M.Sc. (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)

Fourth Semester

Laboratory Record

21-805-0406: Numerical Methods Lab

Submitted in partial fulfillment
of the requirements for the award of degree in
Master of Science (Five Year Integrated)
in Computer Science (Artificial Intelligence & Data Science) of
Cochin University of Science and Technology (CUSAT)
Kochi

Submitted by

SANDRALAYA S
(80521017)

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI-682022

AUGUST 2023

DEPARTMENT OF COMPUTER SCIENCE

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)

KOCHI, KERALA-682022

This is to certify that the software laboratory record for ***21-805-0406: Numerical Methods Lab*** *is a record of work carried out by* ***SANDRALAYA S(80521017)****, in partial fulfillment of the requirements for the award of degree in* ***Master of Science (Five Year Integrated)*** *in* ***Computer Science (Artificial Intelligence & Data Science)*** *of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the first semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

Faculty Member in-charge

Dr. Ajees A P
Assistant Professor
Department of Computer Science
CUSAT

Dr. Madhu S. Nair
Professor and Head
Department of Computer Science
CUSAT

Table of Contents

Bisection Method

AIM

To find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

PROGRAM

```
def func(x,degree,coff):
answer=0
for i in range(degree+1):
answer+=coff[i]*(x)**i
return answer

def bisection(a,b,degree,coff):

if (func(a,degree,coff) * func(b,degree,coff) >= 0):
print("You have not assumed right a and b\n")
return

c = a
while ((b-a) >= 0.01):

# Find middle point
c = (a+b)/2

# Check if middle point is root
if (func(c,degree,coff) == 0.0):
break

# Decide the side to repeat the steps
if (func(c,degree,coff)*func(a,degree,coff) < 0):
b = c
else:
a = c

print("The value of root is : ","%.4f"%c)

degree=int(input('Enter the degree of the funtion: '))
coff=[]
for i in range(0,degree+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
```

```
    coff.append(value)
coff=coff[::-1]
a=int(input('Enter the 1st value for Bisection Method: '))
b=int(input('Enter the 2nd value for Bisection Method: '))
bisection(a, b,degree,coff)
```

SAMPLE INPUT-OUTPUT

Newton Raphson Method

AIM

To calculate the root of a given a function f(x) on floating number x and an initial guess for root, find root of function in interval.

PROGRAM

```
def func(x,degree1,coff1):
answer=0
for i in range(degree1+1):
answer+=coff1[i]*(x)**i
return answer

def derivFunc(x,degree2,coff2):
answer=0
for i in range(degree2+1):
answer+=coff2[i]*(x)**i
return answer

def newtonRaphson(x,degree1,coff1,degree2,coff2):
h = func(x,degree1,coff1) / derivFunc(x,degree2,coff2)
while abs(h) >= 0.0001:
h = func(x,degree1,coff1)/derivFunc(x,degree2,coff2)

# x(i+1) = x(i) - f(x) / f'(x)
x = x - h

print("The value of the root is : ",
"%.4f"% x)

degree1=int(input('Enter the degree of the funtion: '))
coff1=[]
for i in range(0,degree1+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
    coff1.append(value)

degree2=int(input('Enter the degree of the funtion: '))
coff2=[]
for j in range(0,degree2+1):
    value=float(input('Enter the coefficient of x'+str(j)+': '))
```

```
    coff2.append(value)

x0=int(input('Enter the value for Newton Raphson Method: '))

print(coff1,coff2)
newtonRaphson(x0,degree1,coff1,degree2,coff2)
```

