Ezz Aboulezz

260677463

ECSE 443 Assignment 4

Q1.a)

In part a we compute the integral using the mid-point rule. The value computed for the integral was 4.355199572285354 with number of steps = 7.

This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 2.739167815057897e-05

relative error = 6.289459294525523e-06

Q1.b)

In part b we compute the integral using the trapezoidal rule. The value computed for the integral was 4.355199572285354 with number of steps = 8.

This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 2.739335005674803e-05

relative error = 6.289843184323615e-06

Q1.c)

In part c we compute the integral using the Simpson’s rule. The value computed for the integral was 4.355199572285354 with number of steps = 13.

This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 4.259497382808064e-05

relative error = 9.780319138184335e-06

Q2.a)

In part a we compute the integral using the mid-point rule. The value computed for the integral was 7.905347558749129e+02 with number of steps = 371.

This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 9.584108014450976e-04

relative error = 1.212356107542929e-06

Q2.b)

In part b we compute the integral using the trapezoidal rule. The value computed for the integral was 7.905365739128109e+02 with number of steps = 554.

This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 8.596270965881558e-04

relative error = 1.087398179555833e-06

Q2.c)

In part c we compute the integral using the Simpson’s rule. The value computed for the integral was 7.905364603760823e+02 with number of steps = 14.

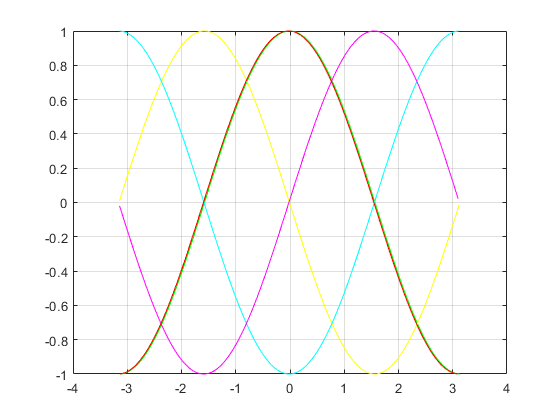
This value was compared with the MATLAB function value. Absolute and Relative errors we found to be:

absolute error = 7.460903680112096e-04

relative error = 9.437781931020496e-07

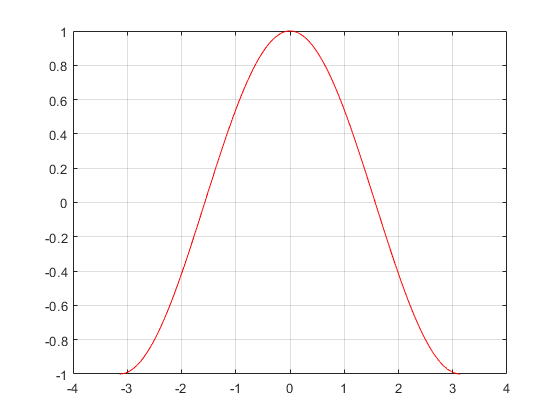
Q3.a)

In part a we compute the fifth backward difference and plot all 5 of them***.***



Q3.b)

In part b we compute the forward difference representation and plot it.



Q4.

f’(0) = 7, f’(2) = -7, f’(4) = -43, f’’(0) = -5.

Appendix: Code used for all questions

Q1.a)

%function, limits, true value, and # of steps

format long;

f = @(x) log(5-4\*cos(x));

LL = 0;

UL = pi;

true = integral(f, LL, UL);

n = 0;

%relative error and midpoint sum

absoluteError = 0;

relativeError = 0;

errorPower = 0;

midSum = 0;

while ~(errorPower == -6)

midSum = 0;

n = n + 1;

LL = 0;

%delta x is difference between bounds divided n

deltaX = (UL - LL)/n;

%compute the integral for n using midpoint rule

for i=1:n

%compute x and y

x = LL + i\*(deltaX) - (deltaX/2);

y = log(5-4\*cos(x));

%multiply y and delta x, and store in sum

midSum = midSum + y\*deltaX;

end

%error calculation

absoluteError = abs(midSum - true);

relativeError = absoluteError/true;

errorPower = ceil(log10(relativeError)-1);

end

disp(n);

disp(midSum);

disp(relativeError);

Q1.b)

%function, limits, true value, and # of steps

f = @(x) log(5-4\*cos(x));

