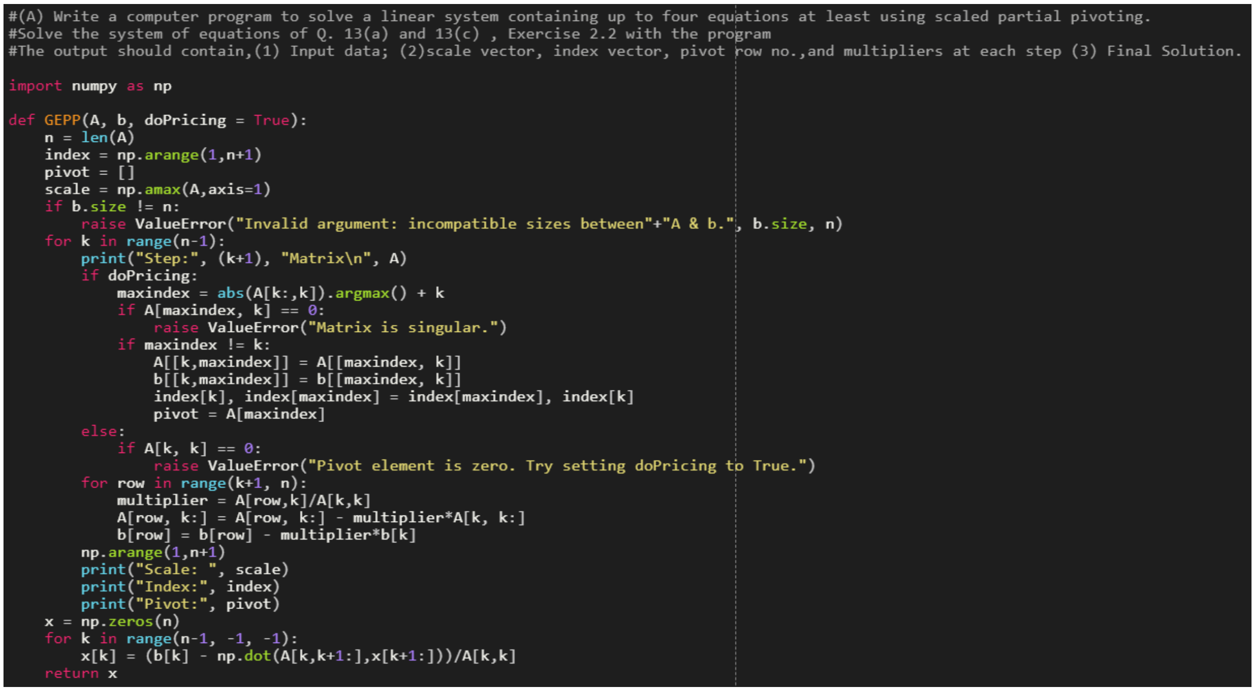
Ocean Lu
CS 3010
Professor Raheja
10/24/19

# Project 1: Report

Please view my code at:
https://colab.research.google.com/drive/1De4PQM87UJqf8soEP8hAUsP6OkLOhwjQ

Task 1:

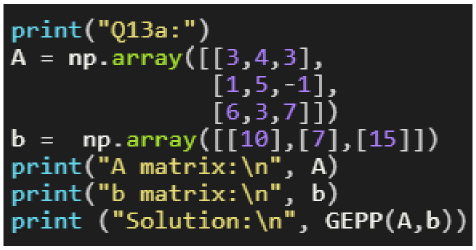


```python
#(A) Write a computer program to solve a linear system containing up to four equations at least using scaled partial pivoting.
#Solve the system of equations of Q. 13(a) and 13(c) , Exercise 2.2 with the program
#The output should contain,(1) Input data; (2)scale vector, index vector, pivot row no.,and multipliers at each step (3) Final Solution.

import numpy as np

def GEPP(A, b, doPricing = True):
    n = len(A)
    index = np.arange(1,n+1)
    pivot = []
    scale = np.amax(A,axis=1)
    if b.size != n:
        raise ValueError("Invalid argument: incompatible sizes between"+"A & b.", b.size, n)
    for k in range(n-1):
        print("Step:", (k+1), "Matrix\n", A)
        if doPricing:
            maxindex = abs(A[k:,k]).argmax() + k
            if A[maxindex, k] == 0:
                raise ValueError("Matrix is singular.")
            if maxindex != k:
                A[[k,maxindex]] = A[[maxindex, k]]
                b[[k,maxindex]] = b[[maxindex, k]]
                index[k], index[maxindex] = index[maxindex], index[k]
                pivot = A[maxindex]
        else:
            if A[k, k] == 0:
                raise ValueError("Pivot element is zero. Try setting doPricing to True.")
        for row in range(k+1, n):
            multiplier = A[row,k]/A[k,k]
            A[row, k:] = A[row, k:] - multiplier*A[k, k:]
            b[row] = b[row] - multiplier*b[k]
        np.arange(1,n+1)
        print("Scale: ", scale)
        print("Index:", index)
        print("Pivot:", pivot)
    x = np.zeros(n)
    for k in range(n-1, -1, -1):
        x[k] = (b[k] - np.dot(A[k,k+1:],x[k+1:]))/A[k,k]
    return x
```

My code has an input with A matrix as a (n x n) nonsingular matrix, and b, as a (n x 1) vector. The output is the solution to Ax = b. k represents the current pivot row, because GE traverses the matrix in the upper right triangle, we also use k for indicating the k-diagonal column index