SAMPLE INPUT-OUTPUT

False Position Method

AIM

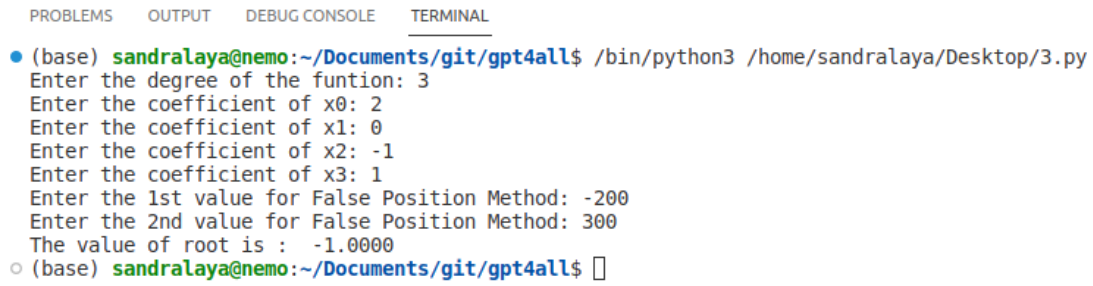
Given a function f(x) on floating number x and two numbers ‘a’ and ‘b’ such that f(a)*f(b) ¡ 0 and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

PROGRAM

```
MAX_ITER = 1000000

def func(x,degree,coff):
answer=0
for i in range(degree+1):
answer+=coff[i]*(x)**i
return answer

def regulaFalsi(a,b,degree,coff):
if func(a,degree,coff) * func(b,degree,coff) >= 0:
print("You have not assumed right a and b")
return -1

c = a

for i in range(MAX_ITER):
c = (a * func(b,degree,coff) - b * func(a,degree,coff))/
  (func(b,degree,coff) - func(a,degree,coff))

if func(c,degree,coff) == 0:
break

elif func(c,degree,coff) * func(a,degree,coff) < 0:
b = c
else:
a = c
print("The value of root is : " , '%.4f' %c)

degree=int(input('Enter the degree of the funtion: '))
coff=[]
for i in range(0,degree+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
```

```
    coff.append(value)
coff=coff[::-1]
a=int(input('Enter the 1st value for False Position Method: '))
b=int(input('Enter the 2nd value for False Position Method: '))
regulaFalsi(a, b,degree,coff)
```

SAMPLE INPUT-OUTPUT



Gaussian Elimination

AIM

The Gaussian elimination method is known as the row reduction algorithm for solving linear equations systems. It consists of a sequence of operations performed on the corresponding matrix of coefficients.

PROGRAM

```
N = 3
def gaussianElimination(mat):
singular_flag = forwardElim(mat)
if (singular_flag != -1):

print("Singular Matrix.")
if (mat[singular_flag][N]):
print("Inconsistent System.")
else:
print("May have infinitely many solutions.")

return
backSub(mat)
def swap_row(mat, i, j):

for k in range(N + 1):

temp = mat[i][k]
mat[i][k] = mat[j][k]
mat[j][k] = temp
def forwardElim(mat):
for k in range(N):
i_max = k
v_max = mat[i_max][k]
for i in range(k + 1, N):
if (abs(mat[i][k]) > v_max):
v_max = mat[i][k]
i_max = i
if not mat[k][i_max]:
return k
if (i_max != k):
swap_row(mat, k, i_max)
```

```
for i in range(k + 1, N):
f = mat[i][k]/mat[k][k]
for j in range(k + 1, N + 1):
mat[i][j] -= mat[k][j]*f
mat[i][k] = 0

return -1

def backSub(mat):

x = [None for _ in range(N)]
for i in range(N-1, -1, -1):

x[i] = mat[i][N]

for j in range(i + 1, N):

x[i] -= mat[i][j]*x[j]

x[i] = (x[i]/mat[i][i])

print("\nSolution for the system:")
for i in range(N):
print("{:.8f}".format(x[i]))

rows=int(input('Enter number of rows of the matrix: '))

mat = []

for i in range(rows):
    values=(input()).split()
    row=[]
    for i in values:
        row.append(float(i))

    mat.append(row)
gaussianElimination(mat)
```

SAMPLE INPUT-OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
(base) sandralaya@nemo:~/Documents/git/gpt4all$ /bin/python3 /home/sandralaya/Desktop/4.py
Enter number of rows of the matrix: 3
3.0 2.0 -4.0 3.0
2.0 3.0 3.0 15.0
5.0 -3.0 1.0 14.0

