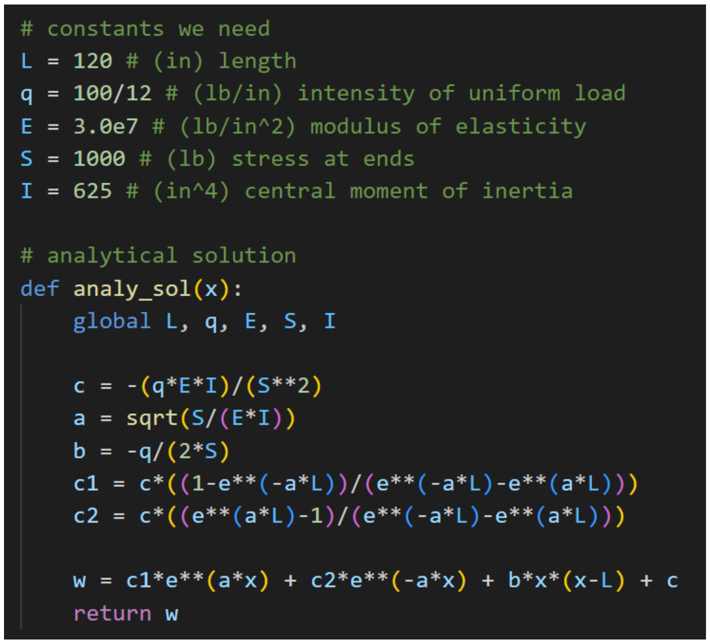
Group 4A
Sissi Zhang
Nicholas Calzada
Lechen Wang
Zun Cao

Group Project 2

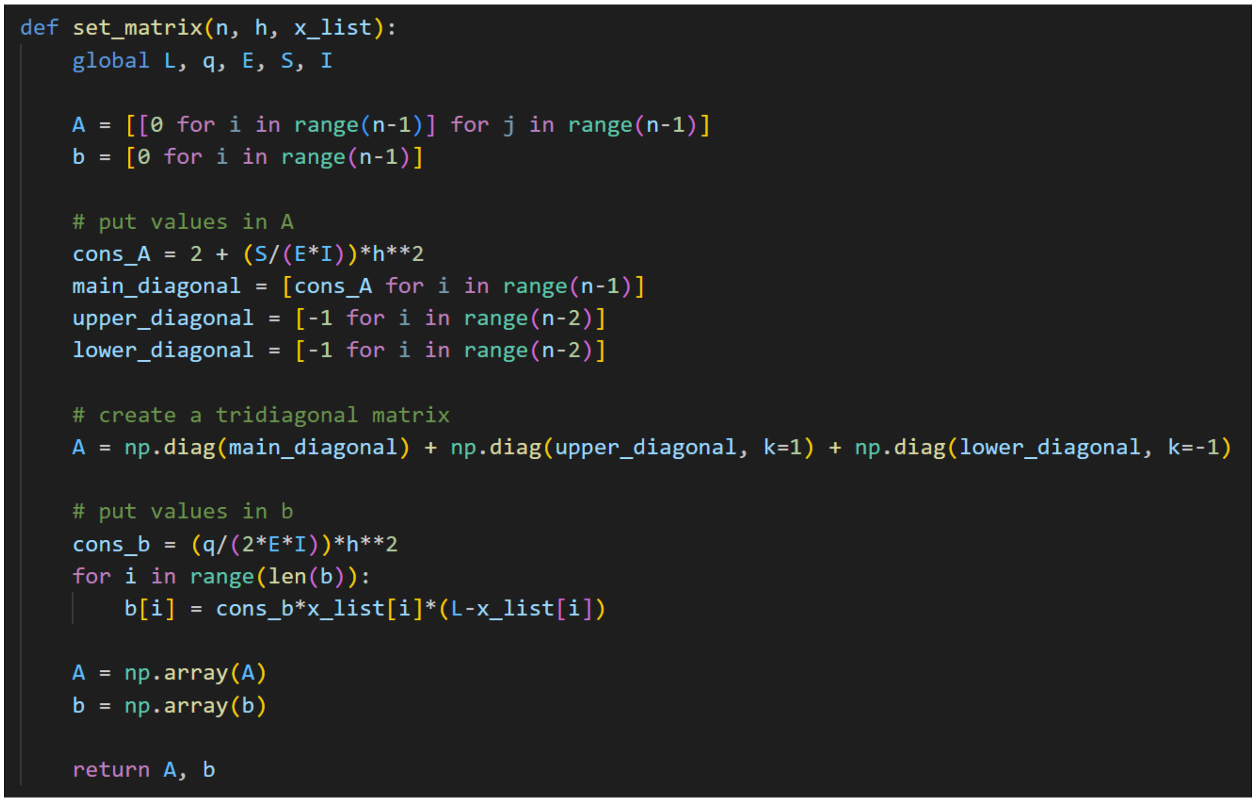
1. Write the MATLAB or Python code defining the true solution w(x). Make sure the value you use for the intensity of uniform load, q, has units lb/in. Otherwise, you will get a big mismatch between the numerical and the ”true” solution.



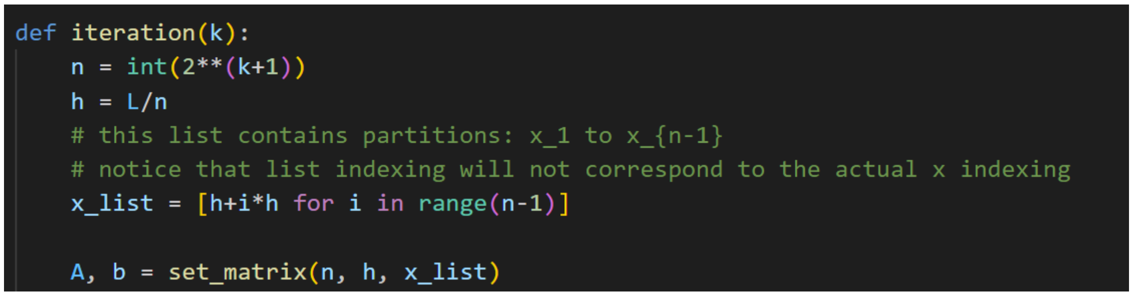
```python
# constants we need
L = 120 # (in) length
q = 100/12 # (lb/in) intensity of uniform load
E = 3.0e7 # (lb/in^2) modulus of elasticity
S = 1000 # (lb) stress at ends
I = 625 # (in^4) central moment of inertia

# analytical solution
def analy_sol(x):
    global L, q, E, S, I

    c = -(q*E*I)/(S**2)
    a = sqrt(S/(E*I))
    b = -q/(2*S)
    c1 = c*((1-e**(-a*L))/(e**(-a*L)-e**(a*L)))
    c2 = c*((e**(a*L)-1)/(e**(-a*L)-e**(a*L)))

    w = c1*e**(a*x) + c2*e**(-a*x) + b*x*(x-L) + c
    return w
```

