PAF-Karachi Institute of Economics and Technology
College of Computing and Information Sciences - North Campus
Numerical Computing (105127/29)

Assignment # 01(Lab)

Ali Afzal 63961

Question 1

We have done five root finding methods up till now i.e. Bisection method, Regula Falsi method, Newton Raphson Method, Secant Method and Muller method. Stopping criteria adopted by us was on the base of tolerance value. IT means that we stopped finding next approximate root when we obtained accuracy to certain number of decimal places.

Now your task is to design a Python program which does the following:
-Ask user to input the equation
-Ask user the name of root finding method to use on the given equation
-Take input of initial values from user on the basis of method name (i.e two values for bisection, Regula Falsi and secant, one for Newton Raphson and three for Muller method)
-Implement the stopping criteria on the base of relative error rather than on the base of tolerance value.
-Display all iterations in the form of a grid like this
-Draw graph of the method entered by user.

Table 1.1 The bisection method for $f(x) = 3x + \sin(x) - e^x = 0$, starting from $x_1 = 0$, $x_2 = 1$, using a tolerance value of 1E-4

Iteration	X_1	X_2	X_3	$F(X_3)$	Maximum error	Actual error
1	0.00000	1.00000	0.50000	0.33070	0.50000	0.13958
2	0.00000	0.50000	0.25000	−0.28662	0.25000	−0.11042
3	0.25000	0.50000	0.37500	0.03628	0.12500	0.01458
4	0.25000	0.37500	0.31250	−0.12190	0.06250	−0.04792
5	0.31250	0.37500	0.34375	−0.04196	0.03125	−0.01667
6	0.34375	0.37500	0.35938	−0.00262	0.01563	−0.00105
7	0.35938	0.37500	0.36719	0.01689	0.00781	0.00677
8	0.35938	0.36719	0.36328	0.00715	0.00391	0.00286
9	0.35938	0.36328	0.36133	0.00227	0.00195	0.00091
10	0.35938	0.36133	0.36035	−0.00018	0.00098	−0.00007
11	0.36035	0.36133	0.36084	0.00105	0.00049	0.00042
12	0.36035	0.36084	0.36060	0.00044	0.00024	0.00017
13	0.36035	0.36060	0.36047	0.00013	0.00012	0.00005

Note: You can use the provided codes and make modifications in them. You have to perform above tasks on all methods. You may find other such tables for reference in book.
Submit the code and output in PDF form. Late submission and copied assignments will not be accepted.

Question 2

Implement any one differentiation method of your choice, without using any built-in function.

Answer1:
Code:

