```python
#Part2 LAB 1
#Solving SLEs using iterative methods

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

#------------------Task 1-------------------

A = np.array ( [[5,10,3,1],
                [6,7,20,-1],
                [12,2,3,-30],
                [15,-1,1,1]])

B = np.array ( [[6.7],
                [5.8],
                [4.3],
                [2.1]])

A_1 = np.linalg.inv(A)

print("det(A) is" , np.linalg.det(A))

#solution using matrix
x = np.dot(A_1,B)
print("X = \n" , x)
```

✓ 1.0s

```
det(A) is 84518.99999999994
X =
 [[ 0.17799785]
 [ 0.57376921]
 [ 0.03425857]
 [-0.03045706]]
```

```python
#--------------------Task 2------------------
#Using Jacobi method
#initial guesses
x1_k = 0
x2_k = 0
x3_k = 0
x4_k = 0


answers = []
answers.append((x1_k,x2_k,x3_k,x4_k))

#calculation loop
decimal_points = 5
#change this to change the precision decimal points in the answer
number_of_interations = 20 #change this to change the number of iterations

i = 1
while i<=number_of_interations:

    x1_k1 = (6.7 - 10*x2_k - 3*x3_k - x4_k)/5
    x2_k1 = (5.8 - 6*x1_k - 20*x3_k + x4_k)/7
    x3_k1 = (4.3 - 12*x1_k - 2*x2_k + 30*x4_k)/3
    x4_k1 = (2.1 - 15*x1_k + x2_k - x3_k)

    answers.append((round(x1_k1,decimal_points),round(x2_k1,decimal_points),
                    round(x3_k1,decimal_points),round(x4_k1,decimal_points)))

    x1_k = x1_k1
    x2_k = x2_k1
    x3_k = x3_k1
    x4_k = x4_k1

    i += 1
```

```python
#results printing
k = 0
while k<=number_of_interations:

    print(k , answers[k])
    k += 1

#since the system is not diagonally dominant, the answer is not converging
```

✓ 0.0s Python

```
0 (0, 0, 0, 0)
1 (1.34, 0.82857, 1.43333, 2.1)
2 (-1.59714, -4.11524, 16.52095, -18.60476)
3 (3.37886, -47.66299, -175.48222, 5.42095)
4 (200.87113, 500.08461, 73.90276, 79.23637)
5 (-1059.01814, -371.17794, -343.0772, -2584.78509)
6 (1466.49921, 1519.52398, -21362.89302, 15859.27143)
7 (6608.17357, 62046.27664, 151715.13481, 887.02882)
8 (-215297.69994, -439008.41554, -58925.15714, -188789.3617)
9 (951131.1377, 325929.39723, -734029.10688, 2849384.34064)
10 (-781316.85847, 1689026.49313, 24472034.02412, -13207006.46144)
11 (-15419870.76845, -71137111.42758, -130070813.40927, -11063252.55397)
12 (222529362.75151, 383267463.72031, -1528300.0808, 290231765.60844)
13 (-823664299.17382, -144911200.49779, 1756688564.03151, -2953144675.37151)
14 (-173561801.00902, -4734990022.16553, -26140182088.58793, 10453364725.17796)
15 (23063416353.78823, 76328339615.54153, 108384554472.02605, 24008619083.65771)
16 (-222489135729.69025, -326008995496.25604, 96946965679.16313, -378007460161.20795)
17 (669451303618.596, -140287422765.78986, -2672778728361.0474, 2914381074772.0347)
18 (1301365867595.141, 7479035402898.172, 26559530481757.926, -7509278248681.582)
19 (-29391933445113.445, -78072583298486.45, -85284269559127.06, -38600983092784.766)
20 (215035924951007.44, 263348001251491.84, -216393708281734.78, 448090687937344.44)
```

```python
#--------------------Task 2-------------------
#Using Gauss Seidel Method
#initial guesses
x1_k = 0
x2_k = 0
x3_k = 0
x4_k = 0


answers = []
answers.append((x1_k,x2_k,x3_k,x4_k))

#calculation loop
decimal_points = 5
#change this to change the precision decimal points in the answer
number_of_interations = 20 #change this to change the number of iterations

i = 1
while i<=number_of_interations:

    x1_k1 = (6.7 - 10*x2_k - 3*x3_k - x4_k)/5
    x2_k1 = (5.8 - 6*x1_k1 - 20*x3_k + x4_k)/7
    x3_k1 = (4.3 - 12*x1_k1 - 2*x2_k1 + 30*x4_k)/3
    x4_k1 = (2.1 - 15*x1_k1 + x2_k1 - x3_k1)

    answers.append((round(x1_k1,decimal_points),round(x2_k1,decimal_points),
                    round(x3_k1,decimal_points),round(x4_k1,decimal_points)))

    x1_k = x1_k1
    x2_k = x2_k1
    x3_k = x3_k1
    x4_k = x4_k1

    i += 1
```

```python
#results printing
k = 0
while k<=number_of_interations:

    print(k , answers[k])
    k += 1

#since the system is not diagonally dominant, the answer is not converging
```