2. For each refinement level k = 1, . . . , 20:

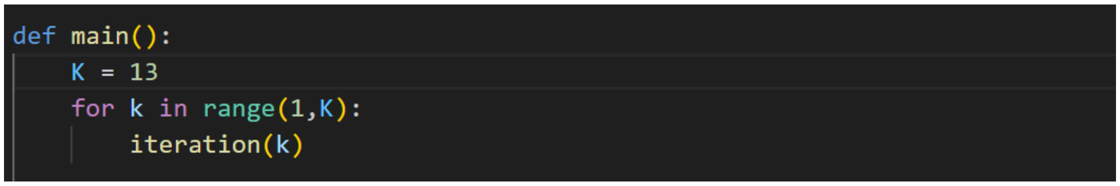
(a) Generate the matrix A and for the vector b.



```python
def set_matrix(n, h, x_list):
    global L, q, E, S, I

    A = [[0 for i in range(n-1)] for j in range(n-1)]
    b = [0 for i in range(n-1)]

    # put values in A
    cons_A = 2 + (S/(E*I))*h**2
    main_diagonal = [cons_A for i in range(n-1)]
    upper_diagonal = [-1 for i in range(n-2)]
    lower_diagonal = [-1 for i in range(n-2)]

    # create a tridiagonal matrix
    A = np.diag(main_diagonal) + np.diag(upper_diagonal, k=1) + np.diag(lower_diagonal, k=-1)

    # put values in b
    cons_b = (q/(2*E*I))*h**2
    for i in range(len(b)):
        b[i] = cons_b*x_list[i]*(L-x_list[i])

    A = np.array(A)
    b = np.array(b)

    return A, b
```



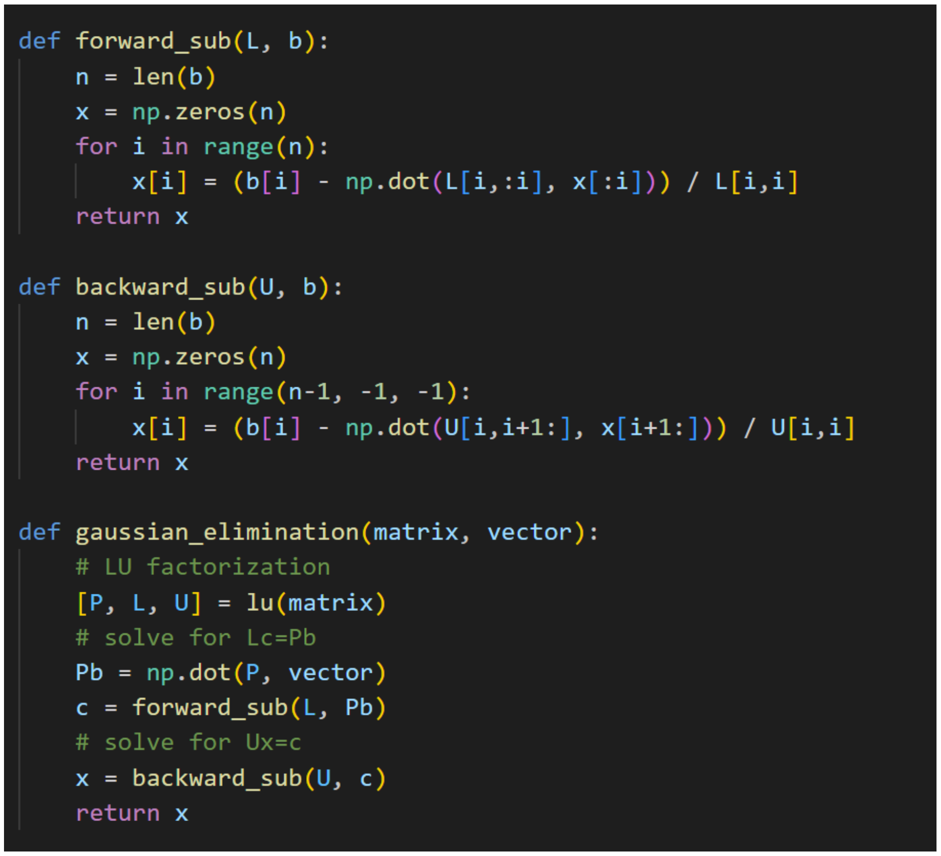
```python
def iteration(k):
    n = int(2**(k+1))
    h = L/n
    # this list contains partitions: x_1 to x_{n-1}
    # notice that list indexing will not correspond to the actual x indexing
    x_list = [h+i*h for i in range(n-1)]

    A, b = set_matrix(n, h, x_list)
```



```python
def main():
    K = 13
    for k in range(1,K):
        iteration(k)
```

Note: K is set to 14 when we actually ran the script.

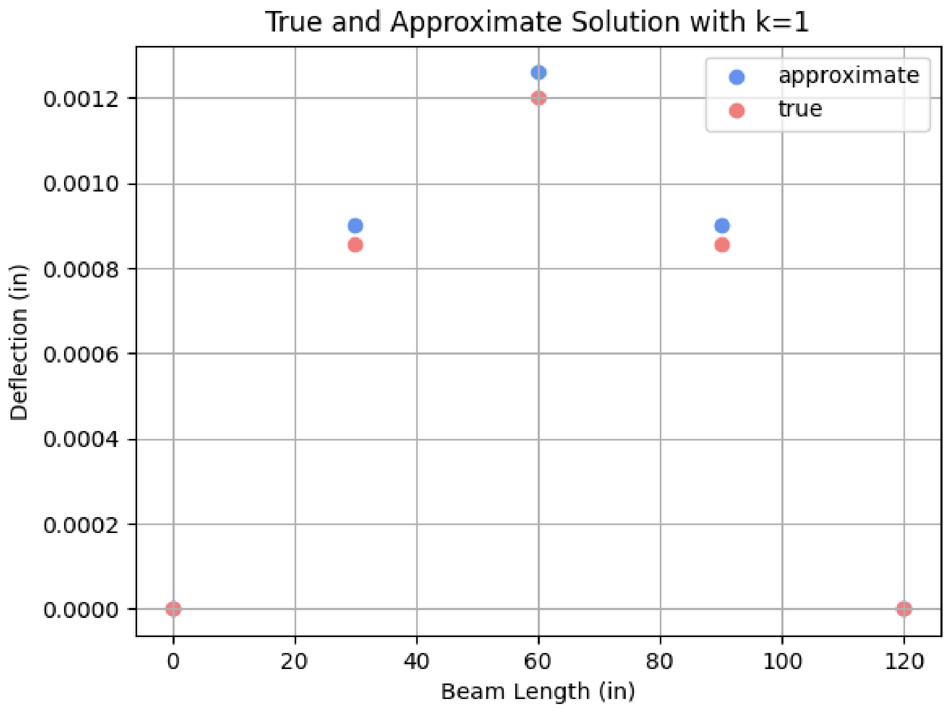
(b) Write the commands needed to solve the system Ay = b using Gaussian Elimination (PA=LU). (It is sufficient to write them just once.)

