Ocean Lu
CS 3010
Professor
11/04/19

Programming Project 2

Please view the link for source code:
https://colab.research.google.com/drive/1LFAN_636FjU95JrHzYgltZuvYlvwxWcO

Bisection Method

My code for this method looks like:

```
# Bisection Method

def f(x):
    return (x**3 + 2*x**2 + 10*x - 20)

def bisection_method(a, b, tol):
    counter = 1
    print("Bisection Method:")
    if f(a)*f(b) > 0:
        print("No root found.")
    else:
        while (b - a)/2.0 > tol:
            midpoint = (a + b)/2.0
            if f(midpoint) == 0:
                return(midpoint)
            elif f(a)*f(midpoint) < 0:
                b = midpoint
            else:
                a = midpoint
            print("Step", counter)
            print("Value of Root:", midpoint)
            print("Value of Function:", f(midpoint))
            print("Error:", (b - a)/2.0, "\n")
            counter = counter + 1
        return(midpoint)

a = 1
b = 2
E = 0.000005
answer = bisection_method(a, b, E)

print("Final answer:", answer)
```

The output looks like:

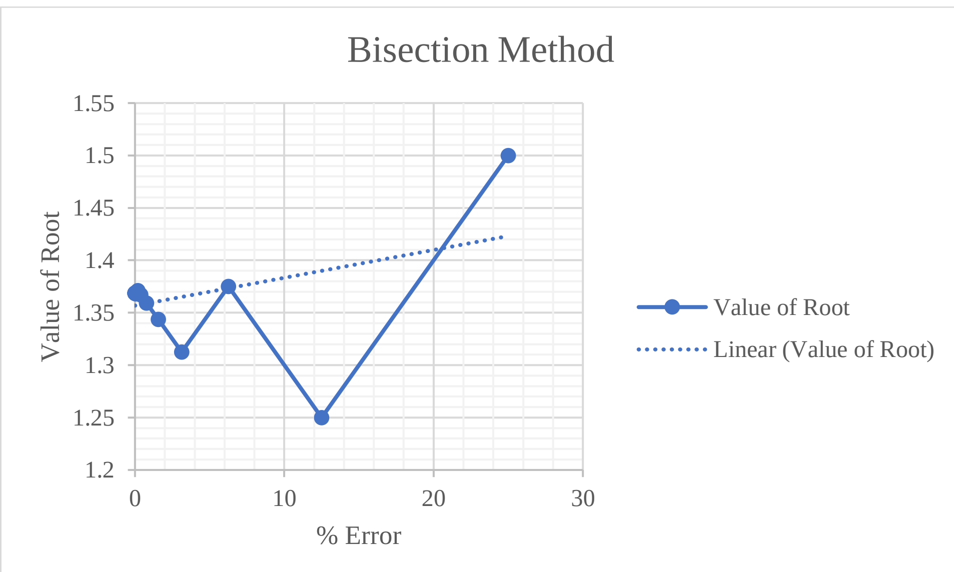
```
Bisection Method:
Step 1
Value of Root: 1.5
Value of Function: 2.875
Error: 0.25

Step 2
Value of Root: 1.25
Value of Function: -2.421875
Error: 0.125

Step 3
Value of Root: 1.375
Value of Function: 0.130859375
Error: 0.0625

Step 4
Value of Root: 1.3125
Value of Function: -1.168701171875
Error: 0.03125

Step 5
Value of Root: 1.34375
Value of Function: -0.524810791015625
Error: 0.015625

Step 6
Value of Root: 1.359375
Value of Function: -0.19845962524414062
Error: 0.0078125

Step 7
Value of Root: 1.3671875
Value of Function: -0.03417253494262695
Error: 0.00390625

Step 8
Value of Root: 1.37109375
Value of Function: 0.04825013875961304
Error: 0.001953125

Step 9
Value of Root: 1.369140625
Value of Function: 0.007015503942966461
Error: 0.0009765625
```

```
Step 10
Value of Root: 1.3681640625
Value of Function: -0.013584337197244167
Error: 0.00048828125

Step 11
Value of Root: 1.36865234375
Value of Function: -0.003285872400738299
Error: 0.000244140625

Step 12
Value of Root: 1.368896484375
Value of Function: 0.001864451784058474
Error: 0.0001220703125

Step 13
Value of Root: 1.3687744140625
Value of Function: -0.000710801299646846
Error: 6.103515625e-05

Step 14
Value of Root: 1.36883544921875
Value of Function: 0.0005768024936969596
Error: 3.0517578125e-05

Step 15
Value of Root: 1.368804931640625
Value of Function: -6.70050900168917e-05
Error: 1.52587890625e-05

Step 16
Value of Root: 1.3688201904296875
Value of Function: 0.0002548972800688887
Error: 7.62939453125e-06

Step 17
Value of Root: 1.3688125610351562
Value of Function: 9.394573958587671e-05
Error: 3.814697265625e-06

Final answer: 1.3688125610351562
```

