**NUMERICAL COMPUTING ASSIGNMENT**

**NAME:**

MUHAMMAD AMMAR SIDDIQUI

**ROLL NUMBER:**

20k-0177

**SECTION:**

BCS-4E

**LANGUAGE:**

Python 3.0

**Root Finding:**

1. **Newton Method:**

def func( x ):

return x \* x \* x - x \* x -1

def derivFunc( x ):

return 3 \* x \* x - 2 \* x

def newton( x ):

h = func(x) / derivFunc(x)

while abs(h) >= 1e-10:

h = func(x)/derivFunc(x)

x = x - h

print("The value of the root is : ","%.9f"% x)

x0 = 1

newton(x0)



1. **Bisection Method:**

def func(x): #function is f(x) = x2 - 3

return x\*x – 3

# Prints root of func(x)

# with error of EPSILON

def bisection(a,b):

if (func(a) \* func(b) >= 0):

print("Interval theorem is failed\n")

return

c = a

while ((b-a) >= 0.0001):

# Find middle point

c = (a+b)/2

# Check if middle point is root

if (func(c) == 0.0):

break

if (func(c)\*func(a) < 0):

b = c

else:

a = c

print("The value of root is : ","%.5f"%c)

a = 1

b = 2

bisection(a, b)



1. **False Position:**

MAX\_ITER = 100

def func( x ):

return (x \* x \* x - x - 1)

def regulaFalsi( a , b):

if func(a) \* func(b) >= 0:

print("Interval theorem has failed")

return -1

c = a

for i in range(MAX\_ITER):

# Using the formula for regular falsi

c = (a \* func(b) - b \* func(a))/ (func(b) - func(a))

# Check if the above found point is root

if func(c) == 0:

break

# Decide the side to repeat the steps

elif func(c) \* func(a) < 0:

b = c

else:

a = c

print("The value of root is : " , '%.6f' %c)

a = 1

b = 2

regulaFalsi(a, b)



1. **Fixed Point:**

import math

def f(x):

return x\*x\*x + x\*x -1

# Re-writing f(x)=0 to x = g(x)

def g(x):

return 1/math.sqrt(1+x)

# Implementing Fixed Point Iteration Method

def fixedPointIteration(x0, e, N):

step = 1

flag = 1

condition = True

while condition:

x1 = g(x0)

x0 = x1

step = step+1

if step > N:

flag=0

break

condition = abs(f(x1)) > e

if flag==1:

print('\nRequired root is: %0.8f' % x1)

else:

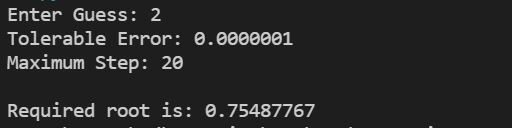
print('\nNot Convergent.')

x0 = float(input('Enter Guess: '))

e = float(input('Tolerable Error: '))

N = int(input('Maximum Step: '))

fixedPointIteration(x0,e,N)



1. **Secant Method**

def secant(f,a,b,N):

if f(a)\*f(b) >= 0:

print("Interval theorem fails. Couldn't find root.")

return None

a\_n = a

b\_n = b

for n in range(1,N+1):

m\_n = a\_n - f(a\_n)\*(b\_n - a\_n)/(f(b\_n) - f(a\_n))

f\_m\_n = f(m\_n)

if f(a\_n)\*f\_m\_n < 0:

a\_n = a\_n

b\_n = m\_n

elif f(b\_n)\*f\_m\_n < 0:

a\_n = m\_n

b\_n = b\_n

elif f\_m\_n == 0:

print("Found exact solution.")

return m\_n

else:

print("Secant method fails.")

return None

return a\_n - f(a\_n)\*(b\_n - a\_n)/(f(b\_n) - f(a\_n))

p = lambda x: x\*\*3 - x\*\*2 -1

approx = secant(p, 1, 2, 20)

if (approx!=None):

print("Answer found %0.4f" %approx)

else:

print("Answer not found\n")



1. **Taylor Polynomial & relative error:**

import sympy as sp

import math

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

def taylor\_polynomial(f,degree):

ans = f

for i in range (0,degree+1):

temp = sp.diff(f,x,i)/math.factorial(i)

ans += temp

return ans

def taylor\_polynomial\_val(f,x1,x0,degree):

#f(x) + f'(x)/1!

ans = 0

for i in range(0,degree+1):

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

valf\_f = (valf\_f(x0))/math.factorial(i)

valf\_f\*= (x1-x0)

ans+=valf\_f

i = i+1

f = sp.diff(f,x,i)

return ans

f = sp.cos(x)

print("Function is: ",f,"\n")

x0 = 0

x0 = int(x0)

x1 = float(input("Enter the value of x "))

degree = int(input("Enter the degree "))

#print("The polynomial is: ",taylor\_polynomial(f,degree),"\n")

ans = taylor\_polynomial\_val(f, x1, x0, degree)

print("Evaluation of polynomial is %0.4f" %ans)

re = abs(ans - math.cos(x0))

re = abs(ans-math.sin(math.radians(x0)))

print("Relative error is %0.4f" %re)

**Lagrange interpolation:**

import numpy as np

# Reading degree

degree = int(input('Enter degree of the polynomial: '))

x = np.zeros((degree+1))

y = np.zeros((degree+1))

# Reading data points

print('Enter the points x & y: ')

for i in range(degree+1):

x[i] = float(input( 'x['+str(i)+']='))

y[i] = float(input( 'y['+str(i)+']='))

xp = float(input('Enter the value of x'))

ans = 0

for i in range(degree+1):

p = 1

for j in range(degree+1):

if i != j: #if we have L(x0) then (x0-x0) shouldn't be included in the solution therefore adding this check

p = p \* (xp - x[j])/(x[i] - x[j])

ans = ans + p \* y[i] #Multiplying with f(x)

# Displaying output

print('Interpolated value at %.3f is %.3f.' % (xp, ans))

**Bound Error:**

import sympy as sp

import math

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

f = sp.cos(x)

n = int(input("Enter the value of n"))

x0 = float(input('Enter the value of x0 '))

x1 = float(input('Enter the value of x1 '))

x2 = float(input('Enter the value of x '))

dfx = sp.diff(f,x,n+1)

dfx\_f = sp.lambdify(x,dfx)

dfx\_f = dfx\_f(x2)/math.factorial(n+1)

be = dfx\_f\*(x2-x0)\*(x2-x1)

print("Bound error is: %0.4f" %be)