Solution for the system:
3.00000000
1.00000000
2.00000000
(base) sandralaya@nemo:~/Documents/git/gpt4all$
```

Gauss–Seidel method

AIM

The Gauss–Seidel method is an iterative technique for solving a square system of n (n=3) linear equations with unknown x.

PROGRAM

```
def seidel(a, x ,b):
n = len(a)
for j in range(0, n):
d = b[j]
for i in range(0, n):
if(j != i):
d-=a[j][i] * x[i]
x[j] = d / a[j][j]
return x

n = int(input('Enter number of rows of the matrix: '))
a = []
b = []
x=[]
for i in range(n):
    x.append(0)
a = []

for i in range(n):
    values=(input()).split()
    row=[]
    for i in values:
        row.append(float(i))

    a.append(row)

values=(input('Enter the matrix B')).split()
b=[]
for i in values:
    b.append(float(i))

for i in range(0, 25):
x = seidel(a, x, b)
```

```
print(x)
```

SAMPLE INPUT-OUTPUT

Gauss-Jordan method

AIM

The Gauss-Jordan method, also known as Gauss-Jordan elimination method is used to solve a system of linear equations and is a modified version of Gauss Elimination Method.

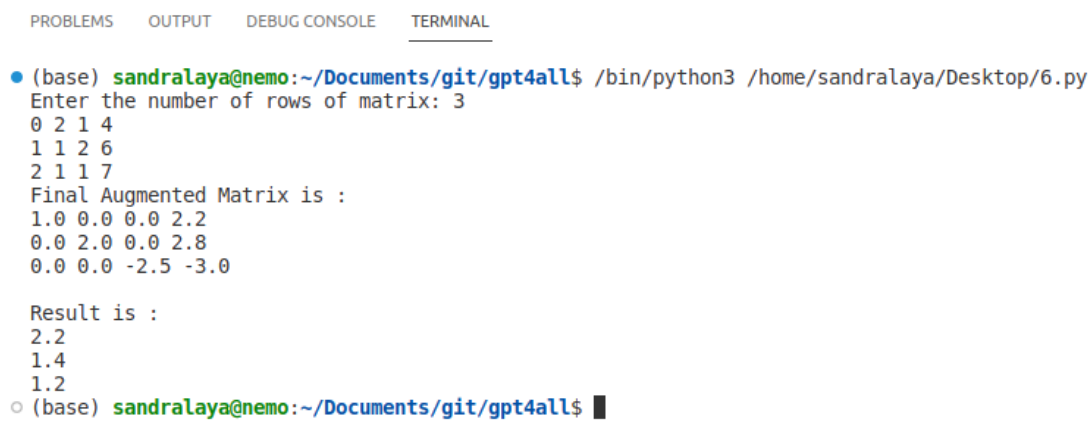
PROGRAM

```
M = 10
def PrintMatrix(a, n):
for i in range(n):
print(*a[i])
def PerformOperation(a, n):
i = 0
j = 0
k = 0
c = 0
flag = 0
m = 0
pro = 0
for i in range(n):
if (a[i][i] == 0):

c = 1
while ((i + c) < n and a[i + c][i] == 0):
c += 1
if ((i + c) == n):

flag = 1
break

j = i
for k in range(1 + n):

temp = a[j][k]
a[j][k] = a[j+c][k]
a[j+c][k] = temp

for j in range(n):
if (i != j):
p = a[j][i] / a[i][i]
```

```
k = 0
for k in range(n + 1):
a[j][k] = a[j][k] - (a[i][k]) * p

return flag
def PrintResult(a, n, flag):

print("Result is : ")

if (flag == 2):
print("Infinite Solutions Exists<br>")
elif (flag == 3):
print("No Solution Exists<br>")
else:
for i in range(n):
print(a[i][n] / a[i][i], end=" ")
print()
def CheckConsistency(a, n, flag):
flag = 3
for i in range(n):
sum = 0
for j in range(n):
sum = sum + a[i][j]
if (sum == a[i][j]):
flag = 2

return flag
a = []
n = int(input('Enter the number of rows of matrix: '))
for i in range(n):
    values=(input()).split()
    row=[]
    for i in values:
        row.append(float(i))

