University of Dhaka

Department of Computer Science & Engineering

CSE 3212: Numerical Methods Lab

3rd Year 2nd Semester

Name of the assignment:

Finding roots of different equations using bisection, false position, Newton Raphson and secant method

Submitted by:

Shahrear Bin Amin , Roll: 55

Submitted to:

Mr. Mubin Ul Haque, Lecturer, Department of CSE

University of Dhaka

Problem statement 1:

The velocity v of a falling parachutist is given by

$$v = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

where g = 9.8 m/s^2. For a parachutist with a drag coefficient c = 15 kg/s, compute the mass m
so that the velocity is v = 35 m/s at t = 9 s.

By using, (a) bisection and (b) false position.
For (a) and (b) use initial guesses from the ***user input,*** and iterate until the approximate error
falls below ***user specified tolerance.***
At first, print the value of m and f(m) from user lower input and user upper input, increasing by
0.1. Then, If the root finding is possible, print the solution, otherwise print no root is possible .
You also need to print the following table in your console view.

iteration	Upper value	Lower value	Xm	f(Xm)	Relative approximate error

Lastly,
Draw six graphs from above solution.
In graph 1: the graph of x_m and relative approximation error (bisection).
In graph 2: the graph of no of iteration and relative approximation error (bisection).
In graph 3: the graph of x_r and relative approximation error (false position).
In graph 4: the graph of no of iteration and relative approximation error (false position).
In graph 5: Compare the relative approximate error with respect to number of iterations between the bisection method and false position method. For comparison, you need to draw the graph of number of iteration and relative approximation error.
In graph 6: Compare the relative approximate error with respect to *x* between the bisection
method and false position method. For comparison, you need to draw the graph of *x* and
relative approximation error.

Solution 1:

```python
#!/usr/bin/env python3
import matplotlib.pyplot as plt

import numpy as np

import csv

def func(m):

   immidiate = (9.8*m)/15

   imi2=-(15/m)*9

   return immidiate*(1-np.exp(imi2))-35

def funciton_value(x_start,x_end):

   print("\t\tx\t\t\tf(x)\n","-"*25)

   while x_start<x_end:

      print("{:12f}{:12f}".format(x_start,func(x_start)))

      x_start=x_start+.1

def univariate(x_value, y_value, graph_info , x1_value=None,y1_value=None):

   if x1_value is not None:

      plt.plot(x_value, y_value,color='blue',label="bisection")

      plt.plot(x1_value,y1_value,color='red',label="false position")

   else:

      plt.plot(x_value, y_value,label=graph_info["method"])

   plt.xlabel(graph_info["x_label"])

   plt.ylabel(graph_info["y_label"])

   plt.axhline(y=0, color='k')

   plt.axvline(x=0, color='k')

   plt.grid()

   plt.title(graph_info["method"]+"\n"+graph_info["title"])

   plt.legend()

   plt.savefig(graph_info["method"]+" "+graph_info["title"]+".png", dpi=100)

   plt.show()

def bisection(a,lo,hi):

   print("Bisection: ")

   xm=0

   Xm=[]
```

```
    Iteration=[]
    error=[]
    if(func(lo)*func(hi)>0):
        print("Value does not exists ")
        return
    else:
        print("Root exists")
    iter_count=0
    print("{}\t{}\t  {}\t\t{}\t\t\t{}\t\t{}"
        "".format("Itr","Upper value","Lower value","Xm","f(Xm)","Relative error"))
