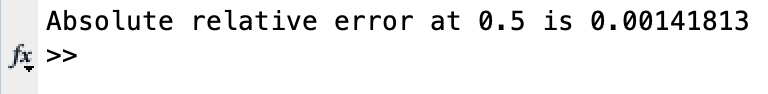
grid on

end

It is the same idea that we have discussed during the lecture and idea can be found in lecture notes.



As you can the found result and the actual result are lies on top of each other.



The relative error is less than euler’s method.

Now, let’s continue with Runge-Kutta.

%% Runge-Kutta Method of order four

x\_real = 0:0.1:0.5;

y0 = 1;

syms y(t) a

eqn = diff(y,t) == t+y;

S = dsolve(eqn);

real\_func = @(x)(2\*exp(x) - x - 1);

y = real\_func(x\_real);

y\_estimate = kuttamethod4(x\_real, y0, y);

error = abs(y - y\_estimate)./abs(y);

fprintf("Absolute relative error at 0.5 is %.8f", error(end));

fprintf("\n");

Here, I am finding the actual function by using dsolve. Then, by using y0, I found the unknown coefficient. Then, I calculate the result of the real values. Then, I send the X array to the kuttamethod4 function.

function Iout = kuttamethod4(X,Y, real)

est\_func = @(x,y)(x + y);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

temp2 = est\_func(X(i-1)+ h/2, y\_est(i-1)+ h/2\*temp);

temp3 = est\_func(X(i-1)+ h/2, y\_est(i-1)+ h/2\*temp2);

temp4 = est\_func(X(i-1)+ h, y\_est(i-1)+ h\*temp3);

y\_est(i) = y\_est(i-1) + 1/6\*(temp + temp4 + 2\*temp2 + 2\*temp3)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

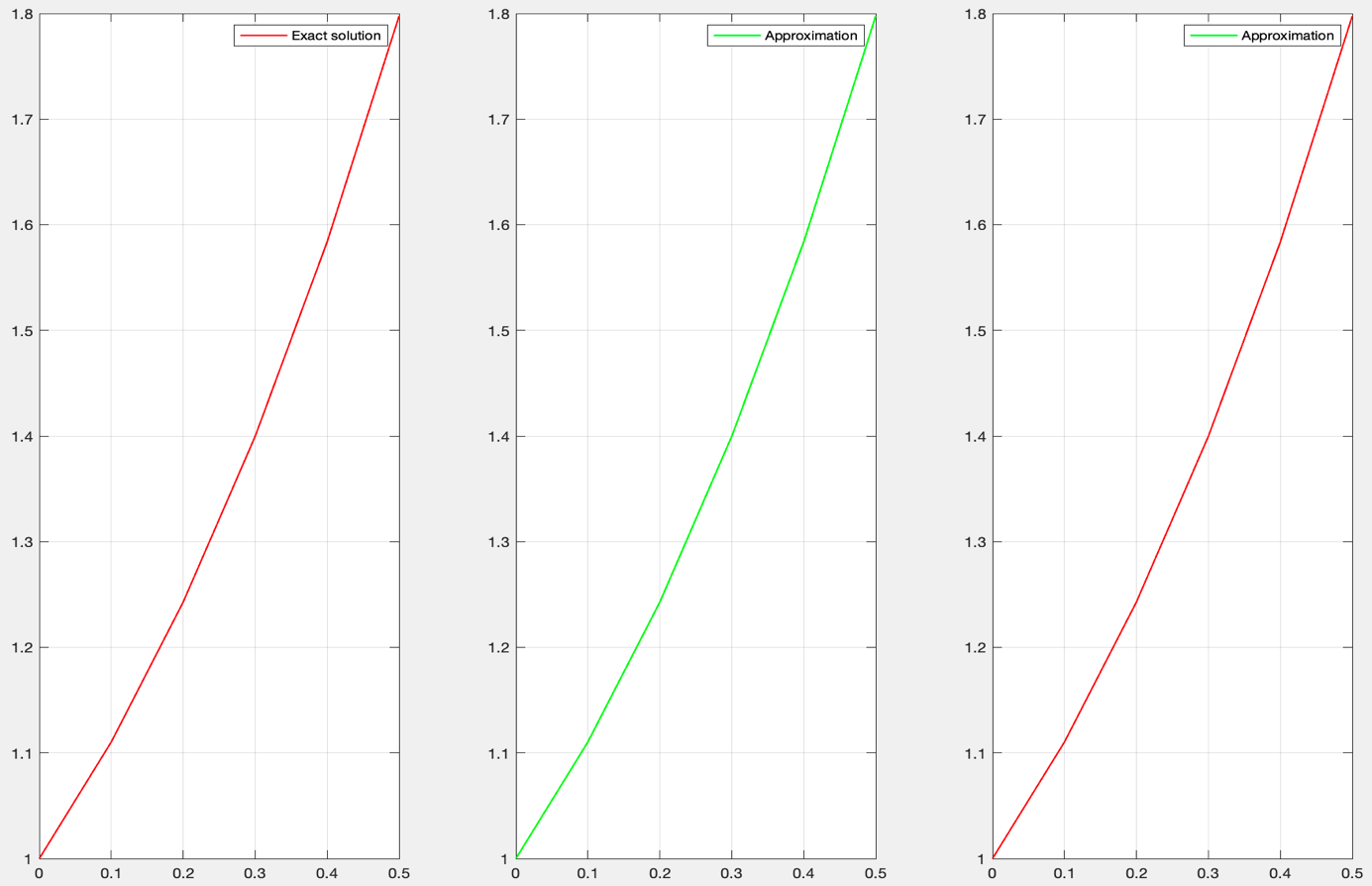
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