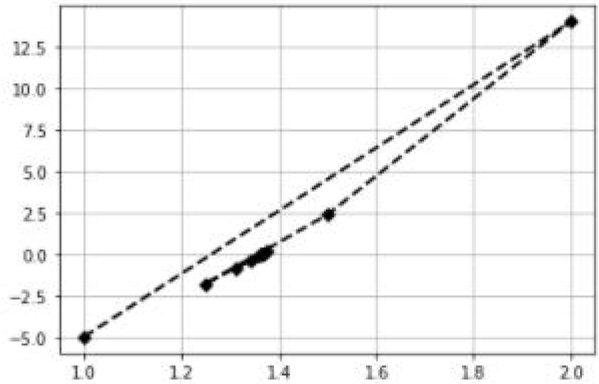
```
import cmath
import pandas
from sympy import var
from scipy import misc
from sympy import sympify
import matplotlib.pyplot as plt
from sympy.utilities.lambdify import lambdify



#Declaration Of Global Variables
x_axis=[]
y_axis=[]
data=[]

#Ask user to input the equation
x = var('x')
user_input = input("Enter your function: ")
expr = sympify(user_input)
f = lambdify(x, expr)


#Ask user the name of root finding method to use on the given equation
print("Enter 0 for Bisection")
print("Enter 1 for Regula Falsi")
print("Enter 2 for Newton Raphson")
print("Enter 3 for Secant")
print("Enter 4 for Muller")
sel = input("Your Selection: ")


print('\n')


#Method Definations
def bissection(a,b):
  x_axis.append(a)
  y_axis.append(f(a))
  x_axis.append(b)
  y_axis.append(f(b))
  niter=1
  mid = 0.1
  oldmid = 0.01
  while((abs(mid-oldmid)/abs(mid))>=0.0001):
    oldmid=mid
    mid=(a+b)/2.0
    prod1=f(a)*f(mid)
    prod2=f(b)*f(mid)
    maxerror=abs(mid-oldmid)
    actualerror=abs(mid-oldmid)/abs(mid)
    data.append([niter,a,b,mid,f(mid),maxerror,actualerror])
    if prod1<0:
      b=mid
    elif prod2<0:
      a=mid
```

```
    niter+=1
    x_axis.append(mid)
    y_axis.append(f(mid))
  print(pandas.DataFrame(data, columns=['Itertaion', 'A', 'B', 'Mid', 'f(Mid)','Max Error','Actual Error']))
  return mid,niter

#------------------------------------------

def rf(a,b):
  x_axis.append(a)
  y_axis.append(f(a))
  x_axis.append(b)
  y_axis.append(f(b))
  niter=1
  m = 0.1
  oldm = 0.01
  while((abs(m-oldm)/abs(m))>=0.0001 and niter <= 50):
    oldm=m
    m=(a*f(b)-b*f(a))/(f(b)-f(a))
    prod1=f(a)*f(m)
    prod2=f(b)*f(m)
    maxerror=abs(m-oldm)
    actualerror=abs(m-oldm)/abs(m)
    data.append([niter,a,b,m,f(m),maxerror,actualerror])
    if prod1<0:
      b=m
    elif prod2<0:
      a=m
    niter+=1
    x_axis.append(m)
    y_axis.append(f(m))
  print(pandas.DataFrame(data, columns=['Itertaion', 'A', 'B', 'Mid', 'f(Mid)','Max Error','Actual Error']))
  return m,niter

#------------------------------------------

def NewtonsMethod(f, x):
    niter=1
    oldx = 0.01
    while True:
        x_axis.append(x)
        y_axis.append(f(x))
        x1 = x - f(x) / misc.derivative(f, x)
        maxerror= abs(x1 - x)
        actualerror= abs(x1 - x)/abs(x1)
        data.append([niter,x,x1,f(x1),maxerror,actualerror])
        if (actualerror) < 0.0001:
            break
        x = x1
        niter+=1
    print(pandas.DataFrame(data, columns=['Itertaion', 'A', 'Root', 'f(Root)','Max Error','Actual Error']))
    return x

#------------------------------------------

def secant(fn,a,b,niter=100):
  x_axis.append(a)
  y_axis.append(f(a))
  x_axis.append(b)
  y_axis.append(f(b))
```

```
 c = 0.1
 oldc = 0.01
 for i in range(niter):
   oldc=c
   c= b-(b-a)/(fn(b)-fn(a))*fn(b)
   x_axis.append(c)
   y_axis.append(f(c))
   maxerror=abs(c-oldc)
   actualerror=abs(c-oldc)/abs(c)
   data.append([i+1,a,b,c,f(c),maxerror,actualerror])
   if (abs(c-oldc)/abs(c))>=0.0001: break
   else:
     a,b=b,c
 else:
   print("Max Iteration Completed")
 print(pandas.DataFrame(data, columns=['Itertaion', 'A', 'B', 'Root', 'f(Root)','Max Error','Actual Error']))
 return c,i

#-------------------------------------------

def muller(f,x0,x1,x2):
 x3 = 0.1
 oldx3 = 0.01
 i=3

 y0=(f(x0))
 y1=(f(x1))
 y2=(f(x2))

 x_axis.append(x0)
 x_axis.append(x1)
 x_axis.append(x2)
 y_axis.append(f(x0))
 y_axis.append(f(x1))
 y_axis.append(f(x2))

 data.append([0,x0,x1,x2,'-','-',0])

 while(abs(x3-oldx3)/abs(x3)>0.01):
   oldx3=x3
   h1=x1-x0
   h2=x0-x2

   h=h2/h1

   a=(h*y1-y0*(1+h)+y2)/h*h1*h1*(1+h)
   b=(y1-y0-(a*h1**2))/h1
   c=f(x2)

   x3=x0-(2*c)/(b+cmath.sqrt(b**2-4*a*c))
   maxerror=abs(x3-oldx3)
   actualerror=abs(x3-oldx3)/abs(x3)
   x_axis.append(x3)
   y_axis.append(f(x3))
   data.append([i,x0.real,x1.real,x2.real,x3.real,f(x3.real),maxerror,actualerror])

   if x3.real>x0.real:
     x2=x0
     x0=x3
   elif x3.real<x0.real:
```