    print("-" * 60)
    with open("data.csv", 'w') as csvfile:
        data = csv.writer(csvfile, delimiter=',')
        while (abs(lo - hi) > a):
            iter_count = iter_count + 1
            xo = xm
            xm = (lo + hi) / 2
            y = func(xm)
            rel_error = round((abs((xm - xo) / xm) * 100), 5);
            Xm.append(round(xm, 5))
            Iteration.append(iter_count)
            error.append(rel_error)
            data.writerow([xm, rel_error])
            print("{:2d} {:12f} {:12f} {:12f} {:15.3e} {:15f}".format(iter_count,
round(hi, 5),
                                                   round(lo, 5), round(xm, 5), round(y, 15),
                                                   round((abs((xm - xo) / xm) * 100), 5)))
            if (y * func(hi) > 0):
                hi = xm
            else:
                lo = xm
        print("\nRoot is : ", xm,"\n")
        return Xm, Iteration, error
def false_position(a,lo,hi):
```

```python
    print("False position: ")
    Xm = []
    Iteration = []
    error = []
    xm = 0
    if(func(lo)*func(hi)>0):
        print("Value does not exists ")
        return
    iter_count=0
    print("{}\t{}\t  {}\t\t{}\t\t\t{}\t\t{}"
          "".format("Itr", "Upper value", "Lower value", "Xm", "f(Xm)", "Relative
error"))
    print("-" * 60)
    while(True):
        iter_count=iter_count+1
        xp=xm
        fx1=func(hi)
        fx0=func(lo)
        xm=-((fx0*(lo-hi))/(fx0-fx1))+lo
        y=func(xm)
        rel_error = round((abs((xm-xp)/xm)*100),5)
        Xm.append(round(xm, 5))
        Iteration.append(iter_count)
        error.append(rel_error)
        print("{:2d} {:12f} {:12f} {:12f} {:15.3e} {:15f}".format(iter_count,round(hi,5),
                round(lo,5),round(xm,5),round(y,15),round((abs((xm-xp)/xm)*100),5)))
        if(y>0):
            hi=xm
        else:
            lo=xm
        if(abs((xm-xp)/xm)<a):
            print("\nRoot is : ",xm)
            break
    return Xm, Iteration, error
```

```python
funciton_value(55,70)

all_input = input("Give a, low ,high\n")

a = float(all_input.split(' ')[0])

lo = float(all_input.split(' ')[1])

hi = float(all_input.split(' ')[2])

xm,iter_b,error_b=bisection(a,lo,hi)

univariate(x_value=xm, y_value=error_b,graph_info={"x_label": "Xm", "y_label": "Relative error",

                 "title":"Xm Vs Relative error","method":"Bisection"})

univariate(x_value=iter_b,y_value=error_b,graph_info= {"x_label": "Iteration", "y_label": "Relative error",

                  "title":"Iteration Vs Relative error","method":"Bisection"})

xr,iter_f,error_f=false_position(a,lo,hi)

univariate(x_value=xr,y_value=error_f, graph_info={"x_label": "Xr", "y_label": "Relative error",

                 "title":"Xr Vs Relative error","method":"False Position"})

univariate(x_value=iter_f,y_value= error_f,graph_info={"x_label": "Iteration", "y_label": "Relative error",

                 "title":"Iteration Vs Relative error","method":"False Position"})

univariate(x_value=iter_b,y_value=error_b,graph_info={"x_label":"Iteration","y_label":"Relative error","title":

                 "Bisection & False Position","method":"Compare"},x1_value=iter_f,y1_value=error_f)

univariate(x_value=xm,y_value=error_b,graph_info={"x_label":"X","y_label":"Relative error","title":

                 "Bisection & False Position","method":"Compare2"},x1_value=xr,y1_value=error_f)
```

