Bisection Method

Submitted by: Md. Tanvir Alam, Roll : 61

Problem 1:

Considering the following function

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-1)^k (\frac{x^2}{4})^k}{k!(n+k)!}$$

Solution 1:

```
import matplotlib.pyplot as plt
import math

class state:
    def __init__(self, x_n, x_p, x_m, err, f_xm):
        self.x_n = x_n
        self.x_p = x_p
        self.x_m = x_m
        self.err = err
        self.f_xm = f_xm


class Bisection:

    def __init__(self, x1, x2):
        if Bisection.function(0, x1)<0 < Bisection.function(0, x2):
            self.x_neg = x1
            self.x_pos = x2
        elif Bisection.function(0, x2)<0 < Bisection.function(0, x1):
            self.x_neg = x2
            self.x_pos = x1
```

```
        else:
            self.x_neg = x2
            self.x_pos = x1
            raise ValueError('No root is possible')
        self.x_mid = None

    @staticmethod
    def function(n, x):
        ans = pow(x / 2, n)
        temp = 0
        for k in range(1, 96):
            a = pow(-1, k)
            b = pow((pow(x, 2) / 4), k)
            c = math.factorial(k) * math.factorial(n + k)
            temp += a * b / c;
        return ans * temp

    @staticmethod
    def error(x_new, x_old):
        if x_old is None:
            return None
        return math.fabs((x_new-x_old)/x_new)

    def run(self):
        new_mid = (self.x_neg + self.x_pos) / 2
        error = Bisection.error(new_mid, self.x_mid)
        self.x_mid = new_mid
        f_x = Bisection.function(0, self.x_mid)
        st = state(self.x_neg,self.x_pos,self.x_mid,error,f_x)
        if f_x > 0:
            self.x_pos = self.x_mid
        else:
            self.x_neg = self.x_mid
        return st


x0 = []
x1 = []
x2 = []
```

```
x=0.0
while x <= 10.0:
    x0.append(Bisection.function(0,x))
    x1.append(Bisection.function(1,x))
    x2.append(Bisection.function(2,x))
    x += 0.1

plt.ylabel('J(0,x)')
plt.xlabel('x')
plt.plot(x0)
plt.savefig('solve1/J0x.png')
plt.show()
plt.ylabel('J(1,x)')
plt.xlabel('x')
plt.plot(x1)
plt.savefig('solve1/J1x.png')
plt.show()
plt.ylabel('J(2,x)')
plt.xlabel('x')
plt.plot(x2)
plt.savefig('solve1/J2x.png')
plt.show()

print('X     f(X)')
x = 1
while x < 3.1:
    print('%2.1f  %10.8f' % (x, Bisection.function(0, x)))
    x += 0.1

print()

bs = Bisection(float(input('Enter x1: ')), float(input('Enter x2: ')))

xm, err, fxm = [], [], []

tolerance = pow(10, -int(input('Enter Tolerance:')))
print(tolerance)
temp_tolerance = None
```

```
i = 0
print('iteration   Upper value   Lower value   Xm   f(Xm)   Relative approximate error')
while temp_tolerance is None or temp_tolerance>tolerance:
    st = bs.run()
    if st.err is not None:
        print('%3d %10.5f %10.5f %10.5f %10.5f  %10.5f '%(i + 1, st.x_n, st.x_p, st.x_m,
st.f_xm, st.err))
    else:
        print('%3d %10.5f %10.5f %10.5f %10.5f    -------  '%(i + 1, st.x_n, st.x_p, st.x_m,
st.f_xm))

    xm.append(st.x_m)
    err.append(st.err)
    fxm.append(st.f_xm)
    temp_tolerance=st.err
    i += 1


plt.ylabel('Relative approximate error')
plt.xlabel('xm')
plt.plot(xm, err)
plt.savefig('solve1/xm vs error.png')
plt.show()
plt.clf()
plt.ylabel('Relative approximate error')
plt.xlabel('Iterations')
plt.plot(err)
plt.savefig('solve1/Iterations vs error.png')
plt.show()
```

