Question 1

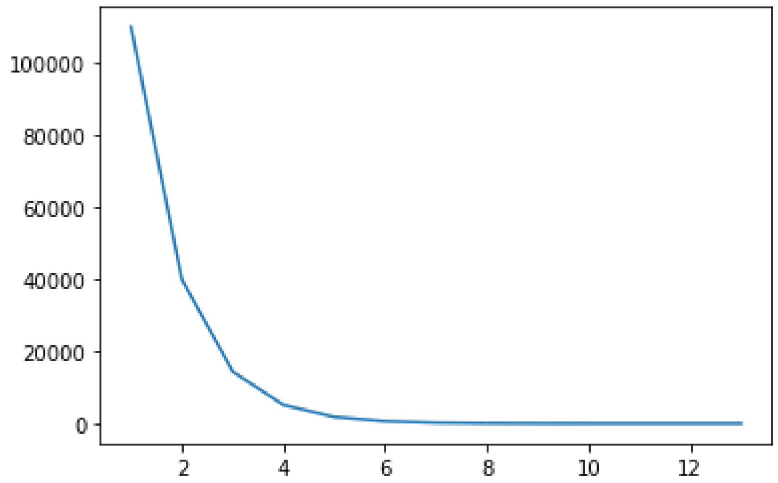
In [ ]:

```
import math
import matplotlib.pyplot as plt
from functionLibrary import newRaph


h = 0.0001 #defining the h that will be used to calculate first derivative of y using first principle
x0 = 10 #dummy variable for root, chosen arbitarirly


y = lambda x: (x-5)*math.exp(x) + 5 #defining the function
dy = lambda x: (y(x+h)-y(x-h))/(2*h) #defining the first derivative of the function using first principle

x, n = newRaph(y, dy, x0, e=0.00001, n=100) #e is taken to be 10^(-5) to ensure b has answer accurate to 10^(-4)
print("The value %f at %d iterations." %(x, n)) #printing the results
print("Now using this value of x, we determine the value of b by using the values given in the question.")
print("b = hc/kx")
b = 6.626*10**(-34)*3*10**8/(1.381*10**(-23)*float(x))
print(b)
```




```
i x_i
1 9.166628835241774
2 8.360077542515926
3 7.589162889621367
4 6.86707462434134
5 6.214045854042731
6 5.661188526380725
7 5.252697608834539
8 5.03008641018726
9 4.9691423601208315
10 4.965130682570582
11 4.965114232019877
12 4.965114231744276
The value 4.965114 at 11 iterations.
Now using this value of x, we determine the value of b by using the values given in the question.
b = hc/kx
0.0028990103307382923
```

Question 2

In [ ]:

```
from functionLibrary import determinant
from functionLibrary import inverse
from functionLibrary import matrixProduct

matrix = open("midsem-q2.txt", "r+")

M = []
for row in matrix:
    e1 = row.split()
    fe1 = []
    for i in range(len(e1)):
        fe1.append(float(e1[i]))
    M.append(fe1)

print("The determinant of the given matrix is: ")

determinant(M)

print("As this value is non-zero, the inverse exists.")
print("Now we will implement Gauss-Jordan elimination algorithm to find the inverse.")


matrix = open("midsem-q2.txt", "r+")

M = []
for row in matrix:
    e1 = row.split()
    fe1 = []
    for i in range(len(e1)):
        fe1.append(float(e1[i]))
    M.append(fe1)
```

```python
print("The inverse of the given matrix is: ")

inverse(M)


#Verification
#the construction of R and S was asked to limit the matrix entries to two decimal spaces.
#first constructing the matrix with string entries and then converting that into floats is the only viable option in Python
#re-constructing the matrix with values upto two decimal places as strings
n = len(M)
R = []
for i in range(len(M)):
    r = []
    for j in range(n, len(M[0])):
        r.append("{:.2f}".format(M[i][j]))
    R.append(r)

#defining matrix to have numbers rounded off to 2 decimal places as floats
S = []
for i in range(len(R)):
    s = []
    for j in range(len(R[0])):
        s.append(float(R[i][j]))
    S.append(s)

print(S)
matrix = open("midsem-q2.txt", "r+")

Q = []
for row in matrix:
    e1 = row.split()
    fe1 = []
    for i in range(len(e1)):
        fe1.append(float(e1[i]))
    Q.append(fe1)


print(Q)
matrixProduct(Q, S)

print("Clearly, above matrix is an identity matrix.")
```

```
The determinant of the given matrix is:
120.0
As this value is non-zero, the inverse exists.
Now we will implement Gauss-Jordan elimination algorithm to find the inverse.
The inverse of the given matrix is:
0.00 0.00 0.00 0.20
0.00 0.00 0.25 0.00
0.00 0.33 0.00 0.00
0.50 0.00 0.00 0.00
[[0.0, 0.0, 0.0, 0.2], [0.0, 0.0, 0.25, 0.0], [0.0, 0.33, 0.0, 0.0], [0.5, 0.0, 0.0, 0.0]]
[[0.0, 0.0, 0.0, 2.0], [0.0, 0.0, 3.0, 0.0], [0.0, 4.0, 0.0, 0.0], [5.0, 0.0, 0.0, 0.0]]
Multiplying the two matrices:
1.0 0.0 0.0 0.0
0.0 0.99 0.0 0.0
0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0
Clearly, above matrix is an identity matrix.
```