Sample Input output:

(a):value of x and f(x) from user lower input and user upper input, increasing by 0.1

```
       x            f(x)
 ------------------------
  55.000000   -2.153420
  55.100000   -2.107506
  55.200000   -2.061652
  55.300000   -2.015860
  55.400000   -1.970130
  55.500000   -1.924460
  55.600000   -1.878852
  55.700000   -1.833305
  55.800000   -1.787819
  55.900000   -1.742393
  56.000000   -1.697029
  56.100000   -1.651726
  56.200000   -1.606483
  56.300000   -1.561301
  56.400000   -1.516180
  56.500000   -1.471120
  56.600000   -1.426120
  56.700000   -1.381180
  56.800000   -1.336301
  56.900000   -1.291482
  57.000000   -1.246723
  57.100000   -1.202025
  57.200000   -1.157387
  57.300000   -1.112808
  57.400000   -1.068290
  57.500000   -1.023832
  57.600000   -0.979433
  57.700000   -0.935095
```

(b): Bisection Method

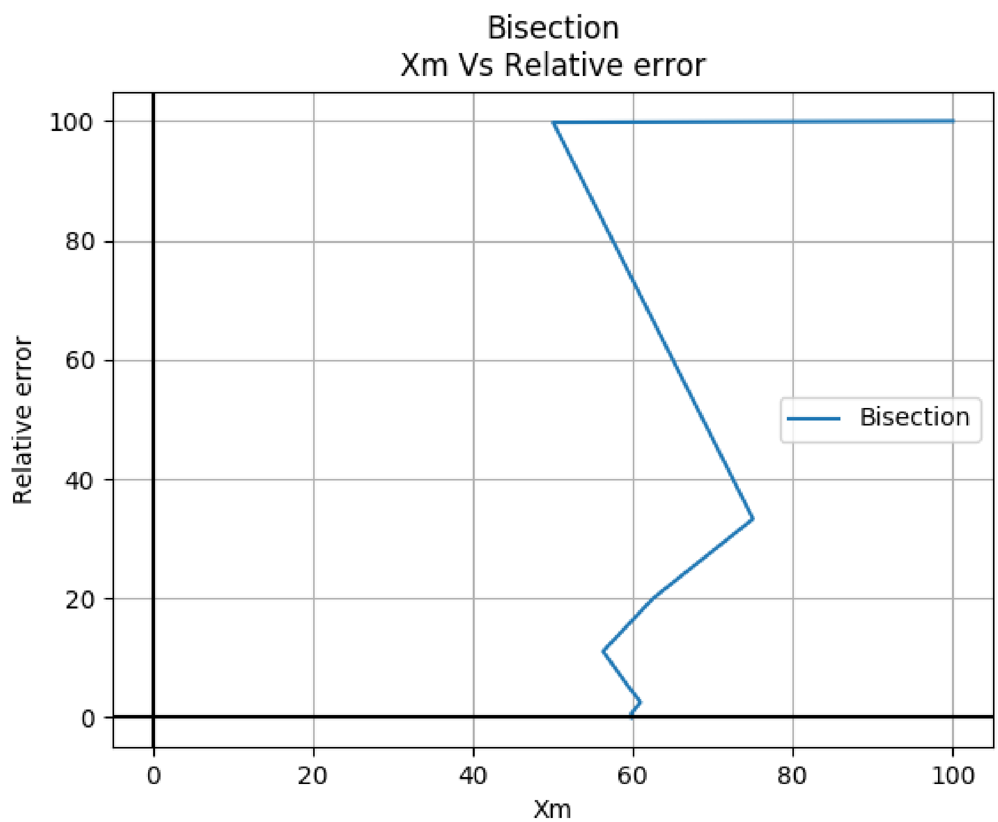
```
Give a, low ,high
>? .00001 .1 200
Bisection:
Root exists
Itr Upper value  Lower value      Xm          f(Xm)       Relative error
------------------------------------------------------------
 1   200.000000    0.100000  100.050000    1.341e+01     100.000000
 2   100.050000    0.100000   50.075000   -4.492e+00      99.800300
 3   100.050000   50.075000   75.062500    5.922e+00      33.288930
 4    75.062500   50.075000   62.568750    1.153e+00      19.968040
 5    62.568750   50.075000   56.321870   -1.551e+00      11.091380
 6    62.568750   56.321870   59.445310   -1.708e-01       5.254300
 7    62.568750   59.445310   61.007030    4.980e-01       2.559900
 8    61.007030   59.445310   60.226170    1.654e-01       1.296540
 9    60.226170   59.445310   59.835740   -2.283e-03       0.652500
10    60.226170   59.835740   60.030960    8.164e-02       0.325190
11    60.030960   59.835740   59.933350    3.971e-02       0.162860
12    59.933350   59.835740   59.884550    1.872e-02       0.081500
13    59.884550   59.835740   59.860140    8.220e-03       0.040760
14    59.860140   59.835740   59.847940    2.969e-03       0.020390
15    59.847940   59.835740   59.841840    3.435e-04       0.010190
16    59.841840   59.835740   59.838790   -9.696e-04       0.005100
17    59.841840   59.838790   59.840320   -3.130e-04       0.002550
18    59.841840   59.840320   59.841080    1.521e-05       0.001270
19    59.841080   59.840320   59.840700   -1.489e-04       0.000640
20    59.841080   59.840700   59.840890   -6.685e-05       0.000320
21    59.841080   59.840890   59.840980   -2.582e-05       0.000160
22    59.841080   59.840980   59.841030   -5.301e-06       0.000080
23    59.841080   59.841030   59.841060    4.957e-06       0.000040
24    59.841060   59.841030   59.841040   -1.720e-07       0.000020
25    59.841060   59.841040   59.841050    2.393e-06       0.000010

Root is :   59.84105030596254
```

(c): False Position Method

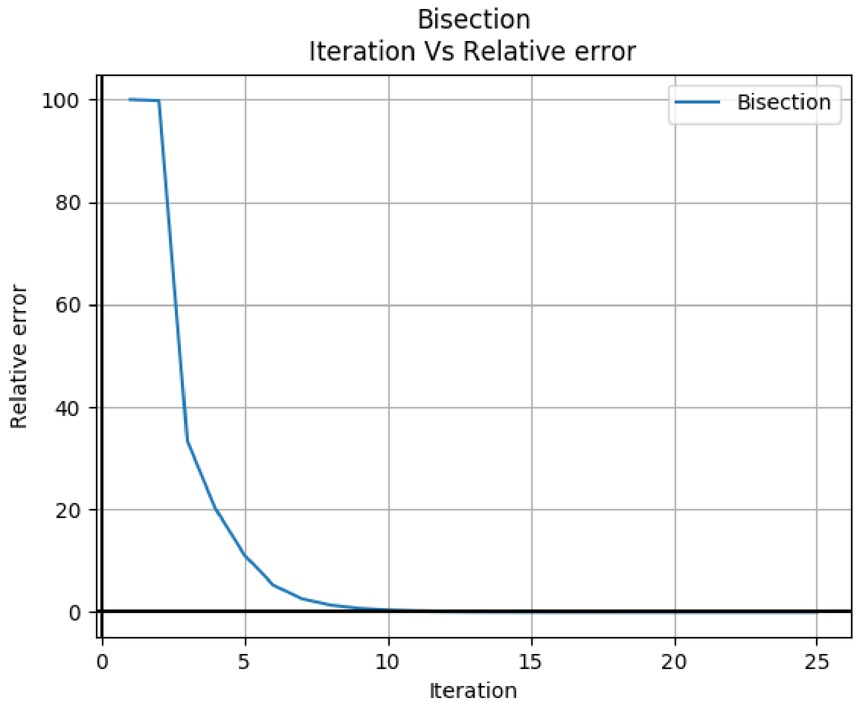
```
False position:
Itr Upper value   Lower value      Xm          f(Xm)       Relative error
-----------------------------------------------------------
 1   200.000000     0.100000   109.094380      1.560e+01      100.000000
 2   109.094380     0.100000    75.452860      6.059e+00       44.586150
 3    75.452860     0.100000    64.315760      1.869e+00       17.316280
 4    64.315760     0.100000    61.054680      5.182e-01        5.341240
 5    61.054680     0.100000    60.163810      1.386e-01        1.480750
 6    60.163810     0.100000    59.926390      3.672e-02        0.396170
 7    59.926390     0.100000    59.863580      9.699e-03        0.104930
 8    59.863580     0.100000    59.846990      2.560e-03        0.027720
 9    59.846990     0.100000    59.842610      6.756e-04        0.007320
10    59.842610     0.100000    59.841460      1.783e-04        0.001930
11    59.841460     0.100000    59.841150      4.704e-05        0.000510

Root is :  59.841154031018206
```

Graphs:

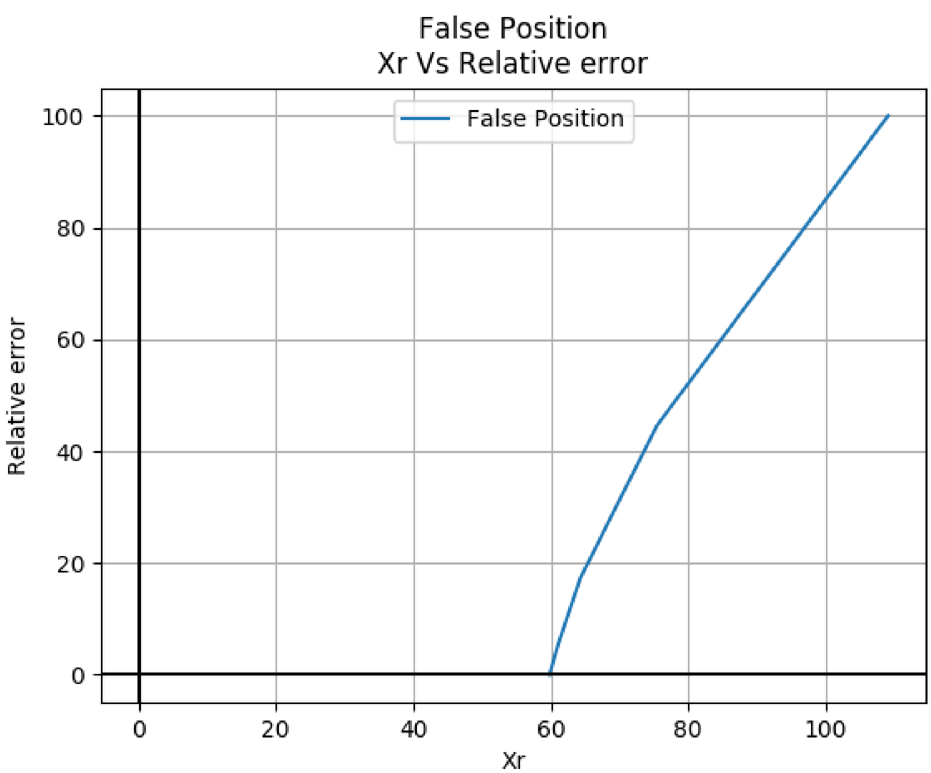
Graph 1: Graph of Xm and relative approximation error (bisection)