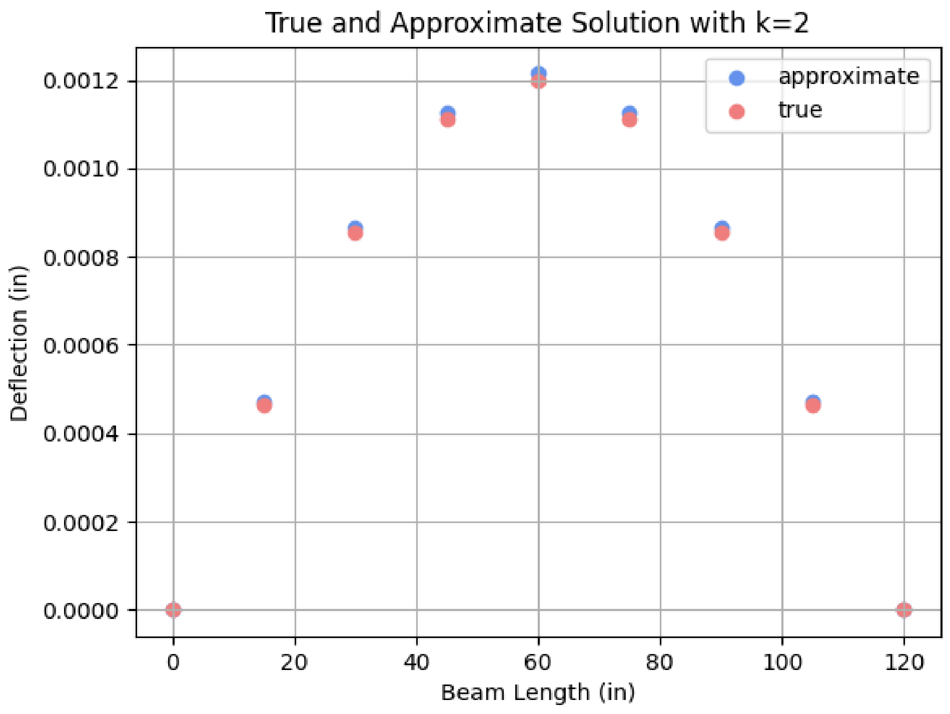


```python
def forward_sub(L, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n):
        x[i] = (b[i] - np.dot(L[i,:i], x[:i])) / L[i,i]
    return x

def backward_sub(U, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (b[i] - np.dot(U[i,i+1:], x[i+1:])) / U[i,i]
    return x

def gaussian_elimination(matrix, vector):
    # LU factorization
    [P, L, U] = lu(matrix)
    # solve for Lc=Pb
    Pb = np.dot(P, vector)
    c = forward_sub(L, Pb)
    # solve for Ux=c
    x = backward_sub(U, c)
    return x
```

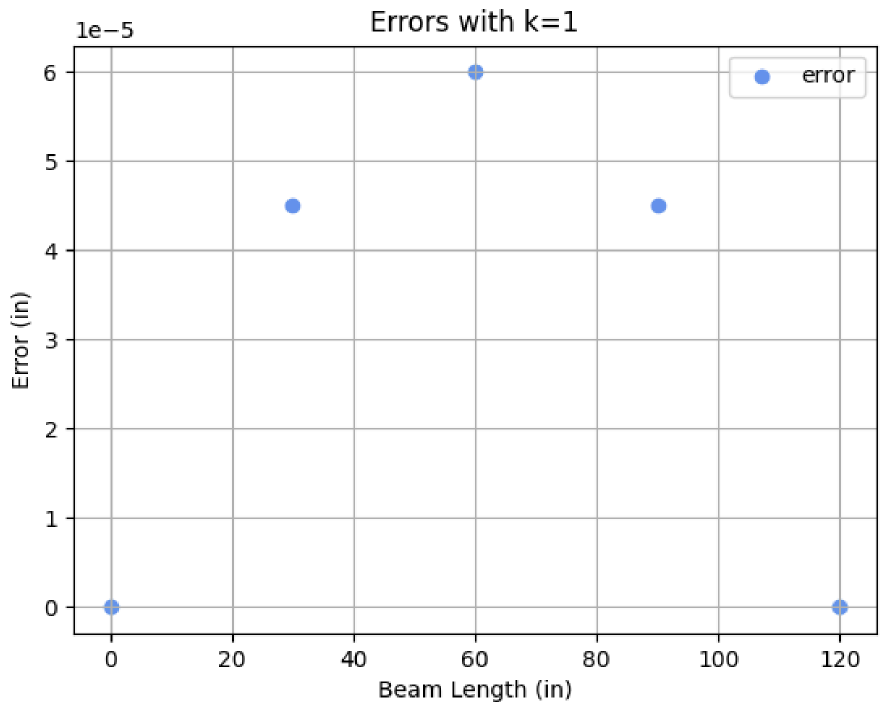
Not necessary. You are allowed to use builtin functions

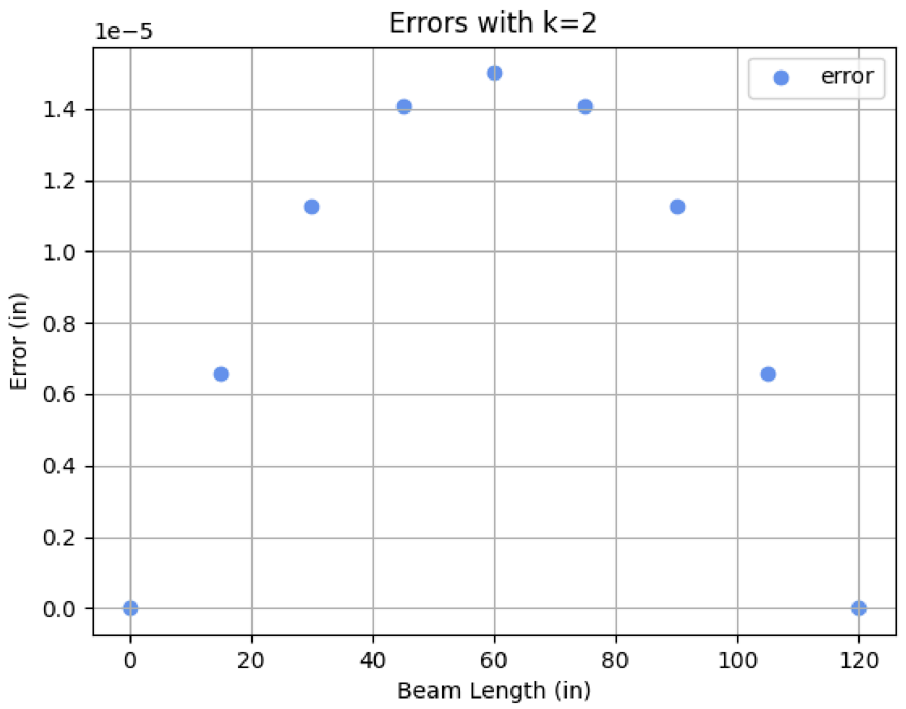
(c) For k = 1, 2, plot on the same figure (one figure per level) the true and the approximate solution. You can plot the numerical solution either as a piecewise line function or as a scatter plot. Annotate the figures correspondingly.