grid on

end

Here, we have temp, temp2, temp3 and temp4 as k1, k2, k3 and k4.



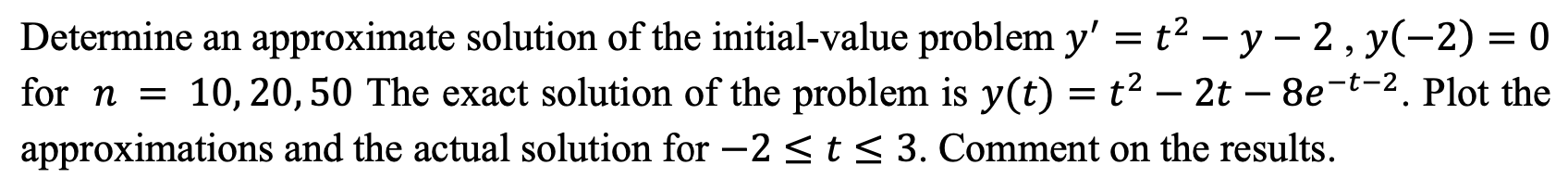
As you can the found result and the actual result are lies on top of each other.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

As you can see, when we use Runge-Kutta 4th order method error decreased significantly.

**Question 5**



**Solution**

The exact solution of the problem is now given. By playing with the size of the step, we will evaluate the performance of each method which are Euler’s method, Midpoint method, Runge-Kutta method. Now let’s start with Euler’s method.

%% Euler's Method

x\_real = -2:0.1:3;

y0 = 0;

real\_func = @(x)(x.^(2)-2\*x - 8\*exp(-x-2));

y = real\_func(x\_real);

y\_estimate = q5euler\_method(x\_real, y0, y);

error = abs(y - y\_estimate);

Here, I am playing with step size at x\_real part. When h = 0.5 we have 10 iterations, h = 0.25 we have 20 iterations and when h = 0.1 we have 50 iterations.

function Iout = q5euler\_method(X,Y, real)

est\_func = @(x,y)(x.^2 - y - 2);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

y\_est(i) = y\_est(i-1) + (temp)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

