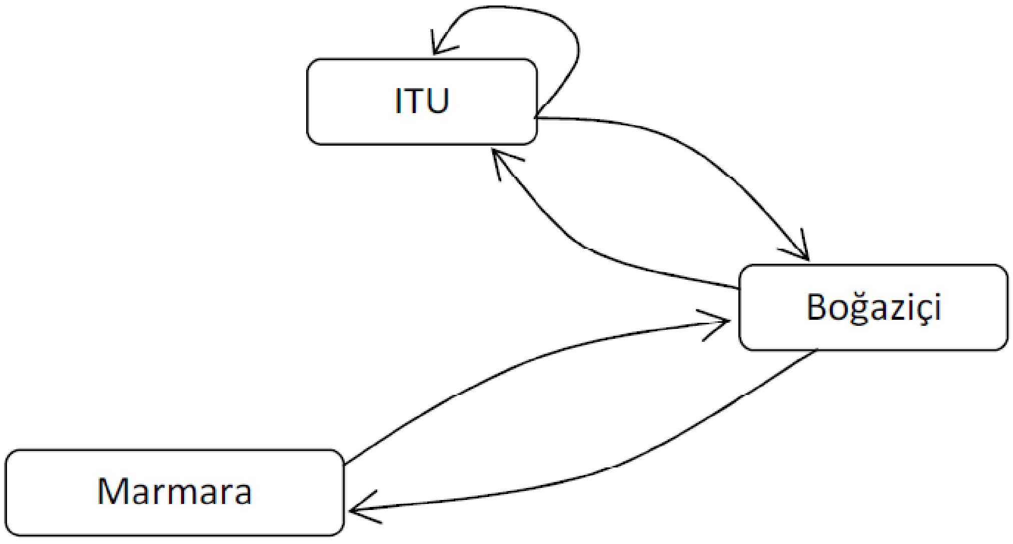
Assignment Report

Eren Ulaş 150114822
Berk Karabacak 150114823

Problem 1

We implement a basic version of PageRank algorithm. First, we implement the graph below by using the sparse matrix representation to show the connections between the websites.

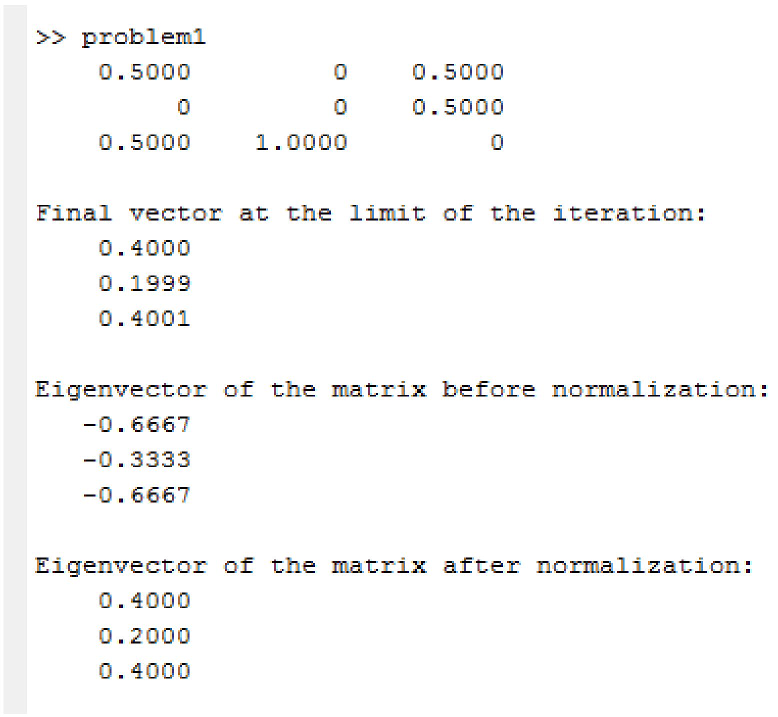


Then we find the importance matrix by replacing the entries in the connectivity matrix with division of one with the sum of the corresponding column in the connectivity matrix.

```
0.5000         0    0.5000
     0         0    0.5000
0.5000    1.0000         0
```

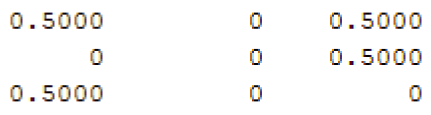
For the first question of the first problem, the importance matrix is shown at left .