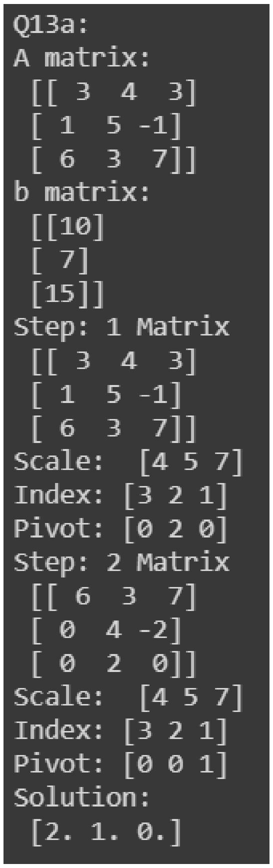
Once in the for loop, I choose the largest pivot element below (and including) k. I swap rows and equate the solution in the column.

To solve Q13a, I call the specific functions:



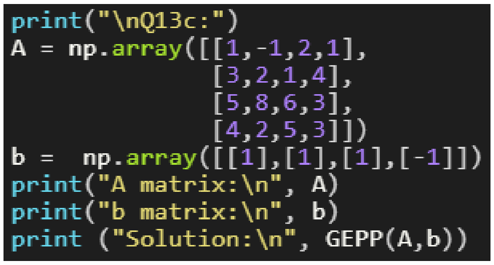
```python
print("Q13a:")
A = np.array([[3,4,3],
              [1,5,-1],
              [6,3,7]])
b =  np.array([[10],[7],[15]])
print("A matrix:\n", A)
print("b matrix:\n", b)
print ("Solution:\n", GEPP(A,b))
```

Here are my outputs:



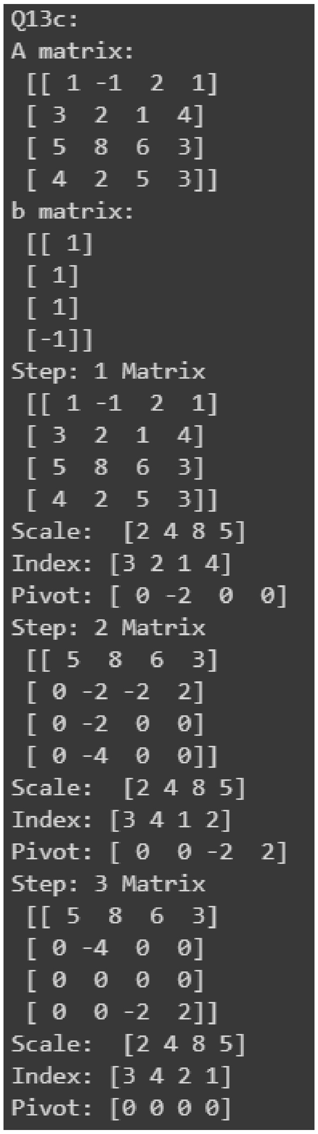
```
Q13a:
A matrix:
 [[ 3  4  3]
 [ 1  5 -1]
 [ 6  3  7]]
b matrix:
 [[10]
 [ 7]
 [15]]
Step: 1 Matrix
 [[ 3  4  3]
 [ 1  5 -1]
 [ 6  3  7]]
Scale:  [4 5 7]
Index: [3 2 1]
Pivot: [0 2 0]
Step: 2 Matrix
 [[ 6  3  7]
 [ 0  4 -2]
 [ 0  2  0]]
Scale:  [4 5 7]
Index: [3 2 1]
Pivot: [0 0 1]
Solution:
 [2. 1. 0.]
```

To solve Q13c, I call the specific functions:

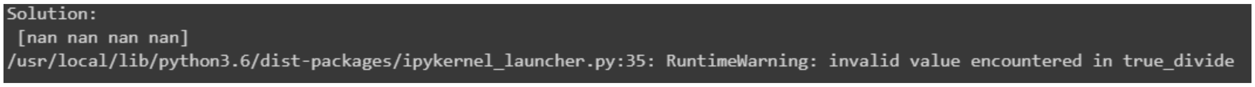


```python
print("\nQ13c:")
A = np.array([[1,-1,2,1],
              [3,2,1,4],
              [5,8,6,3],
              [4,2,5,3]])
b =  np.array([[1],[1],[1],[-1]])
print("A matrix:\n", A)
print("b matrix:\n", b)
print ("Solution:\n", GEPP(A,b))
```

Here are my outputs:



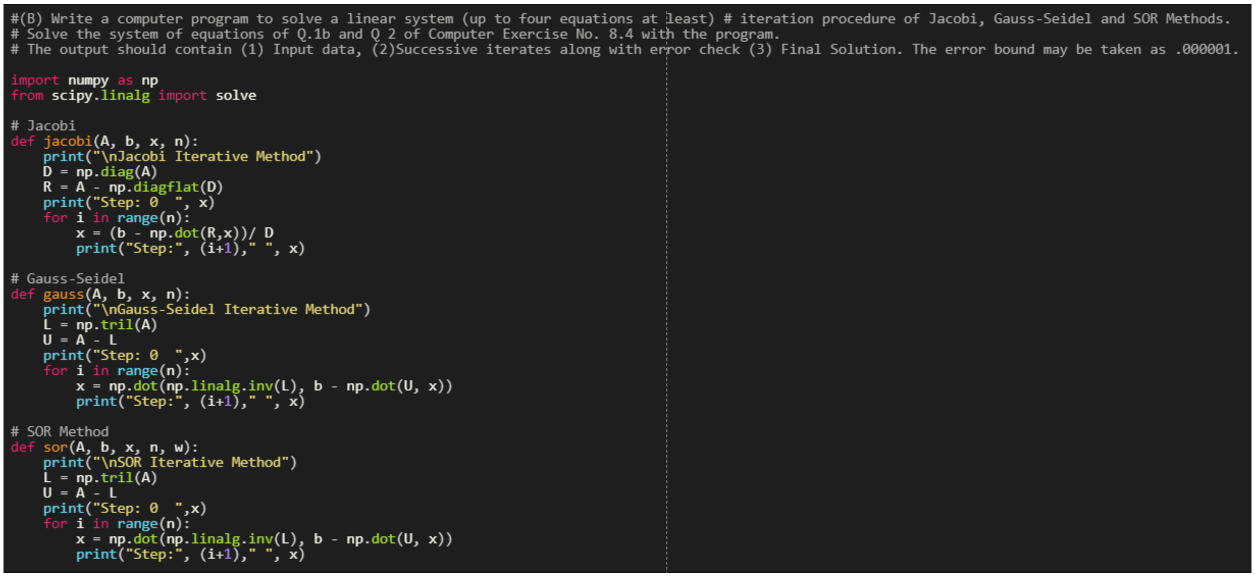
```
Q13c:
A matrix:
 [[ 1 -1  2  1]
 [ 3  2  1  4]
 [ 5  8  6  3]
 [ 4  2  5  3]]
b matrix:
 [[ 1]
 [ 1]
 [ 1]
 [-1]]
Step: 1 Matrix
 [[ 1 -1  2  1]
 [ 3  2  1  4]
 [ 5  8  6  3]
 [ 4  2  5  3]]
Scale:  [2 4 8 5]
Index: [3 2 1 4]
Pivot: [ 0 -2  0  0]
Step: 2 Matrix
 [[ 5  8  6  3]
 [ 0 -2 -2  2]
 [ 0 -2  0  0]
 [ 0 -4  0  0]]
Scale:  [2 4 8 5]
Index: [3 4 1 2]
Pivot: [ 0  0 -2  2]
Step: 3 Matrix
 [[ 5  8  6  3]
 [ 0 -4  0  0]
 [ 0  0  0  0]
 [ 0  0 -2  2]]
Scale:  [2 4 8 5]
Index: [3 4 2 1]
Pivot: [0 0 0 0]
```



```
Solution:
 [nan nan nan nan]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:35: RuntimeWarning: invalid value encountered in true_divide
```

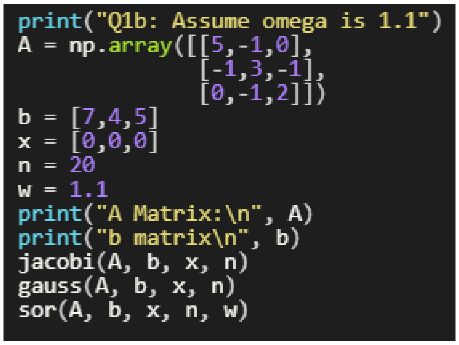
Task 2:

This is my code to find specific iterative methods:



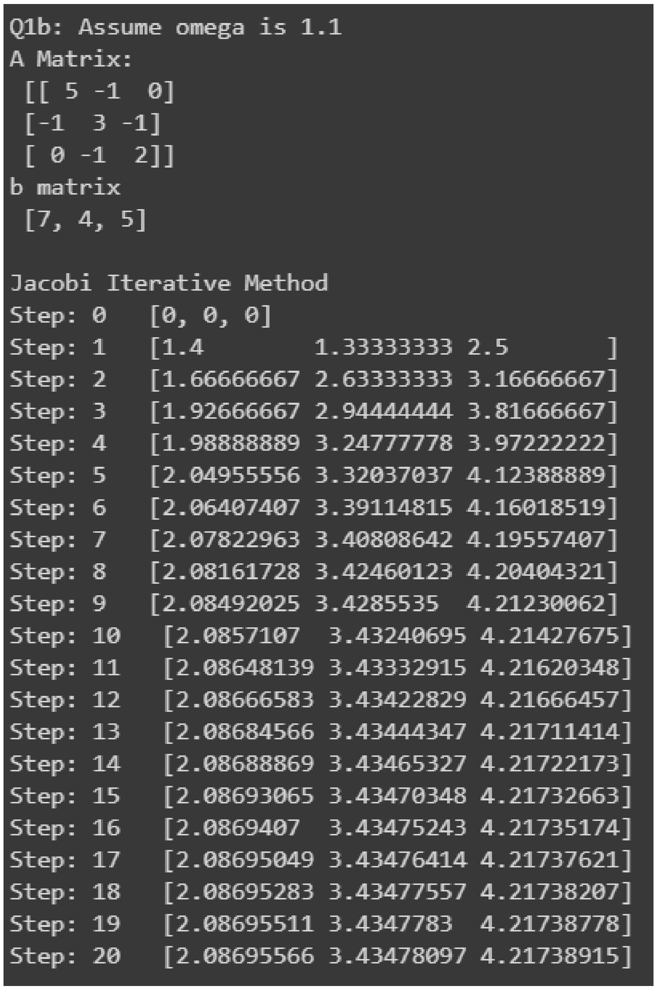
```python
#(B) Write a computer program to solve a linear system (up to four equations at least) # iteration procedure of Jacobi, Gauss-Seidel and SOR Methods.
# Solve the system of equations of Q.1b and Q 2 of Computer Exercise No. 8.4 with the program.
# The output should contain (1) Input data, (2)Successive iterates along with error check (3) Final Solution. The error bound may be taken as .000001.

import numpy as np
from scipy.linalg import solve

# Jacobi
def jacobi(A, b, x, n):
    print("\nJacobi Iterative Method")
    D = np.diag(A)
    R = A - np.diagflat(D)
    print("Step: 0  ", x)
    for i in range(n):
        x = (b - np.dot(R,x))/ D
        print("Step:", (i+1)," ", x)

# Gauss-Seidel
def gauss(A, b, x, n):
    print("\nGauss-Seidel Iterative Method")
    L = np.tril(A)
    U = A - L
    print("Step: 0  ",x)
    for i in range(n):
        x = np.dot(np.linalg.inv(L), b - np.dot(U, x))
        print("Step:", (i+1)," ", x)

# SOR Method
def sor(A, b, x, n, w):
    print("\nSOR Iterative Method")
    L = np.tril(A)
    U = A - L
    print("Step: 0  ",x)
    for i in range(n):
        x = np.dot(np.linalg.inv(L), b - np.dot(U, x))
        print("Step:", (i+1)," ", x)
```