    a.append(row)

flag = 0
flag = PerformOperation(a, n)
```

```
if (flag == 1):
flag = CheckConsistency(a, n, flag)
print("Final Augmented Matrix is : ")
PrintMatrix(a, n)
print()
PrintResult(a, n, flag)
```

SAMPLE INPUT-OUTPUT



Newton Forward Interpolation

AIM

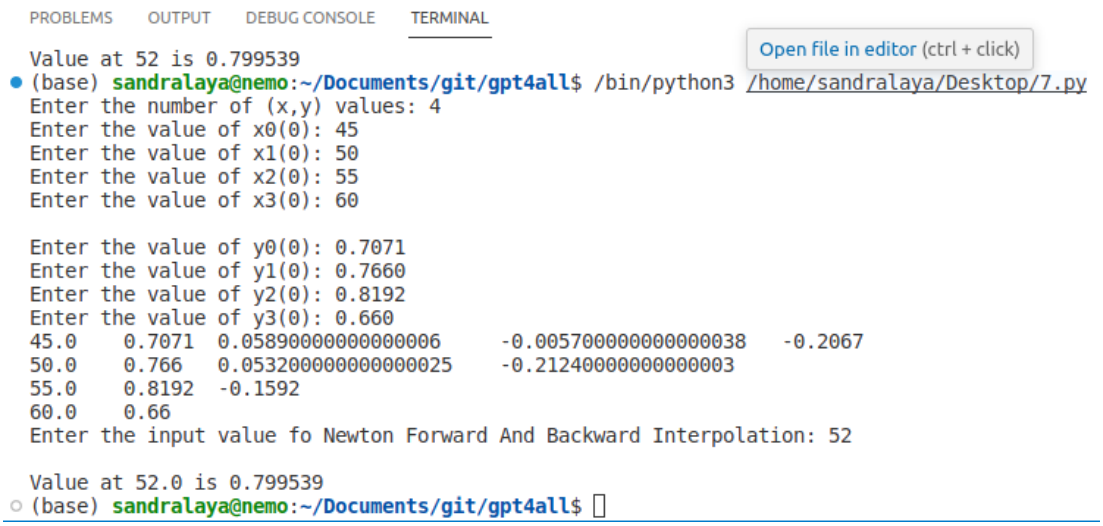
Write a C++ program to implement a class MATRIX with member functions such as matrix_add, matrix_mult, matrix_transpose, matrix_trace etc

PROGRAM

```
def u_cal(u, n):

temp = u
for i in range(1, n):
temp = temp * (u - i)
return temp

def fact(n):
f = 1
for i in range(2, n + 1):
f *= i
return f

n=int(input('Enter the number of (x,y) values: '))
x=[]
for i in range(n):
x.append(float(input('Enter the value of x'+str(i)+'(0): ')))

y = [[0 for i in range(n)]
for j in range(n)]
print()
for i in range(n):
y[i][0]=(float(input('Enter the value of y'+str(i)+'(0): ')))

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

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

```
value=float(input('Enter the input value fo Newton Forward And Backward Interpolation: '))

sum = y[0][0]
u = (value - x[0]) / (x[1] - x[0])
for i in range(1,n):
sum = sum + (u_cal(u, i) * y[0][i]) / fact(i)

print("\nValue at", value,
"is", round(sum, 6))
```