LL = 0;

UL = pi;

true = integral(f, LL, UL);

n = 0;

%relative error and trapezoidal sum

absoluteError = 0;

relativeError = 0;

errorPower = 0;

trapSum = 0;

while ~(errorPower == -6)

trapSum = 0;

n = n + 1;

LL = 0;

%delta x is difference between bounds divided n-1

deltaX = (UL - LL)/(n-1);

for i=1:n

%compute x and y

x = LL + ((i-1)\*(deltaX));

y = log(5-4\*cos(x));

if (i == 0)||(i == n)

coeff = 0.5;

else

coeff = 1;

end

%store in sum

trapSum = trapSum + (coeff\*deltaX\*y);

end

%error calculation

absoluteError = abs(trapSum - true);

relativeError = absoluteError/true;

errorPower = ceil(log10(relativeError)-1);

end

disp(n);

disp(trapSum);

disp(relativeError);

Q1.c)

%function, limits, true value, and # of steps

f = @(x) log(5-4\*cos(x));

LL = 0;

UL = pi;

true = integral(f, LL, UL);

n = 0;

%relative error and Simpson sum

absoluteError = 0;

relativeError = 0;

errorPower = 0;

simSum = 0;

while ~(errorPower == -6)

simSum = 0;

n = n + 1;

LL = 0;

%delta x is difference between bounds divided n-1

deltaX = (UL - LL)/(n-1);

%compute the integral for n using Simpson's rule

for i=1:n

%compute x and y

x = LL + ((i-1)\*(deltaX));

y = log(5-4\*cos(x));

%coefficient we multiply y with based on parity

if (mod(i,2) == 0)

coeff = 4;

else

coeff = 2;

end

if (i == 0)||(i == n)

coeff = 1;

end

%store in sum

simSum = simSum + (coeff\*deltaX\*(1/3)\*y);

end

%error calculation

absoluteError = abs(simSum - true);

relativeError = absoluteError/true;

errorPower = ceil(log10(relativeError)-1);

end

disp(n);

disp(simSum);

disp(relativeError);

Q2.a)

%function, limits, true value, and # of steps

syms x y;

f = x^2 + y;

LLy = x;

ULy = 2\*x^3;

LLx = 2;

ULx = 3;

trueY = int(f, y, LLy, ULy);

trueX = int(trueY, x, 2, 3);

trueX = double(trueX);

n = 370;

%absolute error

absoluteError = 0;

errorPower = 0;

while (errorPower > -4)

n = n + 1;

LLy = x;

%dyIntegral stores the first integration

dyIntegral = 0;

deltaX = (ULy - LLy)/n;

%compute integral if upper and lower limits not equal

while(LLy ~= ULy)

%mid point rule

dyIntegral = dyIntegral + subs(f, y, LLy + (deltaX/2))\*(deltaX);

LLy = LLy + deltaX;

end

LLx = 2;

deltaX = (ULx - LLx)/n;

%dxIntegral stores the final value

dxIntegral = 0;

%compute the integral if lower limit of x is less than upper limit

while(LLx < ULx)

%mid point rule

dxIntegral = dxIntegral + subs(dyIntegral, x, LLx + (deltaX/2))\*(deltaX);

LLx = LLx + deltaX;

end

%error calculation

dxIntegral = double(dxIntegral);

absoluteError = abs(dxIntegral - trueX);

errorPower = ceil(log10(absoluteError)-1);

end

disp(dxIntegral);

disp(n);

disp(absoluteError);

Q2.b)

%function, limits, true value, and # of steps

syms x y;

f = x^2 + y;

LLy = x;

ULy = 2\*x^3;

LLx = 2;

ULx = 3;

trueY = int(f, y, LLy, ULy);

trueX = int(trueY, x, 2, 3);

trueX = double(trueX);

n = 550;

%absolute error

absoluteError = 0;

errorPower = 0;

while (errorPower > -4)

n = n + 1;

LLy = x;

%dyIntegral stores the first integration

dyIntegral = 0;

deltaX = (ULy - LLy)/n;

%compute integral if upper and lower limits are not equal

while(LLy ~= ULy)

%trapezoidal rule

y1 = subs(f, y, LLy);

y2 = subs(f, y, LLy + deltaX);

dyIntegral = dyIntegral + (y1 + y2)\*(deltaX/2);