To solve Q1b, this is my main method code:

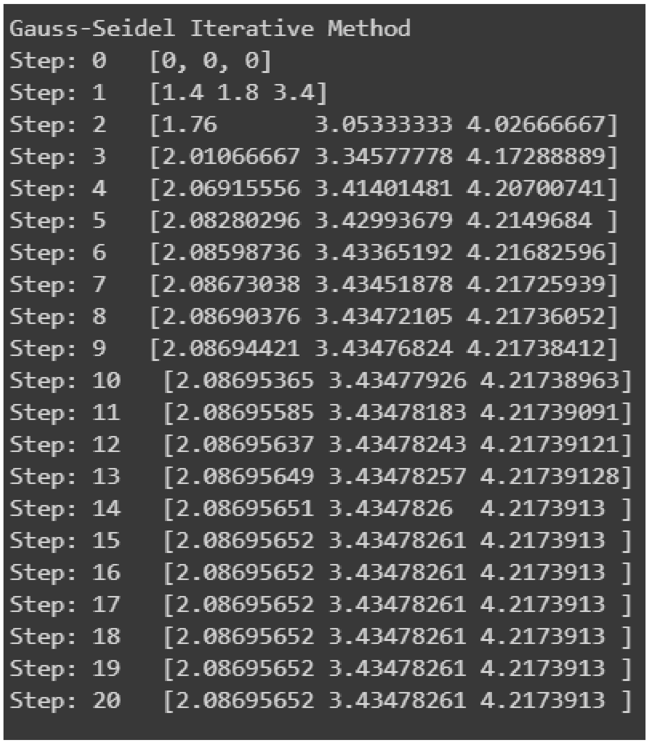


```python
print("Q1b: Assume omega is 1.1")
A = np.array([[5,-1,0],
              [-1,3,-1],
              [0,-1,2]])
b = [7,4,5]
x = [0,0,0]
n = 20
w = 1.1
print("A Matrix:\n", A)
print("b matrix\n", b)
jacobi(A, b, x, n)
gauss(A, b, x, n)
sor(A, b, x, n, w)
```

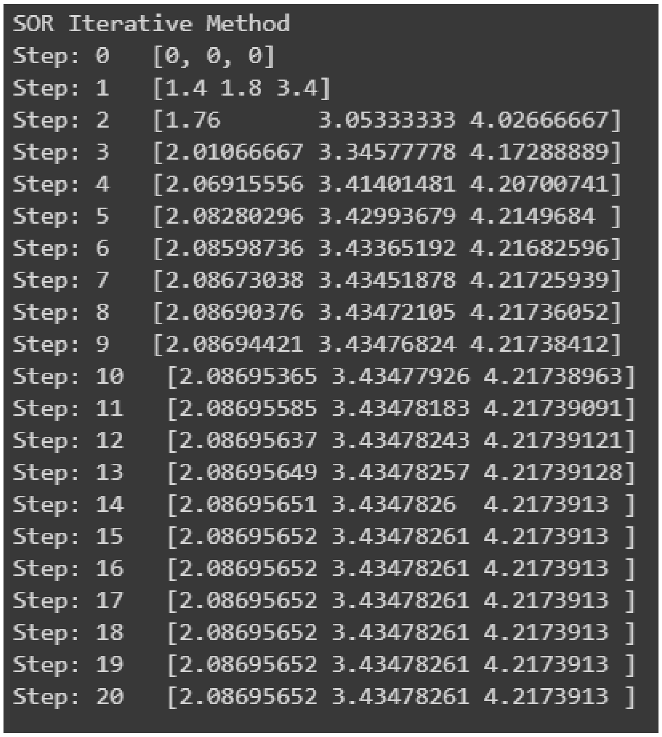
This is the output:



```
Q1b: Assume omega is 1.1
A Matrix:
 [[ 5 -1  0]
 [-1  3 -1]
 [ 0 -1  2]]
b matrix
 [7, 4, 5]

Jacobi Iterative Method
Step: 0   [0, 0, 0]
Step: 1   [1.4        1.33333333 2.5       ]
Step: 2   [1.66666667 2.63333333 3.16666667]
Step: 3   [1.92666667 2.94444444 3.81666667]
Step: 4   [1.98888889 3.24777778 3.97222222]
Step: 5   [2.04955556 3.32037037 4.12388889]
Step: 6   [2.06407407 3.39114815 4.16018519]
Step: 7   [2.07822963 3.40808642 4.19557407]
Step: 8   [2.08161728 3.42460123 4.20404321]
Step: 9   [2.08492025 3.4285535  4.21230062]
Step: 10   [2.0857107  3.43240695 4.21427675]
Step: 11   [2.08648139 3.43332915 4.21620348]
Step: 12   [2.08666583 3.43422829 4.21666457]
Step: 13   [2.08684566 3.43444347 4.21711414]
Step: 14   [2.08688869 3.43465327 4.21722173]
Step: 15   [2.08693065 3.43470348 4.21732663]
Step: 16   [2.0869407  3.43475243 4.21735174]
Step: 17   [2.08695049 3.43476414 4.21737621]
Step: 18   [2.08695283 3.43477557 4.21738207]
Step: 19   [2.08695511 3.4347783  4.21738778]
Step: 20   [2.08695566 3.43478097 4.21738915]
```

