20 May 2022

NUMERICAL COMPUTING (CS325)

Group Members:

- Mohammad Basil Ali Khan (20K-0477)
- Ali Jodat (20K-0155)
- Abdul Ahad Shaikh (20K-0319)
- Mohammad Umer (20K-0225)

Course Instructor:
Sir Jamil Usmani

NUMERICAL COMPUTING (CS325)
PROJECT
LAB – 1

Group Members:

- Mohammad Basil Ali Khan (20K-0477)
- Ali Jodat (20K-0155)
- Abdul Ahad Shaikh (20K-0319)
- Mohammad Umer (20K-0225)

Project Title:

LAB 1: Solution of Non Linear Equation in one Variable f(x) = 0

Aim:

To understand the fundamental concepts of scientific programming using python.

Description:

We selected three methods of Lab1.

1. Bisection Method
2. Regular Falsi Method
3. Secant Method

First we have studied the algorithm of then we have written the programming of that method.

IDE and Programming Language:

We have chosen python programming language and IDE we are using is Visual Studio Code.

Library Used:

We have imported 3 libraries:

1. sympy library for to get equation solution on particular intervals and can initialize symbols.
2. tabulate library to generated table on each iteration.
3. array library to save each iteration values to use in next iteration.

Implementation and Code Snippets:

✓ Bisection Method:

Formula:

$$c = \frac{a+b}{2}\ ; where\ a\ and\ b\ are\ intervals\ [a, b]$$

Algorithm:

Step 1: Find two points, say a and b such that a < b and f(a)* f(b) < 0

Step 2: Find the midpoint of a and b, say "c"

Step 3: c is the root of the given function if f(c) = 0; else follow the next step

Step 4: Divide the interval [a, b] – If f(c)*f(a) <0, there exist a root between t and a
else if f(c) *f (b) < 0, there exist a root between t and b

Step 5: Repeat above three steps until f(c) = 0.

Code Snippets:

```
Bisection_method.py
from sympy import *
from tabulate import tabulate
from array import *

def getFunc():
    func = input("Enter Function: ")
    return func

def getIntervals():
    print("Enter Intervals: ")
    a = input("a = ")
    b = input("b = ")
    return a, b

def f(eq, num):
    x = float(num)
    return eval(eq)

Iteration = 0
Func = getFunc()
a, b = getIntervals()
col_names = ["n", "a", "b", "c", "f(c)"]
arr = [[]]
root_check = false
c = (float(a)+float(b))/2
if(f(Func, a) * f(Func, b) < 0):
    arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
    while(f(Func, c) != 0):
        c = (float(a)+float(b))/2
        if f(Func, a) * f(Func, c) < 0:
            b = c
        else:
            a = c
        Iteration += 1
        arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
        ans = f(Func, c)
    print(tabulate(arr, headers=col_names, tablefmt="fancy_grid"))
    print("Root: ", format(c, ".5f"))
    print("Number of Iterations: ", Iteration)
    print("f(c) = ", ans)
else:
    print("Root is not availble in given Intervals. ")
```

Output:

Input: $x^3 + 4x^2 - 10 = 0$; $[1, 2]$

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ftc\Desktop\NC Project> & C:/Python/Python310/python.exe "c:/Users/ftc/Desktop/NC Project/Bisection_method.py"
Enter Function: x**3 + 4*x**2 - 10
Enter Intervals:
a = 1
b = 2
```

n	a	b	c	f(c)
0	1	2	1.5	2.375
1	1	1.5	1.5	2.375
2	1.25	1.5	1.25	-1.79688
3	1.25	1.375	1.375	0.162109
4	1.3125	1.375	1.3125	-0.848389
5	1.34375	1.375	1.34375	-0.350983
6	1.35938	1.375	1.35938	-0.0964088
7	1.35938	1.36719	1.36719	0.0323558
8	1.36328	1.36719	1.36328	-0.03215
9	1.36328	1.36523	1.36523	7.20248e-05
10	1.36426	1.36523	1.36426	-0.0160467
11	1.36475	1.36523	1.36475	-0.00798926

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

35	1.36523	1.36523	1.36523	-2.3789e-11
36	1.36523	1.36523	1.36523	2.16513e-10
37	1.36523	1.36523	1.36523	9.6362e-11
38	1.36523	1.36523	1.36523	3.62874e-11
39	1.36523	1.36523	1.36523	6.24922e-12
40	1.36523	1.36523	1.36523	-8.76987e-12
41	1.36523	1.36523	1.36523	-1.26121e-12
42	1.36523	1.36523	1.36523	2.49401e-12
43	1.36523	1.36523	1.36523	6.16396e-13
44	1.36523	1.36523	1.36523	-3.21521e-13
45	1.36523	1.36523	1.36523	1.47438e-13
46	1.36523	1.36523	1.36523	-8.70415e-14
47	1.36523	1.36523	1.36523	3.01981e-14
48	1.36523	1.36523	1.36523	-2.84217e-14
49	1.36523	1.36523	1.36523	0