SAMPLE INPUT-OUTPUT



Lagrange's Interpolation

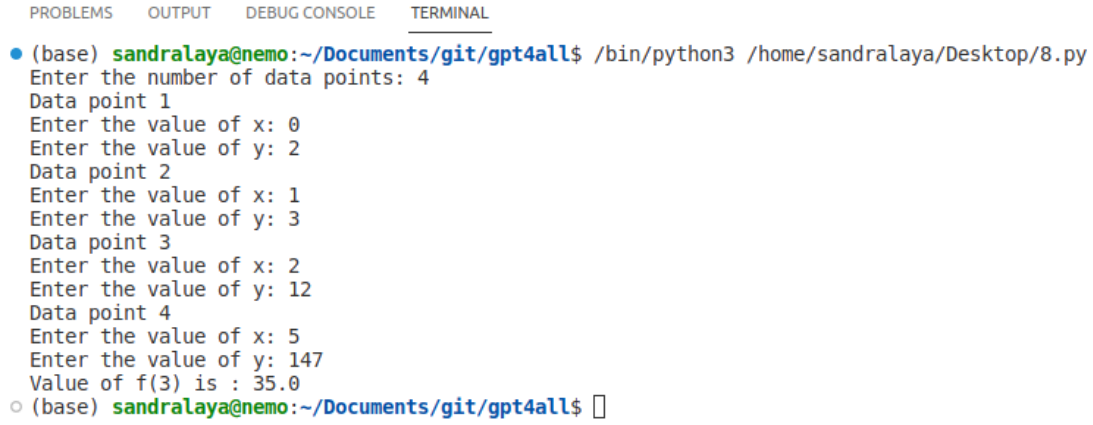
AIM

A method of finding new data points within the range of a discrete set of known data points. In other words interpolation is the technique to estimate the value of a mathematical function, for any intermediate value of the independent variable.

PROGRAM

```
class Data:
def __init__(self, x, y):
self.x = x
self.y = y

def interpolate(f: list, xi: int, n: int) -> float:
result = 0.0
for i in range(n):
term = f[i].y
for j in range(n):
if j != i:
term = term * (xi - f[j].x) / (f[i].x - f[j].x)
result += term

return result
if __name__ == "__main__":
f=[]#[Data(0, 2), Data(1, 3), Data(2, 12), Data(5, 147)]
datapoints=int(input('Enter the number of data points: '))
for i in range(datapoints):
print('Data point '+str(i+1))
x=int(input('Enter the value of x: '))
y=int(input('Enter the value of y: '))
f.append(Data(x,y))

print("Value of f(3) is :", interpolate(f, 3, 4))
```

SAMPLE INPUT-OUTPUT



Simpson's 1/3 Rule

AIM

In numerical analysis, Simpson's 1/3 rule is a method for numerical approximation of definite integrals.

PROGRAM

```
import math
def func(x,degree,coff):
answer=0
for i in range(degree+1):
answer+=coff[i]*(x)**i
return answer
def simpsons_( ll, ul, n ):

# Calculating the value of h
h = ( ul - ll )/n

# List for storing value of x and f(x)
x = list()
fx = list()

# Calculating values of x and f(x)
i = 0
while i<= n:
x.append(ll + i * h)
fx.append(func(x[i],degree,coff))
i += 1

# Calculating result
res = 0
i = 0
while i<= n:
if i == 0 or i == n:
res+= fx[i]
elif i % 2 != 0:
res+= 4 * fx[i]
else:
res+= 2 * fx[i]
i+= 1
```

```
res = res * (h / 3)
print("%.6f"% res)

degree=int(input('Enter the degree of the funtion: '))
coff=[]
for i in range(0,degree+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
    coff.append(value)
coff=coff[::-1]

lower_limit = int(input('Enter the lower limit: '))
upper_limit = int(input('Enter the upper limit: '))
n = int(input('Enter the intervals: '))#6 # Number of interval
simpsons_(lower_limit, upper_limit, n)
```