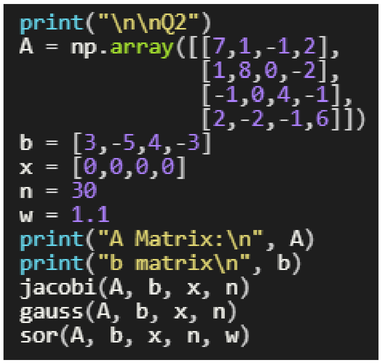


```
Gauss-Seidel Iterative Method
Step: 0   [0, 0, 0]
Step: 1   [1.4 1.8 3.4]
Step: 2   [1.76       3.05333333 4.02666667]
Step: 3   [2.01066667 3.34577778 4.17288889]
Step: 4   [2.06915556 3.41401481 4.20700741]
Step: 5   [2.08280296 3.42993679 4.2149684 ]
Step: 6   [2.08598736 3.43365192 4.21682596]
Step: 7   [2.08673038 3.43451878 4.21725939]
Step: 8   [2.08690376 3.43472105 4.21736052]
Step: 9   [2.08694421 3.43476824 4.21738412]
Step: 10   [2.08695365 3.43477926 4.21738963]
Step: 11   [2.08695585 3.43478183 4.21739091]
Step: 12   [2.08695637 3.43478243 4.21739121]
Step: 13   [2.08695649 3.43478257 4.21739128]
Step: 14   [2.08695651 3.4347826  4.2173913 ]
Step: 15   [2.08695652 3.43478261 4.2173913 ]
Step: 16   [2.08695652 3.43478261 4.2173913 ]
Step: 17   [2.08695652 3.43478261 4.2173913 ]
Step: 18   [2.08695652 3.43478261 4.2173913 ]
Step: 19   [2.08695652 3.43478261 4.2173913 ]
Step: 20   [2.08695652 3.43478261 4.2173913 ]
```