Sample Input Output

The value of the function J(0, x) is always negative for x =1 to 5 (varying k from 1 to 10). So, it will cause an exception.

```
X     f(X)
1.0   -0.23480231
1.1   -0.28037798
1.2   -0.32886726
1.3   -0.37991401
1.4   -0.43314488
1.5   -0.48817233
1.6   -0.54459783
1.7   -0.60201514
1.8   -0.66001359
1.9   -0.71818144
2.0   -0.77610922
2.1   -0.83339302
2.2   -0.88963773
2.3   -0.94446022
2.4   -0.99749232
2.5   -1.04838378
2.6   -1.09680495
2.7   -1.14244937
2.8   -1.18503603
2.9   -1.22431155
3.0   -1.26005195

Enter x1: 1
Enter x2: 3
Traceback (most recent call last):
  File "/home/tanvir/PycharmProjects/NumericLab/Lab01/solve1.py", line 93, in <module>
    bs = Bisection(float(input('Enter x1: ')), float(input('Enter x2: ')))
  File "/home/tanvir/PycharmProjects/NumericLab/Lab01/solve1.py", line 25, in __init__
    raise ValueError('No root is possible')
ValueError: No root is possible
```

However, if we ignore the exception and run the iteration, the output will look like below.

```
X     f(X)
1.0  -0.23480231
1.1  -0.28037798
1.2  -0.32886726
1.3  -0.37991401
1.4  -0.43314488
1.5  -0.48817233
1.6  -0.54459783
1.7  -0.60201514
1.8  -0.66001359
1.9  -0.71818144
2.0  -0.77610922
2.1  -0.83339302
2.2  -0.88963773
2.3  -0.94446022
2.4  -0.99749232
2.5  -1.04838378
2.6  -1.09680495
2.7  -1.14244937
2.8  -1.18503603
2.9  -1.22431155
3.0  -1.26005195

Enter x1: 1
Enter x2: 3
Enter Tolerance:4
0.0001
iteration   Upper value   Lower value   Xm   f(Xm)   Relative approximate error
  1    3.00000    1.00000    2.00000   -0.77611    -------
  2    2.00000    1.00000    1.50000   -0.48817     0.33333
  3    1.50000    1.00000    1.25000   -0.35409     0.20000
  4    1.25000    1.00000    1.12500   -0.29224     0.11111
  5    1.12500    1.00000    1.06250   -0.26293     0.05882
  6    1.06250    1.00000    1.03125   -0.24871     0.03030
  7    1.03125    1.00000    1.01562   -0.24172     0.01538
  8    1.01562    1.00000    1.00781   -0.23825     0.00775
  9    1.00781    1.00000    1.00391   -0.23652     0.00389
 10    1.00391    1.00000    1.00195   -0.23566     0.00195
 11    1.00195    1.00000    1.00098   -0.23523     0.00098
 12    1.00098    1.00000    1.00049   -0.23502     0.00049
 13    1.00049    1.00000    1.00024   -0.23491     0.00024
 14    1.00024    1.00000    1.00012   -0.23486     0.00012
 15    1.00012    1.00000    1.00006   -0.23483     0.00006

Process finished with exit code 0
```

Graphs:

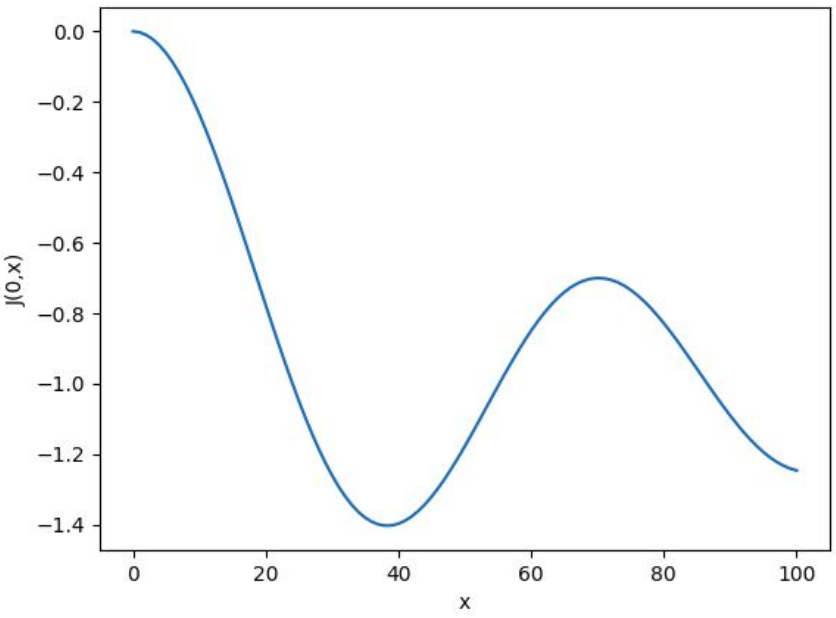

Figure 01: x vs J(0,x)