Question 3

In [ ]:

```python
from functionLibrary import fwdSub
from functionLibrary import bwdSub
from functionLibrary import crout
from functionLibrary import solver

matrixA = open("midsem-q3-A.txt", "r+") #opening the
A = []
for row in matrixA:
    e1 = row.split()
    fe1 = []
    for i in range(len(e1)):
        fe1.append(float(e1[i]))
    A.append(fe1)


matrixb = open("midsem-q3-b.txt", "r+")
b = []
for row in matrixb:
    e1 = row.split()
    for i in range(len(e1)):
        b.append(float(e1[i]))


print("\n" + "The solution using Crout's algorithm:" + "\n")
print( "x = " + str(solver(A,b, crout)) + "\n") #the solution format is x = [x1, x2, x3, x4]
```

```
The solution using Crout's algorithm:

L = [[3.0, 0, 0, 0], [-3.0, -2.0, 0, 0], [6.0, 10.0, -1.0, 0], [-9.0, -16.0, -3.0, -1.0]]

U = [[1, -2.3333333333333335, -0.6666666666666666, 0.6666666666666666], [0, 1, 0.5, -1.0], [0, 0, 1, -1.0], [0, 0, 0, 1]]
```

```
x = [3.000000000000001, 4.0, -6.0, -1.0]
```

Question 4

Using the midpoint method

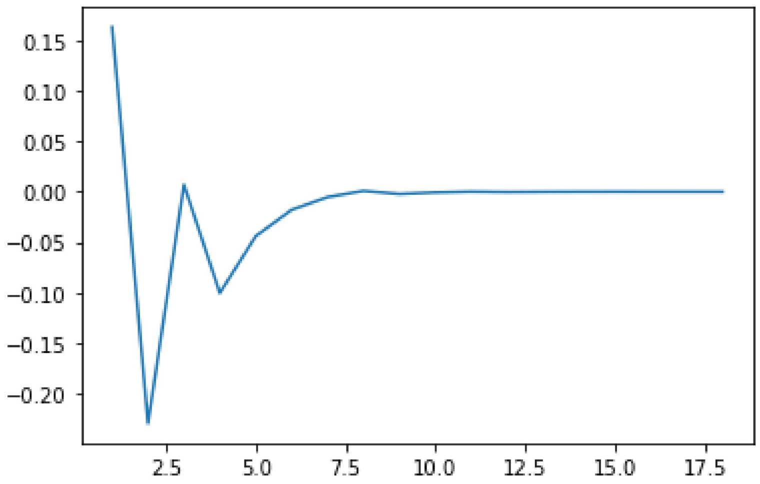
In [ ]:
```python
import math
from functionLibrary import rootBisec

y = lambda x: 4*math.exp(-x)*math.sin(x) - 1 #defining the function

a = float(input("Enter first approx: "))
b = float(input("Enter second approx: "))

rootBisec(y, a, b, 0.0001) #specifying e=10^(-4) as asked in question
```

```
Required root is:  0.3705558776855469
```




```
i x_i
1 0.5
2 0.25
3 0.375
4 0.3125
5 0.34375
6 0.359375
7 0.3671875
8 0.37109375
9 0.369140625
10 0.3701171875
11 0.37060546875
12 0.370361328125
13 0.3704833984375
14 0.37054443359375
15 0.370574951171875
16 0.3705596923828125
17 0.37055206298828125
18 0.3705558776855469
```

Using the regula-falsi method

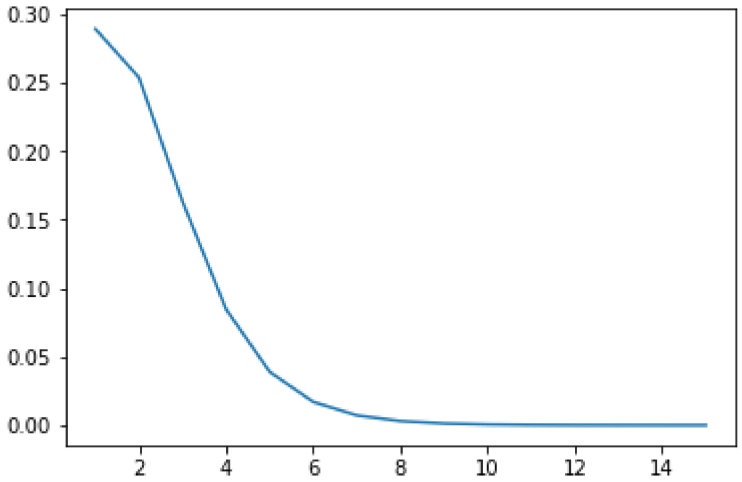
In [ ]:
```python
import math
from functionLibrary import rootRegFalsi

y = lambda x: 4*math.exp(-x)*math.sin(x) - 1 #defining the function

a = float(input("Enter first approx: "))
b = float(input("Enter second approx: "))

rootRegFalsi(y, a, b)
```

```
Required root is:  0.370562300835524
```




```
i x_i
1 0.8075982052665829
2 0.6265494882844935
3 0.49985393187801186
4 0.42979553644622137
5 0.39632542281599925
6 0.3814972968627457
7 0.3751529050070425
```

```
8 0.3724793039057679
9 0.3713598671047275
10 0.3708924286491957
11 0.37069746366711326
12 0.37061618376611605
13 0.37058230528185915
14 0.37056818546599873
15 0.370562300835524
```