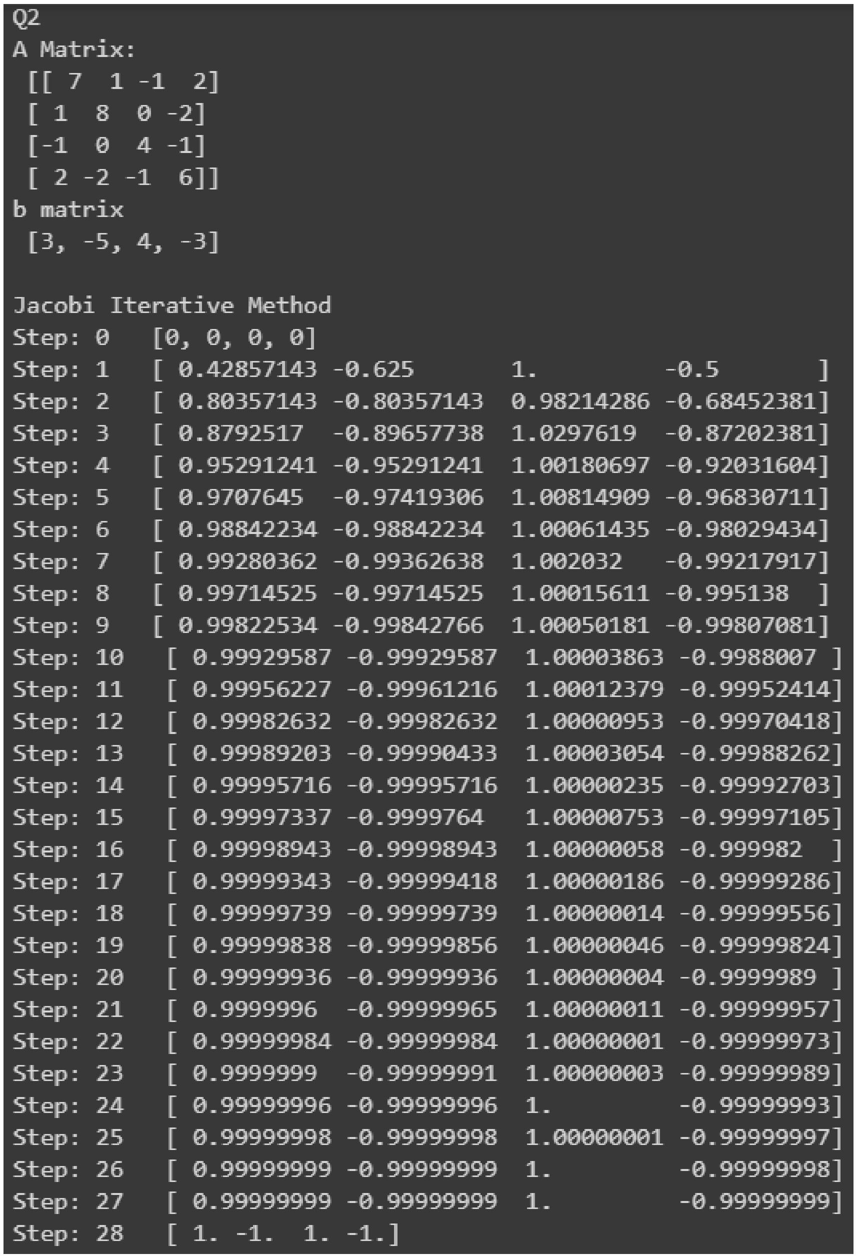
```
SOR Iterative Method
Step: 0   [0, 0, 0]
Step: 1   [1.4 1.8 3.4]
Step: 2   [1.76       3.05333333 4.02666667]
Step: 3   [2.01066667 3.34577778 4.17288889]
Step: 4   [2.06915556 3.41401481 4.20700741]
Step: 5   [2.08280296 3.42993679 4.2149684 ]
Step: 6   [2.08598736 3.43365192 4.21682596]
Step: 7   [2.08673038 3.43451878 4.21725939]
Step: 8   [2.08690376 3.43472105 4.21736052]
Step: 9   [2.08694421 3.43476824 4.21738412]
Step: 10   [2.08695365 3.43477926 4.21738963]
Step: 11   [2.08695585 3.43478183 4.21739091]
Step: 12   [2.08695637 3.43478243 4.21739121]
Step: 13   [2.08695649 3.43478257 4.21739128]
Step: 14   [2.08695651 3.4347826  4.2173913 ]
Step: 15   [2.08695652 3.43478261 4.2173913 ]
Step: 16   [2.08695652 3.43478261 4.2173913 ]
Step: 17   [2.08695652 3.43478261 4.2173913 ]
Step: 18   [2.08695652 3.43478261 4.2173913 ]
Step: 19   [2.08695652 3.43478261 4.2173913 ]
Step: 20   [2.08695652 3.43478261 4.2173913 ]
```

To solve Q2, this is my main method code:

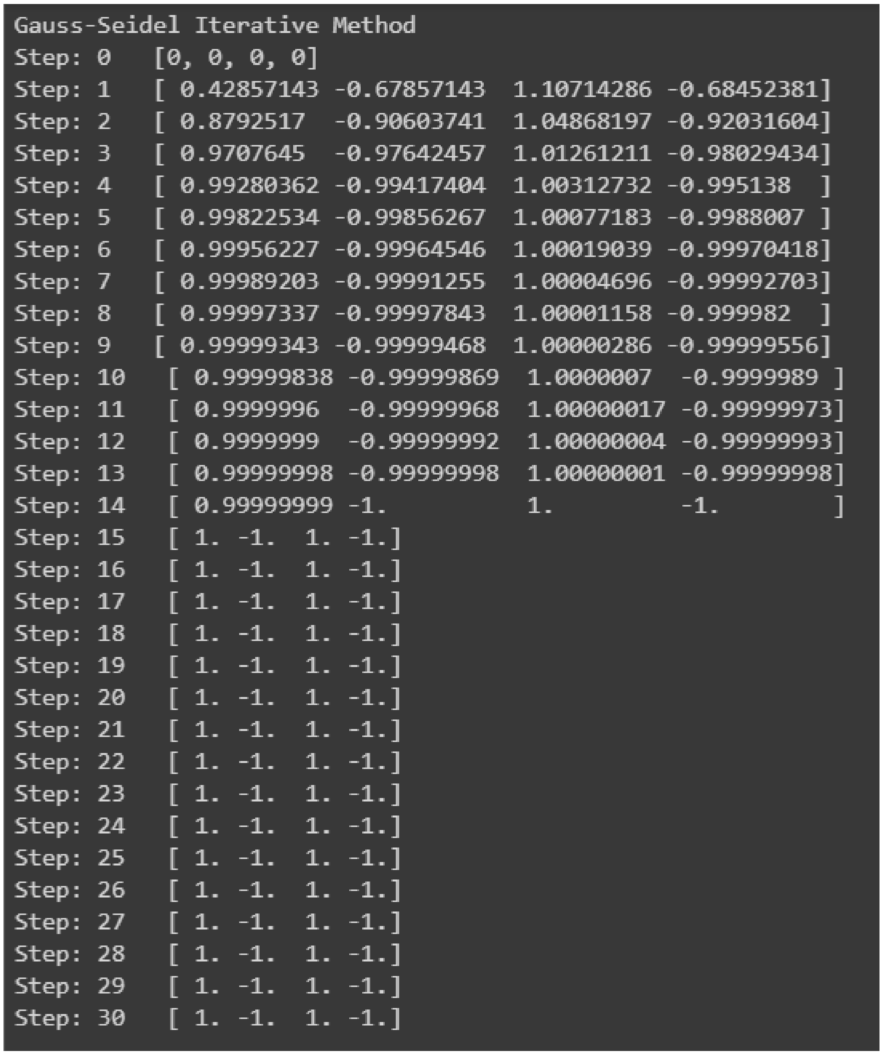


```
print("\n\nQ2")
A = np.array([[7,1,-1,2],
              [1,8,0,-2],
              [-1,0,4,-1],
              [2,-2,-1,6]])
b = [3,-5,4,-3]
x = [0,0,0,0]
n = 30
w = 1.1
print("A Matrix:\n", A)
print("b matrix\n", b)
jacobi(A, b, x, n)
gauss(A, b, x, n)
sor(A, b, x, n, w)
```