✓ 0.0s Python

```
0 (0, 0, 0, 0)
1 (1.34, -0.32, -3.71333, -14.60667)
2 (7.12933, 3.24057, -175.31105, 73.71162)
3 (85.30316, 439.13051, 104.5832, -942.90011)
4 (-751.09093, 211.11163, -6563.9452, 18043.52072)
5 (-91.22029, 21410.79234, 166527.66011, -143746.46335)
6 (-113987.54808, -398624.08244, -715763.61959, 2026954.85838)
7 (821316.70495, 1630618.97436, 15897203.88112, -26586333.38101)
8 (-7482292.26119, -42805235.94803, -207397339.36666, 276826489.43653)
9 (154683578.96876, 499524544.10818, 1816514217.18483, -3637243355.50805)
10 (-1361508946.08565, -4542639145.27011, -27897971672.45781, 43777966720.57248)
11 (17068467951.2404, 71332370352.58374, 321950881833.8074, -506645530747.7296)
12 (-234506163654.566, -791232312212.9543, -3600942444715.6294, 6327302587323.266)
13 (2477569573791.9736, 9068676291269.975, 47316963383886.21, -75411830699493.73)
14 (-31445164472971.586, -119011444505626.11, -548996686099298.8, 901662708688248.6)
15 (387088358933183.1, 1365580896725018.8, 6557886386666409.0, -1.0998630873939136e+1
16 (-4466167450662054.5, -1.647990770047071e+16, -8.1135033803096e+16, 1.316476378625
17 (5.531130811028781e+16, 2.0321149503753558e+17, 9.59756816159386e+17, -1.586214942
18 (-6.650340912154692e+17, -2.398735245524439e+18, -1.160285623255017e+19, 1.9179632
19 (7.923257759527426e+18, 2.909960149272809e+19, 1.407035581859826e+20, -2.30452823
20 (-9.653077327981257e+19, -3.521913353038489e+20, -1.683610914206509e+21, 2.779381
```