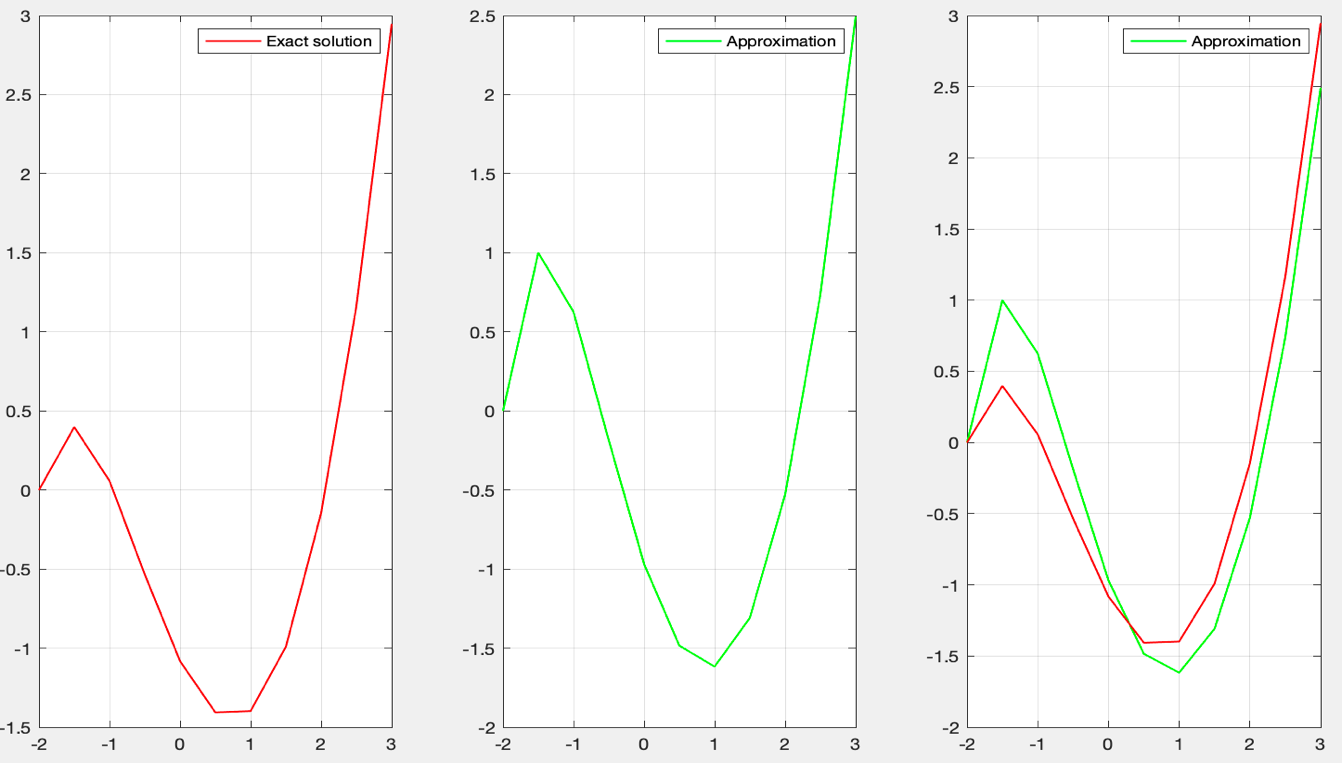
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

