Matthew Horowitz

A20377155

CS 430- Section 5 (Online)

Girish Rajani-Bathija

A20503736

CS 430- Section 4 (Live)

Bhavesh Rajesh Talreja

A20516822

CS 430-Section 4 (Live)

Muhammad Sheheryar Qureshi

A20517229

CS 430-Section 4 (Live)

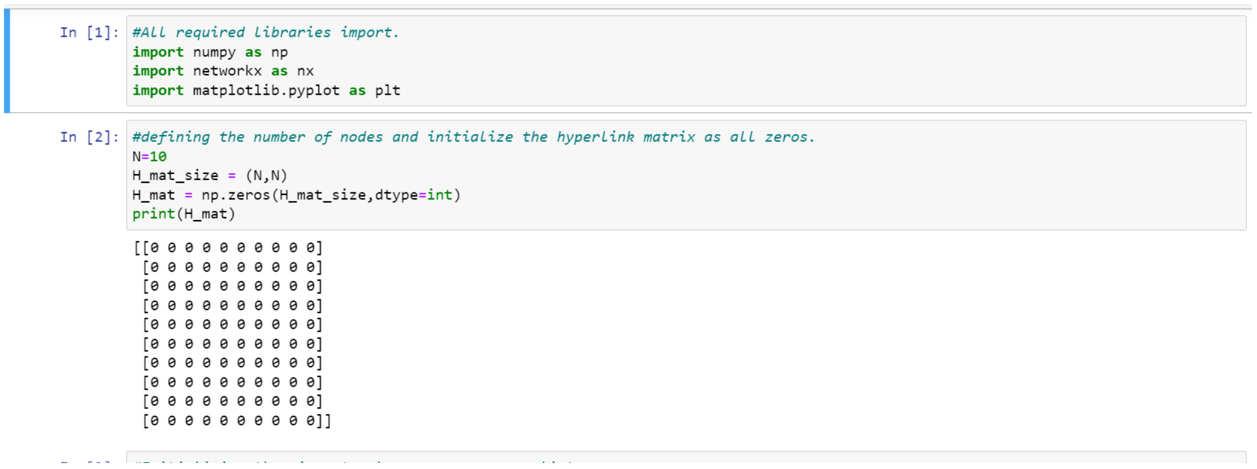
Homework Four

<u>Note: Submitted with this report are all Python Files. Comments can be found in these files that show step by step processes. Professor Chlebus has said that using NumPy is fine for this any question in the assignment).</u>

(a) Find the row-normalized hyperlink matrix H of this graph. Are there any dangling nodes in this hyperlink graph?

Creating the row-normalized hyperlink matrix H:

To generate the row-normalized hyperlink matrix H of this 10 node graph of webpages, first, a 10 x 10 matrix was created, whose values were initialized to 0:



```
In [1]: #All required libraries import.
        import numpy as np
        import networkx as nx
        import matplotlib.pyplot as plt
```

```
In [2]: #defining the number of nodes and initialize the hyperlink matrix as all zeros.
        N=10
        H_mat_size = (N,N)
        H_mat = np.zeros(H_mat_size,dtype=int)
        print(H_mat)

        [[0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]
         [0 0 0 0 0 0 0 0 0 0]]
```

Next, the matrix was populated, this was done by reading in the set of prime numbers, or numbers of prime power supplied in the report brief. The first digit corresponds to the out-bound connection of a webpage node to another node (the rows), while the second digit corresponds to the inbound connection (the columns). For three digit numbers, the first two digits correspond to the outbound connection, while the final digit corresponds to the inbound connection. Should a connection exist, a 1 was recorded, else 0.

```
#Initializing the given topology sequence as a list.
topo_seq = [13,16,17,19,23,25,27,29,31,32,37,41,43,47,49,53,59,61,64,67,71,73,79,81,83,89,97,101,103,107,109]

#In order to create the final hyperlink matrix, we use the topology sequence most and least significance digits
#to get to the appropriate connection.
for m in topo_seq:
    i=int(m/10) #most significant digit.
    j=int(m%10) #least significant digit.
    H_mat[i-1,j-1]=(1) #Link from most significant digit to least significant digit.

#Printing the hyperlink matrix.
print(H_mat)
```

```
[[0 0 1 0 0 1 1 0 1 0]
 [0 0 1 0 1 0 1 0 1 0]
 [1 1 0 0 0 0 1 0 0 0]
 [1 0 1 0 0 0 1 0 1 0]
 [0 0 1 0 0 0 0 0 1 0]
 [1 0 0 1 0 0 1 0 0 0]
 [1 0 1 0 0 0 0 0 1 0]
 [1 0 1 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1 0 0 0]
 [1 0 1 0 0 0 1 0 1 0]]
```