After that, we set the initial vector, and define the tolerance value as 1/1000. We compute the next vector according to the equation ‘A*x = x’ . Then, the loop starts with the condition of the difference between the previous vector and the next vector should be bigger than the tolerance value. At each step, the next vector we have found becomes the previous vector and computes the next vector. After the condition of the loop is no longer satisfied, it normalizes the final vector, and displays it. Finally, for the comparison, we have used the ‘eigs()’ function which is a built-in matlab function to find the eigenvectors of the importance matrix. After normalizing the eigenvector we have seen that the values are equal. For the first question of the first problem the results are shown below.



```
>> problem1
    0.5000         0    0.5000
         0         0    0.5000
    0.5000    1.0000         0

Final vector at the limit of the iteration:
    0.4000
    0.1999
    0.4001

Eigenvector of the matrix before normalization:
   -0.6667
   -0.3333
   -0.6667

Eigenvector of the matrix after normalization:
    0.4000
    0.2000
    0.4000
```

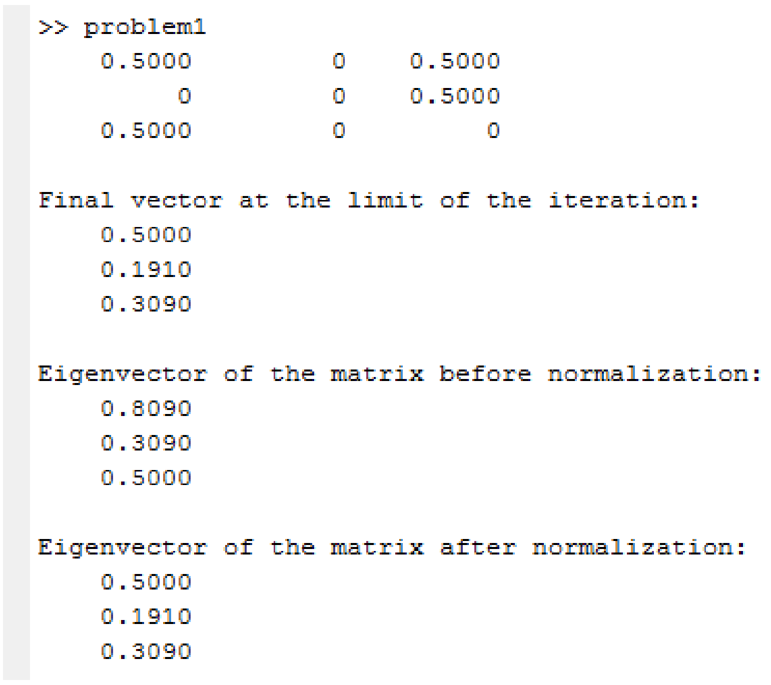
For the second question of the first problem, we remove the link from Marmara to Boğaziçi, so the importance matrix is changed.



```
    0.5000         0    0.5000
         0         0    0.5000
    0.5000         0         0
```

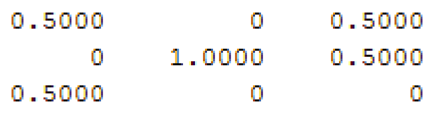
The new importance matrix is shown at left.

After changing the importance matrix, the whole process is repeated, and the results are shown below.



```
>> problem1
    0.5000         0    0.5000
         0         0    0.5000
    0.5000         0         0

Final vector at the limit of the iteration:
    0.5000
    0.1910
    0.3090

Eigenvector of the matrix before normalization:
    0.8090
    0.3090
    0.5000

Eigenvector of the matrix after normalization:
    0.5000
    0.1910
    0.3090
```

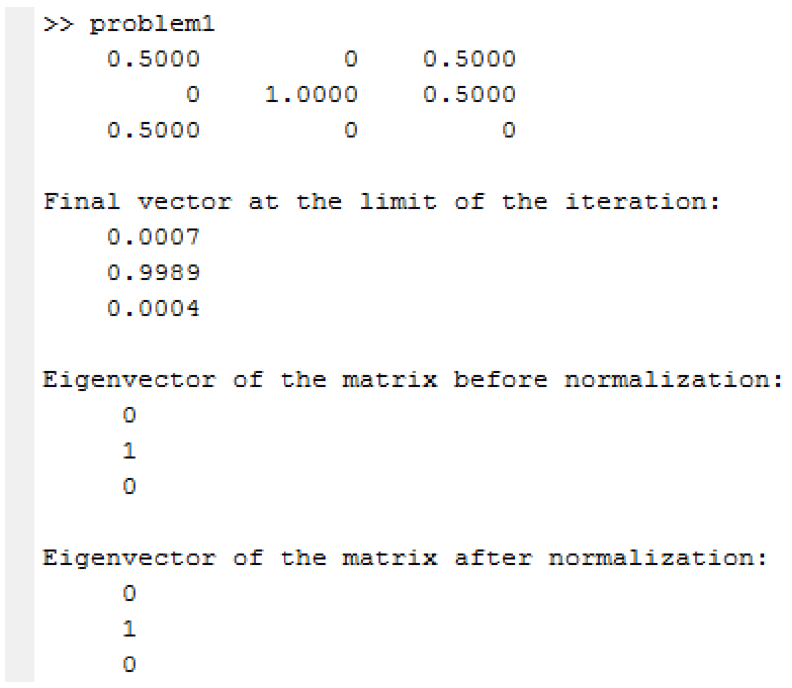
As an answer to the third question of the first problem, we add a link from Marmara to itself to make Marmara the most important site, so importance matrix is changed again.



```
0.5000         0    0.5000
     0    1.0000    0.5000
0.5000         0         0
```

The new importance matrix is shown at left.

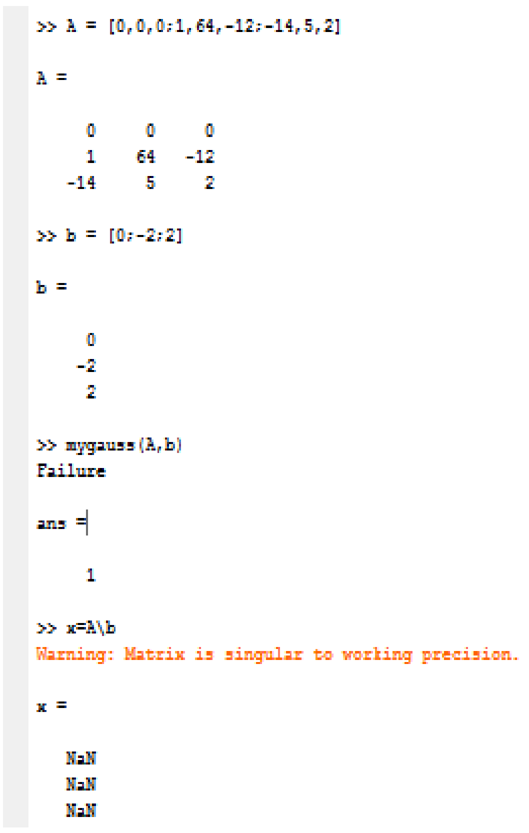
After changing the importance matrix, the whole process is repeated again, and the results are shown below.



```
>> problem1
    0.5000         0    0.5000
         0    1.0000    0.5000
    0.5000         0         0

Final vector at the limit of the iteration:
    0.0007
    0.9989
    0.0004

Eigenvector of the matrix before normalization:
     0
     1
     0

Eigenvector of the matrix after normalization:
     0
     1
     0
```

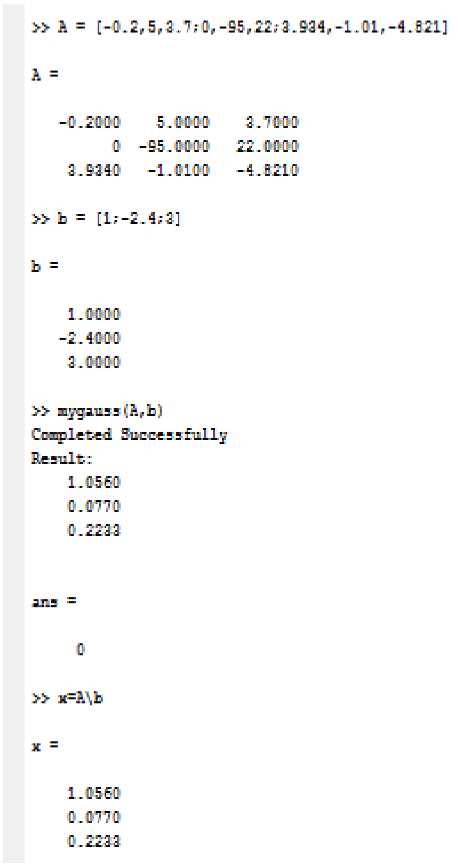
Problem 2

We implement a function called 'mygauss' which finds the solution x of the system A*x = b by using Gaussian Elimination Method with Scaled Column Pivoting. Function takes two inputs as 'A' and 'b'. 'A' is an n-by-n matrix and 'b' is a vector of size n. The function gives two outputs 'singular' and 'x' . 'singular' indicates whether A is nonsingular or not and 'x' is the solution of the system.
Firstly, function creates an augmented matrix out of A and b. Then, it finds the scale factor for each row by comparing each entry at each row, and keeps the scale factors and their row numbers in matrices. After that, it finds the greatest ratio and changes the rows according to that ratio, and repeats this process for all rows. Finally, the function does back substitution and at this part it starts from the bottom and goes to the top of the matrix. The results of four run are shown below.



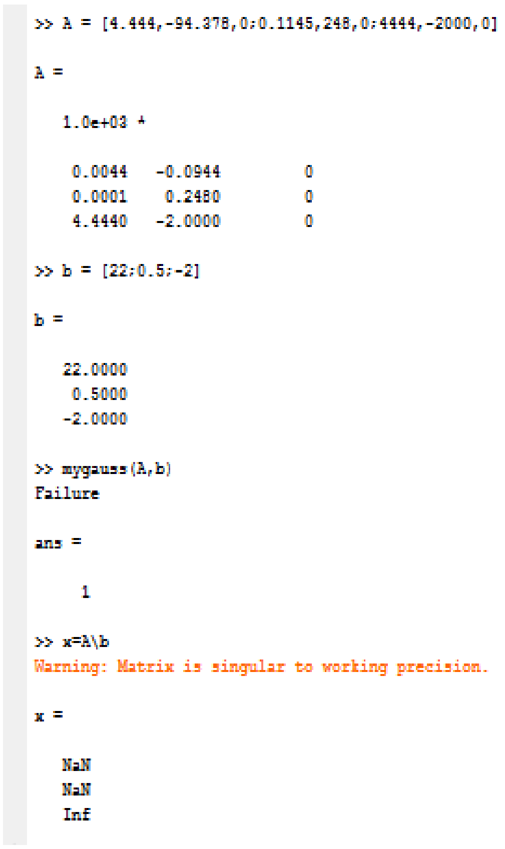
```
>> A = [0,0,0;1,64,-12;-14,5,2]

A =

     0     0     0
     1    64   -12
   -14     5     2

>> b = [0;-2;2]

b =

     0
    -2
     2

>> mygauss(A,b)
Failure

ans =

     1

>> x=A\b
Warning: Matrix is singular to working precision.

x =

   NaN
   NaN
   NaN
```

First Run



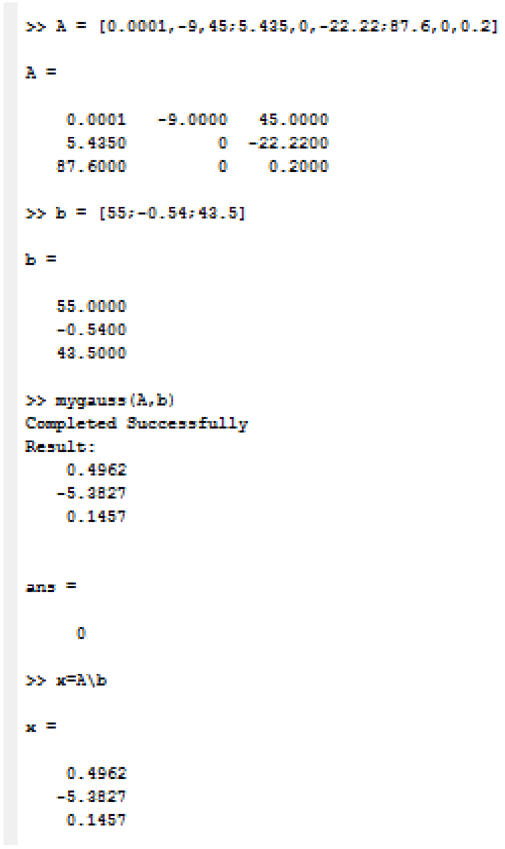
```
>> A = [-0.2,5,3.7;0,-95,22;3.934,-1.01,-4.821]

A =

   -0.2000    5.0000    3.7000
         0  -95.0000   22.0000
    3.9340   -1.0100   -4.8210

>> b = [1;-2.4;3]

b =

    1.0000
   -2.4000
    3.0000

>> mygauss(A,b)
Completed Successfully
Result:
    1.0560
    0.0770
    0.2233


ans =

     0

>> x=A\b

x =

    1.0560
    0.0770
    0.2233
```

Second Run



```
>> A = [4.444,-94.378,0;0.1145,248,0;4444,-2000,0]

A =

   1.0e+03 *

    0.0044   -0.0944         0
    0.0001    0.2480         0
    4.4440   -2.0000         0

>> b = [22;0.5;-2]

b =

   22.0000
    0.5000
   -2.0000

>> mygauss(A,b)
Failure

ans =

     1

>> x=A\b
Warning: Matrix is singular to working precision.

x =

   NaN
   NaN
   Inf
```

Third Run

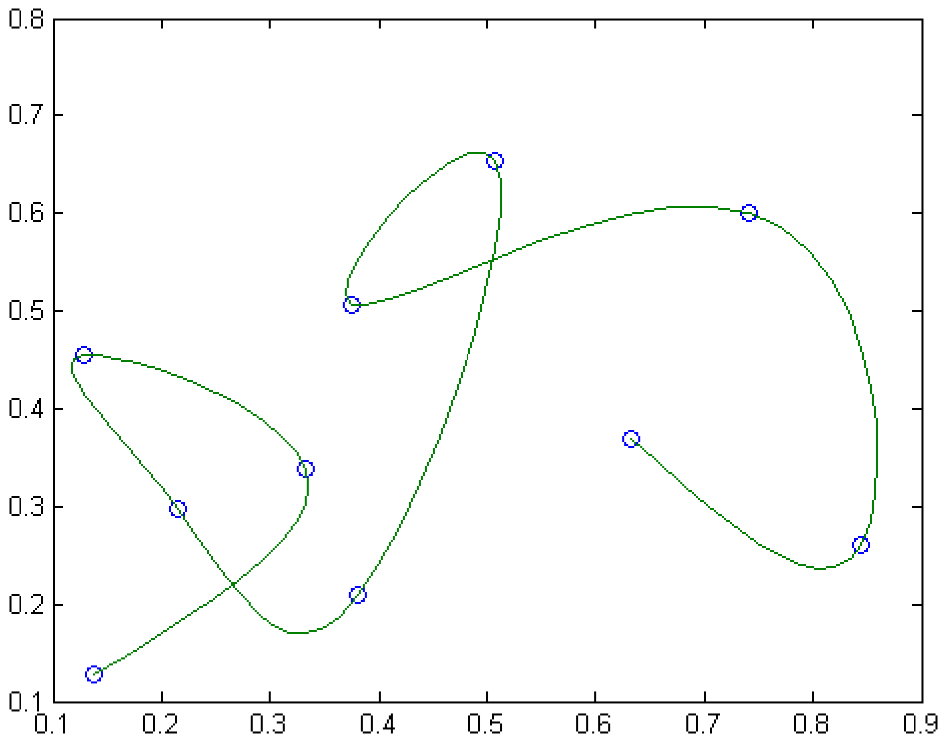


```
>> A = [0.0001,-9,45;5.435,0,-22.22;87.6,0,0.2]

A =

    0.0001   -9.0000   45.0000
    5.4350         0  -22.2200
   87.6000         0    0.2000

>> b = [55;-0.54;43.5]

b =

   55.0000
   -0.5400
   43.5000

>> mygauss(A,b)
Completed Successfully
Result:
    0.4962
   -5.3827
    0.1457


ans =

     0

>> x=A\b

x =

    0.4962
   -5.3827
    0.1457
```

Fourth Run

Problem 3

In this problem we implemented a curve drawing program.To draw smooth curves we used natural cubic splines.
We start program by taking random points from user with MATLAB® built-in ginput() function.Then we assign their coordinates to x and y matrices.Next we create an array called t which acts like time axis in our function.After that we pass these x and y values to algorithm for finding coefficients.In the end we get 4 coefficients(a2x,b2x...,a2y,b2y...) for each x and y matrices.With these coefficients,we create our cubic polynomial.
At last stage of program,we plot the curve.We evaluate our polynomials at different time values between 1 and number of points.We used "10 x number of points" time value in our program.By increasing "10" one can easily get a smoother curve.Finally,we plot our curve with the points that we get from evaluating our polynomials.



An example run of the program with 10 data points is shown above