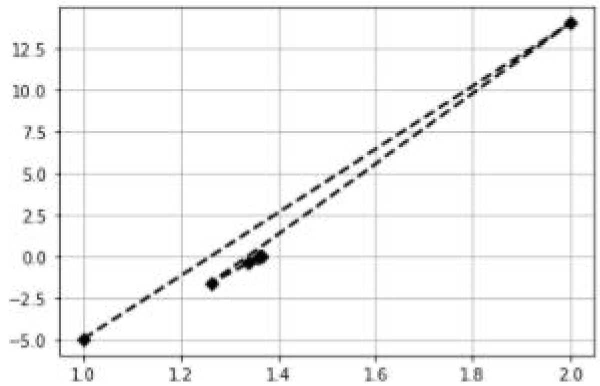
```
      x1=x0
      x0=x3
    i = i + 1
  print(pandas.DataFrame(data, columns=['Itertaion', 'X0', 'X1', 'X2', 'X3', 'f(X3)','Max Error','Actual Error']))
  return x0

#------------------------------------------

#Driver Codes
#Bisection
if sel == '0':
  print("Bisection")
  a = float(input("Value 1: "))
  b = float(input("Value 2: "))
  bissection(a,b)

#Regula Falsi
if sel == '1':
  print("Regula Falsi")
  a = float(input("Value 1: "))
  b = float(input("Value 2: "))
  rf(a,b)

#Newton Raphson
if sel == '2':
  print("Newton Raphson")
  a = float(input("Value 1: "))
  NewtonsMethod(f, a)

#Secant
if sel == '3':
  print("Secant")
  a = float(input("Value 1: "))
  b = float(input("Value 2: "))
  secant(f,a,b)

#Muller
if sel == '4':
  print("Muller")
  a = float(input("Value 1: "))
  b = float(input("Value 2: "))
  c = float(input("Value 3: "))
  muller(f,a,b,c)


#Plotting
plt.plot(x_axis,y_axis,color='k', marker='D',linestyle='--', linewidth=2)
plt.grid()
plt.show()
```

```
Enter your function: x**3+4*x**2-10
Enter 0 for Bisection
Enter 1 for Regula Falsi
Enter 2 for Newton Raphson
Enter 3 for Secant
Enter 4 for Muller
Your Selection: 0


Bisection
Value 1: 1
Value 2: 2
    Itertaion         A         B       Mid   f(Mid)  Max Error  Actual Error
0           1  1.000000  2.000000  1.500000  2.375000   1.400000      0.933333
1           2  1.000000  1.500000  1.250000 -1.796875   0.250000      0.200000
2           3  1.250000  1.500000  1.375000  0.162109   0.125000      0.090909
3           4  1.250000  1.375000  1.312500 -0.848389   0.062500      0.047619
4           5  1.312500  1.375000  1.343750 -0.350983   0.031250      0.023256
5           6  1.343750  1.375000  1.359375 -0.096409   0.015625      0.011494
6           7  1.359375  1.375000  1.367188  0.032356   0.007812      0.005714
7           8  1.359375  1.367188  1.363281 -0.032150   0.003906      0.002865
8           9  1.363281  1.367188  1.365234  0.000072   0.001953      0.001431
9          10  1.363281  1.365234  1.364258 -0.016047   0.000977      0.000716
10         11  1.364258  1.365234  1.364746 -0.007989   0.000488      0.000358
11         12  1.364746  1.365234  1.364990 -0.003959   0.000244      0.000179
12         13  1.364990  1.365234  1.365112 -0.001944   0.000122      0.000089
```

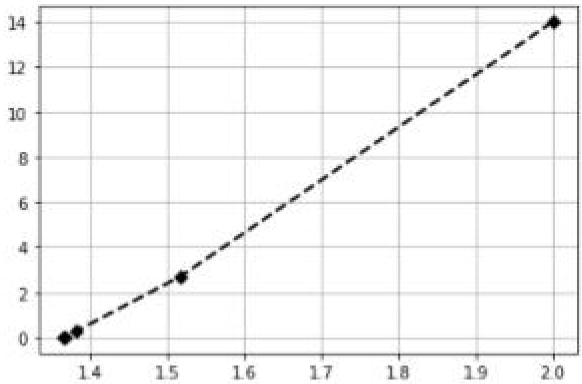


Output:

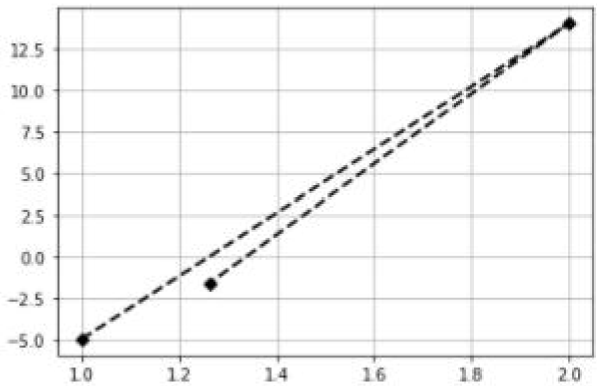
```
Enter your function: x**3+4*x**2-10
Enter 0 for Bisection
Enter 1 for Regula Falsi
Enter 2 for Newton Raphson
Enter 3 for Secant
Enter 4 for Muller
Your Selection: 1


Regula Falsi
Value 1: 1
Value 2: 2
   Itertaion         A    B       Mid    f(Mid)  Max Error  Actual Error
0          1  1.000000  2.0  1.263158 -1.602274   1.163158      0.920833
1          2  1.263158  2.0  1.338828 -0.430365   0.075670      0.056520
2          3  1.338828  2.0  1.358546 -0.110009   0.019719      0.014514
3          4  1.358546  2.0  1.363547 -0.027762   0.005001      0.003668
4          5  1.363547  2.0  1.364807 -0.006983   0.001260      0.000923
5          6  1.364807  2.0  1.365124 -0.001755   0.000317      0.000232
6          7  1.365124  2.0  1.365203 -0.000441   0.000080      0.000058
```



```
Enter your function: x**3+4*x**2-10
Enter 0 for Bisection
Enter 1 for Regula Falsi
Enter 2 for Newton Raphson
Enter 3 for Secant
Enter 4 for Muller
Your Selection: 2


Newton Raphson
Value 1: 2
   Itertaion         A      Root   f(Root)  Max Error  Actual Error
0          1  2.000000  1.517241  2.700808   0.482759      0.318182
1          2  1.517241  1.382497  0.287562   0.134744      0.097464
2          3  1.382497  1.366337  0.018284   0.016161      0.011828
3          4  1.366337  1.365294  0.001052   0.001043      0.000764
4          5  1.365294  1.365234  0.000060   0.000060      0.000044
```




```
Enter your function: x**3+4*x**2-10
Enter 0 for Bisection
Enter 1 for Regula Falsi
Enter 2 for Newton Raphson
Enter 3 for Secant
Enter 4 for Muller
Your Selection: 3


Secant
Value 1: 1
Value 2: 2
   Itertaion    A    B      Root   f(Root)  Max Error  Actual Error
0          1  1.0  2.0  1.263158 -1.602274   1.163158      0.920833
```



Answer2:

Code:

```
def dd(x, y):
    return ((x - y)/2)

def RK(x0, y0, x, h):
    n = int((x - x0)/h)
    y = y0
    for i in range(1, n + 1):
        k1 = h * dd(x0, y)
        k2 = h * dd(x0 + 0.5 * h, y + 0.5 * k1)
        k3 = h * dd(x0 + 0.5 * h, y + 0.5 * k2)
        k4 = h * dd(x0 + h, y + k3)
        y = y + (1.0 / 6.0)*(k1 + 2 * k2 + 2 * k3 + k4)
        x0 = x0 + h
    return y


print ('RK of y at x is:', RK(2, 4, 8, 0.4))
```

Output:

```
In [8]: def dd(x, y):
            return ((x - y)/2)

        def RK(x0, y0, x, h):
            n = int((x - x0)/h)
            y = y0
            for i in range(1, n + 1):
                k1 = h * dd(x0, y)
                k2 = h * dd(x0 + 0.5 * h, y + 0.5 * k1)
                k3 = h * dd(x0 + 0.5 * h, y + 0.5 * k2)
                k4 = h * dd(x0 + h, y + k3)
                y = y + (1.0 / 6.0)*(k1 + 2 * k2 + 2 * k3 + k4)
                x0 = x0 + h
            return y


        print ('RK of y at x is:', RK(2, 4, 8, 0.4))

        RK of y at x is: 6.199157688008517
```