Next, the rows were normalized. This was done by summing each of the rows and dividing the sum into each one of the 1's present in a row. As such, after this process, each row sums to 1.[1]

```
#Part Q1(a).
# Find the row-normalized hyperlink matrix H of this graph. Are there any dangling nodes in this hyperlink graph?

#Row normalizing the hyperlink matrix we created in previous steps.
H_mat_row_sum = H_mat.sum(axis=1) #sum of all the elements of a particular row in the hyperlink matrix.

#Dividing each element of the row by its respective sum obtained in the previous step.
row_norm_H_mat = H_mat/H_mat_row_sum[:,np.newaxis]

#Printing the row normalized hyperlink matrix.
print(row_norm_H_mat)
```

```
[[0.         0.         0.25       0.         0.         0.25
  0.25       0.         0.25       0.        ]
 [0.         0.         0.25       0.         0.25       0.
  0.25       0.         0.25       0.        ]
 [0.33333333 0.33333333 0.         0.         0.         0.
  0.33333333 0.         0.         0.        ]
 [0.25       0.         0.25       0.         0.         0.
  0.25       0.         0.25       0.        ]
 [0.         0.         0.5        0.         0.         0.
  0.         0.         0.5        0.        ]
 [0.33333333 0.         0.         0.33333333 0.         0.
  0.33333333 0.         0.         0.        ]
 [0.33333333 0.         0.33333333 0.         0.         0.
  0.         0.         0.33333333 0.        ]
 [0.33333333 0.         0.33333333 0.         0.         0.
  0.         0.         0.33333333 0.        ]
 [0.         0.         0.         0.         0.         0.
  1.         0.         0.         0.        ]
 [0.25       0.         0.25       0.         0.         0.
  0.25       0.         0.25       0.        ]]
```

Above is the hyperlink matrix that has been row-normalized, Matrix H.

[1] Davey, B Pitkethly, J (2013), "Math Delivers! Google PageRank", Australian Mathematical Science Institute, Available at: https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Pagerank5.pdf.

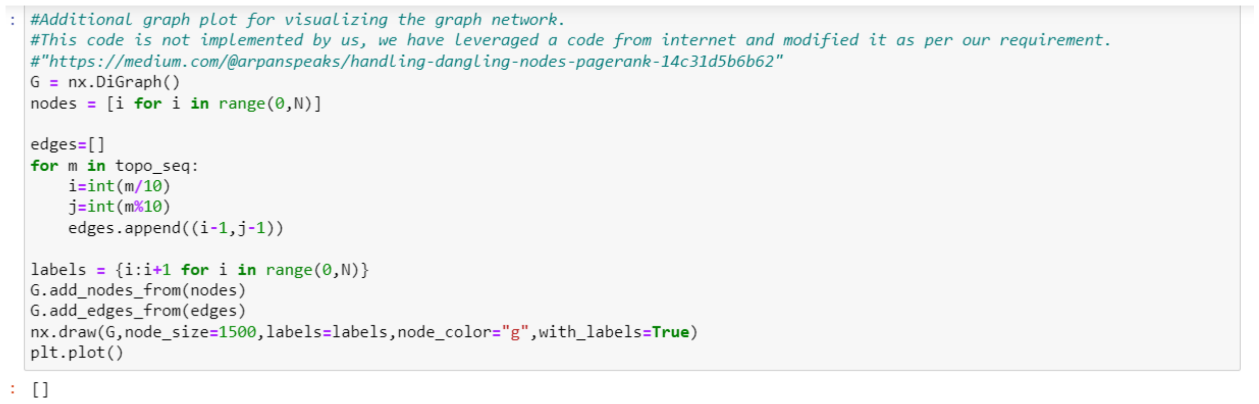
Dangling Nodes:

Next, a quick examination was done to ensure that there are no dangling nodes. Dangling nodes are said to be nodes that have no outbound connections to other nodes, in this case webpages without links to other webpages.[2]

Dangling nodes can be found in two ways. The first method is conducted by examining each row and seeing if there is a row of 0s. Since each row corresponds to a node's out links, if a row has all 0's, then it is a dangling node.