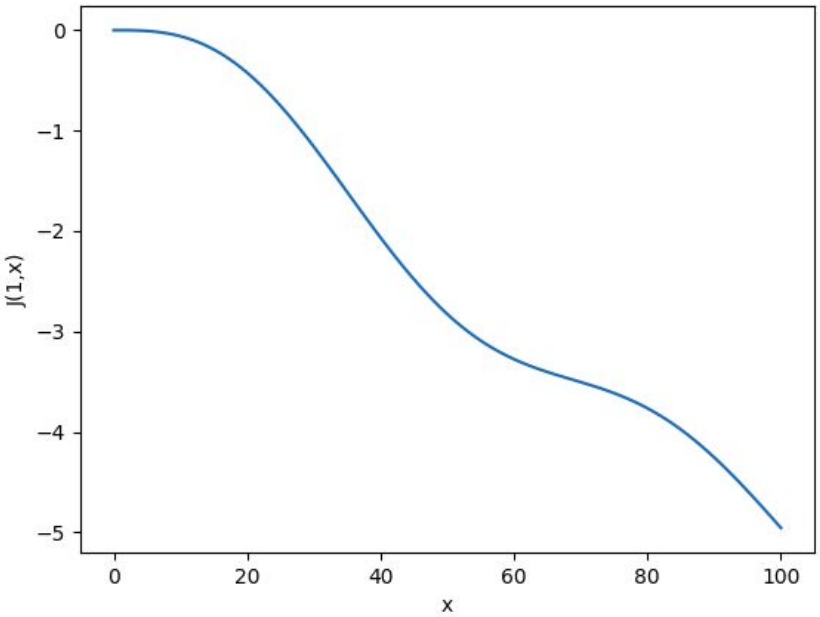

Figure 02: x vs J(1,x)

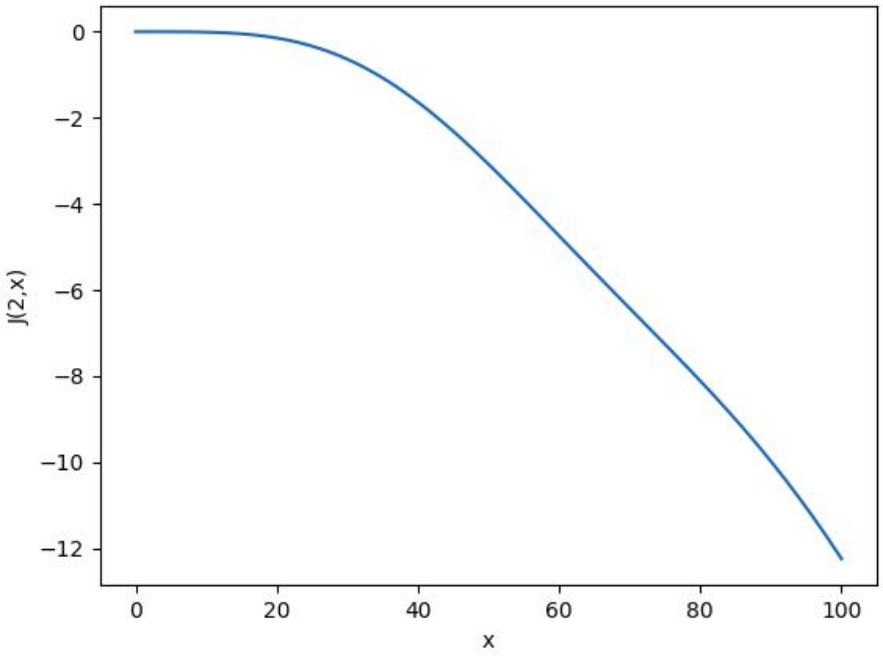

Figure 03: x vs J(2,x)