```
Root:  1.36523
Number of Iterations:  49
f(c) =  0.0
PS C:\Users\ftc\Desktop\NC Project>
```

✓ Secant Method:

Formula:

$$\frac{A(f(b)) - B(f(A))}{f(B) - f(A)} \text{ ; } where\ A\ and\ B\ are\ intervals\ [A, B]$$

Algorithm:

Given an equation f(c) = 0
Let the initial guesses be a and b
Do

$$c = \frac{a(f(b)) - b(f(a))}{f(b) - f(a)}$$

while (f(c) not equals 0)

Code Snippets:

```python
from sympy import *
from tabulate import tabulate
from array import *

def getFunc():
    func = input("Enter Function: ")
    return func

def getIntervals():
    print("Enter Intervals: ")
    a = input("a = ")
    b = input("b = ")
    return a, b

def f(eq, num):
    x = float(num)
    return eval(eq)

Iteration = 0
Func = getFunc()
a, b = getIntervals()
col_names = ["n", "a", "b", "c", "f(c)"]
arr = [[]]
root_check = false
if f(Func, a) * f(Func, b) < 0:
    c = float((float(a)*f(Func, b) - float(b)*f(Func, a))/((f(Func, b) - f(Func, a))))
    arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
    while f(Func, c) != 0:
        a = b
        b = c
        c = float((float(a)*f(Func, b) - float(b)*f(Func, a))/((f(Func, b) - f(Func, a))))
        Iteration +=1
        arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
    print(tabulate(arr, headers=col_names, tablefmt="fancy_grid"))
    print("Root: ", c)
    print("Number of Iterations: ", Iteration)
else:
    print("Root is not availble in given Intervals. ")
```

Output:

Input : $x - \cos x$; $[0, 1]$

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                    Python

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ftc\Desktop\NC Project> & C:/Python/Python310/python.exe "c:/Users/ftc/Desktop/NC Project/Secant_method.py"
Enter Function: x - cos(x)
Enter Intervals:
a = 0
b = 1
```

n	a	b	c	f(c)
0	0	1	0.685073	-0.0892993
1	1	0.685073	0.736299	-0.00466004
2	0.685073	0.736299	0.739119	5.7286e-05
3	0.736299	0.739119	0.739085	-3.52926e-08
4	0.739119	0.739085	0.739085	-2.66787e-13
5	0.739085	0.739085	0.739085	0

```
Root:  0.7390851332151607
Number of Iterations:  5
PS C:\Users\ftc\Desktop\NC Project>
```

✓ Regular Falsi Method:

Formula:

$$\frac{A(f(b)) - B(f(A))}{f(B) - f(A)} \; ; where\ A\ and\ B\ are\ intervals\ [A, B]$$

Algorithm:

Step 1: Find two points, say a and b such that a < b and f(a)* f(b) < 0

Step 2: $c = \frac{a(f(b))-b(f(a))}{f(b)-f(a)}$

Step 3: c is the root of the given function if f(c) = 0; else follow the next step

Step 4: Divide the interval [a, b] – If f(c)*f(a) <0, there exist a root between t and a
else if f(c) *f (b) < 0, there exist a root between t and b

Step 5: Repeat above three steps until f(c) = 0.

Code Snippets:

```python
from sympy import *
from tabulate import tabulate
from array import *

def getFunc():
    func = input("Enter Function : ")
    return func

def getIntervls():
    print("Enter Intervals : ")
    a = input("a = ")
    b = input("b = ")
    return a,b

def f(eq, num):
    x = float(num)
    return eval(eq)

Iteration = 0
Func = getFunc()
a,b = getIntervls()
col_names = ["n", "a", "b", "c", "f(c)"]
arr = [[]]

c = float(float(a) * f(Func,b) - float(b) * f(Func,a))/(f(Func,b) - f(Func,a))

if(f(Func,a) * f(Func,b) < 0):
    while f(Func , c) != 0:
        arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
        c = float(float(a) * f(Func,b) - float(b) * f(Func,a))/(f(Func,b) - f(Func,a))

        if(f(Func, a) * f(Func, c) < 0):
            b = c
        else:
            a = c

        Iteration += 1
        ans = f(Func, c)

    arr.insert(Iteration, [Iteration, a, b, c, f(Func, c)])
    print(tabulate(arr, headers=col_names, tablefmt="fancy_grid"))
    print("Root : ", c)
    print("Number of Iterations performed : ", Iteration)
    print("f(c) = ", ans)
else:
    print("Root is not available in given Intervals. ")
```