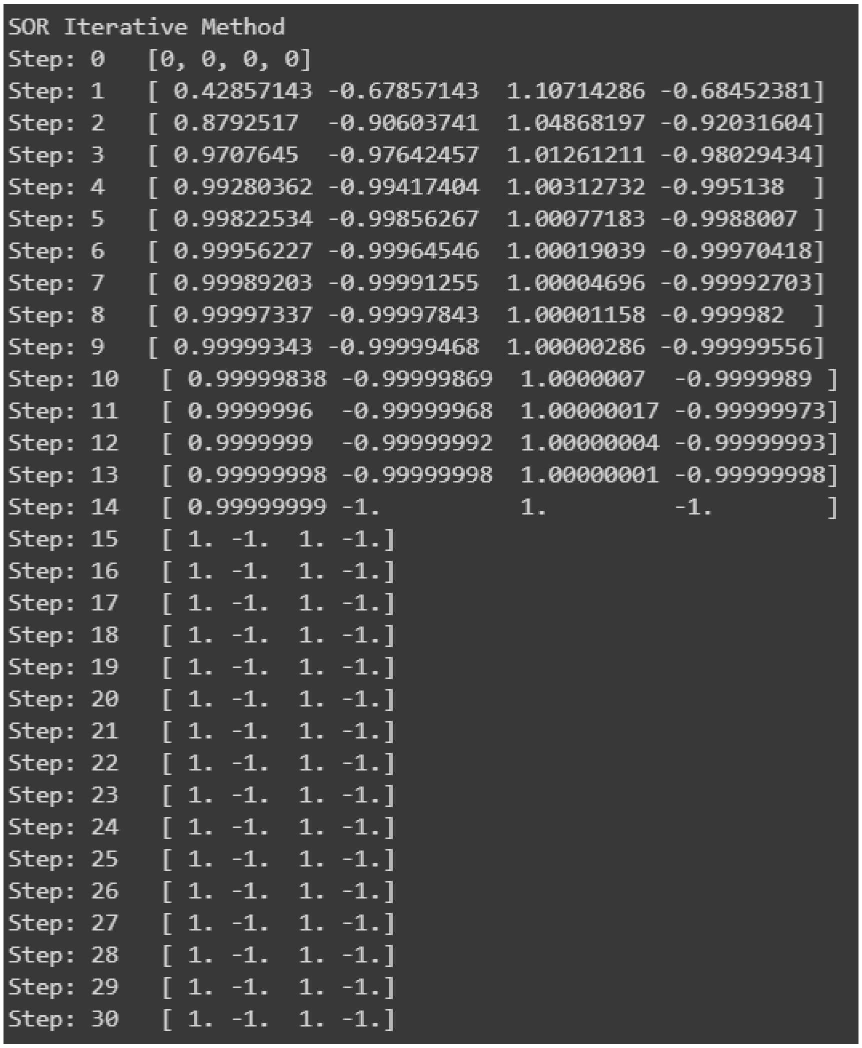
This is the output:



```
Q2
A Matrix:
 [[ 7  1 -1  2]
 [ 1  8  0 -2]
 [-1  0  4 -1]
 [ 2 -2 -1  6]]
b matrix
 [3, -5, 4, -3]

Jacobi Iterative Method
Step: 0   [0, 0, 0, 0]
Step: 1   [ 0.42857143 -0.625       1.         -0.5       ]
Step: 2   [ 0.80357143 -0.80357143  0.98214286 -0.68452381]
Step: 3   [ 0.8792517  -0.89657738  1.0297619  -0.87202381]
Step: 4   [ 0.95291241 -0.95291241  1.00180697 -0.92031604]
Step: 5   [ 0.9707645  -0.97419306  1.00814909 -0.96830711]
Step: 6   [ 0.98842234 -0.98842234  1.00061435 -0.98029434]
Step: 7   [ 0.99280362 -0.99362638  1.002032   -0.99217917]
Step: 8   [ 0.99714525 -0.99714525  1.00015611 -0.995138  ]
Step: 9   [ 0.99822534 -0.99842766  1.00050181 -0.99807081]
Step: 10   [ 0.99929587 -0.99929587  1.00003863 -0.9988007 ]
Step: 11   [ 0.99956227 -0.99961216  1.00012379 -0.99952414]
Step: 12   [ 0.99982632 -0.99982632  1.00000953 -0.99970418]
Step: 13   [ 0.99989203 -0.99990433  1.00003054 -0.99988262]
Step: 14   [ 0.99995716 -0.99995716  1.00000235 -0.99992703]
Step: 15   [ 0.99997337 -0.9999764   1.00000753 -0.99997105]
Step: 16   [ 0.99998943 -0.99998943  1.00000058 -0.999982  ]
Step: 17   [ 0.99999343 -0.99999418  1.00000186 -0.99999286]
Step: 18   [ 0.99999739 -0.99999739  1.00000014 -0.99999556]
Step: 19   [ 0.99999838 -0.99999856  1.00000046 -0.99999824]
Step: 20   [ 0.99999936 -0.99999936  1.00000004 -0.9999989 ]
Step: 21   [ 0.9999996  -0.99999965  1.00000011 -0.99999957]
Step: 22   [ 0.99999984 -0.99999984  1.00000001 -0.99999973]
Step: 23   [ 0.9999999  -0.99999991  1.00000003 -0.99999989]
Step: 24   [ 0.99999996 -0.99999996  1.         -0.99999993]
Step: 25   [ 0.99999998 -0.99999998  1.00000001 -0.99999997]
Step: 26   [ 0.99999999 -0.99999999  1.         -0.99999998]
Step: 27   [ 0.99999999 -0.99999999  1.         -0.99999999]
Step: 28   [ 1. -1.  1. -1.]
```



```
Gauss-Seidel Iterative Method
Step: 0   [0, 0, 0, 0]
Step: 1   [ 0.42857143 -0.67857143  1.10714286 -0.68452381]
Step: 2   [ 0.8792517  -0.90603741  1.04868197 -0.92031604]
Step: 3   [ 0.9707645  -0.97642457  1.01261211 -0.98029434]
Step: 4   [ 0.99280362 -0.99417404  1.00312732 -0.995138  ]
Step: 5   [ 0.99822534 -0.99856267  1.00077183 -0.9988007 ]
Step: 6   [ 0.99956227 -0.99964546  1.00019039 -0.99970418]