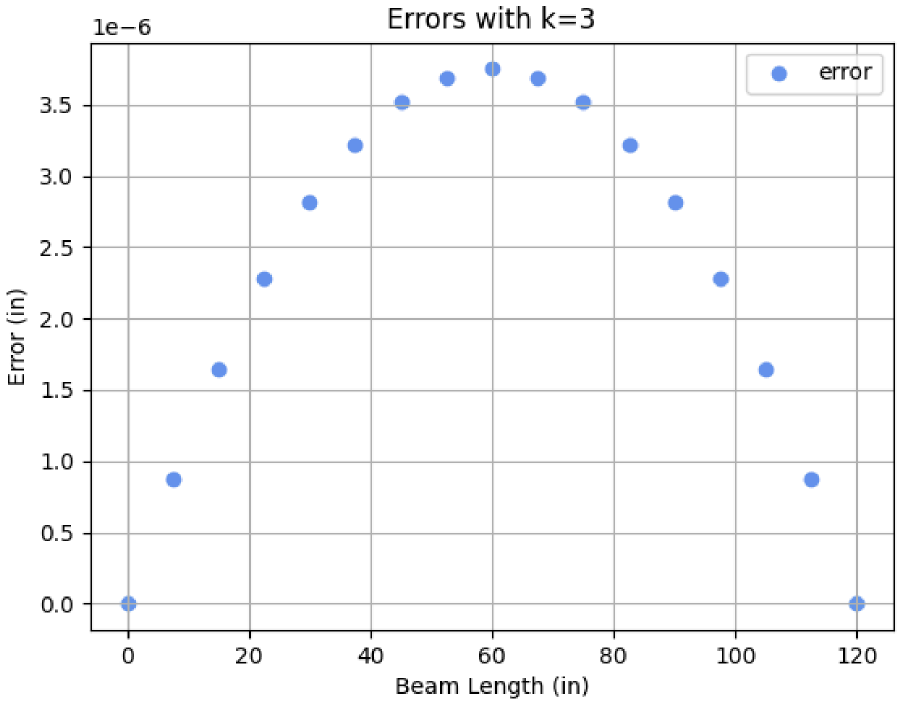


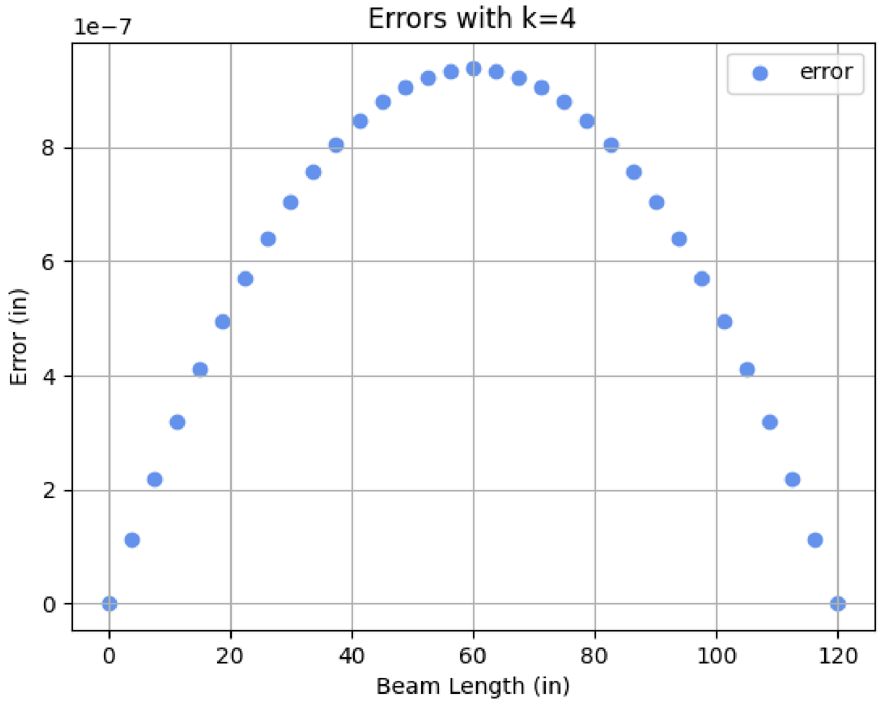


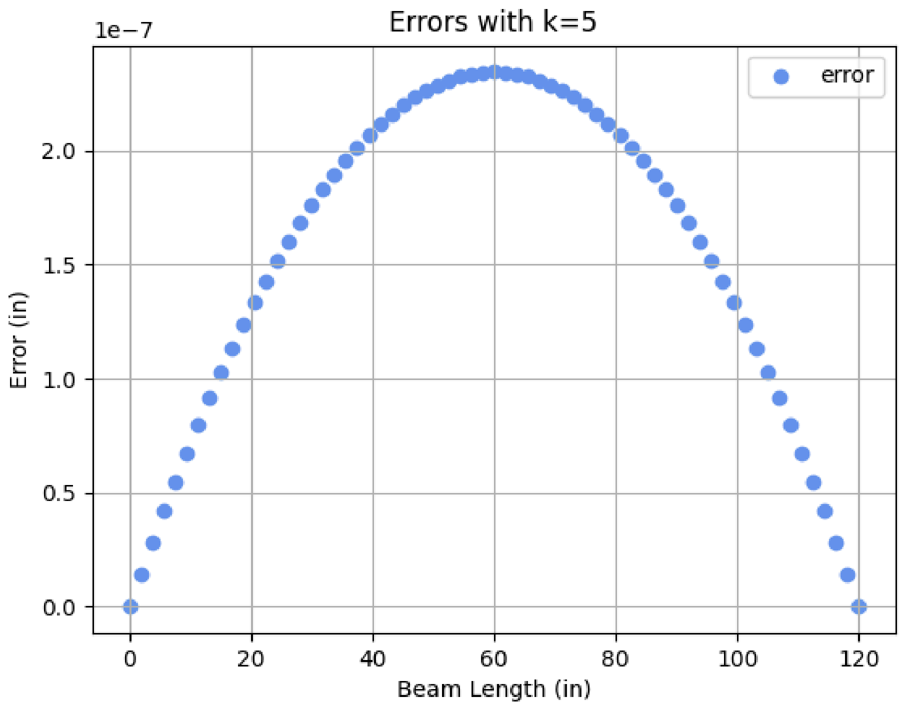
(d) For each level (value of k), plot the error in the solution. Annotate the figure correspondingly.