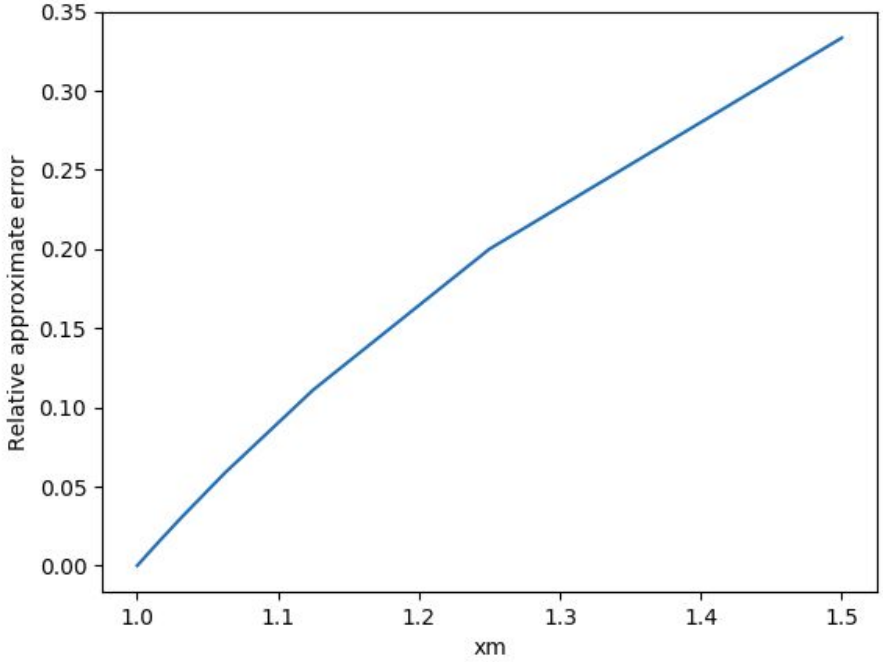

Figure 03: x vs relative approximation error.

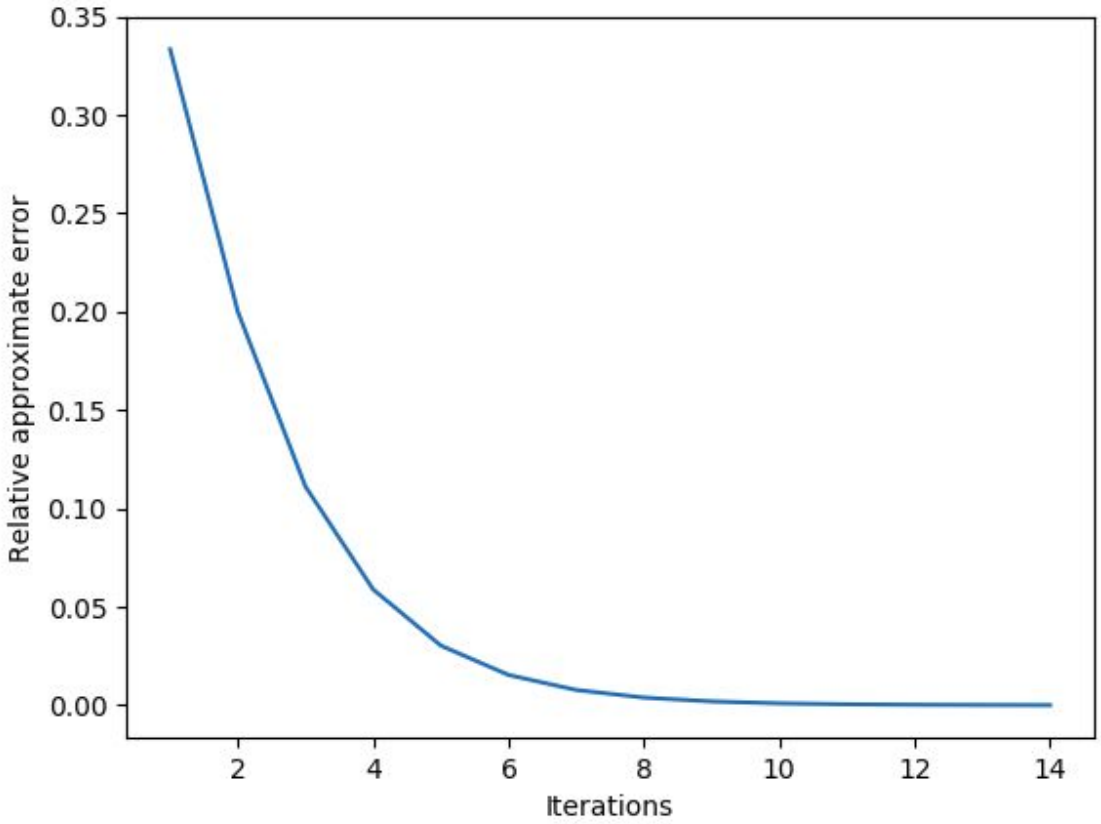

Figure 04: Iterations vs relative approximation error.