Input :

$$x^3 + 4x^2 - 10 = 0 ; [1, 2]$$

Output:

```
PS C:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course related material\Numerical_Computing\Lab_1\Regular_falsi.py"
Enter Function : x**3 + 4*x**2 -10
Enter Intervals :
a = 1
b = 2
```

n	a	b	c	f(c)
0	1	2	1.26316	-1.60227
1	1.26316	2	1.26316	-1.60227
2	1.33883	2	1.33883	-0.430365
3	1.35855	2	1.35855	-0.110009
4	1.36355	2	1.36355	-0.0277621
5	1.36481	2	1.36481	-0.00698342
6	1.36512	2	1.36512	-0.00175521
7	1.3652	2	1.3652	-0.000441063
8	1.36522	2	1.36522	-0.000110828
21	1.36523	2	1.36523	-1.7657e-12
22	1.36523	2	1.36523	-4.47642e-13
23	1.36523	2	1.36523	-1.13687e-13
24	1.36523	2	1.36523	-2.84217e-14
25	1.36523	2	1.36523	-7.10543e-15
26	1.36523	2	1.36523	-3.55271e-15
27	1.36523	2	1.36523	0

```
Root :  1.3652300134140969
Number of Iterations performed :  27
f(c) =  0.0
PS C:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course related material\Numerical_Computing\Lab_1>
```

NUMERICAL COMPUTING (CS325)
PROJECT
LAB – 2

Group Members:

- Mohammad Basil Ali Khan (20K-0477)
- Ali Jodat (20K-0155)
- Abdul Ahad Shaikh (20K-0319)
- Mohammad Umer (20K-0225)

Project Title:

LAB 2: Interpolation and Polynomial Approximation

Aim:

To understand the fundamental concepts of scientific programming using python.

Description:

We selected three methods of Lab2.

1. Lag grange Interpolation
2. Newton Divided Difference
3. Newton Forward and Backward

First we have studied the algorithm of then we have written the programming of that method.

IDE and Programming Language:

We have chosen python programming language and IDE we are using is Visual Studio Code.

Library Used:

✓ Used panda library to make data frame

Implementation and Code Snippets:

- ## Lag grange Interpolation:

Formula:

$$f(x) = f_0\partial_0(x) + f_1\partial_1(x) + f_2\partial_2(x) + \cdots + f_N\partial_N(x)$$

Where $\partial_i(x)$ can be written as;

$$\partial_i(x) = \frac{\prod_{i=0}^{N}; i \neq j\ (x - x_j)}{\prod_{i=0}^{N}; i \neq j\ (x_i - x_j)}$$

Algorithm:

Step 1: Read number of data N.

Step 2: Read data Xi and Yi from I = 0 to I = N.

Step 3: Read value of independent variables say x whose corresponding value of dependent say y is to be determined.

Step 4: Initialize: y = 0

Step 5: For i = 0 to N

Set p = 1

For j =0 to N

If i ≠ j then

Calculate product = product * (x - Xj)/(Xi - Xj)

End If

Next j

Calculate y = y + product * Yi

Next i

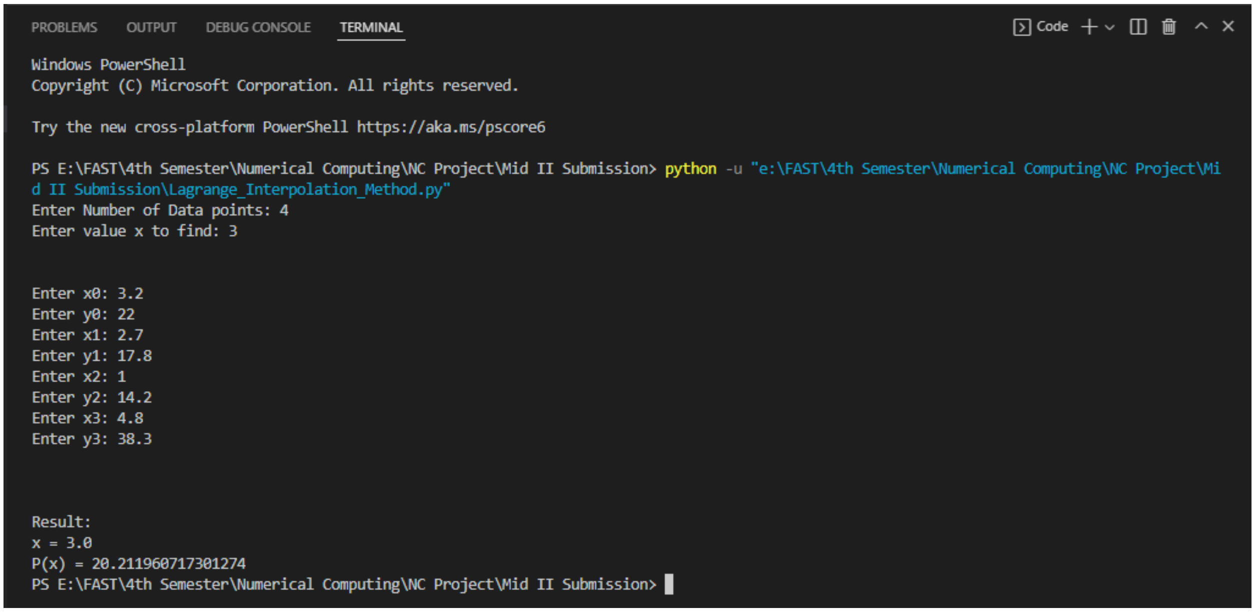
Step 6: Display value of y as interpolated value.