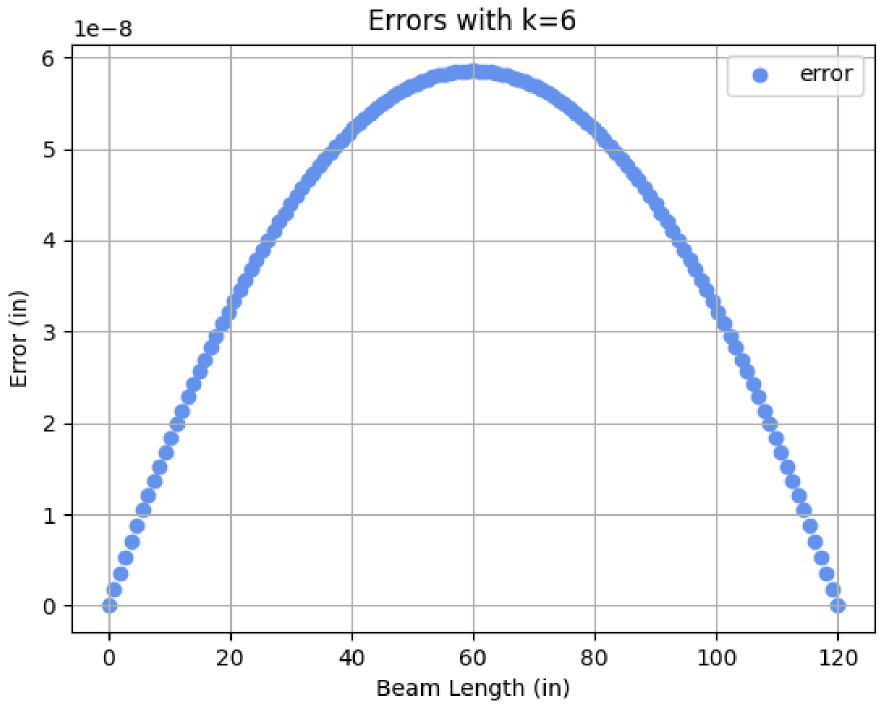


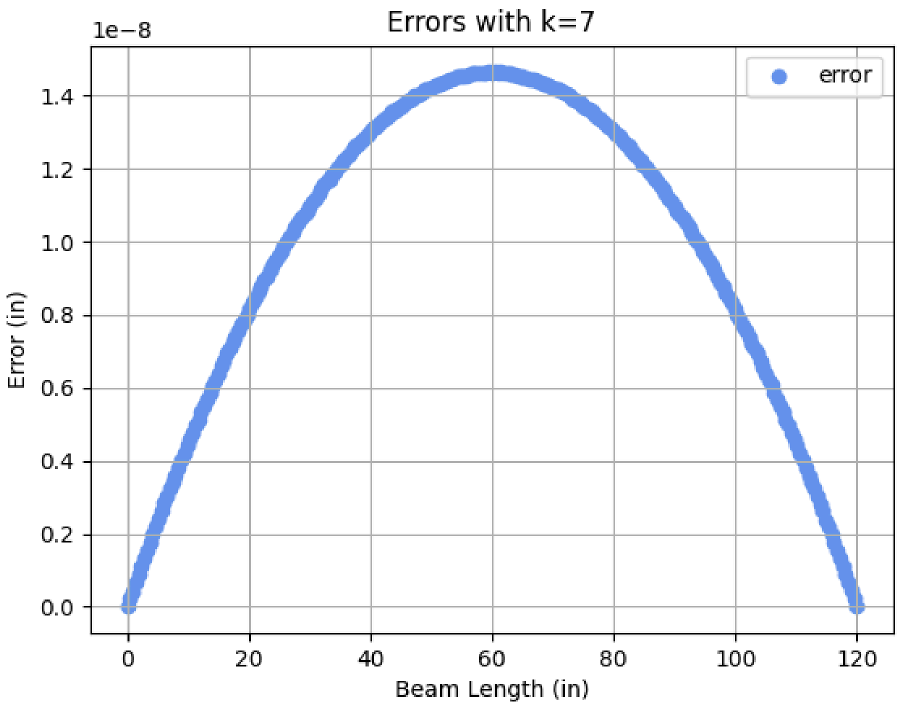



Errors with k=3
1e−6
error
Error (in)
Beam Length (in)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100
120

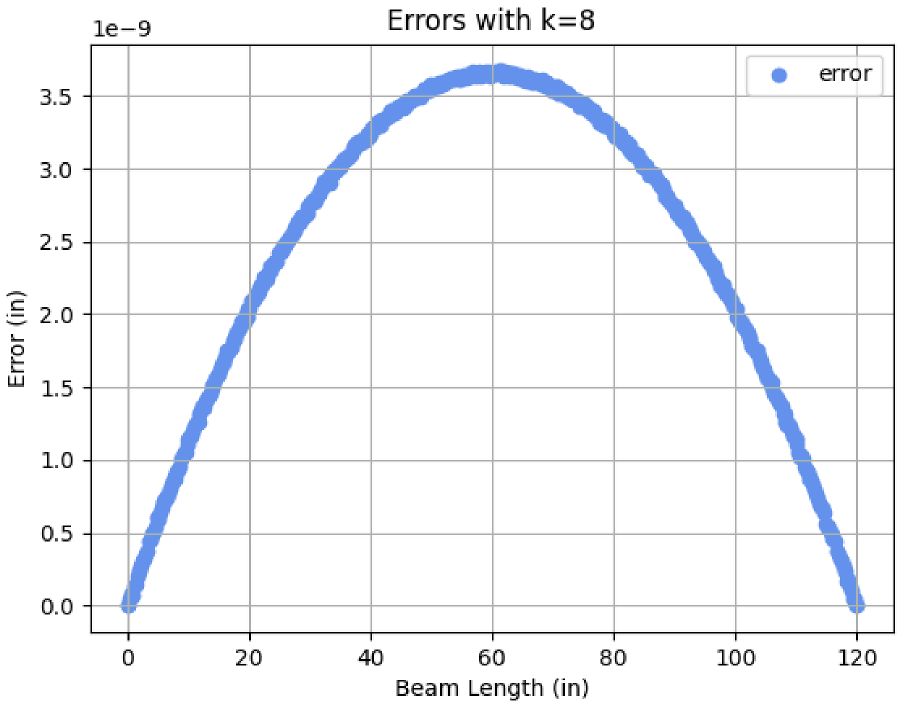

Errors with k=4
1e−7
error
Error (in)
Beam Length (in)
0
2
4
6
8
0
20
40
60
80
100
120


Errors with k=5
1e−7
error
Error (in)
Beam Length (in)
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
120

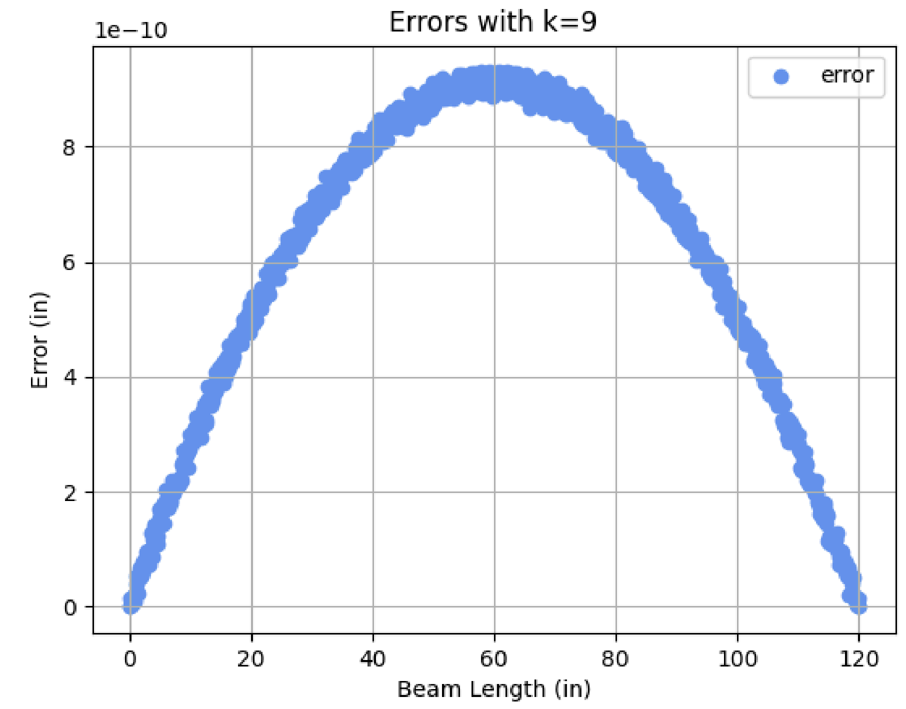

Errors with k=6
1e−8
error
Error (in)
Beam Length (in)
0
1
2
3
4
5
6
0
20
40
60
80
100
120

