**NC ASSIGNMENT 02**

**NAME: KUMAIL HAIDER**

**ROLL NO: 20K-0455**

**SECTION: BSCS (4B)**

**Coding # 04: NUMERICAL DIFFERENTIATION**

**PART a)**

**3 POINT MID-POINT**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <iostream>

#include <iomanip>

#include "../true.h"

using namespace std;

double f(double x);

struct Coord { double x, y = 0; };

struct Coords {

Coord \*c;

Coord operator [] (int i) { return c[i]; }

const Coord operator [] (int i) const { return c[i]; }

Coords(const initializer\_list<double> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = { \*(init.begin() + i), f(\*(init.begin() + i))}; }

Coords(const initializer\_list<Coord> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = \*(init.begin() + i); }

};

double f(double x) {

return

/\* f(x): \*/ x \* exp(x);

;

}

double g(double x) {

return (x + 1) \* exp(x);

}

int main() {

const Coords c = {

// x values (or {x, y} pairs):

{1.8, 10.889365}, {1.9, 12.703199}, {2.0, 14.778112},

{2.1, 17.148957}, {2.2, 19.85503}

};

double h =

/\* x to approximate: \*/ 0.2

;

double x =

/\* x to approximate: \*/ 2.0

;

cout << "true value: " << g(x) << endl;

cout << "formula: " << endl;

cout << "f'(x) = (1/2h) \* [f(x+h) - f(x-h)]" << endl;

printf("f'(%.6g) = (1/2(%.6g)) \* [f(%.6g+%.6g) - f(%.6g-%.6g)]\n", x, h, x, h, x, h);

printf("f'(%.6g) = %.6g \* [f(%.6g) - f(%.6g)]\n", x, (1/(2\*h)), x + h, x - h);

printf("f'(%.6g) = %.6g \* [%.6g - %.6g]\n", x, (1/(2\*h)), f(x + h), f(x - h));

printf("f'(%.6g) = %.6g", x, (1/(2 \* h)) \* (f(x + h) - f(x - h)));

return 0;

}

**3 POINT ENDPOINT**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <iostream>

#include <iomanip>

#include "../true.h"

using namespace std;

double f(double x);

struct Coord { double x, y = 0; };

struct Coords {

Coord \*c;

Coord operator [] (int i) { return c[i]; }

const Coord operator [] (int i) const { return c[i]; }

Coords(const initializer\_list<double> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = { \*(init.begin() + i), f(\*(init.begin() + i))}; }

Coords(const initializer\_list<Coord> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = \*(init.begin() + i); }

};

double f(double x) {

return

/\* f(x): \*/ x \* exp(x);

;

}

double g(double x) {

return (x + 1) \* exp(x);

}

int main() {

const Coords c = {

// x values (or {x, y} pairs):

{1.8, 10.889365}, {1.9, 12.703199}, {2.0, 14.778112},

{2.1, 17.148957}, {2.2, 19.85503}

};

double h =

/\* x to approximate: \*/ 0.1

;

double x =

/\* x to approximate: \*/ 2.0

;

cout << "true value: " << g(x) << endl;

cout << "formula: " << endl;

cout << "f'(x) = (1/2h) \* [-3f(x) + 4f(x+h) - f(x+2h)]" << endl;

printf("f'(%.6g) = (1/2(%.6g)) \* [-3f(%.6g) + 4f(%.6g + %.6g) - f(%.6g+2(%.6g))]\n", x, h, x, x, h, x, h);

printf("f'(%.6g) = %.6g \* [-3f(%.6g) + 4f(%.6g) - f(%.6g)]\n", x, (1/(2\*h)), x, x + h, x + 2\*h);

printf("f'(%.6g) = %.6g \* [-3(%.6g) + 4(%.6g) - %.6g]\n", x, (1/(2\*h)), f(x), f(x + h), f(x + 2\*h));

printf("f'(%.6g) = %.6g", x, (1/(2 \* h)) \* (-3 \* f(x) + 4 \* f(x + h) - f(x + 2 \* h)));

return 0;

**PART b)**

**5 POINT MID-POINT**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <iostream>

#include <iomanip>

#include "../true.h"

using namespace std;

double f(double x);

struct Coord { double x, y = 0; };

struct Coords {

Coord \*c;

Coord operator [] (int i) { return c[i]; }

const Coord operator [] (int i) const { return c[i]; }

Coords(const initializer\_list<double> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = { \*(init.begin() + i), f(\*(init.begin() + i))}; }

Coords(const initializer\_list<Coord> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = \*(init.begin() + i); }

};

double f(double x) {

return

/\* f(x): \*/ x \* exp(x);

;

}

double g(double x) {

return (x + 1) \* exp(x);

}

int main() {

const Coords c = {

// x values (or {x, y} pairs):

{1.8, 10.889365}, {1.9, 12.703199}, {2.0, 14.778112},

{2.1, 17.148957}, {2.2, 19.85503}

};

double h =

/\* x to approximate: \*/ 0.1

;

double x =

/\* x to approximate: \*/ 2.0

;

cout << "true value: " << g(x) << endl;

cout << "formula: " << endl;

cout << "f'(x) = (1/12h) \* [f(x-2h) - 8f(x-h) + 8f(x + h) -f(x+2h)]" << endl;

printf("f'(%.6g) = (1/12(%.6g)) \* [f(%.6g-2(%.6g)) - 8f(%.6g-%.6g) + 8f(%.6g+%.6g) - f(%.6g+2(%.6g))]\n", x, h, x, h, x, h, x, h, x, h);

printf("f'(%.6g) = %.6g \* [f(%.6g) - 8f(%.6g) + 8f(%.6g) - f(%.6g)]\n", x, (1/(12\*h)), x - 2 \* h, x - h, x + h, x + 2 \* h);

printf("f'(%.6g) = %.6g \* [%.6g - 8(%.6g) + 8(%.6g) - %.6g]\n", x, (1/(12\*h)), f(x - 2 \* h), f(x - h), f(x + h), f(x + 2 \* h));

printf("f'(%.6g) = %.6g", x, (1/(12 \* h)) \* (f(x - 2 \* h) - (8 \* f(x - h)) + (8 \* f(x + h)) - f(x + 2 \* h)));

return 0;

}

**5 POINT ENDPOINT**

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <iostream>

#include <iomanip>

#include "../true.h"

using namespace std;

double f(double x);

struct Coord { double x, y = 0; };

struct Coords {

Coord \*c;

Coord operator [] (int i) { return c[i]; }

const Coord operator [] (int i) const { return c[i]; }

Coords(const initializer\_list<double> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = { \*(init.begin() + i), f(\*(init.begin() + i))}; }

Coords(const initializer\_list<Coord> &init)

: c(init.size() ? new Coord[init.size()] : nullptr)

{ for (int i = 0; i < init.size(); i++) \*(c + i) = \*(init.begin() + i); }

};

double f(double x) {

return

/\* f(x): \*/ x \* exp(x);

;

}

double g(double x) {

return (x + 1) \* exp(x);

}

int main() {

const Coords c = {

// x values (or {x, y} pairs):

{1.8, 10.889365}, {1.9, 12.703199}, {2.0, 14.778112},

{2.1, 17.148957}, {2.2, 19.85503}

};

double h =

/\* x to approximate: \*/ 0.1

;

double x =

/\* x to approximate: \*/ 1.8

;

cout << "true value: " << g(x) << endl;

cout << "formula: " << endl;

cout << "f'(x) = (1/12h) \* [-25f(x) + 48(x+h) - 36f(x+2h) + 16f(x+3h) - 3f(x+4h)]" << endl;

printf("f'(%.6g) = (1/12(%.6g)) \* [-25f(%.6g) + 48f(%.6g+%.6g) - 36f(%.6g+2(%.6g)) + 16f(%.6g+3(%.6g)) - 3f(%.6g+4(%.6g))]\n", x, h, x, x, h, x, h, x, h, x, h);