Table of the results for Bisection method:

n	Value of Root	Value of Function	Error	% Error
1	1.5	2.875	0.25	25
2	1.25	-2.421875	0.125	12.5
3	1.375	0.130859375	0.0625	6.25
4	1.3125	-1.168701171875	0.03125	3.125
5	1.34375	-0.524810791015625	0.015625	1.5625
6	1.359375	-0.19845962524414062	0.0078125	0.78125
7	1.3671875	-0.03417253494262695	0.00390625	0.390625
8	1.37109375	0.04825013875961304	0.001953125	0.1953125
9	1.369140625	0.007015503942966461	0.0009765625	0.09765625
10	1.3681640625	-0.013584337197244167	0.00048828125	0.048828125
11	1.36865234375	-0.003285872400738299	0.000244140625	0.024414063
12	1.368896484375	0.001864451784058474	0.0001220703125	0.012207031
13	1.3687744140625	-0.000710801299646846	6.103515625e-05	0.006103516
14	1.36883544921875	0.0005768024936969596	3.0517578125e-05	0.003051758
15	1.368804931640625	-6.70050900168917e-05	1.52587890625e-05	0.001525879
16	1.3688201904296875	0.0002548972800688887	7.62939453125e-06	0.000762939
17	1.3688125610351562	9.394573958587671e-05	3.814697265625e-06	0.00038147

Plot:



False Position Method

My code for this method looks like:

```
# False Position Method

import math
import numpy as np

def f(x):
    return (x**3 + 2*x**2 + 10*x - 20)

def regulaFalsi(a,b,TOL):
    print("False Position Method:")
    i = 1
    FA = f(a)
    while True:
        p = (a*f(b)-b*f(a))/(f(b) - f(a))
        FP = f(p)
        if(FP == 0 or np.abs(f(p)) < TOL):
            break
        else:
            print("Step", i)
            print ("Value of Root:", a)
            print("Value of Function:",f(a))
            print("Error:", np.abs(f(p)), "\n")
        i = i + 1
        if(FA*FP > 0):
            a = p
        else:
            b = p
    print("Final answer: ", a)

a = 1
b = 2
E = 0.000005
regulaFalsi(a, b, E)
```