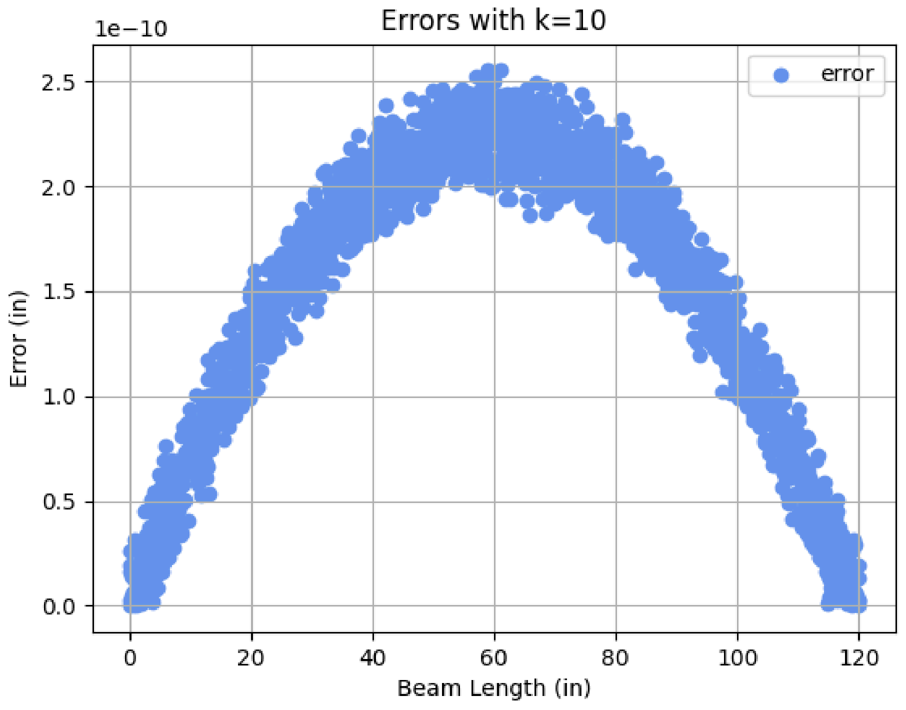


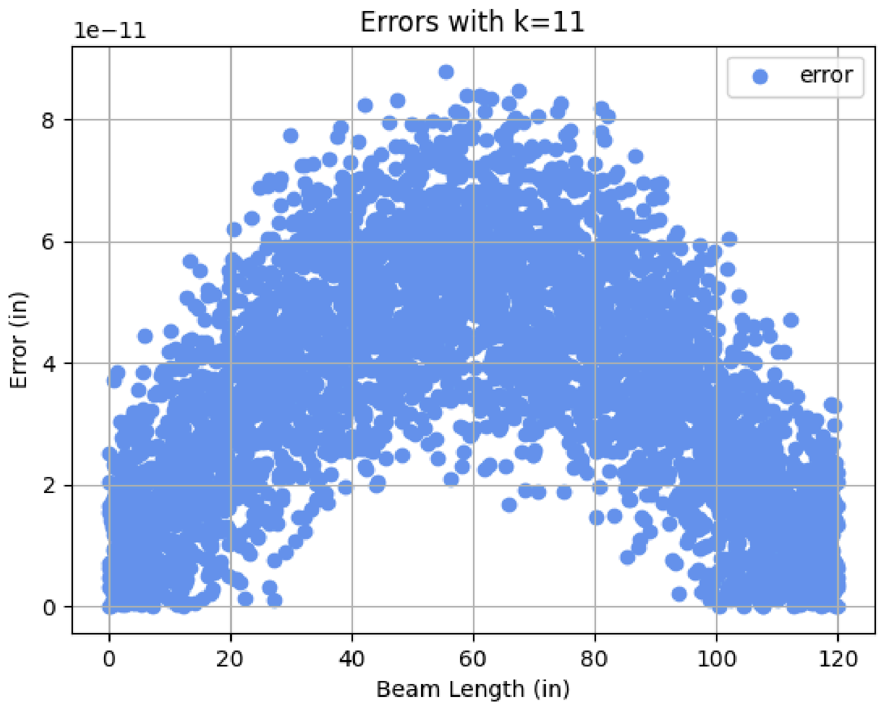
Errors with k=7
1e−8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Error (in)
0
20
40
60
80
100
120
Beam Length (in)
error

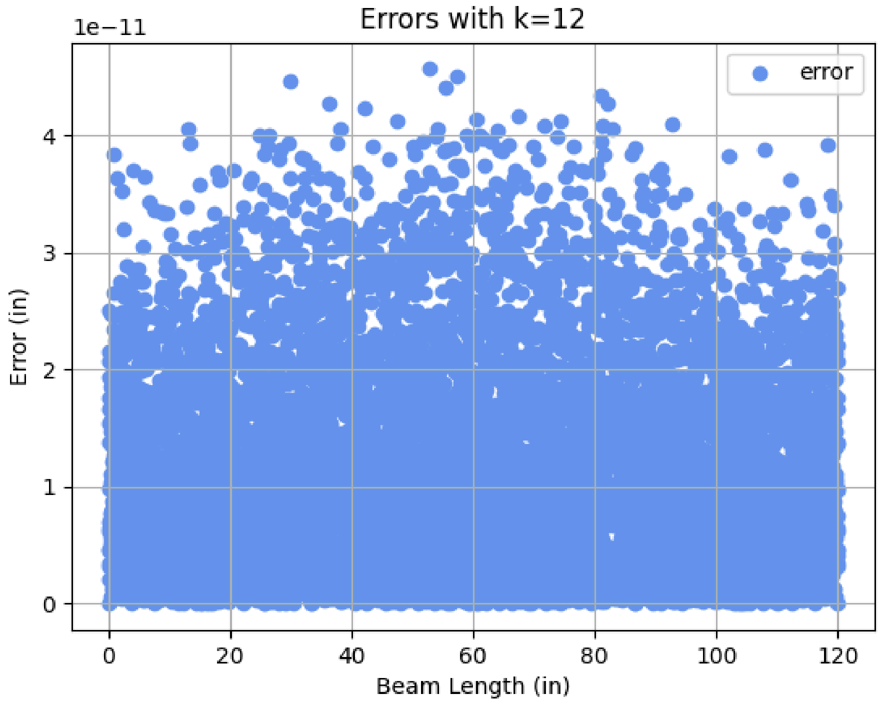


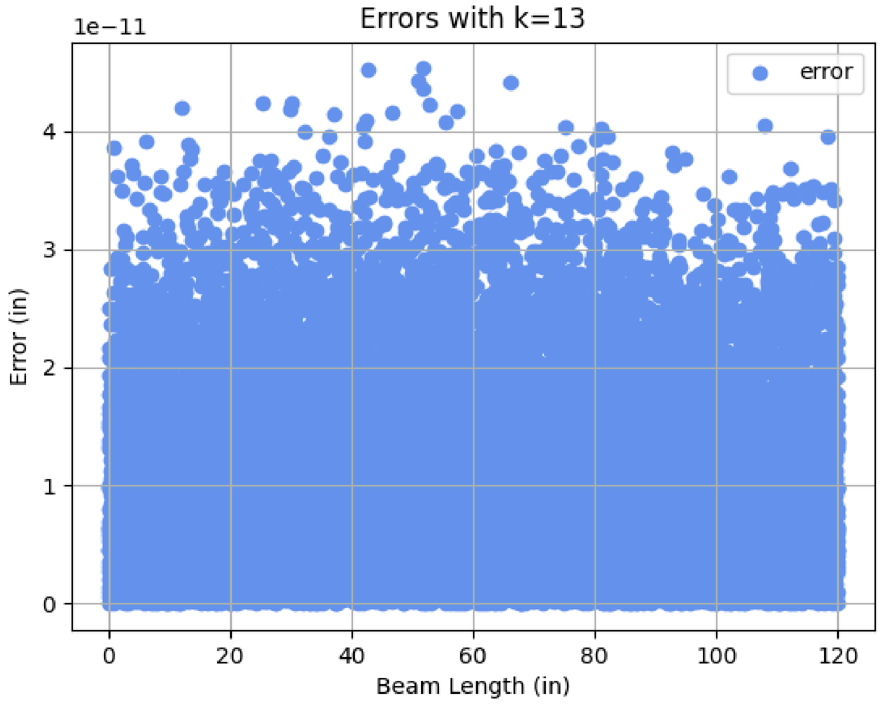
Errors with k=8
1e−9
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Error (in)
0
20
40
60
80
100
120
Beam Length (in)
error