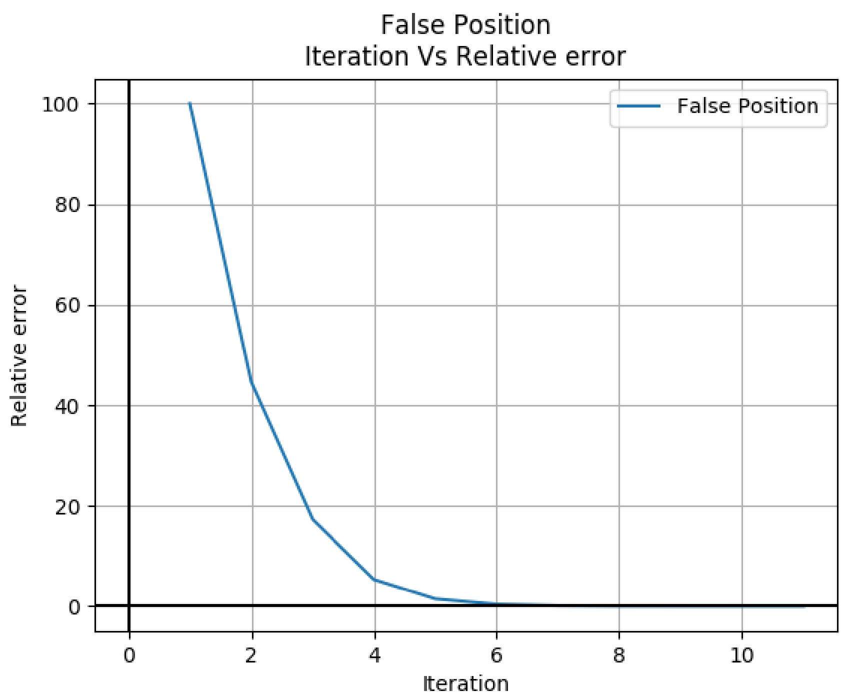
Graph 2: Graph of no of iteration and relative approximation error (bisection)



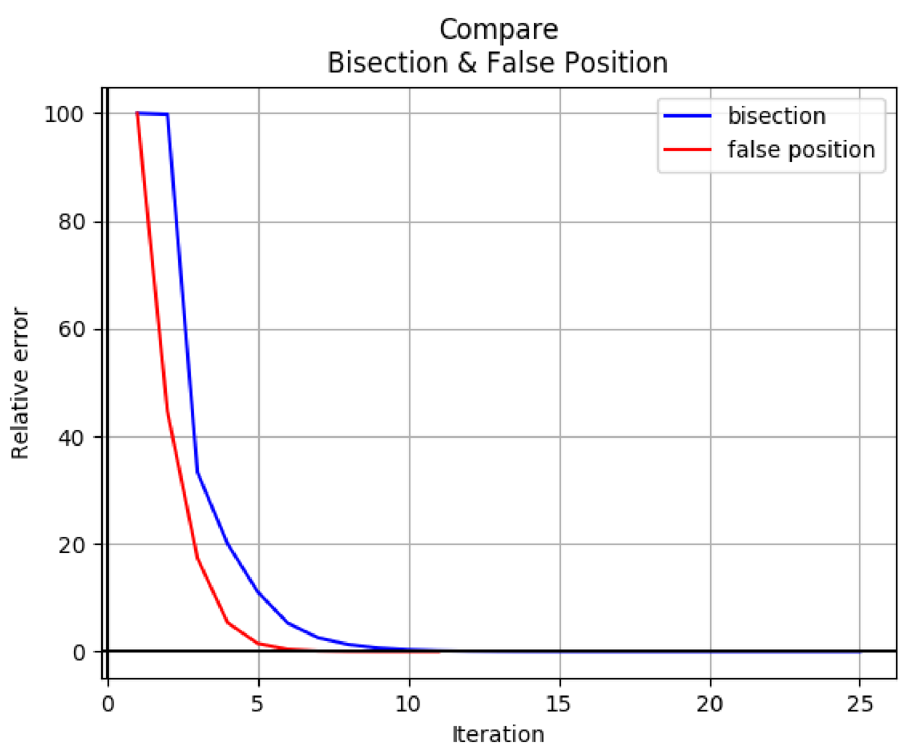
Graph 3: Graph of Xr and relative approximation error (false position)