SAMPLE INPUT-OUTPUT

Simpson's 3/8 rule

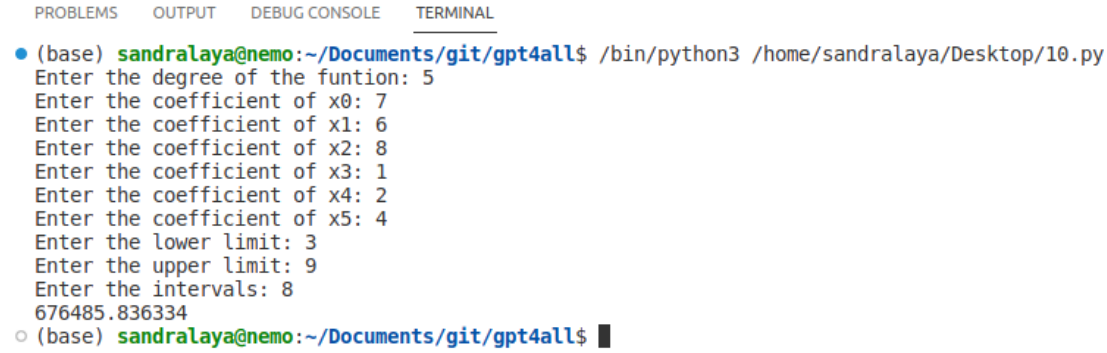
AIM

The Simpson's 3/8 rule was developed by Thomas Simpson. This method is used for performing numerical integrations. This method is generally used for numerical approximation of definite integrals. Here, parabolas are used to approximate each part of curve.

PROGRAM

```
def func(x,degree,coff):
answer=0
for i in range(degree+1):
answer+=coff[i]*(x)**i
return answer

def calculate(lower_limit, upper_limit, interval_limit ):

interval_size = (float(upper_limit - lower_limit) / interval_limit)
sum = func(lower_limit,degree,coff) + func(upper_limit,degree,coff);

for i in range(1, interval_limit ):
if (i % 3 == 0):
sum = sum + 2 * func((lower_limit + i * interval_size),degree,coff)
else:
sum = sum + 3 * func((lower_limit + i * interval_size),degree,coff)

return ((float( 3 * interval_size) / 8 ) * sum )

degree=int(input('Enter the degree of the funtion: '))
coff=[]
for i in range(0,degree+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
    coff.append(value)
coff=coff[::-1]

lower_limit = int(input('Enter the lower limit: '))
upper_limit = int(input('Enter the upper limit: '))
interval_limit = int(input('Enter the intervals: '))
integral_res = calculate(lower_limit, upper_limit, interval_limit)
print (round(integral_res, 6))
```

SAMPLE INPUT-OUTPUT



Trapezoidal Rule

AIM

Trapezoidal rule is used to find the approximation of a definite integral. The basic idea in Trapezoidal rule is to assume the region under the graph of the given function to be a trapezoid and calculate its area.

PROGRAM

```
def func(x,degree,coff):
answer=0
for i in range(degree+1):
answer+=coff[i]*(x)**i
return answer
def trapezoidal (a, b, n,degree,coff):
h = (b - a) / n
s = (func(a,degree,coff) + func(b,degree,coff))
i = 1
while i < n:
s += 2 * func((a + i * h),degree,coff)
i += 1
return ((h / 2) * s)

degree=int(input('Enter the degree of the funtion: '))
coff=[]
for i in range(0,degree+1):
    value=float(input('Enter the coefficient of x'+str(i)+': '))
    coff.append(value)
coff=coff[::-1]
x0=int(input('Enter the upper limit: '))
xn=int(input('Enter the lower limit: '))
n=int(input('Enter the intervals: '))

print ("Value of integral is ",
"%.4f"%trapezoidal(x0, xn, n,degree,coff))
```

SAMPLE INPUT-OUTPUT

Euler Method

AIM

The Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value.

PROGRAM

```
def func( x, y ):
return (x + y + x * y)

def euler( x0, y, h, x ):
temp = -0

while x0 < x:
temp = y
y = y + h * func(x0, y)
x0 = x0 + h

print("Approximate solution at x = ", x, " is ", "%.6f"% y)



x0 = float(input('Enter the x0 value for Eulers Method: '))
y0 = float(input('Enter the y0 value for Eulers Method: '))
h = float(input('Enter the h value for Eulers Method: '))
x = float(input('Enter the x value for Eulers Method: '))

euler(x0, y0, h, x)
```

SAMPLE INPUT-OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
(base) sandralaya@nemo:~/Documents/git/gpt4all$ /bin/python3 /home/sandralaya/Desktop/12.py
Enter the x0 value for Eulers Method: 0
Enter the y0 value for Eulers Method: 1
Enter the h value for Eulers Method: 0.025
Enter the x value for Eulers Method: 0.1
Approximate solution at x =  0.1  is  1.111673
(base) sandralaya@nemo:~/Documents/git/gpt4all$
```