Errors with k=9
1e−10
error
Error (in)
Beam Length (in)
0
2
4
6
8
0
20
40
60
80
100
120

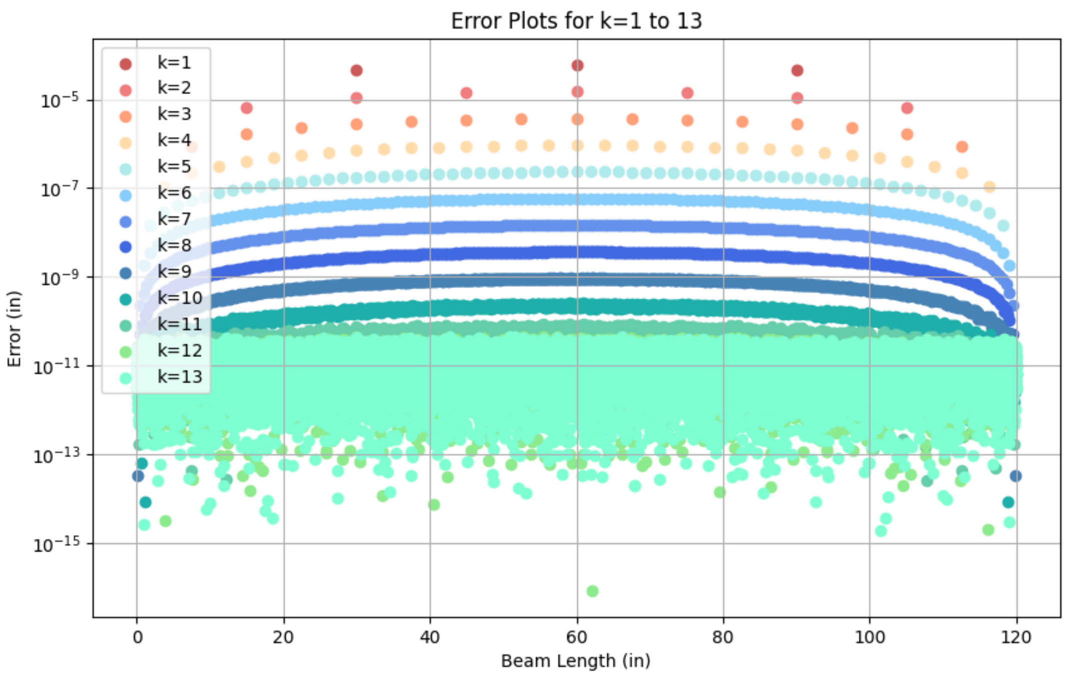

Errors with k=10
1e−10
error
Error (in)
Beam Length (in)
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120


Errors with k=11
1e−11
Error (in)
Beam Length (in)
error
0
2
4
6
8
0
20
40
60
80
100
120

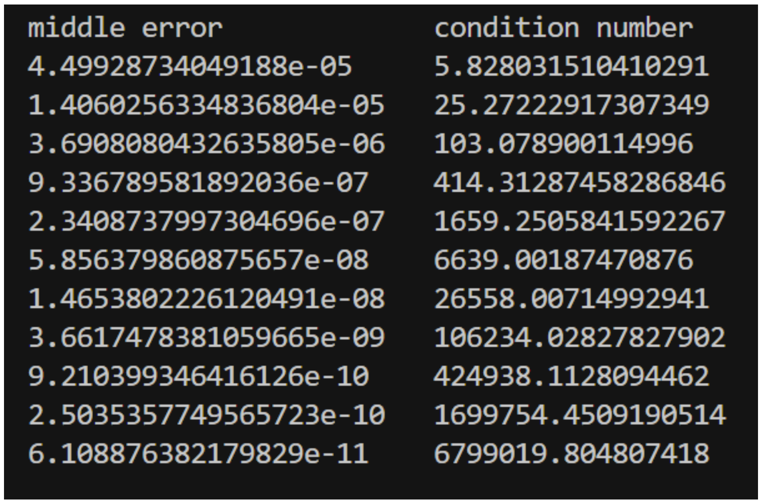

Errors with k=12
1e−11
Error (in)
Beam Length (in)
error
0
1
2
3
4
0
20
40
60
80
100
120


Errors with k=13
1e−11
Error (in)
Beam Length (in)
error
0
1
2
3
4
0
20
40
60
80
100
120

(e) (Bonus) Plot on the same figure all error plots using logarithmic y-scale.



(f) Calculate and store the error at the middle of the beam and the condition number of matrix A in vectors E and KN, respectively. (These are just suggestions for names.)



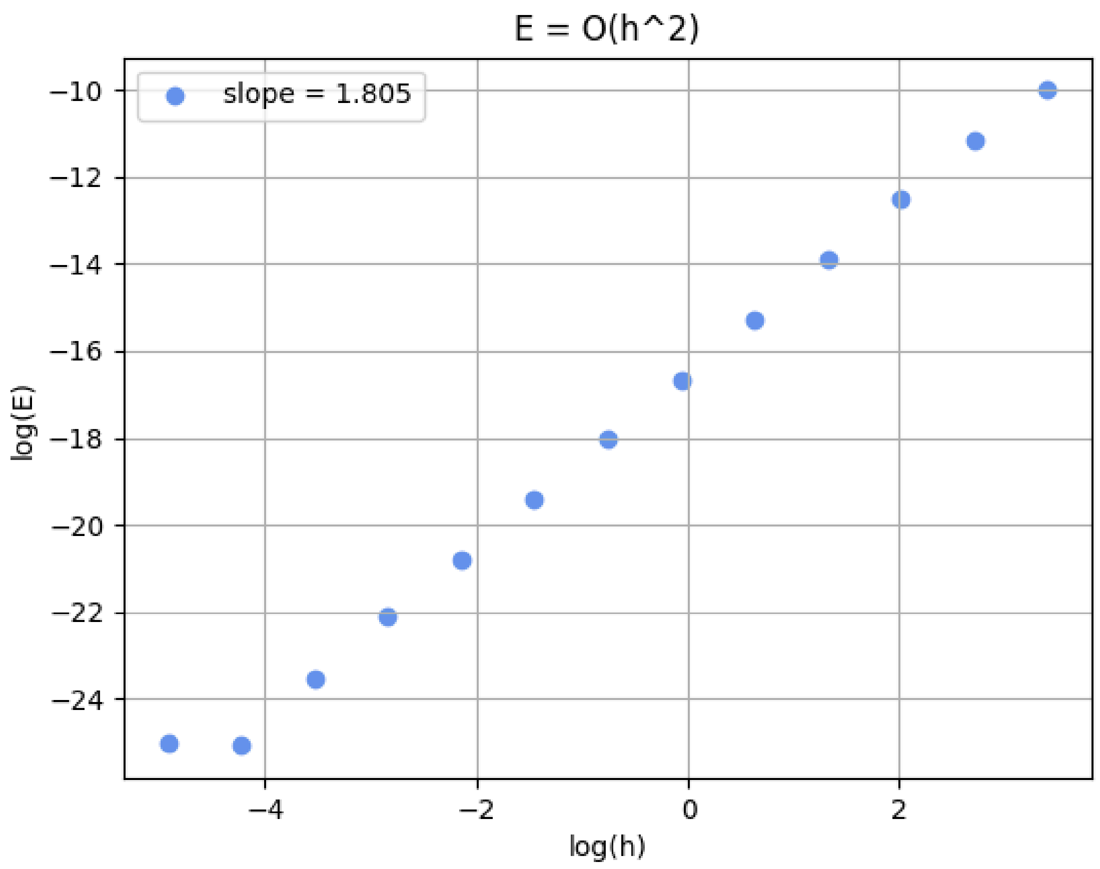
```
middle error              condition number
4.49928734049188e-05      5.828031510410291
1.4060256334836804e-05    25.27222917307349
3.6908080432635805e-06    103.078900114996
9.336789581892036e-07     414.31287458286846
2.3408737997304696e-07    1659.2505841592267
5.856379860875657e-08     6639.00187470876
1.4653802226120491e-08    26558.00714992941
3.6617478381059665e-09    106234.02827827902
9.210399346416126e-10     424938.1128094462
2.5035357749565723e-10    1699754.4509190514
6.108876382179829e-11     6799019.804807418
```