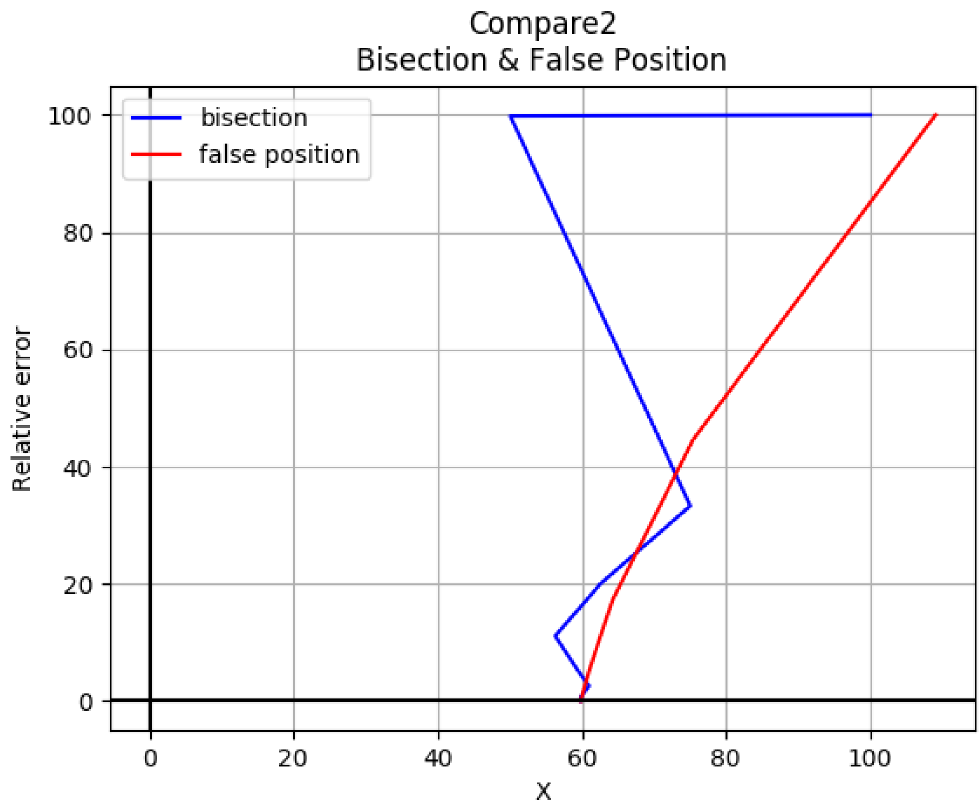
Graph 4: Graph of no of iteration and relative approximation error (false position)



Graph 5: Compare the relative approximate error with respect to number of iteration between the bisection method and false position method



Graph 6: Compare the relative approximate error with respect to x between the bisection method and false position method



Problem Statement 2

Write a single program (source **file name must be** problem2. extension) to

solve the following

(a) Use the Newton-Raphson method to determine a root of $f(x) = -x^2 + 1.8x + 2.5$ using $x_0 = 5$.
Perform the computation until εa is less than user specified tolerance.
Also perform an error check of your final answer as the following table.
(b) Use the Newton-Raphson method to find the root of

$$f(x) = e^{-0.5x}(4 - x) - 2$$

Employ initial guesses of (i) 2, (ii) 6, and (iii) 8.
Explain your results.
You also need to print the following table in your console view.