grid on

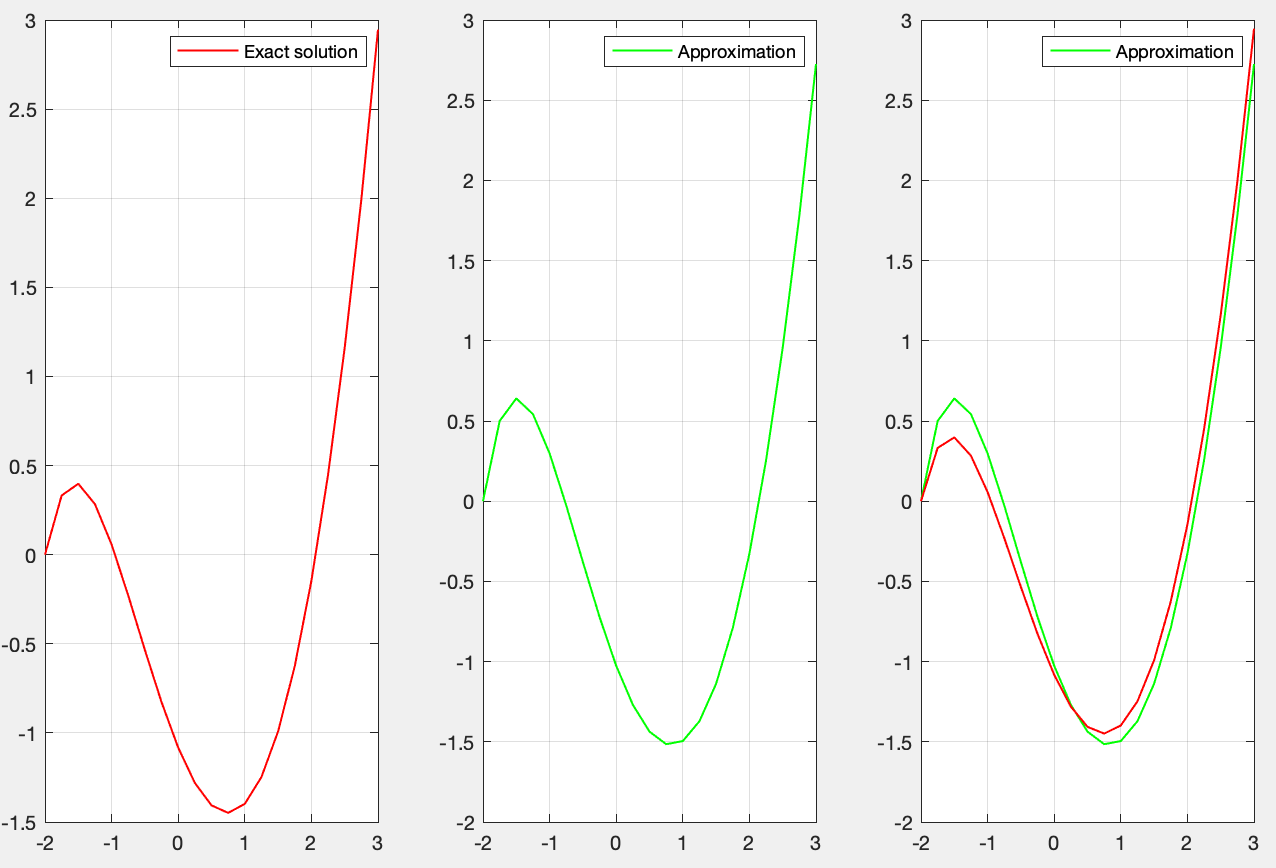
end

According to the selected h size, the value of h in the function changes. Now, let’s look at the plottings. First, let’s start with n = 10.

****

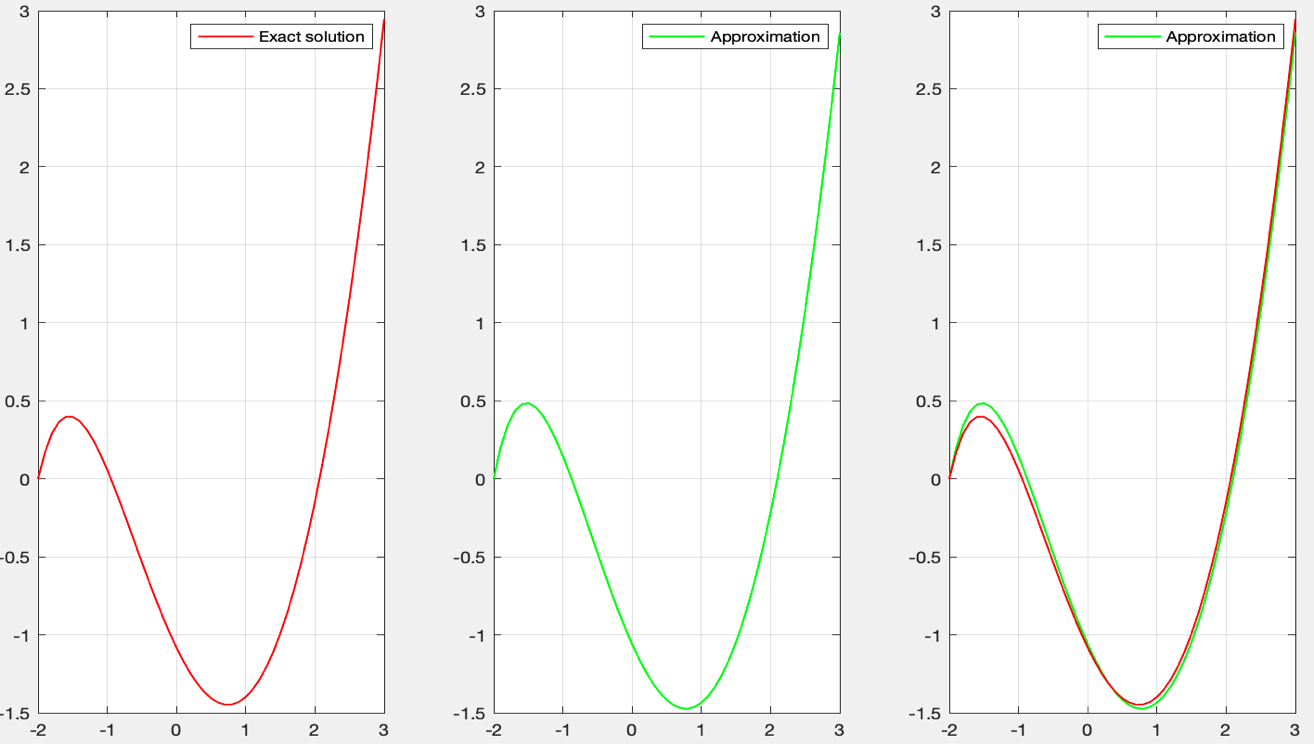
As you can see the results are close but there is a visible error between actual solution and approximation.