The Gauss seidel method is used to speed up the iteration process. Here we can see that from the Gauss seidel method, after 20 iterations it returns a very large value compared to the answer given by the Jacobi method after 20 iterations.

```
#-----------------Task 3------------------
#Using jacobi method
#initial guesses
x1_k = 5
x2_k = -3
x3_k = 5
x4_k = 0


answers = []
answers.append((x1_k,x2_k,x3_k,x4_k))


decimal_points = 5
#change this to change the precision decimal points in the answer

#stopping criteria setting
number_of_interations = 20 #change this to change the number of iterations
tolerance = 0.00001 #change this to change the tolerance value of the norm

#calculation loop
i = 1
while True:

    #equations
    x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
    x2_k1 = (6.7 - 5*x1_k - 3*x3_k - x4_k)/10
    x3_k1 = (5.8 - 6*x1_k - 7*x2_k + x4_k)/20
    x4_k1 = (4.3 - 12*x1_k - 2*x2_k - 3*x3_k)/-30

    answers.append((round(x1_k1,decimal_points),round(x2_k1,decimal_points),
                    round(x3_k1,decimal_points),round(x4_k1,decimal_points)))
```

```
    norm = math.sqrt((x1_k1-x1_k)**2 + (x2_k1-x2_k)**2 +
                     (x3_k1-x3_k)**2 + (x4_k1-x4_k)**2)

    #stopping criteria check and exit
    if ((i==number_of_interations) or (norm <= tolerance)):
        break

    #if not stopped, prep for next iteration
    x1_k = x1_k1
    x2_k = x2_k1
    x3_k = x3_k1
    x4_k = x4_k1

    i += 1

print("Number of iterations : " , i)
#results printing
k = 0
while k<=i:

    print(k , answers[k])
    k += 1
```

```
Number of iterations :  13
0 (5, -3, 5, 0)
1 (-0.39333, -3.33, -0.16, 2.15667)
2 (-0.21511, 0.699, 1.68133, -0.53867)
3 (0.11042, 0.32702, 0.08295, -0.01464)
4 (0.15725, 0.59137, 0.14168, -0.06907)
5 (0.17458, 0.55578, 0.03239, -0.02684)
6 (0.17668, 0.57567, 0.04176, -0.03321)
7 (0.17781, 0.57245, 0.03385, -0.03011)
8 (0.17791, 0.57395, 0.03479, -0.03066)
9 (0.17799, 0.57367, 0.03421, -0.03042)
10 (0.17799, 0.57379, 0.0343, -0.03047)
11 (0.178, 0.57376, 0.03425, -0.03045)
12 (0.178, 0.57377, 0.03426, -0.03046)
13 (0.178, 0.57377, 0.03426, -0.03046)
```

```
#-------------Task 3----------------
#Using gauss seidel method
#initial guesses
x1_k = 5
x2_k = -3
x3_k = 5
x4_k = 0


answers = []
answers.append((x1_k,x2_k,x3_k,x4_k))


decimal_points = 5
#change this to change the precision decimal points in the answer

#stopping criteria setting
number_of_interations = 20 #change this to change the number of iterations
tolerance = 0.00001 #change this to change the tolerance value of the norm

#calculation loop
i = 1
while True:

    #equations
    x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
    x2_k1 = (6.7 - 5*x1_k1 - 3*x3_k - x4_k)/10
    x3_k1 = (5.8 - 6*x1_k1 - 7*x2_k1 + x4_k)/20
    x4_k1 = (4.3 - 12*x1_k1 - 2*x2_k1 - 3*x3_k1)/-30

    answers.append((round(x1_k1,decimal_points),round(x2_k1,decimal_points),
                    round(x3_k1,decimal_points),round(x4_k1,decimal_points)))
```

```python
        norm = math.sqrt((x1_k1-x1_k)**2 + (x2_k1-x2_k)**2 +
                         (x3_k1-x3_k)**2 + (x4_k1-x4_k)**2)

        #stopping criteria check and exit
        if ((i==number_of_interations) or (norm <= tolerance)):
            break

        #if not stopped, prep for next iteration
        x1_k = x1_k1
        x2_k = x2_k1
        x3_k = x3_k1
        x4_k = x4_k1

        i += 1

    print("Number of iterations : " , i)
    #results printing
    k = 0
    while k<=i:

        print(k , answers[k])
        k += 1
```

✓ 0.0s

```
Number of iterations :  7
0 (5, -3, 5, 0)
1 (-0.39333, -0.63333, 0.62967, -0.27992)
2 (0.07446, 0.47186, 0.08851, -0.07324)
3 (0.17044, 0.56555, 0.03726, -0.03373)
4 (0.17747, 0.57346, 0.03436, -0.03068)
5 (0.17799, 0.57377, 0.03425, -0.03046)
6 (0.178, 0.57377, 0.03426, -0.03046)
7 (0.178, 0.57377, 0.03426, -0.03046)
```

From the results of both methods we can see that using gauss seidel method returns the answer from comparatively less amount of iterations.