iteration	xi	f(xi)	f'(xi)	Relative approximate error

Solution:

```python
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
def func1(x):
    return -x**2+1.8*x+2.5
def derivative1(x):
    return -2*x+1.8
def func2(x):
    return np.exp(-.5*x)*(4-x)-2
def derivative2(x):
    return np.exp(-.5*x)*(.5*x-3)
def graph_plot(x_start,x_end):
    y = []
    x_list = []
    x =x_start
    while x <= x_end:
        y.append(derivative2(x))
        x_list.append(x)
        x = x+.1
    plt.plot(x_list,y)
    plt.ylabel("Derivative value")
    plt.xlabel("x value")
    plt.title("X vs Derivative")
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.savefig("analysis"+".png", dpi=100)
    plt.show()
    return
def newton_raphson(func,derivative,x0,a,iteration):
    itr_count=0
    print(x0)
```

```
    print("{}\t\t\t{}\t\t{}\t\t{}\t\t\t{}"
        "".format("Itr", "Xi", "f(Xi)", "f'(Xi)", "Relative error"))
    print("-"*30,"Xo =",x0,"-"*30)
    while True:
        x1 = x0 - (func(x0) / derivative(x0))
        itr_count = itr_count + 1
        rel_error = abs((x0 - x1) / x1)
        if(itr_count==1):
            print("{:2d} {:12f} {:12f} {:12f}".format(itr_count, round(x1, 5),
            round(func(x1), 5), round(derivative(x1), 15)))
        else:
            print("{:2d} {:12f} {:12f} {:12f} {:15f}".format(itr_count, round(x1, 5),
            round(func(x1), 5), round(derivative(x1), 15),round(rel_error*100, 5)))
        if rel_error < a:
            print("\nRoot is : ",x1,"\n")
            break
        if itr_count == iteration:
            print("Max number of iteration completed")
            break
        x0 = x1
print("-x^2+1.8x+2.5")
all_input = input("Give Xo, Accuracy\n")
Xo = float(all_input.split(' ')[0])
a = float(all_input.split(' ')[1])
newton_raphson(func1,derivative1,Xo,a,100)
print("-"*60)
print("e^(-.5x)(4-x)-2")
newton_raphson(func2,derivative2,2,a,100)
newton_raphson(func2,derivative2,4,a,100)
graph_plot(-3,20)
```

Sample input output:

(a) Equation 1, Using initial guess 5

```
-x^2+1.8x+2.5
Give Xo, Accuracy
>? 5 .00001
5.0
Itr         Xi      f(Xi)       f'(Xi)         Relative error
------------------------------ Xo = 5.0 ----------------------------
 1     3.353660    -2.710440    -4.907317
 2     2.801330    -0.305060    -3.802665       19.716560
 3     2.721110    -0.006440    -3.642217        2.948200
 4     2.719340    -0.000000    -3.638683        0.064980
 5     2.719340    -0.000000    -3.638681        0.000030

Root is :  2.7193405398662276
```

(b). Equation 2

Using the initial guess of 2:

```
------------------------------------------------------------
e^(-.5x)(4-x)-2
2
Itr         Xi      f(Xi)       f'(Xi)         Relative error
------------------------------ Xo = 2 ---------------------------
 1     0.281720     1.229740    -2.483483
 2     0.776890     0.185630    -1.770927       63.737550
 3     0.881710     0.006580    -1.646776       11.888410
 4     0.885700     0.000010    -1.642207        0.451090
 5     0.885710     0.000000    -1.642201        0.000630

Root is :  0.8857088019940231
```

Using the initial guess of 6:

```
6
Itr         Xi      f(Xi)       f'(Xi)         Relative error
------------------------------ Xo = 6 ------------------------------
 1         inf         nan          nan
 2         nan         nan          nan             nan
 3         nan         nan          nan             nan
 4         nan         nan          nan             nan
 5         nan         nan          nan             nan
 6         nan         nan          nan             nan
 7         nan         nan          nan             nan
 8         nan         nan          nan             nan
 9         nan         nan          nan             nan
10         nan         nan          nan             nan
```

we can’t find root using newton-raphson method.

Using the initial guess of 8:

```
8
Itr         Xi      f(Xi)       f'(Xi)         Relative error
------------------------------ Xo = 8 ------------------------------
 1   121.196300    -2.000000     0.000000
 2 7212131452880262772293632.000000    -2.000000     0.000000      100.000000
 3          inf          nan          nan             nan
 4          nan          nan          nan             nan
 5          nan          nan          nan             nan
 6          nan          nan          nan             nan
 7          nan          nan          nan             nan
 8          nan          nan          nan             nan
```

Explanation:

Using this method, we get valid roots for all the given guess points but do not get expected result for 6 and 8. For initial guess 6 we do not get proper root since f’(6) = 0, and Newton-Raphson method uses f(x)/f’(x) to get the next approximation. As we can see from the plot of the function, the slope of f(x) at 8 is very close to zero. So we do not get approximation using this method.