printf("f'(%.6g) = %.6g \* [-25f(%.6g) + 48f(%.6g) - 36f(%.6g) + 16f(%.6g) - 3f(%.6g)]\n", x, (1/(12\*h)), x, x + h, x + 2 \* h, x + 3 \* h, x + 4 \* h);

printf("f'(%.6g) = %.6g \* [-25(%.6g) + 48(%.6g) - 36(%.6g) + 16(%.6g) - 3(%.6g)]\n", x, (1/(12\*h)), f(x), f(x + h), f(x + 2 \* h), f(x + 3 \* h), f(x + 4 \* h));

printf("f'(%.6g) = %.6g", x, (1/(12 \* h)) \* ((-25 \* f(x)) + (48 \* f(x + h)) - (36 \* f(x + 2 \* h)) + (16 \* f(x + 3\* h)) - (3 \* f(x + 4 \* h))));

return 0;

}

**Coding # 05 : NUMERICAL INTEGRATION**

**PART a)**

**Open and Closed Newton Cotes**

import numpy as np

from scipy.integrate import newton\_cotes

def f(x):

return np.sin(x)

a = 0

b = np.pi

exact = 2

for N in [2, 4, 6, 8, 10]:

x = np.linspace(a, b, N + 1)

an, B = newton\_cotes(N, 1)

dx = (b - a) / N

quad = dx \* np.sum(an \* f(x))

error = abs(quad - exact)

print('{:2d} {:10.9f} {:.5e}'.format(N, quad, error))

**PART b)**

**TRAPEZOIDAL FORMULA**

import numpy as np

print(Trapezoidal Implementation \n)

import matplotlib.pyplot as plt

try:

#data set test 1

# xArr = np.array([1.,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8])

# yArr = np.array([1.543,1.669,1.811,1.971,2.151,2.352,2.577,2.828,3.107])

#data set test 2

# xArr = np.array([2.1,2.4,2.7,3.0,3.3,3.6])

# yArr = np.array([3.2,2.7,2.9,3.5,4.1,5.2])

# xArr = np.array([1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2, 3.4])

# yArr = np.array([6.050, 7.389, 9.025, 11.023, 13.464, 16.445, 20.086, 24.533, 29.964])

# Input

xValues = input(Enter Input x Values , Separated: \n)

xArr = np.array([])

for I in xValues.split(,):

xArr = np.append(xArr, i)

xArr = xArr.astype(float)

print(x: , xArr)

yValues = input(Enter Input y Values , Separated: \n)

yArr = np.array([])

for I in yValues.split(,):

yArr = np.append(yArr, i)

yArr = yArr.astype(float)

print(f(x): , yArr)

h = xArr[1] xArr[0]

temp\_mid\_add = 0

# print(xArr.size)

diff\_div = h/2

for I in range(1, xArr.size-1):

temp\_mid\_add += yArr[i]

if xArr.size-2 == i:

lastIndex = yArr[i+1]

# print(lastIndex)

area = diff\_div\*(yArr[0] + 2\*(temp\_mid\_add) + lastIndex)

print(area, sq.m)

plt.title(Trapezoidal, fontsize=24)

plt.xlabel(x, fontsize=14)

plt.ylabel(F(x), fontsize=14)

plt.plot(yArr, xArr)

plt.show()

except:

print(Error Deducted)

**SIMPSON’S FORMULA**

import numpy as np

import matplotlib.pyplot as plt

print(Simpson 1/3 Implementation \n)

try:

#data set test 1

# xArr = np.array([1.,1.1,1.2,1.3,1.4])

# yArr = np.array([1.543,1.669,1.811,1.971,2.151])

#data set test 2

# xArr = np.array([2.1,2.4,2.7,3.0,3.3,3.6])

# yArr = np.array([3.2,2.7,2.9,3.5,4.1,5.2])

# xArr = np.array([0.0, 1.0, 2.0, 3.0, 4., 5., 6.])

# yArr = np.array([1.,2.,5.,10.,17.,26.,37.])

# inputs

xValues = input(Enter Input x Values , Separated: \n)

xArr = np.array([])

for I in xValues.split(,):

xArr = np.append(xArr, i)

xArr = xArr.astype(float)

print(x: , xArr)

yValues = input(Enter Input y Values , Separated: \n)

yArr = np.array([])

for I in yValues.split(,):

yArr = np.append(yArr, i)

yArr = yArr.astype(float)

print(f(x): , yArr)

h = xArr[1] xArr[0]

temp\_mid\_add = 0

four\_mul = 0

two\_mul = 0

diff\_div = h/3

if xArr.size == yArr.size:

if xArr.size % 2:

for I in range(1, xArr.size-1):

#temp\_mid\_add += yArr[i]

if I % 2:

four\_mul += 4\*yArr[i]

# print(four\_mul)

else:

two\_mul += 2\*yArr[i]

# print(two\_mul)

if xArr.size-2 == i:

lastIndex = yArr[i+1]

# print(lastIndex)

area = diff\_div\*((yArr[0] + lastIndex) + four\_mul + two\_mul)

print(area, sq.m)

# plt.title(Simpson, fontsize=24)

# plt.xlabel(x, fontsize=14)

# plt.ylabel(F(x), fontsize=14)

# plt.plot(yArr, xArr)

# plt.show()

else:

print(Dataset should be odd)

print(xArr.size)

else:

print(Values of x or f(x) is not Correct)

except:

print(Error Deducted)

**MID-POINT FORMULA**

import math

#enter your function here

def dydx(x, y):

return (x \* math.exp(3\*x) 2 \* y)

def midpoint(x0, y0, x, h):

n = (int)((x x0)/h)

y = y0

x = x0

for I in range(0, n ):

x = x0 + 0.5 \* h

y1 = y + 0.5 \* h \* dydx(x0, y)

y = y + h \* dydx(x, y1)

x0 = x0 + h

print(iteration ,i+1)

print(x(,i+0.5,) = ,x,, x(,i+1,) = ,x0,, y(,i+0.5,)= ,round(y1,6),, y(,i+1,)= ,round(y,6))

print( )

#Enter arguments

x0 = 0

y = 0

x = 1

h = 0.5

midpoint(x0, y, x, h)

**CODING # 06**

**DIVIDED DIFFERENCE TABLE FIVE VALUES AND CENTRAL DIFFERENCE TABLE FIVE VALUES**

# Function to find the product term

def proterm(i, value, x):

pro = 1;

for j in range(i):

pro = pro \* (value - x[j]);

return pro;

# Function for calculating divided difference table

def dividedDiffTable(x, y, n):

for i in range(1, n):

for j in range(n - i):

y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) /

(x[j] - x[i + j]));

return y;

# Function for applying Newton's divided difference formula

def applyFormula(value, x, y, n):

sum = y[0][0];

for i in range(1, n):

sum = sum + (proterm(i, value, x) \* y[0][i]);

return sum;

# Function for displaying divided

# difference table

def printDiffTable(y, n):

for i in range(n):

for j in range(n - i):

print(round(y[i][j], 4), "\t",

end = " ");

print("");

# number of inputs given

n = 4;

y = [[0 for i in range(10)]

for j in range(10)];

x = [ 5, 6, 9, 11 ];

# y[][] is used for divided difference

# table where y[][0] is used for input

y[0][0] = 12;

y[1][0] = 13;

y[2][0] = 14;

y[3][0] = 16;

# calculating divided difference table

y=dividedDiffTable(x, y, n);

# displaying divided difference table

printDiffTable(y, n);

# value to be interpolated

value = 7;

# printing the value

print("\nValue at", value, "is",

round(applyFormula(value, x, y, n), 2))

def central(order: int, f: 'Callable', x: float, h: float, exact: bool = False) -> float:

return sum([(-1)\*\*(k) \* sci.comb(order, k, exact) \* f(x - (order/2 - k)\*h) for k in range(order+1)]) / h\*\*order