Now, let’s look at when n = 20

****

The result in these figures show that when we increased the iteration number and decreased the step size, we get a closer approximation to the actual values.

Now, let’s look at when n = 50



We get a better result than the previous one. We can say that when we decrease the step size we get better results for this case.

Now, let’s continue with midpoint method.

%% Midpoint Method

x\_real = -2:0.1:3;

y0 = 0;

real\_func = @(x)(x.^(2)-2\*x - 8\*exp(-x-2));

y = real\_func(x\_real);

y\_estimate = q5midpoint\_method(x\_real, y0,y);

error = abs(y - y\_estimate);

The part in the q5main.m is similar in each section.

function Iout = q5midpoint\_method(X, Y, real)

est\_func = @(x,y)(x.^2 - y - 2);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

y\_est(i) = y\_est(i-1) + est\_func(X(i-1) + h/2, y\_est(i-1) + h/2\*temp)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

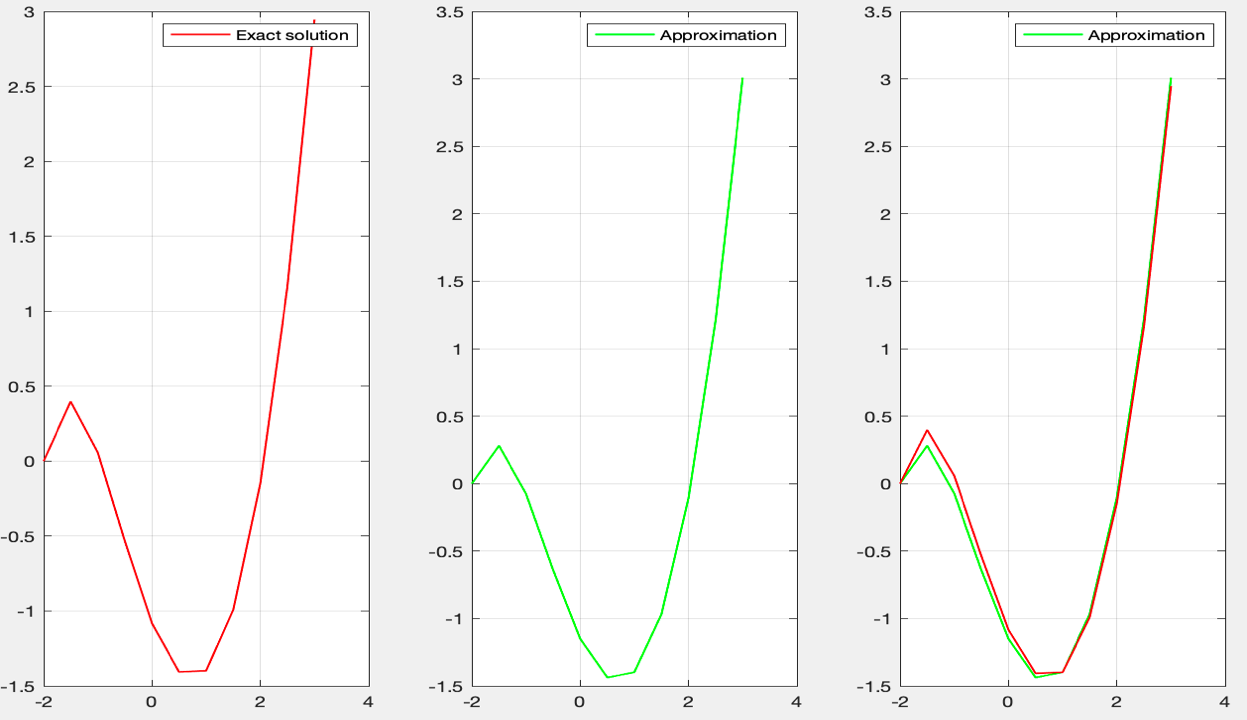
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