Code Snippets:

Lagrange_Interpolation_Method.py

```python


def EnterNumberOfData():
    data = input("Enter Number of Data points: ")
    return int(data)

def Enter_Value():
    val = input("Enter value x to find: ")
    return float(val)

def getData(num):
    for i in range(num):
        val1 = input("Enter x" + str(i) + ": ")
        val2 = input("Enter y" + str(i) + ": ")
        X_values.append(float(val1))
        Y_values.append(float(val2))
    return

Num_Of_Data = EnterNumberOfData()

Degree = Num_Of_Data - 1

x = Enter_Value()
y = 0

X_values = []
Y_values = []

getData(Num_Of_Data)
print("\n")
for i in range(Degree+1):
    product = 1
    for j in range(Degree+1):
        if j != i:
            product = product * ((x - X_values[j])/(X_values[i]-X_values[j]))
    y = y + Y_values[i] * product

print("\nResult: ")
print("x = " + str(x))
print("P(x) = " + str(y))
```

Output:



- ## Newton Forward and Backward:

Formula:

- **Forward**

$$P_n(x) = \sum_{i=0}^{n} \binom{p}{i} \Delta^i f_i(x) \; ; p = \frac{x - x_0}{h} \; ; h = x_1 - x_0$$

- **Backward**

$$P_n(x) = \sum_{i=0}^{n} \binom{p}{i} \nabla^i f_i(x) \; ; p = \frac{x - x_n}{h} \; ; h = x_1 - x_0$$

Algorithm:

- **Forward**

Step 1: Read number of data (n)

Step 2: Read data points for x and y:

For i = 0 to n-1

Read Xi and Yi,0

Next i

Step 3: Read calculation point where derivative is required (xp)

Step 4: Generate forward difference table

For i = 1 to n-1

For j = 0 to n-1-i

Yj,i = Yj+1,i-1 - Yj,i-1

Next j

Next i

Step 5: Calculate finite difference: h = X1 - X0

Step 6: Set sum = 0 and sign = 1

Step 7: Calculate sum of different terms in formula to find derivatives using Newton's forward difference formula:

```
 For  i = 1 to n-1-index
        term = (Yindex, i)i / i
        sum = sum + sign * term
        sign = -sign
    Next i
```

Step 8: Divide sum by finite difference (h) to get result first_derivative = sum/h

Step 9: Display value of first_derivative

- **Backward**

Step 1: Read number of data (n)

Step 2: Read data points for x and y:

```
For i = 0 to n-1
        Read Xi and Yi,0
    Next i
```

Step 3: Read calculation point where derivative is required (xp)

Step 4: Generate backward difference table

```
    For i = 1 to n-1
        For j = n-1 to i (Step -1)
            Yj,i = Yj,i-1 - Yj-1,i-1
        Next j
    Next i
```

Step 5: Calculate finite difference: h = X1 - X0

Step 6: Set sum = 0

Step 7: Calculate sum of different terms in formula to find derivatives using Newton's backward difference formula:

```
     For  i = 1 to index
         term = (Yindex, i)i / i
         sum = sum + term
     Next i
```

Step 8: Divide sum by finite difference (h) to get result

first_derivative = sum/h

Step 9: Display value of first_derivative

Code Snippets:

- **Forward**

Newton_Forward_Formula_Method.py

```python
import pandas as pd

def Calculating_p(p, n):
    temp = p
    for i in range(1, n):
        temp = temp * (p - i)
    return float(temp)

def Factorial(n):
    Fact = 1
    for i in range(2, n + 1):
        Fact = Fact * i
    return int(Fact)

Num_of_data = int(input("Enter number of data: "))
print("\nEnter values of X: ")
x = []
for i in range(Num_of_data):
    num1 = input("X" + str(i) + " : ")
    x.append(num1)

y = [[0 for i in range(Num_of_data)] for j in range(Num_of_data)]
print("\nEnter f(x) values: ")
for i in range(Num_of_data):
    num2 = input("Y" + str(i) + " : ")
    y[i][0] = float(num2)

for i in range(1, Num_of_data):
    for j in range(Num_of_data - i):
        y[j][i] = float(y[j + 1][i - 1]) - float(y[j][i - 1])

print(pd.DataFrame(y))

value = float(input("\nEnter Value to Interpolate: "))

sum = y[0][0]
p = (float(value) - float(x[0])) / (float(x[1]) - float(x[0]))
for i in range(1,Num_of_data):
    sum = float(sum) + (Calculating_p(p, i) * float(y[0][i])) / Factorial(i)

print("\nValue at ", value, "is", sum)
```