Problem 2:

Considering the following function:

$$K = \frac{(c_{c,0} + x)}{(c_{a,0} - 2x)^2 (c_{b,0} - x)}$$

Solution 2(a):

```
import matplotlib.pyplot as plt

def function(x):
    ca0 = 42
    cb0 = 28
    cc0 = 4
    k = 0.016
    return ((cc0 + x) / (pow((ca0 - 2 * x), 2) * (cb0 - x))) - k

print(' x     f(x)')

graph=[]
for i in range(1,21):
    graph.append(function(i))
    print('%2d %10.5f'%(i, function(i)))

plt.ylabel('f(x)')
plt.xlabel('x')
plt.savefig('solve2/a.png')
plt.plot(graph)
plt.show()
```

Sample Output:

x	f(x)
1	-0.01588
2	-0.01584
3	-0.01578
4	-0.01571
5	-0.01562
6	-0.01549
7	-0.01533
8	-0.01511
9	-0.01481
10	-0.01439
11	-0.01379
12	-0.01291
13	-0.01157
14	-0.00944
15	-0.00585
16	0.00067
17	0.01383
18	0.04511
19	0.14372
20	0.73400

Graph:

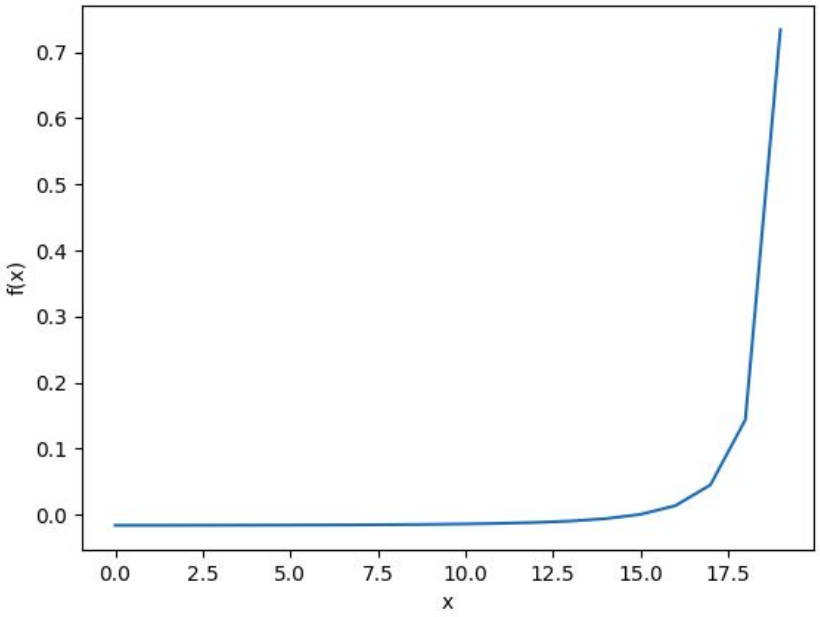

Figure 05: x vs f(x)