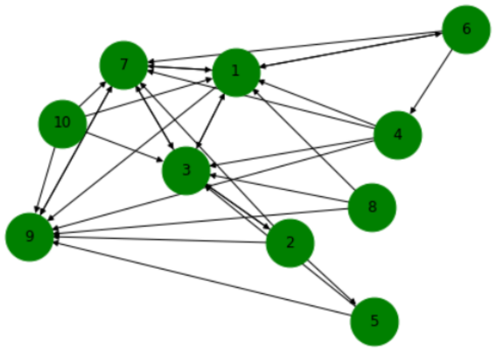
Using the matrix populated matrices presented above, there are no rows with only 0's, and such, there are no dangling nodes

To verify this finding, the second method of verification was employed. The graph of the webpage nodes, with their in and out links was generated, and checked:



```
#Additional graph plot for visualizing the graph network.
#This code is not implemented by us, we have leveraged a code from internet and modified it as per our requirement.
#"https://medium.com/@arpanspeaks/handling-dangling-nodes-pagerank-14c31d5b6b62"
G = nx.DiGraph()
nodes = [i for i in range(0,N)]

edges=[]
for m in topo_seq:
    i=int(m/10)
    j=int(m%10)
    edges.append((i-1,j-1))

labels = {i:i+1 for i in range(0,N)}
G.add_nodes_from(nodes)
G.add_edges_from(edges)
nx.draw(G,node_size=1500,labels=labels,node_color="g",with_labels=True)
plt.plot()
```

```
[]
```



Similar to row evaluation, one can see that there are no dangling nodes, and that every node has outbound connections.

Dangling nodes are an important concept, because if there are dangling nodes, the row reduced matrix H cannot be used as a transition matrix or a stochastic matrix, needed to iteratively

[2] Ibid., pp. 13.

calculate the “importance” of each node (webpage). (Section C discusses the idea of Markov chains). This is because dangling nodes represent webpages for which one cannot leave simply by clicking referential weblinks. As such, a dangling node does not contribute any information to iterative model. As such, should this condition arise, with dangling nodes found, each 0 in the row that corresponds to the node would have to be recalibrated by transforming each entry into an equal probability, resulting in usable stochastic matrix. Such a step can be done, since Google assumes should a user find themselves on a dangling node webpages, then there is an equal probability they will select any of the other webpages in a network.[3] Since this does not occur for the given graph of web page links, this step was not taken, and as such the matrix H can be used as a stochastic matrix S.

(b) Find the Google matrix G of this graph. Assume the damping factor d=0.85.

To find the Google matrix G for this graph, the following steps were taken in accordance with the procedure described in documentation.[4]

The Google Matrix formula: G=dS+(1-d)N.

In the above formula:

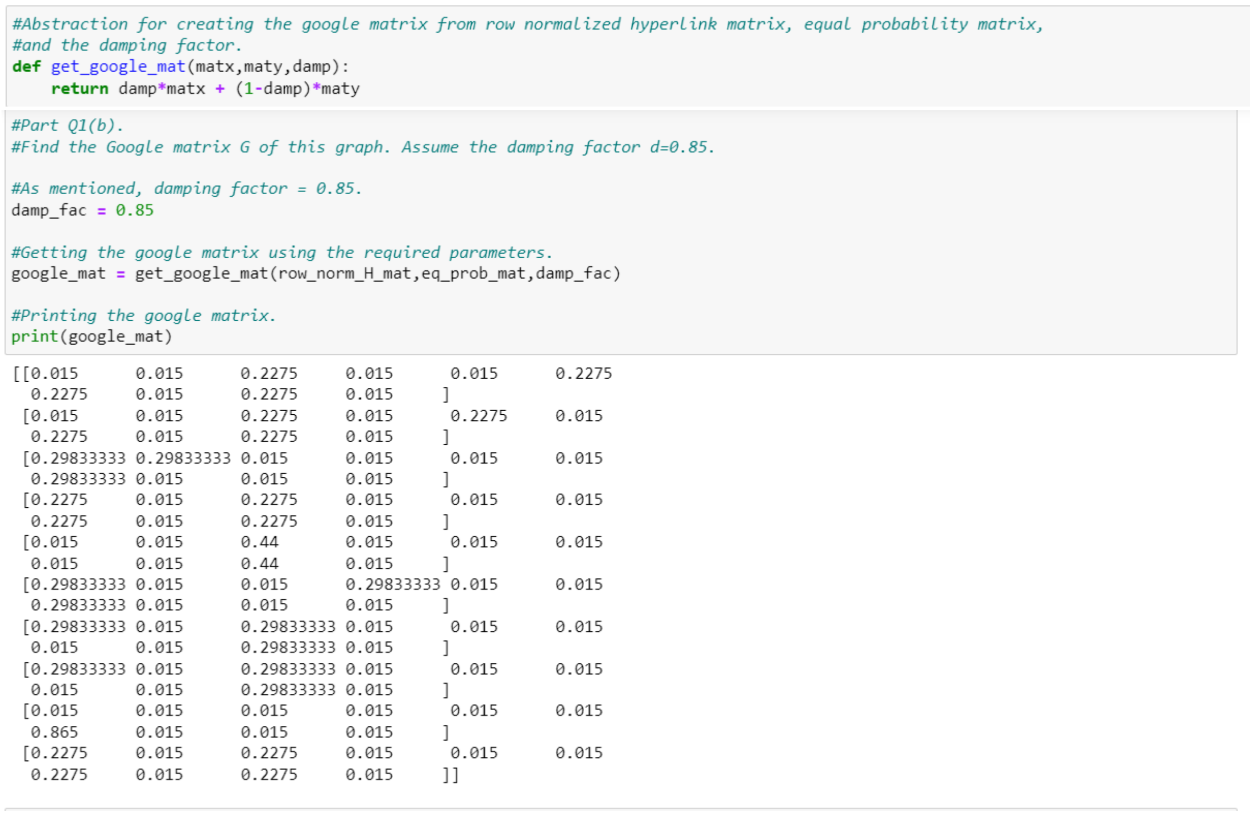
a. d is the damping factor scalar, representing the proportion of time a user chooses to click a weblink when on a webpage, rather than simply selecting a random unlinked webpage. Since it is a proportion, it is represented as: $0 \leq d \leq 1$
b. S is the row normalized stochastic matrix representing in-network probabilities. Since there are no dangling nodes, matrix H can be used as a stochastic matrix.
c. (1-d) represents the proportion of time a user leaves the linked network and goes to page outside of the above graph
d. N is the non-network stochastic matrix, where all webpages have an equal probability of selection.