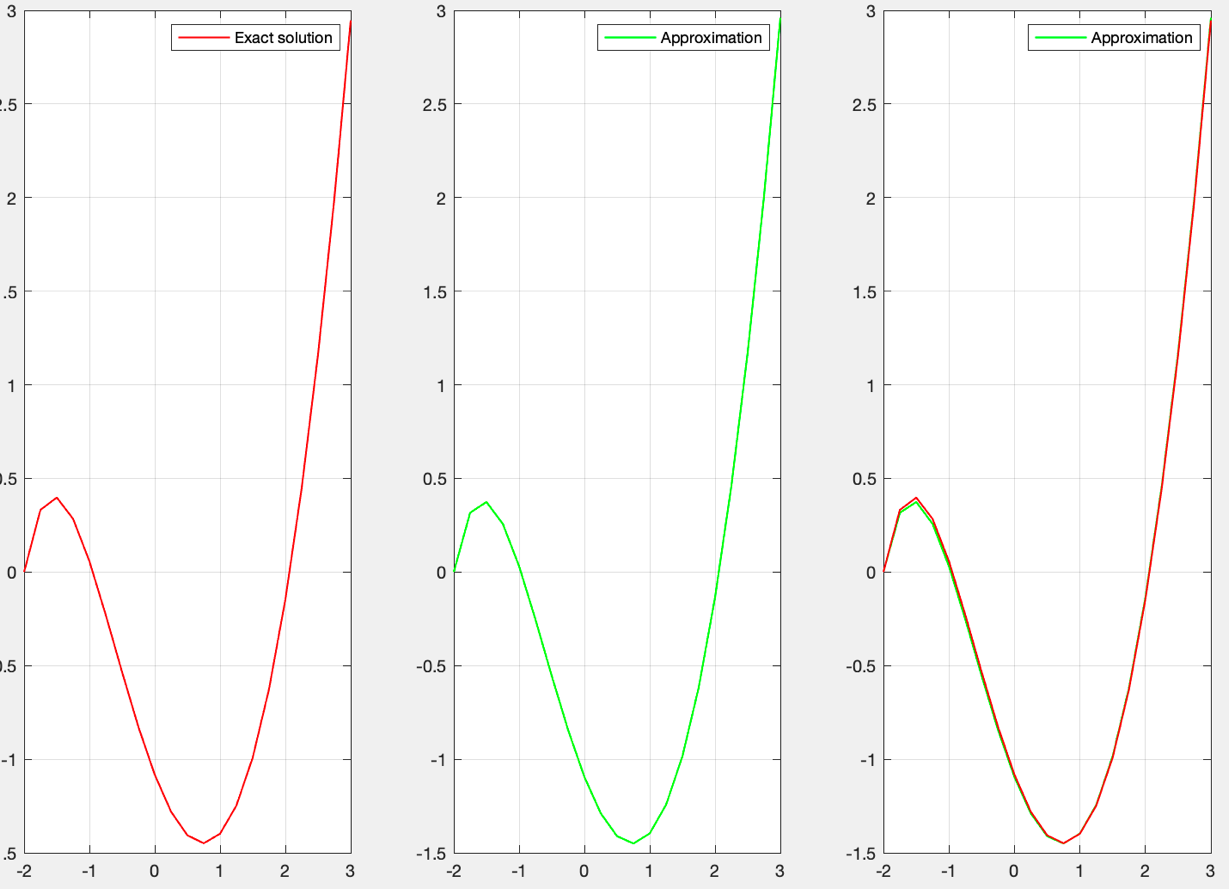
grid on

end

The function is actually same like we did in question 4. Now let’s look at the plots of each iteration numbers consecutively. Firstly, let’s continue with n = 10