Solution 2(b):

```
import matplotlib.pyplot as plt
import math

class state:
    def __init__(self, x_n, x_p, x_m, err, f_xm):
        self.x_n = x_n
        self.x_p = x_p
        self.x_m = x_m
        self.err = err
        self.f_xm = f_xm


class FalsePosition:

    def __init__(self, x1, x2):
        if FalsePosition.function(x1)<0 < FalsePosition.function(x2):
            self.x_neg = x1
            self.x_pos = x2
        elif FalsePosition.function(x2)<0 < FalsePosition.function(x1):
            self.x_neg = x2
            self.x_pos = x1
        else:
            raise ValueError('Invalid Initial Value')

        self.x_mid = None

    @staticmethod
    def function(x):
        ca0 = 42
        cb0 = 28
        cc0 = 4
        k = 0.016
        return ((cc0 + x) / (pow((ca0 - 2 * x), 2) * (cb0 - x))) - k

    @staticmethod
    def error(x_new, x_old):
```

```
        if x_old is None:
            return None
        return math.fabs((x_new-x_old)/x_new)

    @staticmethod
    def get_mid(x1,x2):
        return
((-FalsePosition.function(x1)*(x1-x2))/(FalsePosition.function(x1)-FalsePosition.function(
x2)))+x1

    def run(self):
        new_mid = FalsePosition.get_mid(self.x_neg,self.x_pos)
        error = FalsePosition.error(new_mid, self.x_mid)
        self.x_mid = new_mid
        f_x = FalsePosition.function(self.x_mid)
        st = state(self.x_neg,self.x_pos,self.x_mid,error,f_x)
        if f_x > 0:
            self.x_pos = self.x_mid
        else:
            self.x_neg = self.x_mid
        return st


fp = FalsePosition(float(input('Enter x1: ')), float(input('Enter x2: ')))

xm, err, fxm = [], [], []


tolerance = pow(10, -int(input('Enter Tolerance:')))
print('iteration   Upper value   Lower value   Xm   f(Xm)   Relative approximate error')

temp_tolerance = None
i = 0
while temp_tolerance is None or temp_tolerance > tolerance:
    st = fp.run()
    print(i + 1, st.x_p, st.x_n, st.x_m, st.f_xm, st.err)
    xm.append(st.x_m)
    err.append(st.err)
    fxm.append(st.f_xm)
```

```
    temp_tolerance = st.err
    i += 1

plt.ylabel('Error')
plt.xlabel('Iteration')
plt.plot(err)
plt.savefig('fp.png')
plt.show()
```

Sample output:

```
Enter x1: 0
Enter x2: 20
Enter Tolerance:4
iteration   Upper value   Lower value    Xm    f(Xm)   Relative approximate error
  1   20.00000    0.00000    0.42455   -0.01591    -------
  2   20.00000    0.42455    0.83974   -0.01589     0.49442
  3   20.00000    0.83974    1.24575   -0.01587     0.32592
  4   20.00000    1.24575    1.64277   -0.01586     0.24167
  5   20.00000    1.64277    2.03097   -0.01584     0.19114
  6   20.00000    2.03097    2.41052   -0.01582     0.15746
  7   20.00000    2.41052    2.78160   -0.01580     0.13341
  8   20.00000    2.78160    3.14438   -0.01577     0.11537
  9   20.00000    3.14438    3.49901   -0.01575     0.10135
 10   20.00000    3.49901    3.84565   -0.01572     0.09014
 11   20.00000    3.84565    4.18445   -0.01570     0.08097
 12   20.00000    4.18445    4.51558   -0.01567     0.07333
 13   20.00000    4.51558    4.83917   -0.01563     0.06687
 14   20.00000    4.83917    5.15537   -0.01560     0.06133
 15   20.00000    5.15537    5.46432   -0.01556     0.05654
 16   20.00000    5.46432    5.76616   -0.01553     0.05235
 17   20.00000    5.76616    6.06102   -0.01549     0.04865
 18   20.00000    6.06102    6.34903   -0.01544     0.04536
 19   20.00000    6.34903    6.63033   -0.01540     0.04243
 20   20.00000    6.63033    6.90504   -0.01535     0.03978
 21   20.00000    6.90504    7.17327   -0.01530     0.03739
 22   20.00000    7.17327    7.43515   -0.01524     0.03522
 23   20.00000    7.43515    7.69080   -0.01519     0.03324
 24   20.00000    7.69080    7.94034   -0.01513     0.03143
 25   20.00000    7.94034    8.18386   -0.01506     0.02976
 26   20.00000    8.18386    8.42149   -0.01500     0.02822
 27   20.00000    8.42149    8.65334   -0.01493     0.02679
 28   20.00000    8.65334    8.87949   -0.01485     0.02547
 29   20.00000    8.87949    9.10007   -0.01478     0.02424
 30   20.00000    9.10007    9.31517   -0.01470     0.02309
 31   20.00000    9.31517    9.52489   -0.01461     0.02202
 32   20.00000    9.52489    9.72932   -0.01452     0.02101
 33   20.00000    9.72932    9.92857   -0.01443     0.02007
```

Graph:

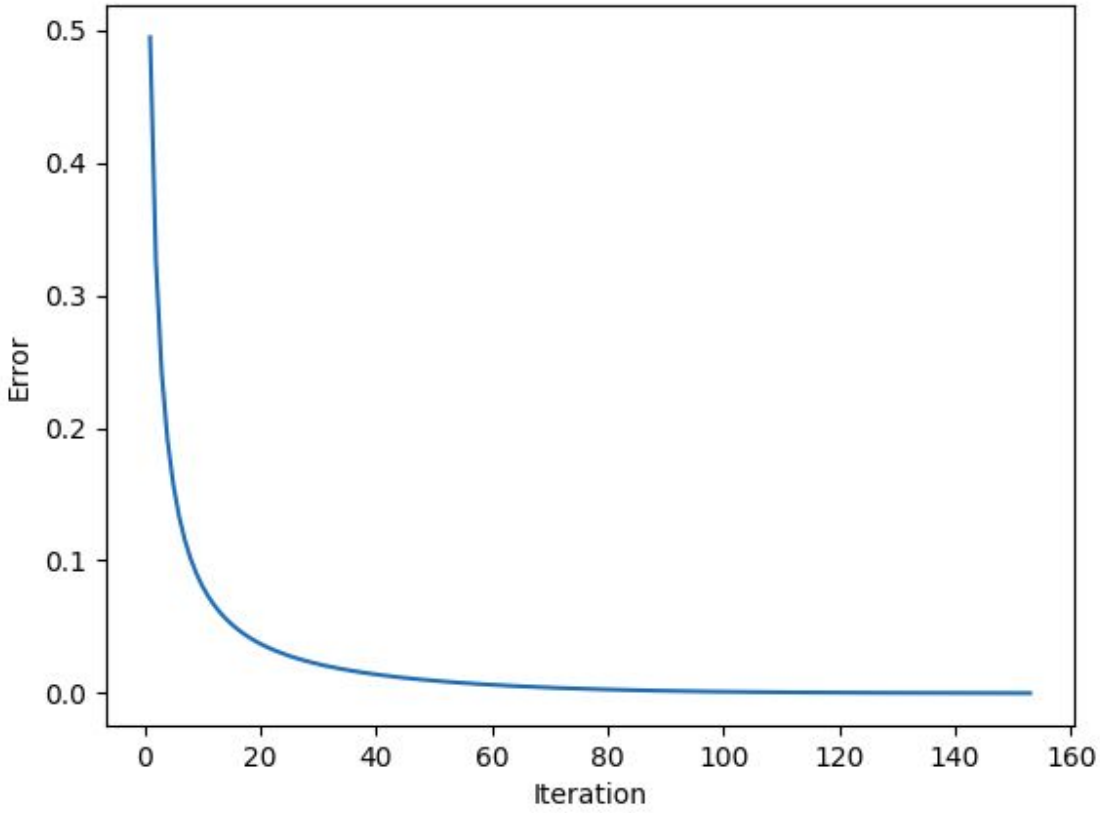

Figure 06: Iteration vs error for false position method

Solution 2(c):

```
import numpy as np
import matplotlib.pyplot as plt
import math

class state:
    def __init__(self, x_n, x_p, x_m, err, f_xm):
        self.x_n = x_n
        self.x_p = x_p
        self.x_m = x_m
        self.err = err
        self.f_xm = f_xm


class Bisection:

    def __init__(self, x1, x2):
        if Bisection.function(x1)<0 < Bisection.function(x2):
            self.x_neg = x1
            self.x_pos = x2
        elif Bisection.function(x2)<0 < Bisection.function(x1):
            self.x_neg = x2
            self.x_pos = x1
        else:
            self.x_neg = x2
            self.x_pos = x1
            '''raise ValueError('Solution Impossible')'''
        self.x_mid = None

    @staticmethod
    def function(x):
        ca0 = 42
        cb0 = 28
        cc0 = 4
        k = 0.016
        return ((cc0 + x) / (pow((ca0 - 2 * x), 2) * (cb0 - x)))-k
```

```
    @staticmethod
    def error(x_new, x_old):
        if x_old is None:
            return None
        return math.fabs((x_new-x_old)/x_new)

    def run(self):
        new_mid = (self.x_neg + self.x_pos) / 2
        error = Bisection.error(new_mid, self.x_mid)
        self.x_mid = new_mid
        f_x = Bisection.function(self.x_mid)
        st = state(self.x_neg,self.x_pos,self.x_mid,error,f_x)
        if f_x > 0:
            self.x_pos = self.x_mid
        else:
            self.x_neg = self.x_mid
        return st

bs = Bisection(float(input('Enter x1: ')), float(input('Enter x2: ')))
xm, err, fxm = [], [], []

tolerance = pow(10, -int(input('Enter Tolerance:')))
print('iteration   Upper value   Lower value   Xm   f(Xm)   Relative approximate error')
temp_tolerance = None
i = 0
while temp_tolerance is None or temp_tolerance > tolerance:
    st = bs.run()
    if st.err is not None:
        print('%3d %10.5f %10.5f %10.5f %10.5f  %10.5f '%(i + 1, st.x_p, st.x_n, st.x_m,
st.f_xm, st.err))
    else:
        print('%3d %10.5f %10.5f %10.5f %10.5f    -------  '%(i + 1, st.x_p, st.x_n, st.x_m,
st.f_xm))
    xm.append(st.x_m)
    err.append(st.err)
    fxm.append(st.f_xm)
    temp_tolerance = st.err
    i += 1
```

```
plt.ylabel('Error')
plt.xlabel('Iteration')
plt.plot(err)
plt.savefig('solve2/bs.png')
plt.show()
```