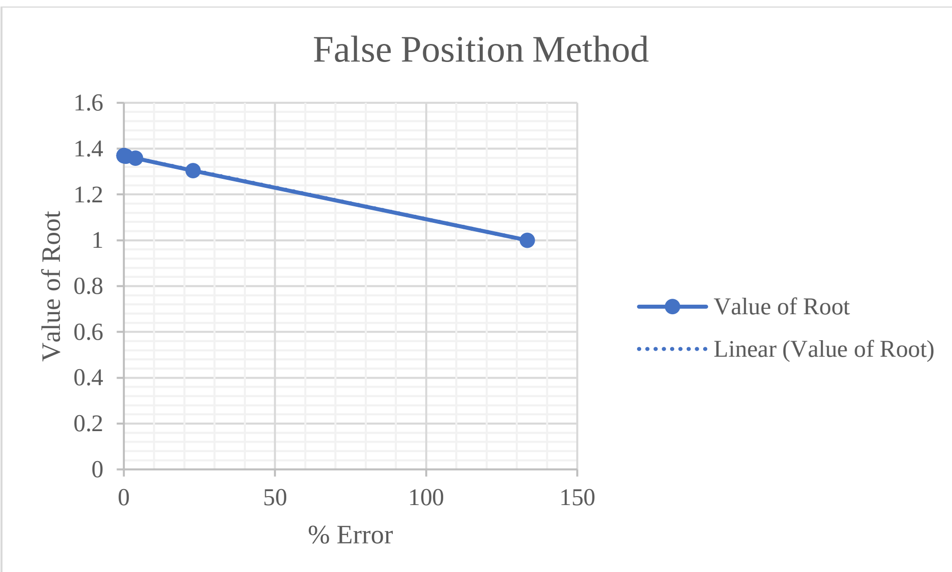
The output looks like:

```
False Position Method:
Step 1
Value of Root: 1
Value of Function: -7
Error: 1.3347579518369344

Step 2
Value of Root: 1.3043478260869565
Value of Function: -1.3347579518369344
Error: 0.22913572958733397

Step 3
Value of Root: 1.3579123046578667
Value of Function: -0.22913572958733397
Error: 0.03859187677837639

Step 4
Value of Root: 1.3669778048165133
Value of Function: -0.03859187677837639
Error: 0.006478728147058632

Step 5
Value of Root: 1.3685009755999702
Value of Function: -0.006478728147058632
Error: 0.001087042825339779

Step 6
Value of Root: 1.368756579007422
Value of Function: -0.001087042825339779
Error: 0.00018237436024648446

Step 7
Value of Root: 1.3687994628833735
Value of Function: -0.00018237436024648446
Error: 3.059667520943776e-05

Step 8
Value of Root: 1.3688066574760007
Value of Function: -3.059667520943776e-05
Error: 5.133145471347689e-06

Final answer:  1.3688078644997985
```

Table of the results for False Position Method:

n	Value of Root	Value of Function	Error	% Error
1	1	-7	1.3347579518369344	133.4757952
2	1.3043478260869565	-1.3347579518369344	0.22913572958733397	22.91357296
3	1.3579123046578667	-0.22913572958733397	0.03859187677837639	3.859187678
4	1.3669778048165133	-0.03859187677837639	0.006478728147058632	0.647872815
5	1.3685009755999702	-0.006478728147058632	0.001087042825339779	0.108704283
6	1.368756579007422	-0.001087042825339779	0.00018237436024648446	0.018237436
7	1.3687994628833735	-0.00018237436024648446	3.059667520943776e-05	0.003059668
8	1.3688066574760007	-3.059667520943776e-05	5.133145471347689e-06	0.000513315

Plot:



Newton Method

My code for this method looks like:

```
# Newton Method

def f(x):
    return (x**3 + 2*x**2 + 10*x - 20)
def df( x ):
    return (3*x**2 + 4*x + 10)

def newtonRaphson(x, TOL):
    i = 1
    h = f(x) / df(x)
    print("Newton Method:")
    while abs(h) >= TOL:
      h = f(x)/df(x)
      x = x - h
      print("Step", i)
      print("Value of Root:", x)
      print("Value of Function:", f(x))
      print("Error:", h, "\n")
      i = i + 1
    print("Final answer:", x)

x0 = 2
E = 0.000005
newtonRaphson(x0, E)
```