Hence the solution for the system of linear equation using Jacobi method and Gauss seidel method is,'
x1 = 0.178
x2 = 0.57377
x3 = 0.03426
x4 = -0.03046

```python
#-----------------Task 4--------------------
#Using jacobi method

tenth_power = range(11)
number_of_iterations_jacobi = []
number_of_iterations_gseidel = []

for i in range(11):

    #initial guesses
    x1_k = x[0]*(10**i)
    x2_k = x[1]*(10**i)
    x3_k = x[2]*(10**i)
    x4_k = x[3]*(10**i)


    decimal_points = 5
    #change this to change the precision decimal points in the answer

    #stopping criteria setting
    #number_of_interations = 20 #change this to change the number of iterations
    tolerance = 10e-8 #change this to change the tolerance value of the norm

    #calculation loop
    j = 1
    while True:

        #equations
        x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
        x2_k1 = (6.7 - 5*x1_k - 3*x3_k - x4_k)/10
        x3_k1 = (5.8 - 6*x1_k - 7*x2_k + x4_k)/20
        x4_k1 = (4.3 - 12*x1_k - 2*x2_k - 3*x3_k)/-30

        norm = math.sqrt((x1_k1-x1_k)**2 + (x2_k1-x2_k)**2 +
                         (x3_k1-x3_k)**2 + (x4_k1-x4_k)**2)
```

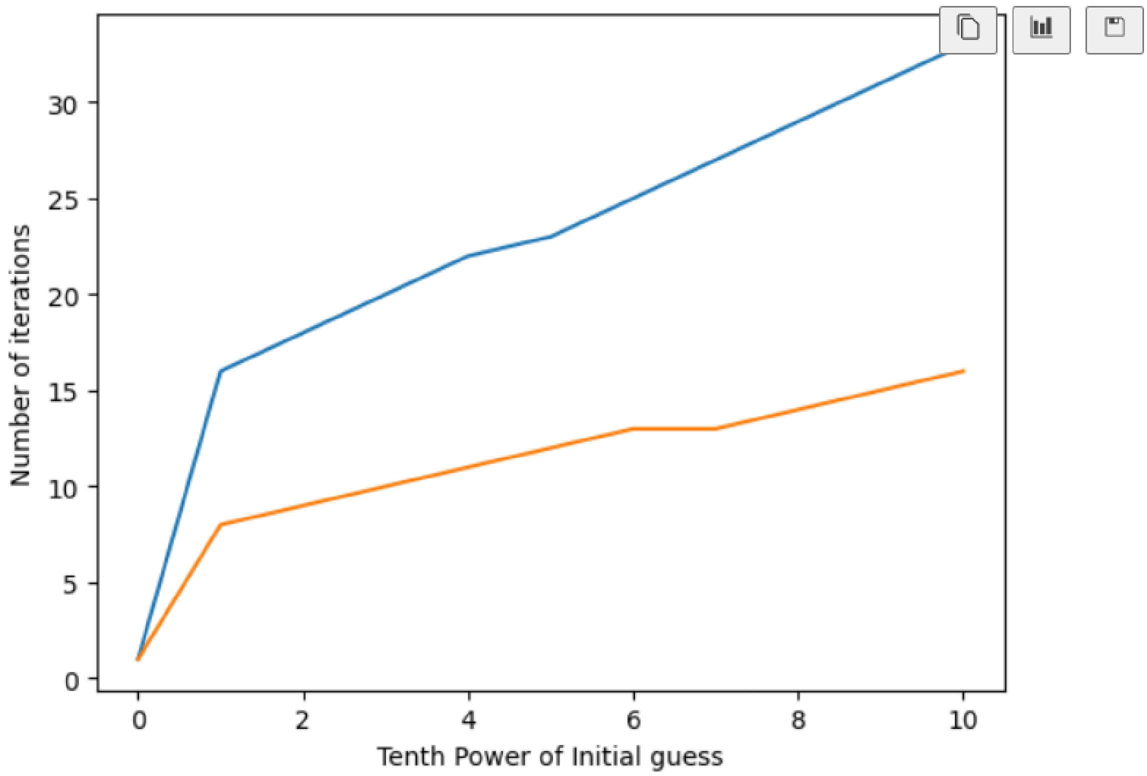
```python
        #stopping criteria check and exit
        if (norm <= tolerance):
            break

        #if not stopped, prep for next iteration
        x1_k = x1_k1
        x2_k = x2_k1
        x3_k = x3_k1
        x4_k = x4_k1


        j += 1

    number_of_iterations_jacobi.append(j)
```

```python
#using gauss seidel method

for i in range(11):

    #initial guesses
    x1_k = x[0]*(10**i)
    x2_k = x[1]*(10**i)
    x3_k = x[2]*(10**i)
    x4_k = x[3]*(10**i)

    #stopping criteria setting
    #number_of_interations = 20 #change this to change the number of iterations
    tolerance = 10e-8 #change this to change the tolerance value of the norm

    #calculation loop
    k = 1
    while True:

        #equations
        x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
        x2_k1 = (6.7 - 5*x1_k1 - 3*x3_k - x4_k)/10
        x3_k1 = (5.8 - 6*x1_k1 - 7*x2_k1 + x4_k)/20
        x4_k1 = (4.3 - 12*x1_k1 - 2*x2_k1 - 3*x3_k1)/-30



        norm = math.sqrt((x1_k1-x1_k)**2 + (x2_k1-x2_k)**2 +
                         (x3_k1-x3_k)**2 + (x4_k1-x4_k)**2)

        #stopping criteria check and exit
        if (norm <= tolerance):
            break
```

```python
        #if not stopped, prep for next iteration
        x1_k = x1_k1
        x2_k = x2_k1
        x3_k = x3_k1
        x4_k = x4_k1

        k += 1

    number_of_iterations_gseidel.append(k)


print(list(tenth_power))
print(number_of_iterations_jacobi)
print(number_of_iterations_gseidel)

#graph plotting
plt.plot((tenth_power), (number_of_iterations_jacobi))
plt.plot((tenth_power), (number_of_iterations_gseidel))

plt.xlabel("Tenth Power of Initial guess")
plt.ylabel("Number of iterations")
plt.show()
```