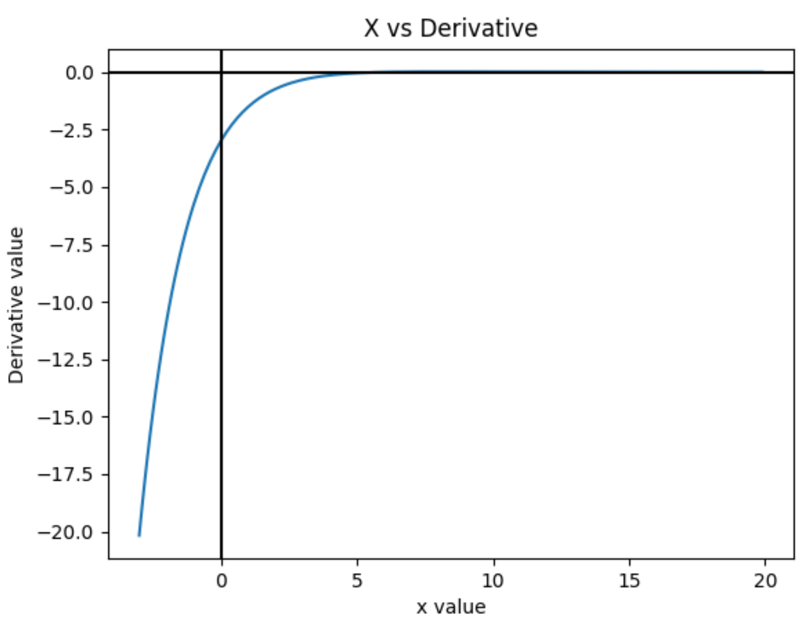


Fig: Slope of the function

Problem Statement 3

Write a single program (source **file name must be** problem3. extension) to solve the following

(a) Consider following easily differentiable function,

$$f(x) = 8\sin(x)e^{-x} - 1$$

Use the secant method, when initial guesses of $x_{i-1} = 0.5$ and $x_i = 0.4$ with user specified
tolerance. You also need to print the following table in your console view.

iteration	Upper value	Lower value	Xm	f(Xm)	Relative approximate error

Solution:

```
#!/usr/bin/env python3
import numpy as np

def func(x):

   return 8*np.sin(x)*np.exp(-x)-1

def secant(x0,x00,a,iteration):

   print("{}\t\t{}\t\t\t{}\t\t{}\t\t\t{}\t\t{}"

       "".format("Itr", "Upper", "Lower", "Xm", "f(Xm)","Relative error"))

   print("-" * 70)

   itr_count=0

   while True:

      x1 = x0 - (func(x0) * (x0 - x00)) / (func(x0) - func(x00))

      itr_count = itr_count + 1

      rel_error = abs((x0 - x1) / x1)

      if(itr_count==1):

         print("{:2d} {:12f} {:12f} {:12f} {:15f}".format(itr_count, round(x0, 5),

         round(x00, 5), round(x1, 15), round(func(x1), 15)))

      else:

         print("{:2d} {:12f} {:12f} {:12f} {:15f} {:15f}".format(itr_count, round(x0,
5),

          round(x00, 5), round(x1, 15), round(func(x1), 15),round(rel_error * 100, 5)))

      if rel_error < a:

         break
```

```python
        if itr_count == iteration:
            print("Max number of iteration completed")
            break
        x00 = x0
        x0 = x1
    print("Root is : ",x1)
print("8sin(x)e^(-x)-1")
all_input = input("Give Xo,Xo-1 ,Accuracy\n")
Xo = float(all_input.split(' ')[0])
Xoo = float(all_input.split(' ')[1])
accuracy=float(all_input.split(' ')[2])
secant(Xo,Xoo,accuracy,200)
```

Sample input output:

```
8sin(x)e^(-x)-1
Give Xo,Xo-1 ,Accuracy
>? .5 .4 .00001
Itr     Upper           Lower       Xm          f(Xm)       Relative error
----------------------------------------------------------------------
 1     0.500000     0.400000    -0.057239      -1.484624
 2    -0.057240     0.500000     0.237075       0.482310      124.143980
 3     0.237070    -0.057240     0.164906       0.113625       43.763260
 4     0.164910     0.237070     0.142665      -0.013780       15.590130
 5     0.142660     0.164910     0.145070       0.000325        1.658300
 6     0.145070     0.142660     0.145015       0.000001        0.038170
 7     0.145010     0.145070     0.145015      -0.000000        0.000110
Root is :  0.14501481252318532
```