**Lab Task 11**

**Numerical Computing Lab(105127)**

**Ali Afzal(63961)**

Code:

# Python3 program for implementing

# Newton divided difference formula

terms = []

# Function to find the product term

def proterm(i, value, x):

pro = 1;

for j in range(i):

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

#Karname start here

terms.append("(x-"+str(x[j])+")")

#Karname end here

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("");

# Driver Code

# 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))

print(y[0][1])

#Karname start here

eq = "f(x) = "

eq = eq+str(y[0][0])

eq = eq + " + "

le = len(terms)

brk = 0

j = 1

for i in range(le):

eq = eq + terms[i]

if i == brk:

eq = eq + " \* "

eq = eq + str(y[0][j])

eq = eq + " + "

j = j + 1

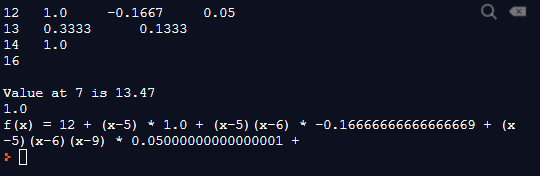
brk = j+i

print(eq)

#Karname end here

# This code is contributed by mits

Output:



Working:

I have edited the code at two places and the start and end of both the sections is:

def proterm(i, value, x):

pro = 1;

for j in range(i):

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

#Karname start here

terms.append("(x-"+str(x[j])+")")

#Karname end here

return pro;

Here I have appended the factors I got in pro term function to a list called terms. From there at the end of code I made a logic which helps me print the output eq the way I want it to be printed. As everything is done prettymuch dynamically I the same working will get any set of roots to the required stage.

#Karname start here

eq = "f(x) = "

eq = eq+str(y[0][0])

eq = eq + " + "

le = len(terms)

brk = 0

j = 1

for i in range(le):

eq = eq + terms[i]

if i == brk:

eq = eq + " \* "

eq = eq + str(y[0][j])

eq = eq + " + "

j = j + 1

brk = j+i

print(eq)

#Karname end here

Problems:

THe problem I am facing from here is that it is hard for me to evaluate the x term as we do in a math equation without any library but my output does provide the equation it is that it needs to be simplified more to get the “actual” equation.