(g) Note: depending on your computer, you might not be able to run all twenty levels, and that’s OK. Just report as many levels as you can.

3. Verify that E = O(h^2):

(a) Include the graph and the vector with slopes described above.

h slopes = [1.6780736088357224, 1.9296142857256098, 1.9829382304828507, 1.995879411691188, 1.9989661195882855, 1.9987340841531769, 2.000670699484238, 1.991196829626069, 1.8792966389924255, 2.0349881605402125, 2.254572428726943, -0.08850460275433059]



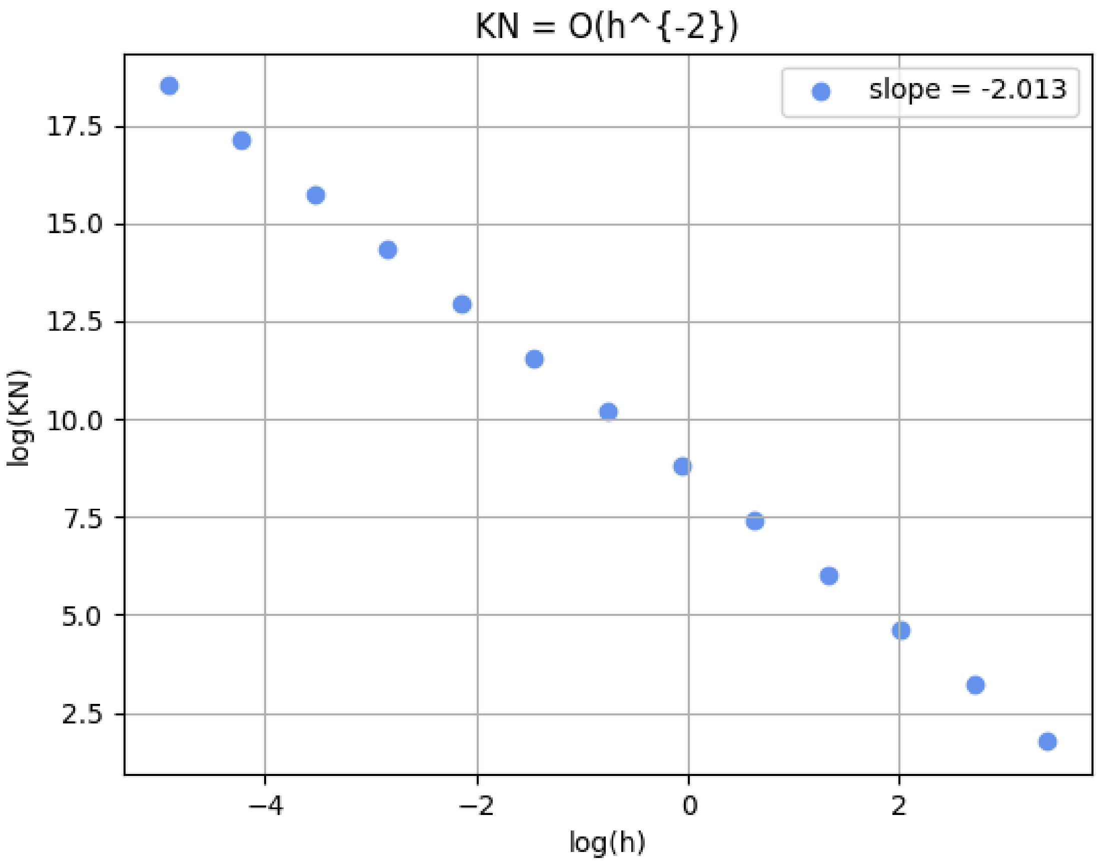
(b) Does the slope match your expectations?

Yes, as shown in the vector ‘h_slopes,’ between each point in the log plot, the slope appears to converge to a value of 2. In other words, the order appears to be 2 as theoretically predicted. Specifically, the calculated slope is around 1.805 (as indicated in the plot), it is close but slightly less than the expected value. This might suggest a slightly lower than quadratic convergence rate or could be influenced by other numerical factors such as the discretization method, round-off errors, or the specifics of the system being modeled.

4. Verify that KN = O(h^{−2}):

(a) Include the graph and the vector with slopes described above.

KN Slopes = [-2.116472342118695, -2.028124218387516, -2.0069716037290677, -2.0017392249349055, -2.0004345770968084, -2.000108629964575, -2.000027156592638, -2.0000067891230717, -2.000001697234283, -2.000000424412298, -2.000000104773298, -2.0000000369577395]



(b) Does the slope match your expectations?

Similarly, the slopes between each point on the plot appear to all be approximately -2, confirming the order of -2, indicating that as h gets smaller, the condition number grows at a rate inversely proportional to the square of h, as expected.

5. What conclusions can you draw about this practical application?

- As h decreases, the discretization of the beam becomes finer, resulting in more accurate approximations of the deflection. However, this also leads to larger condition numbers, indicating that the system becomes more sensitive to numerical errors.

- The finite difference method appears to be a valid approach for solving this BVP, with the error decreasing proportionally to h^2 until a certain point. After this point, numerical instability might increase due to the condition number increase.

- In real-world applications, one must consider the trade-off between increasing the model's refinement and the computational cost and numerical stability. In cases where higher precision is needed, advanced numerical techniques or increased computational precision might be required.

6. (Bonus) Using the general meaning of "well-conditioned problem", is finding a numerical approximation of this BVP a well-conditioned problem? Why or why not?

A well-conditioned problem is one in which a small change in the input causes only a small change in the output. Given that the condition number of the matrix A grows inversely with the square of h, the problem becomes more ill-conditioned as h decreases. This indicates that for very fine discretizations, even small numerical errors can cause large deviations in the solution. Therefore, finding a numerical approximation of this BVP becomes a less well-conditioned problem as the discretization is refined. This is typical for problems involving the finite difference method applied to differential equations.

7. Give feedback.

(a) Did you struggle with any part of the group project?

Coding in Python from scratch is hard. please use the provided functions next time

(b) What do you think we should change so our next sessions run better?

Gives a coding guide for Python programming instead of MATLAB.

(c) Do you have any conceptual questions that didn't get answered?

No, but we were confused about the notation and meaning of O(h^2) and O(h^-2).

in this case, order of the approximation That is, the error associated with the approximation you are using. In these cases, the error scales with h^2 and h^{-2}.

(d) Feel free to add any comments that you want to make but I haven't listed.

8. Submit on Canvas your report (.pdf, .doc, .docx, or .odt file).