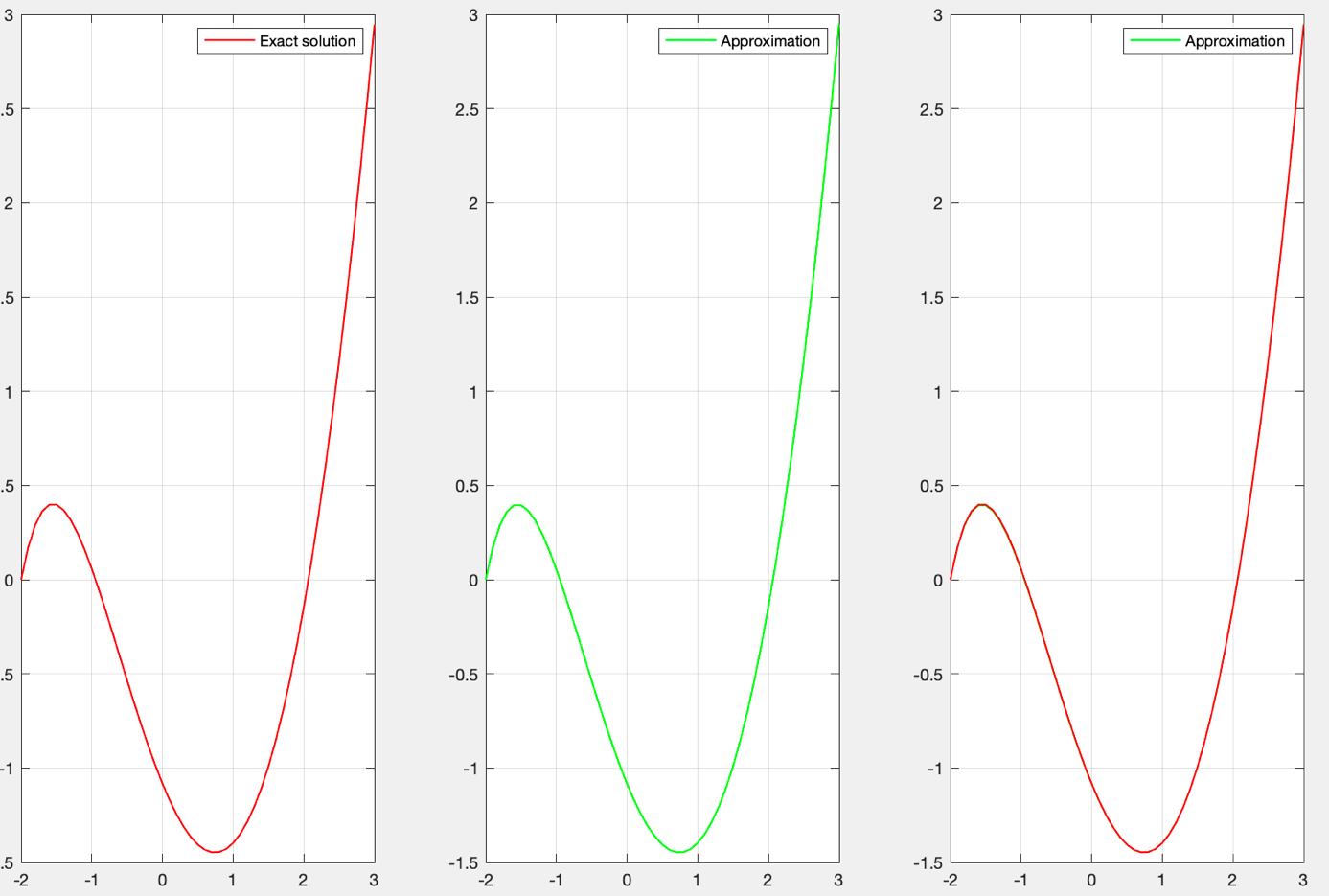
****

When we look at the figures and compare it with the euler method, we get a better result in midpoint method. Now, let’s continue with n = 20



When we look at the figure, the lines started lie on top of each other more than the previous one.

Now let’s look at when n = 50



When we look at the, we can not visibly differ approximation line from the exact solution. Thus, we get a very accurate approximation when we increase the n so that decrease the step size.

Now, let’s continue with Runge-Kutta.

%% Runge-Kutta Method of order four

x\_real = -2:0.1:3;

y0 = 0;

real\_func = @(x)(x.^(2)-2\*x - 8\*exp(-x-2));

y = real\_func(x\_real);

y\_estimate = q5kuttamethod4(x\_real, y0,y);

error = abs(y - y\_estimate);

error = error';

Now, let’s look at the function

function Iout = q5kuttamethod4(X,Y, real)

est\_func = @(x,y)(x.^2 - y - 2);

h = 0.1;

y\_est(1) = Y;

for i = 2:1:size(X,2)

temp = est\_func(X(i-1), y\_est(i-1));

temp2 = est\_func(X(i-1)+ h/2, y\_est(i-1)+ h/2\*temp);

temp3 = est\_func(X(i-1)+ h/2, y\_est(i-1)+ h/2\*temp2);

temp4 = est\_func(X(i-1)+ h, y\_est(i-1)+ h\*temp3);

y\_est(i) = y\_est(i-1) + 1/6\*(temp + temp4 + 2\*temp2 + 2\*temp3)\*h;

end

Iout = y\_est;

figure

subplot(1,3,1)

plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Exact solution");

grid on

subplot(1,3,2)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

legend("Approximation");

grid on

subplot(1,3,3)

plot(X,y\_est, 'LineWidth', 1, 'color', 'green')

grid on

hold on; axis on;