✓ 0.3s

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 16, 18, 20, 22, 23, 25, 27, 29, 31, 33]
[1, 8, 9, 10, 11, 12, 13, 13, 14, 15, 16]
```



Here also we can see that the gauss seidel method reaches the tolerance level in a very less number of iterations compared to jacobi method.

<u>Tenth power of initial guesses vs Number of iterations</u>

```python
#-------------------Task 5------------------
#Using Jacobi method

#initial guesses
x1_k = 5
x2_k = -3
x3_k = 5
x4_k = 0

iteration_number = range(1,21,1)

error_array_jacobi = []
error_array_gseidel =[]

decimal_points = 5
#change this to change the precision decimal points in the answer

#calculation loop - jacobi
j = 1
while j<=20:

    #equations
    x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
    x2_k1 = (6.7 - 5*x1_k - 3*x3_k - x4_k)/10
    x3_k1 = (5.8 - 6*x1_k - 7*x2_k + x4_k)/20
    x4_k1 = (4.3 - 12*x1_k - 2*x2_k - 3*x3_k)/-30

    #error calculation for each variable
    e1_k = (x1_k1 - x[0])**2
    e2_k = (x2_k1 - x[1])**2
    e3_k = (x3_k1 - x[2])**2
    e4_k = (x4_k1 - x[3])**2