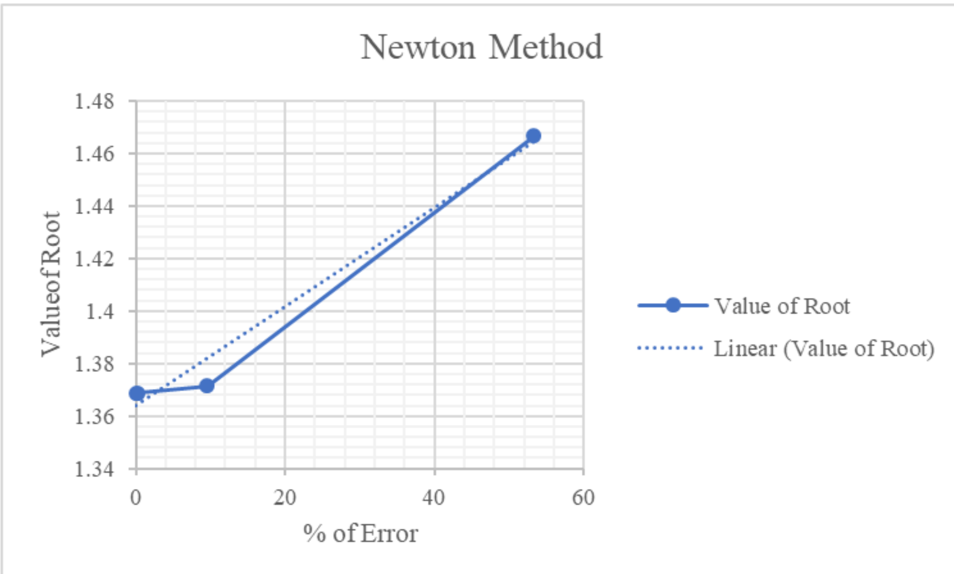
The output looks like:

```
Newton Method:
Step 1
Value of Root: 1.4666666666666668
Value of Function: 2.123851851851853
Error: 0.5333333333333333

Step 2
Value of Root: 1.3715120138059207
Value of Function: 0.05708664190432344
Error: 0.0951546528607461

Step 3
Value of Root: 1.3688102226338952
Value of Function: 4.4614406963461306e-05
Error: 0.0027017911720254935

Step 4
Value of Root: 1.3688081078226673
Value of Function: 2.731326276261825e-11
Error: 2.1148112279961915e-06

Final answer: 1.3688081078226673
```

Table of the results for Newton Method:

n	Value of Root	Value of Function	Error	% Error
1	1.4666666666666668	2.123851851851853	0.5333333333333333	53.33333333
2	1.3715120138059207	0.05708664190432344	0.0951546528607461	9.515465286
3	1.3688102226338952	4.4614406963461306e-05	0.0027017911720254935	0.270179117
4	1.3688081078226673	2.731326276261825e-11	2.1148112279961915e-06	0.000211481

Plot:



Secant Method

My code for this method looks like:

```python
# Secant method

def f(x):
    return (x**3 + 2*x**2 + 10*x - 20)

def secant(x1, x2, E):
    print("Secant Method:")
    n = 0; xm = 0; x0 = 0; c = 0;
    if (f(x1) * f(x2) < 0):
      while True:
        x0 = ((x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1)));
        c = f(x1) * f(x0);
        x1 = x2;
        x2 = x0;
        n += 1;
        print("Step", n)
        print("Value of Root:", x0)
        print("Value of Function:", c)
        if (c == 0):
          break;
        xm = ((x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1)));
        print("Error:", abs(xm - x0), "\n")
        if(abs(xm - x0) < E):
          break;
    else:
      print("Can not find a root in ", "the given inteval");
    print("Final answer:", x0)

x0 = 2;
x1 = 1;
E = 0.000005;
secant(x0, x1, E);
```

The output looks like:

```
Secant Method:
Step 1
Value of Root: 1.3043478260869565
Value of Function: -21.35612722939095
Error: 0.07170579430504076

Step 2
Value of Root: 1.3760536203919973
Value of Function: -1.0722130640657355
Error: 0.007381666860655445

Step 3
Value of Root: 1.3686719535313419
Value of Function: 0.003833714047466499
Error: 0.00013586899387907359

Step 4
Value of Root: 1.368807822525221
Value of Function: -9.218959719539044e-07
Error: 2.8530739548671136e-07

Final answer: 1.368807822525221
```

Table of the results for Secant Method:

n	Value of Root	Value of Function	Error	% Error
1	1.3043478260869565	-21.35612722939095	0.07170579430504076	7.170579431
2	1.3760536203919973	-1.0722130640657355	0.007381666860655445	0.738166686
3	1.3686719535313419	0.003833714047466499	0.00013586899387907359	0.013586899
4	1.368807822525221	-9.218959719539044e-07	2.8530739548671136e-07	2.85307E-05

Plot: