Lab 6 Report

Amlan Nayak MS18197

19th March 2021

1 Write a code to get the inverse of a 20 x 20 matrix using Partition matrix method

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix} A^{-1} = \begin{bmatrix} X & Y \\ Z & V \end{bmatrix}$$

1.1 Algorithm & Discussion

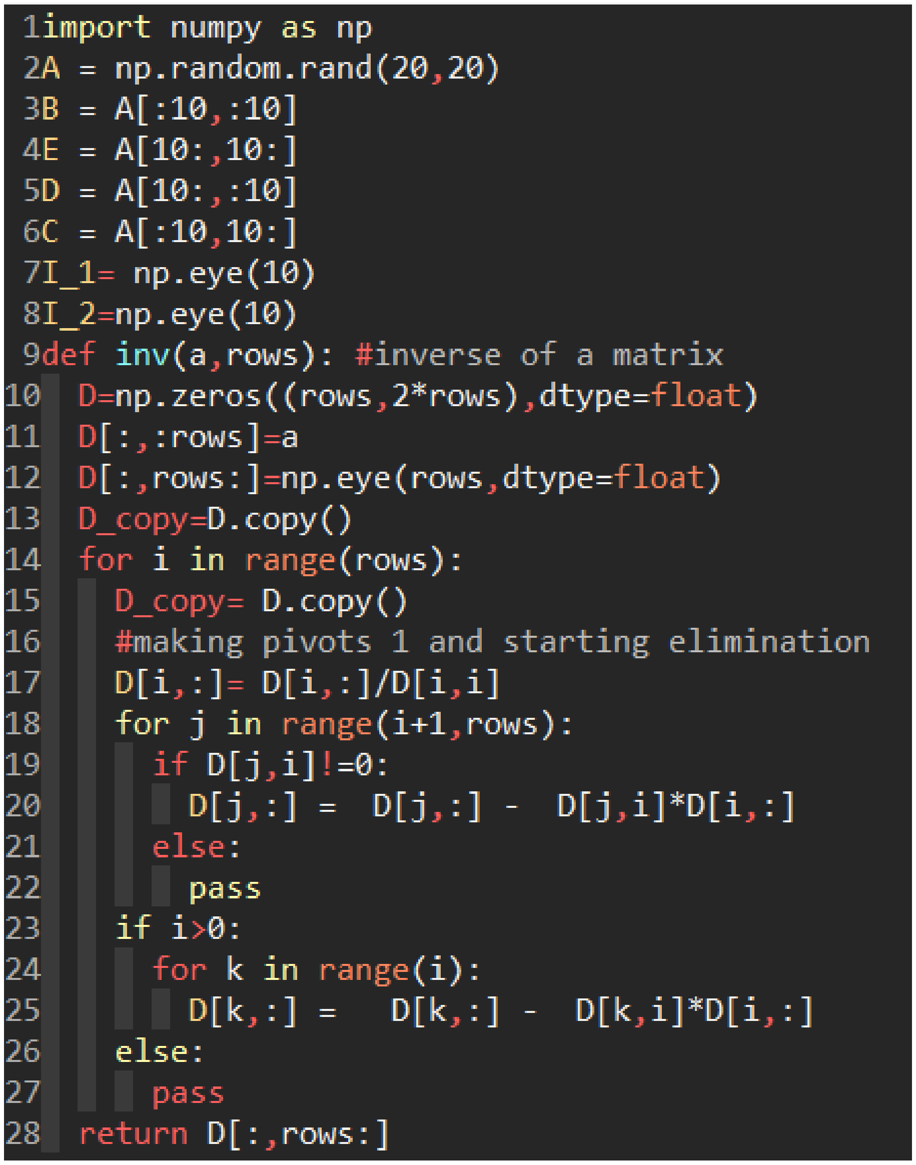
We mainly use the relations for the partition matrix derived in class alongside numpy module. We know dimensions of $(B, X) = r \times r, (Y, C) = s \times s, (Z, D) = r \times s$ and $(V, E) = s \times r$. Let dimensions of $I_1 = r \times r$ and $I_2 = s \times s$. We use:

$$\begin{aligned} V &= [E - DB^{-1}C]^{-1}I_2 & \qquad Z &= -[VDB^{-1}] \\ Y &= -[B^{-1}CV & \qquad X &= B^{-1}[I_1 - CZ] \end{aligned}$$

Out of 1000 loops for the partition method, best of 5 was 1.93 ms per loop. While for the same matrix, the inverse through LU decomposition method has a time of 2.95 ms.

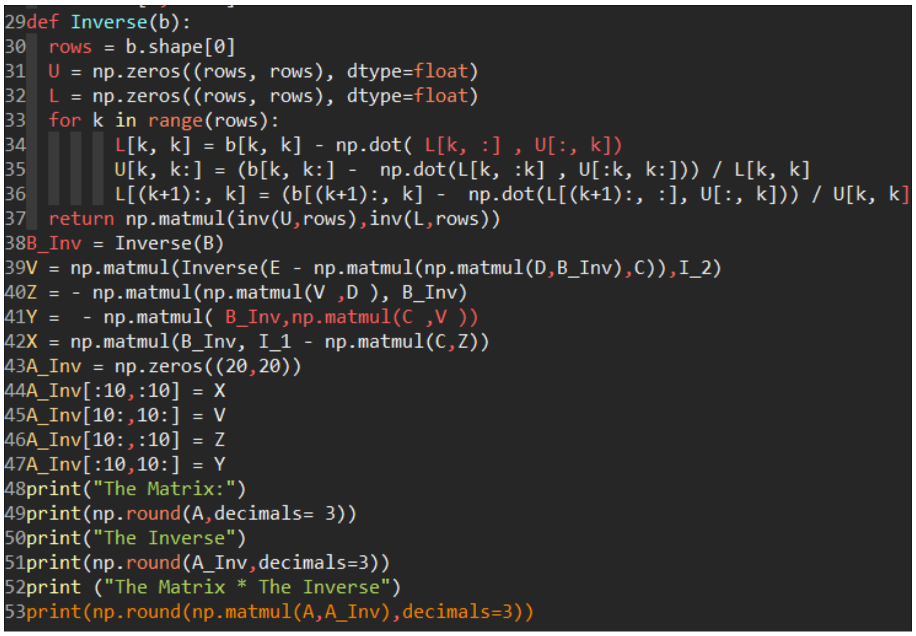
Even though we used the LU decomposition to find the inverse of smaller partition matrices in our computation, the method was still much faster than inverse through full LU decomposition of the matrix.

1.2 Code and Output



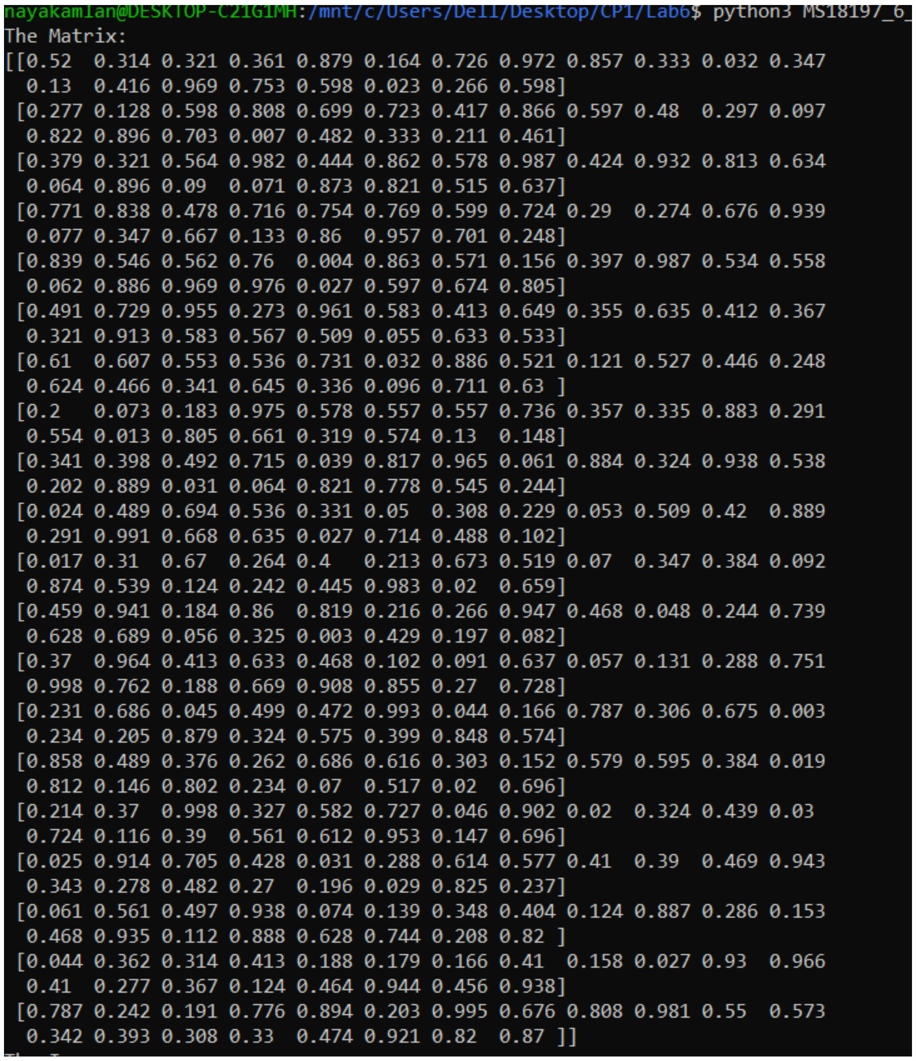
```
import numpy as np
A = np.random.rand(20,20)
B = A[:10,:10]
E = A[10:,10:]
D = A[10:,:10]
C = A[:10,10:]
I_1= np.eye(10)
I_2=np.eye(10)
def inv(a,rows): #inverse of a matrix
    D=np.zeros((rows,2*rows),dtype=float)
    D[:,:rows]=a
    D[:,rows:]=np.eye(rows,dtype=float)
    D_copy=D.copy()
    for i in range(rows):
        D_copy= D.copy()
        #making pivots 1 and starting elimination
        D[i,:]= D[i,:]/D[i,i]
        for j in range(i+1,rows):
            if D[j,i]!=0:
                D[j,:] =  D[j,:] -  D[j,i]*D[i,:]
            else:
                pass
        if i>0:
            for k in range(i):
                D[k,:] =   D[k,:] -  D[k,i]*D[i,:]
        else:
            pass
    return D[:,rows:]
```

Figure 1: Code in emacs part 1



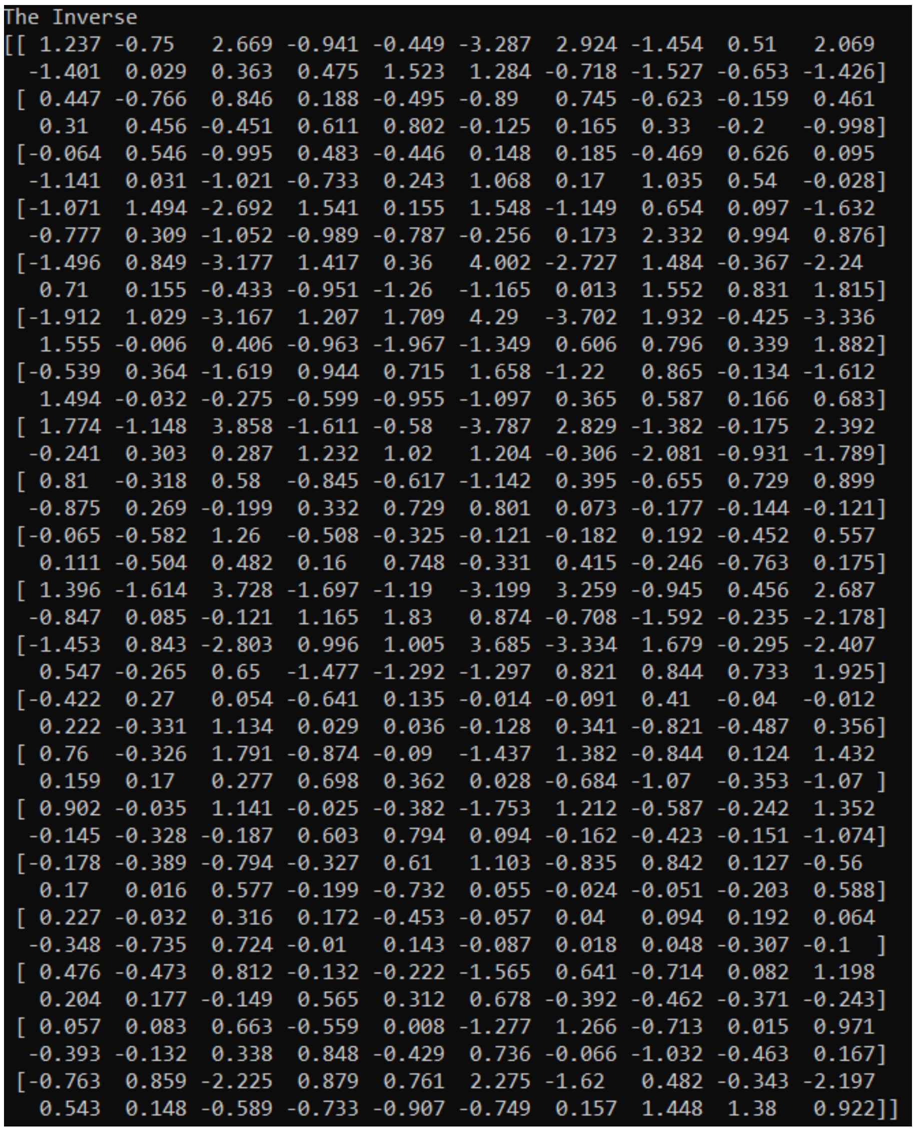
```python
def Inverse(b):
    rows = b.shape[0]
    U = np.zeros((rows, rows), dtype=float)
    L = np.zeros((rows, rows), dtype=float)
    for k in range(rows):
            L[k, k] = b[k, k] - np.dot( L[k, :] , U[:, k])
            U[k, k:] = (b[k, k:] -  np.dot(L[k, :k] , U[:k, k:])) / L[k, k]
            L[(k+1):, k] = (b[(k+1):, k] -  np.dot(L[(k+1):, :], U[:, k])) / U[k, k]
    return np.matmul(inv(U,rows),inv(L,rows))
B_Inv = Inverse(B)
V = np.matmul(Inverse(E - np.matmul(np.matmul(D,B_Inv),C)),I_2)
Z = - np.matmul(np.matmul(V ,D ), B_Inv)
Y =  - np.matmul( B_Inv,np.matmul(C ,V ))
X = np.matmul(B_Inv, I_1 - np.matmul(C,Z))
A_Inv = np.zeros((20,20))
A_Inv[:10,:10] = X
A_Inv[10:,10:] = V
A_Inv[10:,:10] = Z
A_Inv[:10,10:] = Y
print("The Matrix:")
print(np.round(A,decimals= 3))
print("The Inverse")
print(np.round(A_Inv,decimals=3))
print ("The Matrix * The Inverse")
print(np.round(np.matmul(A,A_Inv),decimals=3))
```

Figure 2: Code in emacs part 2



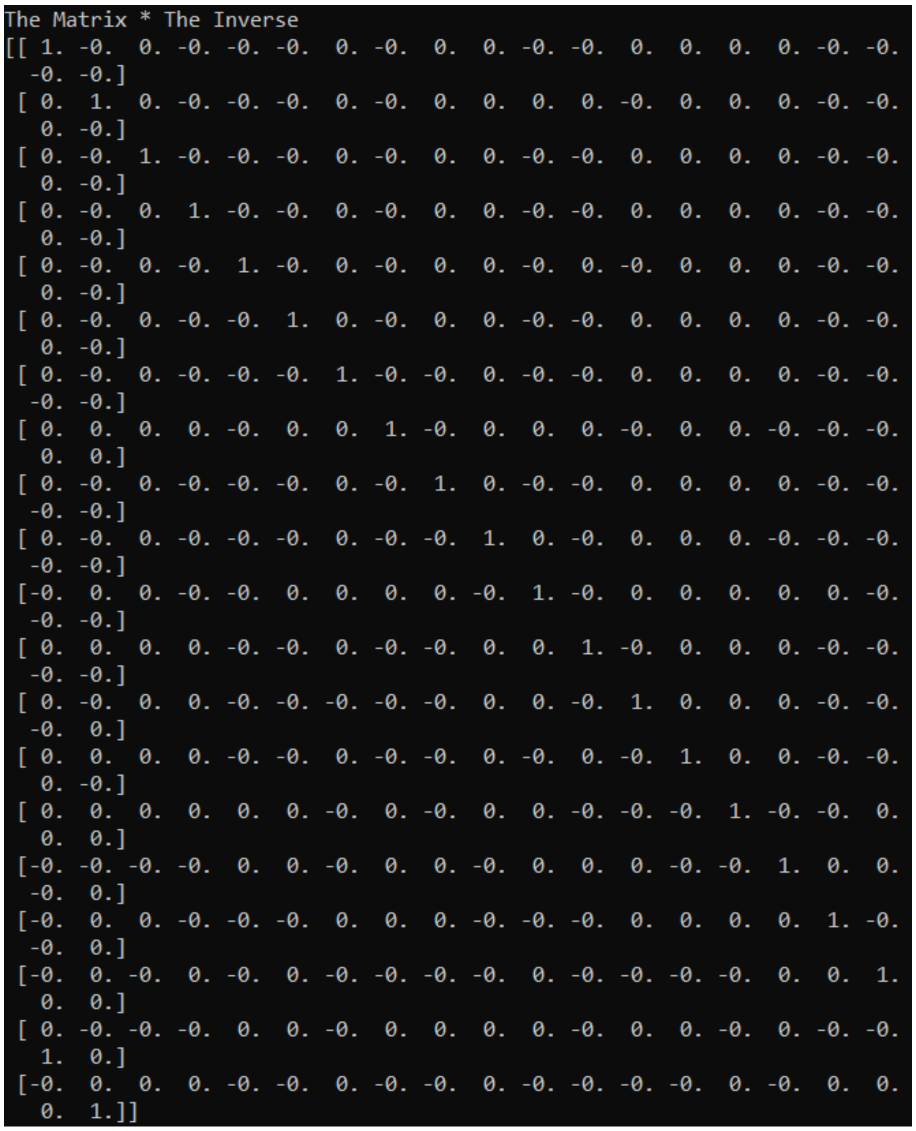
```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab6$ python3 MS18197_6_
The Matrix:
[[0.52  0.314 0.321 0.361 0.879 0.164 0.726 0.972 0.857 0.333 0.032 0.347
  0.13  0.416 0.969 0.753 0.598 0.023 0.266 0.598]
 [0.277 0.128 0.598 0.808 0.699 0.723 0.417 0.866 0.597 0.48  0.297 0.097
  0.822 0.896 0.703 0.007 0.482 0.333 0.211 0.461]
 [0.379 0.321 0.564 0.982 0.444 0.862 0.578 0.987 0.424 0.932 0.813 0.634
  0.064 0.896 0.09  0.071 0.873 0.821 0.515 0.637]
 [0.771 0.838 0.478 0.716 0.754 0.769 0.599 0.724 0.29  0.274 0.676 0.939
  0.077 0.347 0.667 0.133 0.86  0.957 0.701 0.248]
 [0.839 0.546 0.562 0.76  0.004 0.863 0.571 0.156 0.397 0.987 0.534 0.558
  0.062 0.886 0.969 0.976 0.027 0.597 0.674 0.805]
 [0.491 0.729 0.955 0.273 0.961 0.583 0.413 0.649 0.355 0.635 0.412 0.367
  0.321 0.913 0.583 0.567 0.509 0.055 0.633 0.533]
 [0.61  0.607 0.553 0.536 0.731 0.032 0.886 0.521 0.121 0.527 0.446 0.248
  0.624 0.466 0.341 0.645 0.336 0.096 0.711 0.63 ]
 [0.2   0.073 0.183 0.975 0.578 0.557 0.557 0.736 0.357 0.335 0.883 0.291
  0.554 0.013 0.805 0.661 0.319 0.574 0.13  0.148]
 [0.341 0.398 0.492 0.715 0.039 0.817 0.965 0.061 0.884 0.324 0.938 0.538
  0.202 0.889 0.031 0.064 0.821 0.778 0.545 0.244]
 [0.024 0.489 0.694 0.536 0.331 0.05  0.308 0.229 0.053 0.509 0.42  0.889
  0.291 0.991 0.668 0.635 0.027 0.714 0.488 0.102]
 [0.017 0.31  0.67  0.264 0.4   0.213 0.673 0.519 0.07  0.347 0.384 0.092
  0.874 0.539 0.124 0.242 0.445 0.983 0.02  0.659]
 [0.459 0.941 0.184 0.86  0.819 0.216 0.266 0.947 0.468 0.048 0.244 0.739
  0.628 0.689 0.056 0.325 0.003 0.429 0.197 0.082]
 [0.37  0.964 0.413 0.633 0.468 0.102 0.091 0.637 0.057 0.131 0.288 0.751
  0.998 0.762 0.188 0.669 0.908 0.855 0.27  0.728]
 [0.231 0.686 0.045 0.499 0.472 0.993 0.044 0.166 0.787 0.306 0.675 0.003
  0.234 0.205 0.879 0.324 0.575 0.399 0.848 0.574]
 [0.858 0.489 0.376 0.262 0.686 0.616 0.303 0.152 0.579 0.595 0.384 0.019
  0.812 0.146 0.802 0.234 0.07  0.517 0.02  0.696]
 [0.214 0.37  0.998 0.327 0.582 0.727 0.046 0.902 0.02  0.324 0.439 0.03
  0.724 0.116 0.39  0.561 0.612 0.953 0.147 0.696]
 [0.025 0.914 0.705 0.428 0.031 0.288 0.614 0.577 0.41  0.39  0.469 0.943
  0.343 0.278 0.482 0.27  0.196 0.029 0.825 0.237]
 [0.061 0.561 0.497 0.938 0.074 0.139 0.348 0.404 0.124 0.887 0.286 0.153
  0.468 0.935 0.112 0.888 0.628 0.744 0.208 0.82 ]
 [0.044 0.362 0.314 0.413 0.188 0.179 0.166 0.41  0.158 0.027 0.93  0.966
  0.41  0.277 0.367 0.124 0.464 0.944 0.456 0.938]
 [0.787 0.242 0.191 0.776 0.894 0.203 0.995 0.676 0.808 0.981 0.55  0.573
  0.342 0.393 0.308 0.33  0.474 0.921 0.82  0.87 ]]
```

Figure 3: Output - The value of the matrix



```
The Inverse
[[ 1.237 -0.75   2.669 -0.941 -0.449 -3.287  2.924 -1.454  0.51   2.069
  -1.401  0.029  0.363  0.475  1.523  1.284 -0.718 -1.527 -0.653 -1.426]
 [ 0.447 -0.766  0.846  0.188 -0.495 -0.89   0.745 -0.623 -0.159  0.461
   0.31   0.456 -0.451  0.611  0.802 -0.125  0.165  0.33  -0.2   -0.998]
 [-0.064  0.546 -0.995  0.483 -0.446  0.148  0.185 -0.469  0.626  0.095
  -1.141  0.031 -1.021 -0.733  0.243  1.068  0.17   1.035  0.54  -0.028]
 [-1.071  1.494 -2.692  1.541  0.155  1.548 -1.149  0.654  0.097 -1.632
  -0.777  0.309 -1.052 -0.989 -0.787 -0.256  0.173  2.332  0.994  0.876]
 [-1.496  0.849 -3.177  1.417  0.36   4.002 -2.727  1.484 -0.367 -2.24
   0.71   0.155 -0.433 -0.951 -1.26  -1.165  0.013  1.552  0.831  1.815]
 [-1.912  1.029 -3.167  1.207  1.709  4.29  -3.702  1.932 -0.425 -3.336
   1.555 -0.006  0.406 -0.963 -1.967 -1.349  0.606  0.796  0.339  1.882]
 [-0.539  0.364 -1.619  0.944  0.715  1.658 -1.22   0.865 -0.134 -1.612
   1.494 -0.032 -0.275 -0.599 -0.955 -1.097  0.365  0.587  0.166  0.683]
 [ 1.774 -1.148  3.858 -1.611 -0.58  -3.787  2.829 -1.382 -0.175  2.392
  -0.241  0.303  0.287  1.232  1.02   1.204 -0.306 -2.081 -0.931 -1.789]
 [ 0.81  -0.318  0.58  -0.845 -0.617 -1.142  0.395 -0.655  0.729  0.899
  -0.875  0.269 -0.199  0.332  0.729  0.801  0.073 -0.177 -0.144 -0.121]
 [-0.065 -0.582  1.26  -0.508 -0.325 -0.121 -0.182  0.192 -0.452  0.557
   0.111 -0.504  0.482  0.16   0.748 -0.331  0.415 -0.246 -0.763  0.175]
 [ 1.396 -1.614  3.728 -1.697 -1.19  -3.199  3.259 -0.945  0.456  2.687
  -0.847  0.085 -0.121  1.165  1.83   0.874 -0.708 -1.592 -0.235 -2.178]
 [-1.453  0.843 -2.803  0.996  1.005  3.685 -3.334  1.679 -0.295 -2.407
   0.547 -0.265  0.65  -1.477 -1.292 -1.297  0.821  0.844  0.733  1.925]
 [-0.422  0.27   0.054 -0.641  0.135 -0.014 -0.091  0.41  -0.04  -0.012
   0.222 -0.331  1.134  0.029  0.036 -0.128  0.341 -0.821 -0.487  0.356]
 [ 0.76  -0.326  1.791 -0.874 -0.09  -1.437  1.382 -0.844  0.124  1.432
   0.159  0.17   0.277  0.698  0.362  0.028 -0.684 -1.07  -0.353 -1.07 ]
 [ 0.902 -0.035  1.141 -0.025 -0.382 -1.753  1.212 -0.587 -0.242  1.352
  -0.145 -0.328 -0.187  0.603  0.794  0.094 -0.162 -0.423 -0.151 -1.074]
 [-0.178 -0.389 -0.794 -0.327  0.61   1.103 -0.835  0.842  0.127 -0.56
   0.17   0.016  0.577 -0.199 -0.732  0.055 -0.024 -0.051 -0.203  0.588]
 [ 0.227 -0.032  0.316  0.172 -0.453 -0.057  0.04   0.094  0.192  0.064
  -0.348 -0.735  0.724 -0.01   0.143 -0.087  0.018  0.048 -0.307 -0.1  ]
 [ 0.476 -0.473  0.812 -0.132 -0.222 -1.565  0.641 -0.714  0.082  1.198
   0.204  0.177 -0.149  0.565  0.312  0.678 -0.392 -0.462 -0.371 -0.243]
 [ 0.057  0.083  0.663 -0.559  0.008 -1.277  1.266 -0.713  0.015  0.971
  -0.393 -0.132  0.338  0.848 -0.429  0.736 -0.066 -1.032 -0.463  0.167]
 [-0.763  0.859 -2.225  0.879  0.761  2.275 -1.62   0.482 -0.343 -2.197
   0.543  0.148 -0.589 -0.733 -0.907 -0.749  0.157  1.448  1.38   0.922]]
```

Figure 4: Output - The value of the matrix's inverse through partition method



```
The Matrix * The Inverse
[[ 1. -0.  0. -0. -0. -0.  0. -0.  0.  0. -0. -0.  0.  0.  0.  0. -0. -0.
  -0. -0.]
 [ 0.  1.  0. -0. -0. -0.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0. -0. -0.
   0. -0.]
 [ 0. -0.  1. -0. -0. -0.  0. -0.  0.  0. -0. -0.  0.  0.  0.  0. -0. -0.
   0. -0.]
 [ 0. -0.  0.  1. -0. -0.  0. -0.  0.  0. -0. -0.  0.  0.  0.  0. -0. -0.
   0. -0.]
 [ 0. -0.  0. -0.  1. -0.  0. -0.  0.  0. -0.  0. -0.  0.  0.  0. -0. -0.
   0. -0.]
 [ 0. -0.  0. -0. -0.  1.  0. -0.  0.  0. -0. -0.  0.  0.  0.  0. -0. -0.
   0. -0.]
 [ 0. -0.  0. -0. -0. -0.  1. -0. -0.  0. -0. -0.  0.  0.  0.  0. -0. -0.
  -0. -0.]
 [ 0.  0.  0.  0. -0.  0.  0.  1. -0.  0.  0.  0. -0.  0.  0. -0. -0. -0.
   0.  0.]
 [ 0. -0.  0. -0. -0. -0.  0. -0.  1.  0. -0. -0.  0.  0.  0.  0. -0. -0.
  -0. -0.]
 [ 0. -0.  0. -0. -0. -0.  0. -0. -0.  1.  0. -0.  0.  0.  0. -0. -0. -0.
  -0. -0.]
 [-0.  0.  0. -0. -0.  0.  0.  0.  0. -0.  1. -0.  0.  0.  0.  0.  0. -0.
  -0. -0.]
 [ 0.  0.  0.  0. -0. -0.  0. -0. -0.  0.  0.  1. -0.  0.  0.  0. -0. -0.
  -0. -0.]
 [ 0. -0.  0.  0. -0. -0. -0. -0. -0.  0.  0. -0.  1.  0.  0.  0. -0. -0.
  -0.  0.]
 [ 0.  0.  0.  0. -0. -0.  0. -0. -0.  0. -0.  0. -0.  1.  0.  0. -0. -0.
   0. -0.]
 [ 0.  0.  0.  0.  0.  0. -0.  0. -0.  0.  0. -0. -0. -0.  1. -0. -0.  0.
   0.  0.]
 [-0. -0. -0. -0.  0.  0. -0.  0.  0. -0.  0.  0.  0. -0. -0.  1.  0.  0.
  -0.  0.]
 [-0.  0.  0. -0. -0. -0.  0.  0.  0. -0. -0. -0.  0.  0.  0.  0.  1. -0.
  -0.  0.]
 [-0.  0. -0.  0. -0.  0. -0. -0. -0. -0.  0. -0. -0. -0. -0.  0.  0.  1.
   0.  0.]
 [ 0. -0. -0. -0.  0.  0. -0.  0.  0.  0.  0. -0.  0.  0. -0.  0. -0. -0.
   1.  0.]
 [-0.  0.  0.  0. -0. -0.  0. -0. -0.  0. -0. -0. -0. -0.  0. -0.  0.  0.
   0.  1.]]
```

Figure 5: Output - The value of product

2 Solve the following using Jacobi & Gauss-Seidel method

$$10x_1 - 2x_2 - x_3 - x_4 = 3$$

$$-2x_1 + 10x_2 - x_3 - x_4 = 15$$

$$-x_1 - x_2 + 10x_3 - 2x_4 = 27$$

$$-x_1 - x_2 - 2x_3 + 10x_4 = -9$$

2.1 Algorithm & Discussion

For Jacobi method, we use the following relation:

$$X_i^{k+1} = \frac{1}{A_{ii}}(B_i - \sum_{k=1}^{i-1} A_{ik}X_k) - \sum_{k=i+1}^{n} A_{ik}X_k)$$

$$i = 1, 2, 3....n$$

Here k represent the iteration number that tells us which iteration is going on.

For Gauss-Seidel method, we first determine X_1^{k+1} from X_1^k like Jacobi method. Now we determine X_2^{k+1} from X_1^{k+1} instead of calculation X_2^{k+1} from X_2^k in Jacobi method. We use the following relation to determine successive values in X^{k+1}:

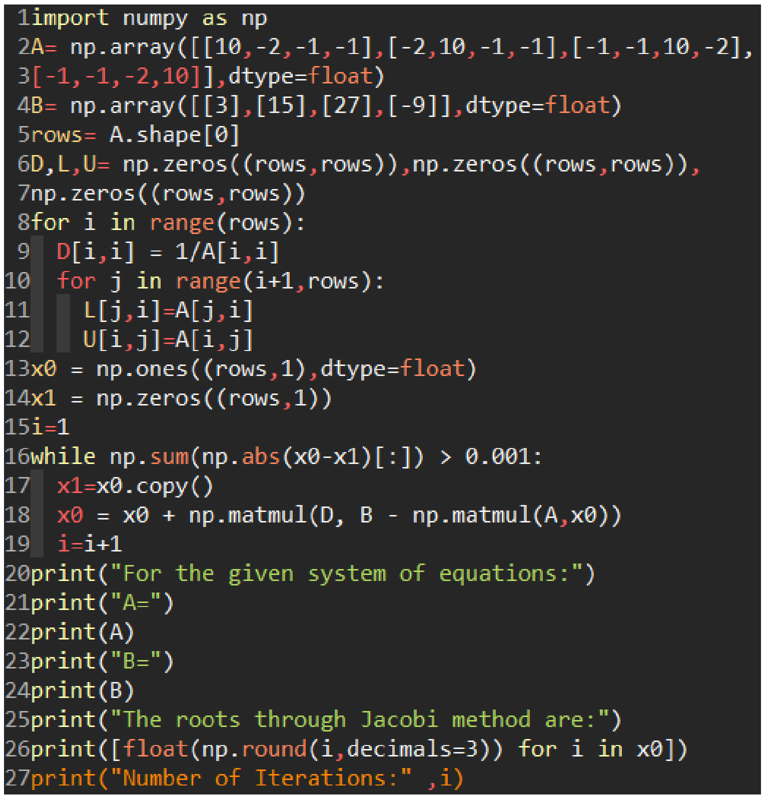
$$X_i^{k+1} = \frac{1}{A_{ii}}(B_i - \sum_{m=1}^{i-1} A_{im}X_m^{k+1} - \sum_{m=i+1}^{n} A_{im}X_m^k)$$

$$i = 1, 2, 3....n$$

Here k represent the iteration number that tells us which iteration is going on. When we compare the two iterations for the given equations, Jacobi method took 10 iterations to compute the solutions while Gauss-Seidel computed the solutions in 6 iterations.

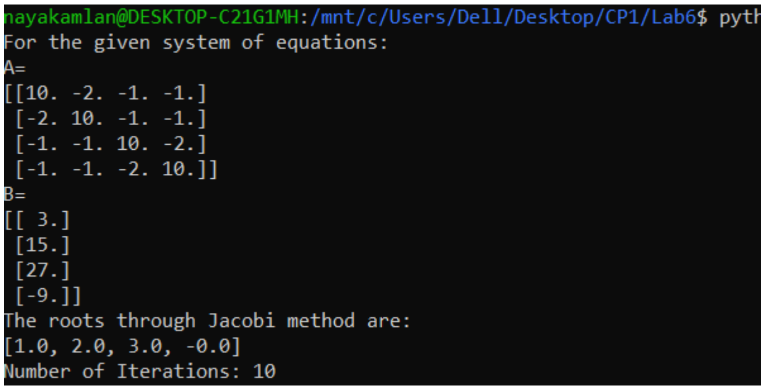
The code written is general and one can change the input matrices in the code to get the required answers.

2.2 Code and Output



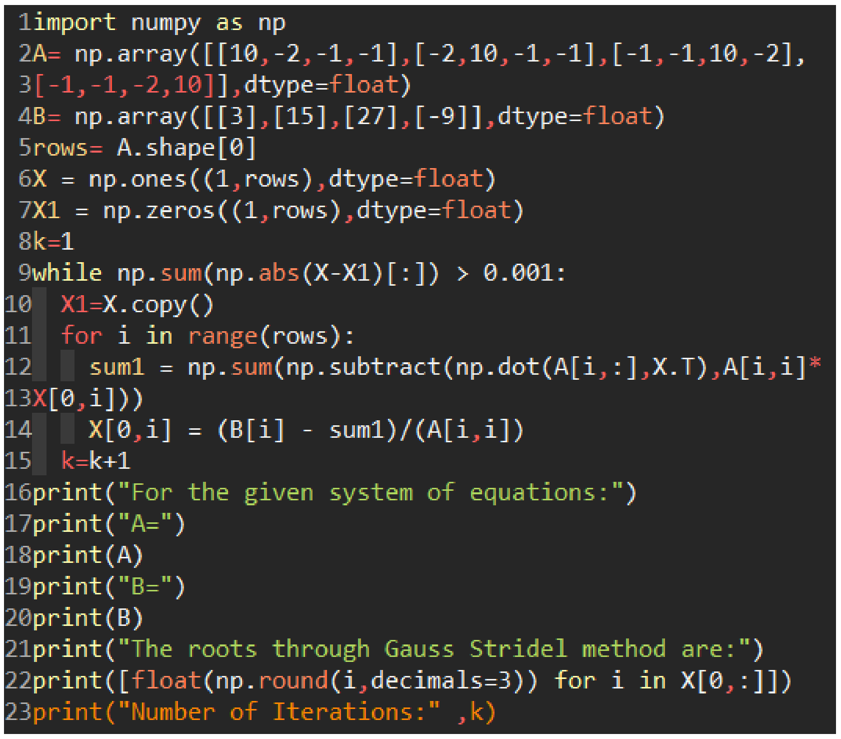
```python
import numpy as np
A= np.array([[10,-2,-1,-1],[-2,10,-1,-1],[-1,-1,10,-2],
[-1,-1,-2,10]],dtype=float)
B= np.array([[3],[15],[27],[-9]],dtype=float)
rows= A.shape[0]
D,L,U= np.zeros((rows,rows)),np.zeros((rows,rows)),
np.zeros((rows,rows))
for i in range(rows):
  D[i,i] = 1/A[i,i]
  for j in range(i+1,rows):
    L[j,i]=A[j,i]
    U[i,j]=A[i,j]
x0 = np.ones((rows,1),dtype=float)
x1 = np.zeros((rows,1))
i=1
while np.sum(np.abs(x0-x1)[:]) > 0.001:
  x1=x0.copy()
  x0 = x0 + np.matmul(D, B - np.matmul(A,x0))
  i=i+1
print("For the given system of equations:")
print("A=")
print(A)
print("B=")
print(B)
print("The roots through Jacobi method are:")
print([float(np.round(i,decimals=3)) for i in x0])
print("Number of Iterations:" ,i)
```

Figure 6: Code of Jacobi method



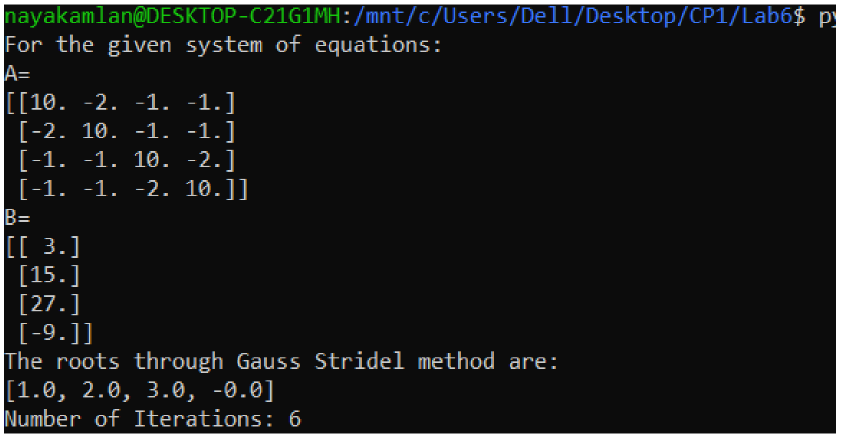
```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab6$ pyth
For the given system of equations:
A=
[[10. -2. -1. -1.]
 [-2. 10. -1. -1.]
 [-1. -1. 10. -2.]
 [-1. -1. -2. 10.]]
B=
[[ 3.]
 [15.]
 [27.]
 [-9.]]
The roots through Jacobi method are:
[1.0, 2.0, 3.0, -0.0]
Number of Iterations: 10
```

Figure 7: Output of Jacobi method



```python
import numpy as np
A= np.array([[10,-2,-1,-1],[-2,10,-1,-1],[-1,-1,10,-2],
[-1,-1,-2,10]],dtype=float)
B= np.array([[3],[15],[27],[-9]],dtype=float)
rows= A.shape[0]
X = np.ones((1,rows),dtype=float)
X1 = np.zeros((1,rows),dtype=float)
k=1
while np.sum(np.abs(X-X1)[:]) > 0.001:
  X1=X.copy()
  for i in range(rows):
    sum1 = np.sum(np.subtract(np.dot(A[i,:],X.T),A[i,i]*
X[0,i]))
    X[0,i] = (B[i] - sum1)/(A[i,i])
  k=k+1
print("For the given system of equations:")
print("A=")
print(A)
print("B=")
print(B)
print("The roots through Gauss Stridel method are:")
print([float(np.round(i,decimals=3)) for i in X[0,:]])
print("Number of Iterations:" ,k)
```

Figure 8: Code of Gauss-Seidel method



```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab6$ py
For the given system of equations:
A=
[[10. -2. -1. -1.]
 [-2. 10. -1. -1.]
 [-1. -1. 10. -2.]
 [-1. -1. -2. 10.]]
B=
[[ 3.]
 [15.]
 [27.]
 [-9.]]
The roots through Gauss Stridel method are:
[1.0, 2.0, 3.0, -0.0]
Number of Iterations: 6
```

Figure 9: Output of Gauss-Seidel method

3 Solve the following equations using relaxation method.

$$x_1 + 3x_2 + 2x_3 - x_4 = 9$$
$$3x_1 - 3x_2 + 2x_3 + 4x_4 = 19$$
$$-4x_1 + 2x_2 + 5x_3 + x_4 = 27$$
$$-x_1 + 2x_2 - 3x_3 + 5x_4 = 14$$

3.1 Algorithm & Discussion

As derived in class, we use the following relation for our iteration:

$$X_{k+1} = X_k + (D + wL)^{-1} w[B - AX_k]$$

where w is the relaxation parameter. D is the matrix with all the diagonal entries of A, L is the matrix with all the entries of A below the diagonal and U is the matrix with all the entries of A above the diagonal.

Varying the relaxation parameter for the given system showed **divergence and numerical instability of the equations when w $>$ 0.71**. As for the values less than that, a plot of the time taken to converge to a solution:

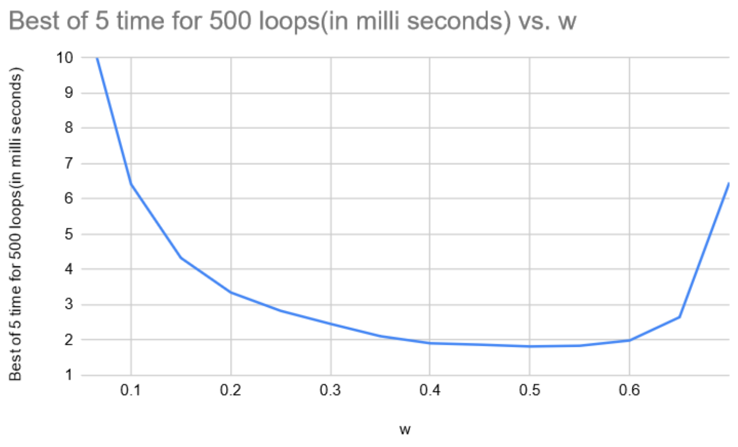


Figure 10: Time vs w graph

As for the values less 0.71, a plot of the iterations taken to converge for different w:

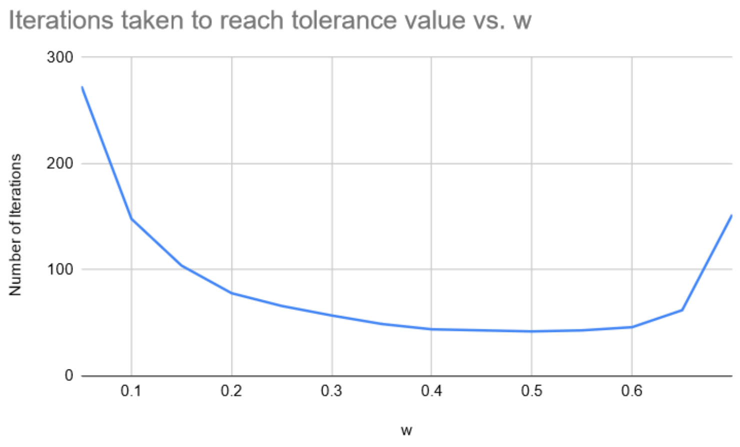


Figure 11: Iterations vs w graph

The code from Lab7 Problem 1 was used to plot the Gerschgorin circles. The convergence becomes faster when we approach a certain value of w. The Gerschgorin circle:

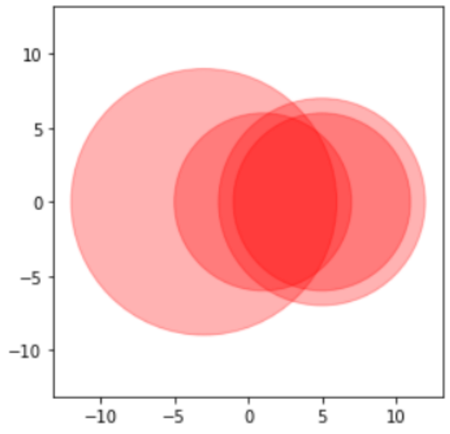


Figure 12: Gerschgorin circles for rows

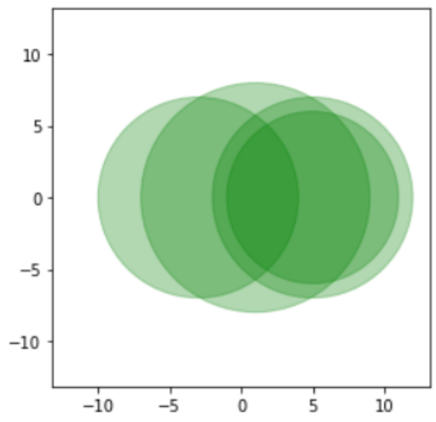


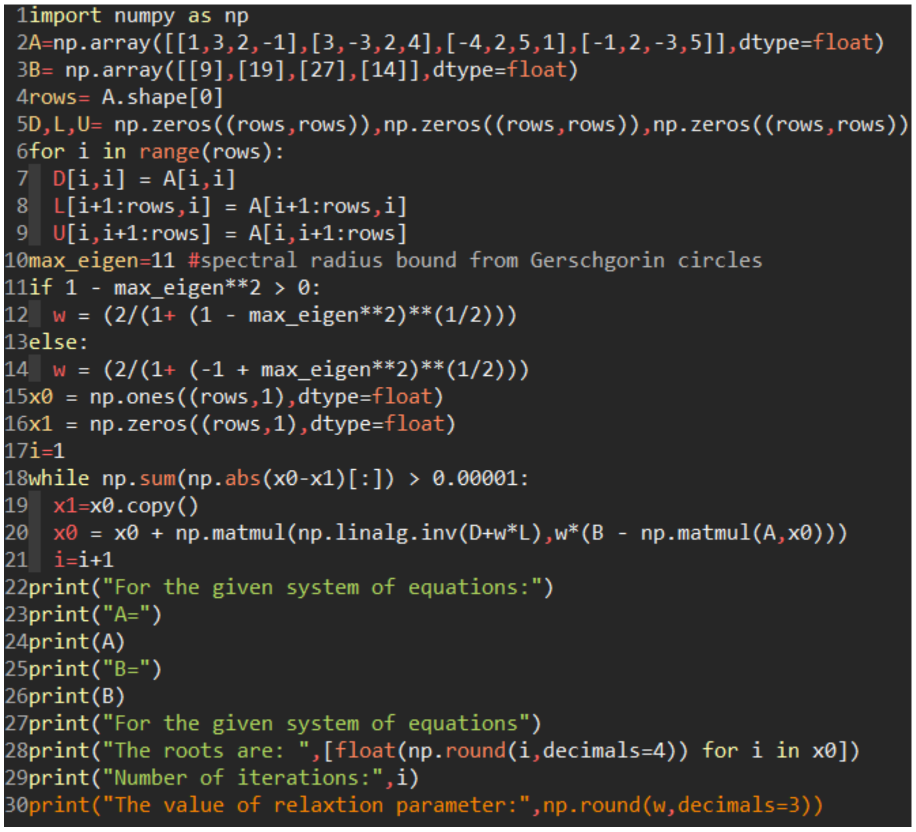
Figure 13: Gerschgorin circles for columns

Using the intersection of the union of the row and column plots, the bounds for eigenvalues were approximately (-10,11). Hence maximum possible eigenvalue could be 11. This is the spectral radius. The formula for optimum relaxation parameter derived in class was:

$$w_{opt} = \frac{2}{1+\sqrt{1-\mu^2}}$$

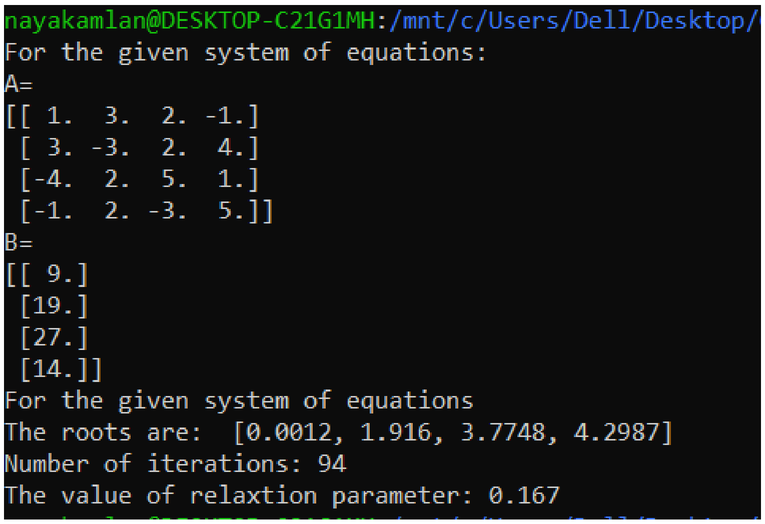
$w_{opt} = 0.154$ was a rough calculation from Gerschgorin circles. The plots of iterations vs w shows w_{opt} would be in range of $(0.5, 0.55)$

3.2 Code and Output



```
import numpy as np
A=np.array([[1,3,2,-1],[3,-3,2,4],[-4,2,5,1],[-1,2,-3,5]],dtype=float)
B= np.array([[9],[19],[27],[14]],dtype=float)
rows= A.shape[0]
D,L,U= np.zeros((rows,rows)),np.zeros((rows,rows)),np.zeros((rows,rows))
for i in range(rows):
    D[i,i] = A[i,i]
    L[i+1:rows,i] = A[i+1:rows,i]
    U[i,i+1:rows] = A[i,i+1:rows]
max_eigen=11 #spectral radius bound from Gerschgorin circles
if 1 - max_eigen**2 > 0:
    w = (2/(1+ (1 - max_eigen**2)**(1/2)))
else:
    w = (2/(1+ (-1 + max_eigen**2)**(1/2)))
x0 = np.ones((rows,1),dtype=float)
x1 = np.zeros((rows,1),dtype=float)
i=1
while np.sum(np.abs(x0-x1)[:]) > 0.00001:
    x1=x0.copy()
    x0 = x0 + np.matmul(np.linalg.inv(D+w*L),w*(B - np.matmul(A,x0)))
    i=i+1
print("For the given system of equations:")
print("A=")
print(A)
print("B=")
print(B)
print("For the given system of equations")
print("The roots are: ",[float(np.round(i,decimals=4)) for i in x0])
print("Number of iterations:",i)
print("The value of relaxtion parameter:",np.round(w,decimals=3))
```

Figure 14: Code in emacs



```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/
For the given system of equations:
A=
[[ 1.  3.  2. -1.]
 [ 3. -3.  2.  4.]
 [-4.  2.  5.  1.]
 [-1.  2. -3.  5.]]
B=
[[ 9.]
 [19.]
 [27.]
 [14.]]
For the given system of equations
The roots are:  [0.0012, 1.916, 3.7748, 4.2987]
Number of iterations: 94
The value of relaxtion parameter: 0.167
```

Figure 15: Output

4 Get the eigenvalues for following matrix using Rutishauser method

$$\begin{bmatrix} 4 & 3 & 2 & 19 \\ 6 & 7 & 8 & 49 \\ 10 & 11 & 12 & 13 \\ 15 & 16 & 17 & 18 \end{bmatrix}$$

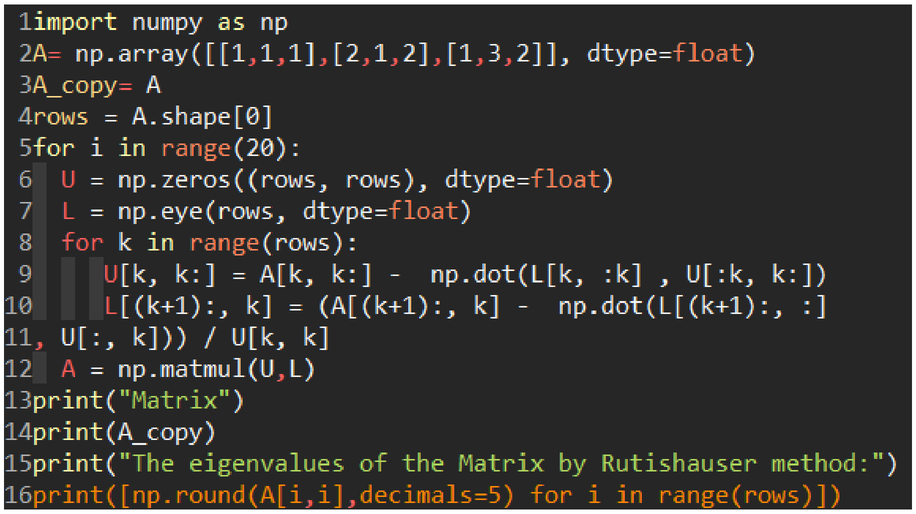
4.1 Algorithm & Discussion

As derived in class, we are going to use the following relation:

$$A_{k+1} = U_k L_k$$
$$k = 1, 2, 3...$$

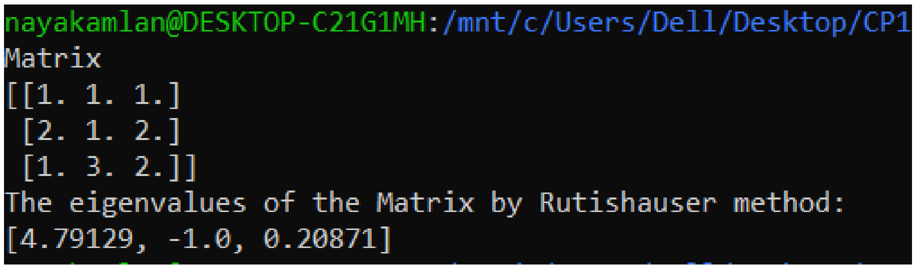
Here U_k & L_k represent the LU decomposition of A_k. We first find A_1 from LU decomposition of the given matrix. We are then going to find successive A_k till the off diagonal terms of A_k are zero.

4.2 Code and Output



```
import numpy as np
A= np.array([[1,1,1],[2,1,2],[1,3,2]], dtype=float)
A_copy= A
rows = A.shape[0]
for i in range(20):
    U = np.zeros((rows, rows), dtype=float)
    L = np.eye(rows, dtype=float)
    for k in range(rows):
        U[k, k:] = A[k, k:] -  np.dot(L[k, :k] , U[:k, k:])
        L[(k+1):, k] = (A[(k+1):, k] -  np.dot(L[(k+1):, :]
, U[:, k])) / U[k, k]
    A = np.matmul(U,L)
print("Matrix")
print(A_copy)
print("The eigenvalues of the Matrix by Rutishauser method:")
print([np.round(A[i,i],decimals=5) for i in range(rows)])
```

Figure 16: Code for Rutishauser method in emacs



```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1
Matrix
[[1. 1. 1.]
 [2. 1. 2.]
 [1. 3. 2.]]
The eigenvalues of the Matrix by Rutishauser method:
[4.79129, -1.0, 0.20871]
```

Figure 17: Output