- **Backward**

Newton_Backward_Formula_Method.py

```python
import pandas as pd

def Calculating_p(p, n):
    temp = p
    for i in range(1, n):
        temp = temp * (p + i)
    return float(temp)

def Factorial(n):
    Fact = 1
    for i in range(2, n + 1):
        Fact = Fact * i
    return int(Fact)

Num_of_data = int(input("Enter number of data: "))
print("\nEnter values of X: ")
x = []
for i in range(Num_of_data):
    num1 = input("X" + str(i) + " : ")
    x.append(num1)

y = [[0 for i in range(Num_of_data)] for j in range(Num_of_data)]
print("\nEnter f(x) values: ")
for i in range(Num_of_data):
    num2 = input("Y" + str(i) + " : ")
    y[i][0] = float(num2)

for i in range(1, Num_of_data):
    for j in range(Num_of_data-1, i-1, -1):
        y[j][i] = float(y[j][i - 1]) - float(y[j - 1][i - 1])

print(pd.DataFrame(y))
```

```
value = float(input("\nEnter Value to Interpolate: "))

sum = float(y[Num_of_data-1][0])
p = (float(value) - float(x[Num_of_data-1])) / (float(x[1]) - float(x[0]))
for i in range(1,Num_of_data):
    sum = float(sum) + (Calculating_p(p, i) * float(y[Num_of_data-1][i])) / Factorial(i)

print("\nValue at ", value, "is", sum)
```

Output:

- **Forward**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Mid II Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Mi
d II Submission\Newton_Forward_Formula_Method.py"
Enter number of data: 4

Enter values of X:
X0 : 1.7
X1 : 1.8
X2 : 1.9
X3 : 2

Enter f(x) values:
Y0 : 0.3979849
Y1 : 0.3399864
Y2 : 0.2818186
Y3 : 0.2238908
          0         1         2         3
0  0.397985 -0.057998 -0.000169  0.000409
1  0.339986 -0.058168  0.000240  0.000000
2  0.281819 -0.057928  0.000000  0.000000
3  0.223891  0.000000  0.000000  0.000000

Enter Value to Interpolate: 1.72

Value at  1.72 is 0.38641839040000003
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Mid II Submission>
```

- **Backward**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Mid II Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Mi
d II Submission\Newton_Backward_Formula_Method.py"
Enter number of data: 4

Enter values of X:
X0 : 1.7
X1 : 1.8
X2 : 1.9
X3 : 2

Enter f(x) values:
Y0 : 0.3979849
Y1 : 0.3399864
Y2 : 0.2818186
Y3 : 0.2238908
          0         1         2         3
0  0.397985  0.000000  0.000000  0.000000
1  0.339986 -0.057998  0.000000  0.000000
2  0.281819 -0.058168 -0.000169  0.000000
3  0.223891 -0.057928  0.000240  0.000409

Enter Value to Interpolate: 1.72

Value at  1.72 is 0.3864183903999998
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Mid II Submission>
```

- # **Newton Divided Difference:**

Formula:

$$f(x) = f[x_0] + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x_0, x_1, x_2] + \ldots + (x - x_0)(x - x_1) \ldots (x - x_k-1)\, f[x_0, x_1, \ldots, x_k]$$

Where

$$f[x_0, x_1, \ldots, x_k] = (f[x_1, x_2, \ldots, x_k] - f[x_0, x_1, \ldots, x_{k-1}]) / (x_k - x_0)$$

Code Snippets:

Get Started | Newton_DividedDifference_method.py × | Lagrange_Interpolation_Method.py

C: > Users > Dell > Documents > BS(CS)NUCES > BS(CS) Course related material > Numerical_Computing > Lab_2 > Newton_DividedDifference_method.py > printDif

```python

def EnterNumberOfData():
    data = input("Enter Number of Data points: ")
    return int(data)

def Enter_Value():
    val = input("Enter value x to find: ")
    return float(val)

def getData(num):
    for i in range(num):
        val1 = input("Enter x" + str(i) + ": ")
        val2 = input("Enter y" + str(i) + ": ")
        x.append(float(val1))
        y[i][0] = (float(val2))
    return

def product_x(i, value, x):
    prod = 1;
    for j in range(i):
        prod = prod * (value - x[j]);
    return prod;

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 + (product_x(i, value, x) * y[0][i]);

    return sum;

def printDiffTable(y, n):
    print("\t DIVIDED DIFFERENCE TABLE")
    print("f(x) \t\t", end="")
    for z in range(1,n):
        print(str(z)+"DD  \t\t\t",end="")

    print("\n")
    for i in range(n):
```

```python
    print("\n")
    for i in range(n):
        for j in range(n - i):
            print(round(y[i][j], 7), "\t\t",
                                end = " ");

        print(" ");

n = EnterNumberOfData();
y = [[0 for i in range(10)]
        for j in range(10)];
x = [];

getData(n)
value = Enter_Value()
y=dividedDiffTable(x, y, n);

printDiffTable(y, n);

print("\nValue at", value, "is",
        round(applyFormula(value, x, y, n), 7))
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\Dell> python -u "c:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course related material\Numerical_Computing\Lab_2\Newton_
Enter Number of Data points: 5
Enter x0: 1.0
Enter y0: .7651977
Enter x1: 1.3
Enter y1: .6200860
Enter x2: 1.6
Enter y2: .4554022
Enter x3: 1.9
Enter y3: .2818186
Enter x4: 2.2
Enter y4: .1103623
Enter value x to find: 2.8
         DIVIDED DIFFERENCE TABLE
f(x)            1DD                     2DD                     3DD                     4DD

0.7651977               -0.4837057              -0.1087339              0.0658784               0.0018251
0.620086                -0.548946               -0.0494433              0.0680685
0.4554022               -0.578612               0.0118183
0.2818186               -0.571521
0.1103623

Value at 2.8 is -0.180286
PS C:\Users\Dell> python -u "c:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course related material\Numerical_Computing\Lab_2\Newton_
```

NUMERICAL COMPUTING (CS325)
PROJECT
LAB – 3

Group Members:

- Mohammad Basil Ali Khan (20K-0477)
- Ali Jodat (20K-0155)
- Abdul Ahad Shaikh (20K-0319)
- Mohammad Umer (20K-0225)

Project Title:

LAB 3: Numerical Integration

Aim:

To understand the fundamental concepts of scientific programming using python.

Description:

We selected three methods of Lab1.

1. Newton Cotes CLOSED quadrature formula.
2. Newton Cotes OPEN quadrature method.
3. Composite Midpoint rule

First we have studied the algorithm of then we have written the programming of that method.

IDE and Programming Language:

We have chosen python programming language and IDE we are using is Visual Studio Code.

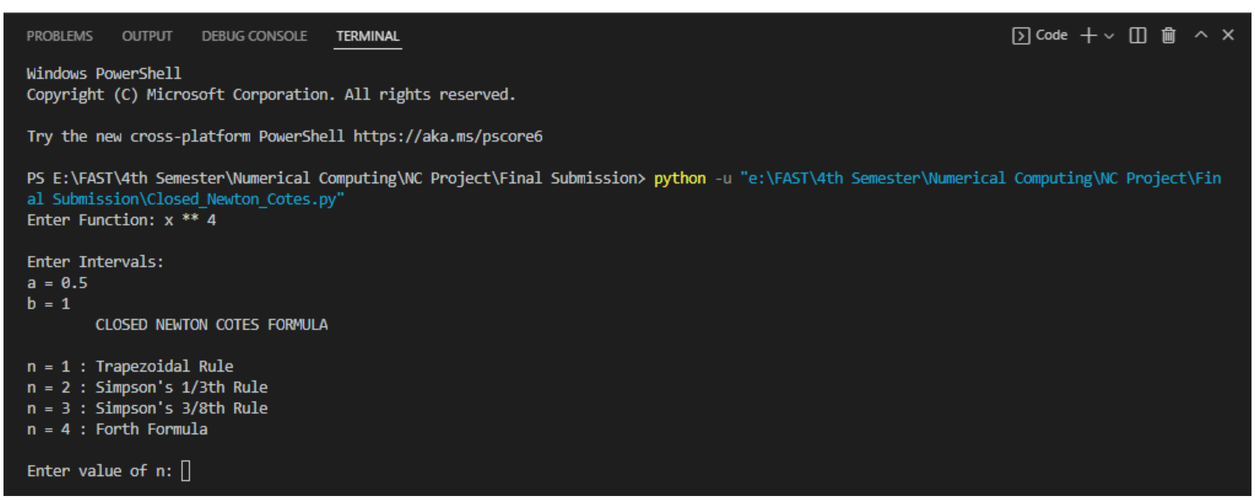
Library Used:

We have imported 3 libraries:

- ✓ sympy library for to get equation solution on particular intervals and can initialize symbols.

Implementation and Code Snippets:

CLOSED NEWTON COTES:



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Fin
al Submission\Closed_Newton_Cotes.py"
Enter Function: x ** 4

Enter Intervals:
a = 0.5
b = 1
        CLOSED NEWTON COTES FORMULA

n = 1 : Trapezoidal Rule
n = 2 : Simpson's 1/3th Rule
n = 3 : Simpson's 3/8th Rule
n = 4 : Forth Formula

Enter value of n:
```

Closed_Newton_Cotes.py

Closed_Newton_Cotes.py > ...