    total_error = round(math.sqrt(e1_k + e2_k + e3_k + e4_k),decimal_points)
```

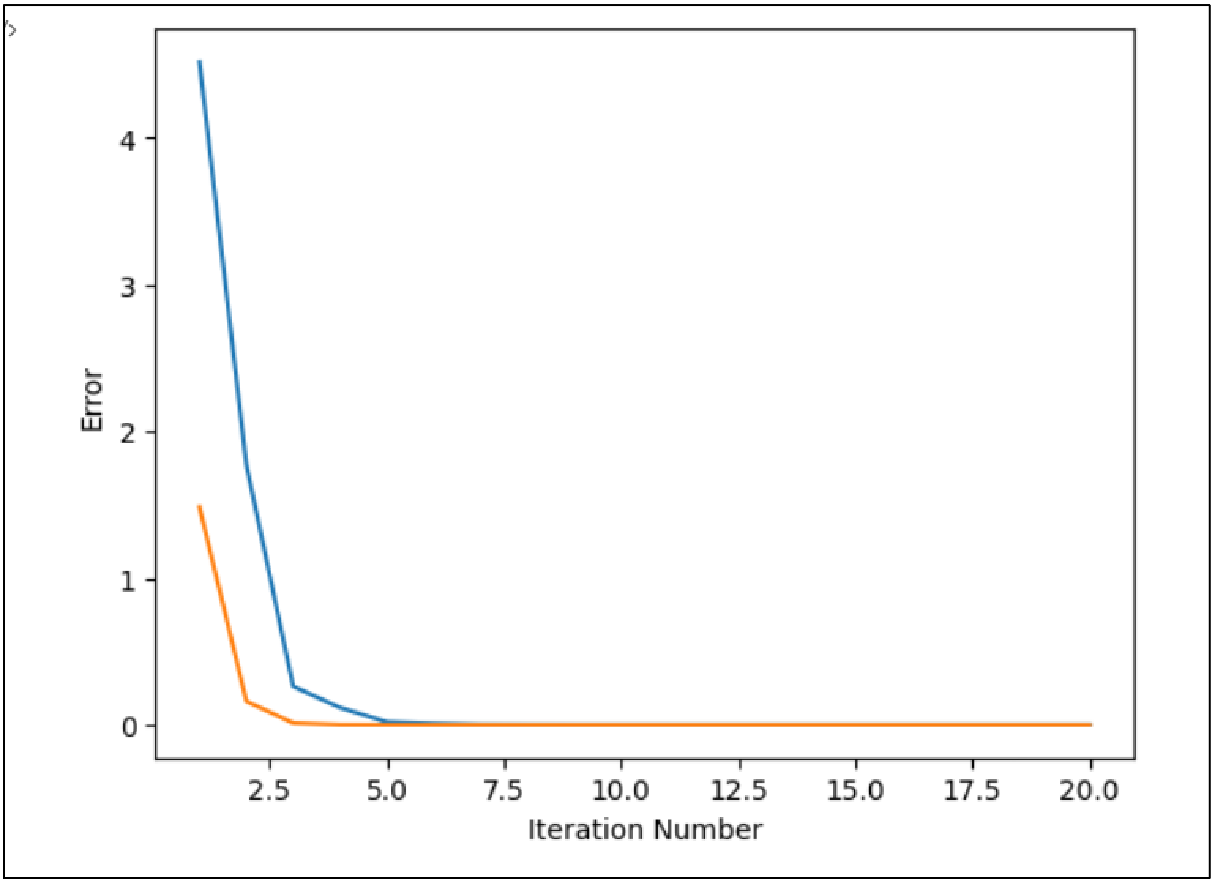
```python
    error_array_jacobi.append(total_error)

    #value assigning for the next loop
    x1_k = x1_k1
    x2_k = x2_k1
    x3_k = x3_k1
    x4_k = x4_k1

    j += 1
```

```python
#calculation loop - gauss seidel

#initial guesses
x1_k = 5
x2_k = -3
x3_k = 5
x4_k = 0

k = 1
while k<=20:

    #equations
    x1_k1 = (2.1 + x2_k - x3_k - x4_k)/15
    x2_k1 = (6.7 - 5*x1_k1 - 3*x3_k - x4_k)/10
    x3_k1 = (5.8 - 6*x1_k1 - 7*x2_k1 + x4_k)/20
    x4_k1 = (4.3 - 12*x1_k1 - 2*x2_k1 - 3*x3_k1)/-30

    #error calculation for each variable
    e1_k = (x1_k1 - x[0])**2
    e2_k = (x2_k1 - x[1])**2
    e3_k = (x3_k1 - x[2])**2
    e4_k = (x4_k1 - x[3])**2

    total_error = round(math.sqrt(e1_k + e2_k + e3_k + e4_k),decimal_points)

    error_array_gseidel.append(total_error)

    #value assigning for the next loop
    x1_k = x1_k1
    x2_k = x2_k1
    x3_k = x3_k1
    x4_k = x4_k1

    k += 1
```

```python
print(list(iteration_number))
print(error_array_jacobi)
print(error_array_gseidel)

#graph plotting
plt.plot((iteration_number), (error_array_jacobi))
plt.plot((iteration_number), (error_array_gseidel))

plt.xlabel("Iteration Number")
plt.ylabel("Error")
plt.show()
```

✓ 0.2s Python

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[4.51521, 1.77239, 0.26091, 0.11735, 0.01876, 0.00832, 0.00144, 0.00061, 0.00011, 5e-
[1.48333, 0.16087, 0.01202, 0.00066, 2e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```



Iteration Number vs Error