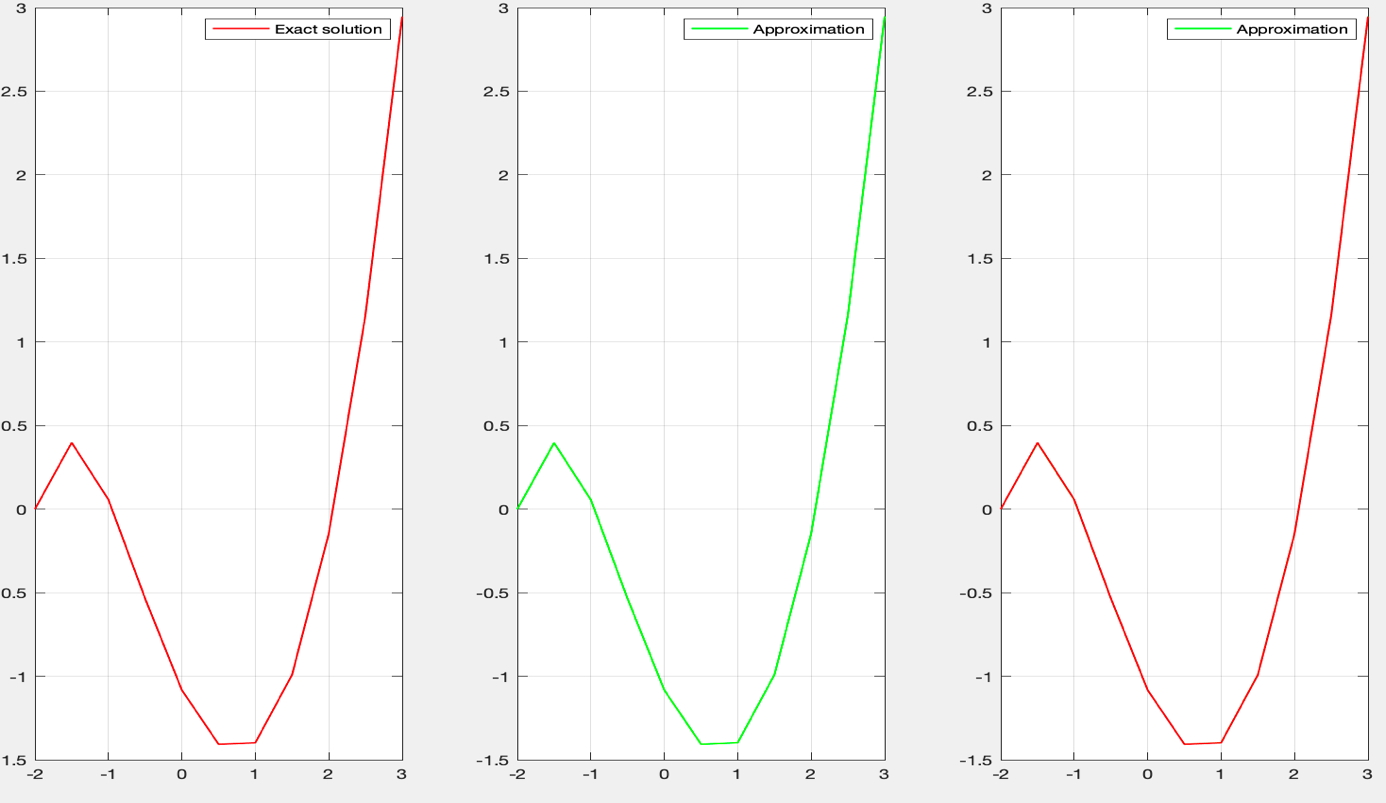
plot(X,real, 'LineWidth', 1, 'color', 'red')

legend("Approximation");

grid on

end

Now, it is time for plot process. Initially, we are going to show when n = 10.



As you can the approximation and exact solution lies on top of each other.

Now, since when n = 20 and n = 50 will be also like each other, let’s check the error values.

Now, let’s continue with n = 20. I will show the absolute error.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

As you can see the errors are very small.

Now, let’s look at when n = 50.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

As you can see when we decrease the step size, the error reduces significantly.