**Three point & Five point Formulas:**

import math

import sympy as sp

def three\_midpoint\_differentiate(f, x0, h):

valf\_f = sp.lambdify(x,f)

return (valf\_f(x0+h)-valf\_f(x0-h))/(2\*h)

def three\_endpoint\_differentiate(f, x0, h):

valf\_f = sp.lambdify(x,f)

return (-3\*valf\_f(x0)+4\*valf\_f(x0+h)-valf\_f(x0+2\*h))/(2\*h)

def five\_midpoint\_differentiate(f, x0, h):

f = sp.lambdify(x,f)

return (f(x0-2\*h) -8\*f(x0-h)+8\*f(x0+h)-f(x0+2\*h))/(12\*h)

def five\_endpoint\_differentiate(f, x0, h):

f = sp.lambdify(x,f)

return (-25\*f(x0)+48\*f(x0+h)-36\*f(x0+2\*h)+16\*f(x0+3\*h)-3\*f(x0+4\*h))/(12\*h)

x = sp.Symbol('x',real=True)

f = x\*sp.exp(x)

# we have table as follows

# x f(x)

# 1.8 10.889365

# 1.9 12.701399

# 2.0 14.778112

# 2.1 17.148957

# 2.2 19.8550130

ans = three\_endpoint\_differentiate(f, 1.8, 0.1)

print("The value of 1.8 using three point is: %0.4f"%ans)

ans = three\_endpoint\_differentiate(f, 2.2, 0.1)

print("The value of 2.2 using three point is: %0.4f"%ans)

ans = three\_midpoint\_differentiate(f, 2.0, 0.1)

print("The value of 2.0 using three point is: %0.4f"%ans)

ans = five\_endpoint\_differentiate(f, 1.8, 0.1)

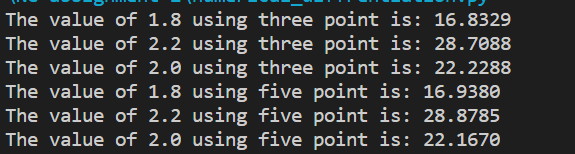
print("The value of 1.8 using five point is: %0.4f"%ans)

ans = five\_endpoint\_differentiate(f, 2.2, 0.1)

print("The value of 2.2 using five point is: %0.4f"%ans)

ans = five\_midpoint\_differentiate(f, 2.0, 0.1)

print("The value of 2.0 using five point is: %0.4f"%ans)



**Numerical Integration:**

**Composite Simpson:**

def composite\_simpson\_integral(f, a, b, n):

if n%2 == 0:

h = (b-a)/n

XI\_0 = f(a) + f(b)

XI\_1 = 0

XI\_2 = 0

for i in range(n-1):

X = a + i\*h

if i%2==0:

XI\_2 = XI\_2+f(X)

else:

XI\_1 = XI\_1+f(X)

XI = (XI\_0 + 2\*XI\_2 +4\*XI\_1)

return XI \* (h / 3)

else:

return 'n should be even positive integer'

**Composite Trapezoidal:**

def composite\_trapezoid\_integral(f, a, b, n):

if n%2 == 0:

x = a

h = (b-a)/n

T = f(a) + f(b)

for i in range(1, n):

x += h

T += 2 \* f(x)

return T \* (h/2)

else:

return 'n should be even positive integer'

Newton Divided Difference:

def proterm(i, value, x):

pro = 1;

for j in range(i):

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

return pro;

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;

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;

def printDiffTable(y, n):

for i in range(n):

for j in range(n - i):

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

end = " ");

print("");

n = 4;

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

for j in range(10)];

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

y[0][0] = 12;

y[1][0] = 13;

y[2][0] = 14;

y[3][0] = 16;

y=dividedDiffTable(x, y, n);

printDiffTable(y, n);

value = 7;

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

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

**Newton Cotes:**

from scipy.integrate import newton\_cotes

import numpy as np

def f(x):

return np.sin(x)

a = 0

b = (np.pi)/4

exact = 1

for N in [1, 2, 3, 4, 5]:

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

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

dx = (b - a) / N

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

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