Step: 7   [ 0.99989203 -0.99991255  1.00004696 -0.99992703]
Step: 8   [ 0.99997337 -0.99997843  1.00001158 -0.999982  ]
Step: 9   [ 0.99999343 -0.99999468  1.00000286 -0.99999556]
Step: 10   [ 0.99999838 -0.99999869  1.0000007  -0.9999989 ]
Step: 11   [ 0.9999996  -0.99999968  1.00000017 -0.99999973]
Step: 12   [ 0.9999999  -0.99999992  1.00000004 -0.99999993]
Step: 13   [ 0.99999998 -0.99999998  1.00000001 -0.99999998]
Step: 14   [ 0.99999999 -1.          1.         -1.        ]
Step: 15   [ 1. -1.  1. -1.]
Step: 16   [ 1. -1.  1. -1.]
Step: 17   [ 1. -1.  1. -1.]
Step: 18   [ 1. -1.  1. -1.]
Step: 19   [ 1. -1.  1. -1.]
Step: 20   [ 1. -1.  1. -1.]
Step: 21   [ 1. -1.  1. -1.]
Step: 22   [ 1. -1.  1. -1.]
Step: 23   [ 1. -1.  1. -1.]
Step: 24   [ 1. -1.  1. -1.]
Step: 25   [ 1. -1.  1. -1.]
Step: 26   [ 1. -1.  1. -1.]
Step: 27   [ 1. -1.  1. -1.]
Step: 28   [ 1. -1.  1. -1.]
Step: 29   [ 1. -1.  1. -1.]
Step: 30   [ 1. -1.  1. -1.]
```



```
SOR Iterative Method
Step: 0   [0, 0, 0, 0]
Step: 1   [ 0.42857143 -0.67857143  1.10714286 -0.68452381]
Step: 2   [ 0.8792517  -0.90603741  1.04868197 -0.92031604]
Step: 3   [ 0.9707645  -0.97642457  1.01261211 -0.98029434]
Step: 4   [ 0.99280362 -0.99417404  1.00312732 -0.995138  ]
Step: 5   [ 0.99822534 -0.99856267  1.00077183 -0.9988007 ]
Step: 6   [ 0.99956227 -0.99964546  1.00019039 -0.99970418]
Step: 7   [ 0.99989203 -0.99991255  1.00004696 -0.99992703]
Step: 8   [ 0.99997337 -0.99997843  1.00001158 -0.999982  ]
Step: 9   [ 0.99999343 -0.99999468  1.00000286 -0.99999556]
Step: 10   [ 0.99999838 -0.99999869  1.0000007  -0.9999989 ]
Step: 11   [ 0.9999996  -0.99999968  1.00000017 -0.99999973]
Step: 12   [ 0.9999999  -0.99999992  1.00000004 -0.99999993]
Step: 13   [ 0.99999998 -0.99999998  1.00000001 -0.99999998]
Step: 14   [ 0.99999999 -1.          1.         -1.        ]
Step: 15   [ 1. -1.  1. -1.]
Step: 16   [ 1. -1.  1. -1.]
Step: 17   [ 1. -1.  1. -1.]
Step: 18   [ 1. -1.  1. -1.]
Step: 19   [ 1. -1.  1. -1.]
Step: 20   [ 1. -1.  1. -1.]
Step: 21   [ 1. -1.  1. -1.]
Step: 22   [ 1. -1.  1. -1.]
Step: 23   [ 1. -1.  1. -1.]
Step: 24   [ 1. -1.  1. -1.]
Step: 25   [ 1. -1.  1. -1.]
Step: 26   [ 1. -1.  1. -1.]
Step: 27   [ 1. -1.  1. -1.]
Step: 28   [ 1. -1.  1. -1.]
Step: 29   [ 1. -1.  1. -1.]
Step: 30   [ 1. -1.  1. -1.]
```