LLy = LLy + deltaX;

end

LLx = 2;

%dxIntegral stores the final value

dxIntegral = 0;

deltaX = (ULx - LLx)/n;

%compute the integral if lower limit of x is less than upper limit

while(LLx < ULx)

%trapezoidal rule

y3 = subs(dyIntegral, x, LLx);

y4 = subs(dyIntegral, x, LLx + deltaX);

dxIntegral = dxIntegral + (y3 + y4)\*(deltaX/2);

LLx = LLx + deltaX;

end

%error calculation

dxIntegral = double(dxIntegral);

absoluteError = abs(dxIntegral - trueX);

errorPower = ceil(log10(absoluteError)-1);

end

disp(dxIntegral);

disp(n);

disp(absoluteError);

Q2.c)

%function, limits, true value, and # of steps

syms x y;

f = x^2 + y;

LLy = x;

ULy = 2\*x^3;

LLx = 2;

ULx = 3;

trueY = int(f, y, LLy, ULy);

trueX = int(trueY, x, 2, 3);

trueX = double(trueX);

n = 0;

%absolute error

absoluteError = 0;

errorPower = 0;

while (errorPower > -4)

n = n + 1;

LLy = x;

t1 = subs(f, y, LLy);

t2 = subs(f, y, ULy);

%dyIntegral stores the first integration

dyIntegral = t1 + t2;

deltaX = (ULy - LLy)/n;

%Simpson rule

for i=1:2:n-1

dyIntegral = dyIntegral + 4\*subs(f, y, LLy + i\*deltaX);

end

for j=2:2:n-2

dyIntegral = dyIntegral + 2\*subs(f, y, LLy + j\*deltaX);

end

dyIntegral = (deltaX/3)\*dyIntegral;

LLx = 2;

t3 = subs(dyIntegral, x, LLx);

t4 = subs(dyIntegral, x, ULx);

%dxIntegral stores the final value

dxIntegral = t3 + t4;

deltaX = (ULx - LLx)/n;

%Simpson rule

for i=1:2:n-1

dxIntegral = dxIntegral + 4\*subs(dyIntegral, x, LLx + i\*deltaX);

end

for j=2:2:n-2

dxIntegral = dxIntegral + 2\*subs(dyIntegral, x, LLx + j\*deltaX);

end

dxIntegral = (deltaX/3)\*dxIntegral;

%error calculation

dxIntegral = double(dxIntegral);

absoluteError = abs(dxIntegral - trueX);

errorPower = ceil(log10(absoluteError)-1);

end

disp(dxIntegral);

disp(n);

disp(absoluteError);

Q3.

%step size, range, function

h = 0.01;

x = -pi:h:pi;

f = sin(x);

%compute derivatives using finite difference

y1 = diff(f)/h;

y2 = diff(y1)/h;

y3 = diff(y2)/h;

y4 = diff(y3)/h;

y5 = diff(y4)/h;

plot(x(:,1:length(y1)),y1,'g', x(:,1:length(y2)),y2,'y', x(:,1:length(y3)),y3,'c', x(:,1:length(y4)),y4,'m', x(:,1:length(y5)),y5,'r')

grid

%b

%step size, range, function

h = 0.001;

x = -pi:h:pi;

f = sin(x);

sum = zeros(1, length(f)-3);

%compute forward difference representation

for i=1:(length(f)-3)

sum(i) = ((-11/6)\*f(i) + (3)\*(f(i+1)) - (1.5)\*f(i+2) + (1/3)\*f(i+3))/h;

end

figure;

plot(x(:,1:length(sum)), sum, 'r');

grid

Q4.

x = [0, 1, 2, 3, 4];

f = [30, 33, 28, 12, -22];

%step size

h = 1;

f0FirstOrder = (-f(3) + 4\*f(2) -3\*f(1))/2\*h;

f2FirstOrder = (-f(5) + 4\*f(4) -3\*f(3))/2\*h;

f4FirstOrder = (3\*f(5) - 4\*f(4) + f(3))/(2\*h);

f0SecondOrder = (-f(4) + 4\*f(3) - 5\*f(2) + 2\*f(1))/(h^2);

disp(f0FirstOrder);

disp(f2FirstOrder);

disp(f4FirstOrder);

disp(f0SecondOrder);