Sample Output:

```
Enter x1: 0
Enter x2: 20
Enter Tolerance:4
iteration   Upper value   Lower value   Xm   f(Xm)   Relative approximate error
  1   20.00000    0.00000   10.00000   -0.01439    -------
  2   20.00000   10.00000   15.00000   -0.00585     0.33333
  3   20.00000   15.00000   17.50000    0.02579     0.14286
  4   17.50000   15.00000   16.25000    0.00310     0.07692
  5   16.25000   15.00000   15.62500   -0.00228     0.04000
  6   16.25000   15.62500   15.93750    0.00012     0.01961
  7   15.93750   15.62500   15.78125   -0.00114     0.00990
  8   15.93750   15.78125   15.85938   -0.00052     0.00493
  9   15.93750   15.85938   15.89844   -0.00021     0.00246
 10   15.93750   15.89844   15.91797   -0.00004     0.00123
 11   15.93750   15.91797   15.92773    0.00004     0.00061
 12   15.92773   15.91797   15.92285   -0.00000     0.00031
 13   15.92773   15.92285   15.92529    0.00002     0.00015
 14   15.92529   15.92285   15.92407    0.00001     0.00008

Process finished with exit code 0
```

Graph:

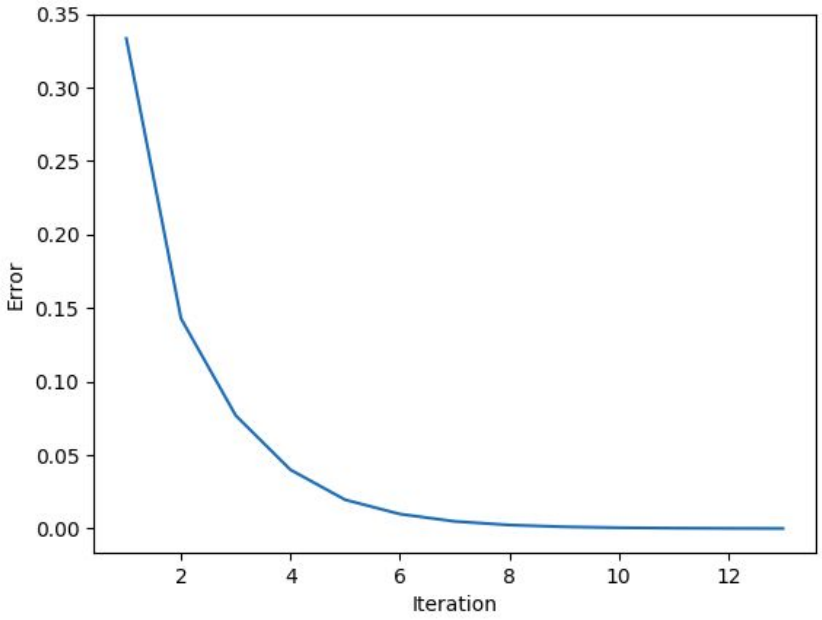

Figure 07: Iteration vs error for bisection method