```python
from sympy import *

func = input("Enter Function: ")
print("\nEnter Intervals: ")
a = float(input("a = "))
b = float(input("b = "))
print("\tCLOSED NEWTON COTES FORMULA")
print("\nn = 1 : Trapezoidal Rule\nn = 2 : Simpson's 1/3th Rule\nn = 3 : Simpson's 3/8th Rule\nn = 4 : Fort
n = int(input("\nEnter value of n: "))

h = float((b-a)/float(n))

if n == 1:
    result = 0
    x = a
    y0 = float(eval(func))
    x = b
    y1 = float(eval(func))
    ans = y0  + y1
    result = (h / 2) * ans
    print("\nResult using Trapezoidal Rule: ", result)
elif n == 2:
    result = 0
    sum = 0
    x = a
    y = []
    y.append(eval(func))
    for i in range(0, 2):
        x = x + h
        y.append(eval(func))
    y[1] = 4 * y[1]
    for i in range(n+1):
        sum = sum + float(y[i])
```

```python
    sum = 0
    x = a
    y = []
    y.append(eval(func))
    for i in range(0, 3):
        x = x + h
        y.append(eval(func))
    y[1] = 3 * y[1]
    y[2] = 3 * y[2]
    for i in range(n+1):
        sum = sum + float(y[i])
    result = ((3 * h)/ 8) * sum
    print("\nResult using Simpson's 3/8th Rule: ", result)
elif n == 4:
    result = 0
    sum = 0
    x = a
    y = []
    y.append(eval(func))
    for i in range(0, 4):
        x = x + h
        y.append(eval(func))
    y[0] = 7 * y[0]
    y[1] = 32 * y[1]
    y[2] = 12 * y[2]
    y[3] = 32 * y[3]
    y[4] = 7 * y[4]
    for i in range(n+1):
        sum = sum + float(y[i])
    result = ((2*h)/45) * sum
    print("\nResult using Forth Formula: ", result)
else:
    print("\nChoose Valid Option !!!")
```

- ✓ **Trapezoidal Rule:**

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Fin
al Submission\Closed_Newton_Cotes.py"
Enter Function: exp(3*x) * sin(2*x)

Enter Intervals:
a = 0
b = 0.7853981634
        CLOSED NEWTON COTES FORMULA

n = 1 : Trapezoidal Rule
n = 2 : Simpson's 1/3th Rule
n = 3 : Simpson's 3/8th Rule
n = 4 : Forth Formula

Enter value of n: 1

Result using Trapezoidal Rule:  4.143259655239261
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

- ✓ **Simpsons 1/3rd Rule:**

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Fin
al Submission\Closed_Newton_Cotes.py"
Enter Function: exp(3*x) * sin(2*x)

Enter Intervals:
a = 0
b = 0.7853981634
        CLOSED NEWTON COTES FORMULA

n = 1 : Trapezoidal Rule
n = 2 : Simpson's 1/3th Rule
n = 3 : Simpson's 3/8th Rule
n = 4 : Forth Formula

Enter value of n: 2

Result using Simpson's 1/3rd Rule:  2.583696403274123
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

- ✓ **Simpsons 3/8th Rule:**

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Fin
al Submission\Closed_Newton_Cotes.py"
Enter Function: exp(3*x) * sin(2*x)

Enter Intervals:
a = 0
b = 0.7853981634
        CLOSED NEWTON COTES FORMULA

n = 1 : Trapezoidal Rule
n = 2 : Simpson's 1/3th Rule
n = 3 : Simpson's 3/8th Rule
n = 4 : Forth Formula

Enter value of n: 3

Result using Simpson's 3/8th Rule:  2.585789051658317
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

✓ **N = 4:**

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Closed_Newton_Cotes.py"
Enter Function: exp(3*x) * sin(2*x)

Enter Intervals:
a = 0
b = 0.7853981634
        CLOSED NEWTON COTES FORMULA

n = 1 : Trapezoidal Rule
n = 2 : Simpson's 1/3th Rule
n = 3 : Simpson's 3/8th Rule
n = 4 : Forth Formula

Enter value of n: 4

Result using Forth Formula:  2.5879684568329377
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

OPEN NEWTON COTES

```python
# Open_Newton_Cotes.py
from sympy import *

func = input("Enter Function: ")
print("\nEnter Intervals: ")
a = float(input("a = "))
b = float(input("b = "))
print("\n\tOPEN NEWTON COTES FORMULA")
print("\nn = 0 ?\nn = 1 ?\nn = 2 ?\nn = 3 ?")
n = int(input("\nEnter value of n: "))

h = float((b-a)/float(n+2))

if n == 0:
    result = 0
    x = a + h
    y0 = float(eval(func))
    result = (2 * h) * y0
    print("\nResult using n = 0: ", result)
elif n == 1:
    result = 0
    sum = 0
    x = a + h
    y = []
    y.append(eval(func))
    x = a + (2 * h)
    y.append(eval(func))
    for i in range(n+1):
        sum = sum + float(y[i])
    result = ((3 * h) / 2) * sum
    print("\nResult using n = 1: ", result)
elif n == 2:
    result = 0
    sum = 0
```

```python
        result = 0
        sum = 0
        x = a + h
        y = []
        y.append(eval(func))
        for i in range(0, 3):
            x = a + (i+2) * h
            y.append(eval(func))
        sum = (2 * y[0]) + (2 * y[2])
        sum = sum - y[1]
        result = ((4 * h)/ 3) * sum
        print("\nResult using n = 2: ", result)
    elif n == 3:
        result = 0
        sum = 0
        x = a + h
        y = []
        y.append(eval(func))
        for i in range(0, 4):
            x = a + (i+2) * h
            y.append(eval(func))
        y[0] = 11 * y[0]
        y[3] = 11 * y[3]
        for i in range(n+1):
            sum = sum + float(y[i])
        result = ((5*h)/24) * sum
        print("\nResult using n = 3: ", result)
    else:
        print("\nChoose Valid Option !!!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Open_Newton_Cotes.py"
Enter Function: x ** 4

Enter Intervals:
a = 0.5
b = 1

        OPEN NEWTON COTES FORMULA

n = 0 ?
n = 1 ?
n = 2 ?
n = 3 ?

Enter value of n:
```

- ✓ **N = 0:**

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Open_Newton_Cotes.py"
Enter Function: sin(x)

Enter Intervals:
a = 0
b = 0.7853981634

        OPEN NEWTON COTES FORMULA

n = 0 ?
n = 1 ?
n = 2 ?
n = 3 ?

Enter value of n: 0

Result using n = 0:  0.3005588649440754
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

✓ **N = 1:**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Open_Newton_Cotes.py"
Enter Function: sin(x)

Enter Intervals:
a = 0
b = 0.7853981634

        OPEN NEWTON COTES FORMULA

n = 0 ?
n = 1 ?
n = 2 ?
n = 3 ?

Enter value of n: 1

Result using n = 1:  0.297987542189132
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

✓ **N = 2:**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Open_Newton_Cotes.py"
Enter Function: sin(x)

Enter Intervals:
a = 0
b = 0.7853981634

        OPEN NEWTON COTES FORMULA

n = 0 ?
n = 1 ?
n = 2 ?
n = 3 ?

Enter value of n: 2

Result using n = 2:  0.292858659194394
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

✓ **N = 3:**

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission> python -u "e:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission\Open_Newton_Cotes.py"
Enter Function: sin(x)

Enter Intervals:
a = 0
b = 0.7853981634

        OPEN NEWTON COTES FORMULA

n = 0 ?
n = 1 ?
n = 2 ?
n = 3 ?

Enter value of n: 3

Result using n = 3:  0.29286922813788824
PS E:\FAST\4th Semester\Numerical Computing\NC Project\Final Submission>
```

Composite Midpoint Rule

Composite_Midpoint.py ×

C: > Users > Dell > Documents > BS(CS)NUCES > BS(CS) Course related material > Numerical_Computing > Lab_3 >

```python
from sympy import *

func = input("Enter Function: ")
print("\nEnter Intervals: ")
a = float(input("a = "))
b = float(input("b = "))

print("\n\tUsing Midpoint Composite Formula")
print("\nDo you want to enter value of \n (1) n \n (2) h")
choice = int(input("Enter 1 or 2 : "))
if choice == 1:
    n = int(input("Enter value of n : "))
    h = float((b-a)/float(n+2))
elif choice == 2:
    h = float(input("Enter value of h : "))
    n = int(((b-a)/float(h))-2)
else :
    print("Wrong choice")
    exit(0)

sum = 0
result = 0
x = a + h
y = []
t = int((n/2) + 1)

for i in range(t):
    y.append(eval(func))
    sum = sum + float(y[i])
    x = x + 2*h

result = 2*h*float(sum)
print("\nResult using Midpoint Formula with value of n =",n, "is :" , result)
```

✓ **When 'n' is given :**

```
PS C:\Users\Dell> python -u "c:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course rel
Enter Function: (x**2)*ln(x+1)

Enter Intervals:
a = 0
b = 2

        Using Midpoint Composite Formula

Do you want to enter value of
 (1) n
 (2) h
Enter 1 or 2 : 1
Enter value of n : 6

Result using Midpoint Formula with value of n = 6 is : 2.346918303762O858
PS C:\Users\Dell>
```

✓ **When ‘h’ is given :**

```
PS C:\Users\Dell> python -u "c:\Users\Dell\Documents\BS(CS)NUCES\BS(CS) Course related material\Nu
Enter Function: (x**2)*ln(x+1)

Enter Intervals:
a = 0
b = 2

        Using Midpoint Composite Formula

Do you want to enter value of
 (1) n
 (2) h
Enter 1 or 2 : 2
Enter value of h : 0.25

Result using Midpoint Formula with value of n = 6 is : 2.3469183037620858
PS C:\Users\Dell>
```