The general idea of this formula is that the damped factor * the row normalized gives the probability of selected a webpage, using the weblink connections, in a network given a proportion of time (damping factor) that a user choses to stay in the network, plus the probability that a user will chose a webpage outside of the network’s weblinks (represented by 1-damping factor * the non-network matrix N). The Matrix N is used here because Google assumes that all other pages outside the network have an equal probability of being selected.[5]

For this assignment, the damping factor of .85 is multiplied into matrix H, found in part a. Next, a stochastic matrix for non-network webpages is created. In this matrix, each row sums to, and given that 10 elements are in each row and column, each entry is .1. Then, the 1-the damping

[3] Ibid.
[4] Ibid., 13.
[5] Ibid., pp. 13.

factor is multiplied into this stochastic matrix. The two matrices are then summed to form the Google Matrix G.



```
#Abstraction for creating the google matrix from row normalized hyperlink matrix, equal probability matrix,
#and the damping factor.
def get_google_mat(matx,maty,damp):
    return damp*matx + (1-damp)*maty
```

```
#Part Q1(b).
#Find the Google matrix G of this graph. Assume the damping factor d=0.85.

#As mentioned, damping factor = 0.85.
damp_fac = 0.85

#Getting the google matrix using the required parameters.
google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

#Printing the google matrix.
print(google_mat)
```

```
[[0.015      0.015      0.2275     0.015      0.015      0.2275
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.2275     0.015      0.2275     0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.29833333 0.29833333 0.015      0.015      0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.44       0.015      0.015      0.015
  0.015      0.015      0.44       0.015     ]
 [0.29833333 0.015      0.015      0.29833333 0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.015      0.015      0.015      0.015      0.015      0.015
  0.865      0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]]
```

(c) You are NOT allowed to use the formula for the number of iterations (given in [1], page 15, item 2b) needed to obtain numerical values of the PageRank row vector v. Propose an alternative termination criterion for this iterative algorithm.

Concept of PageRank:

Once the Google Matrix is found, representing the importance of a webpage, by its connected nodes, factoring in the probability a user choses to leave a network, it is now important to actually calculate the probability of a user selecting a webpage. The greater the probability the greater the importance score Google assigns a page.

It is important to note that users can access a webpage B from webpage A, where they are both in the same network in two ways:

a) Directly by clicking a weblink on page A that takes them to B
b) Indirectly by clicking a weblink on page A that takes them to another webpage and from there clicking on another weblink (and so on) until they arrive at page B.
c) Web Users continue to move through the network for an indeterminate time period. As such, a direct probability cannot be calculated, due to the dynamic nature of moving through a network, even though the structure of the graph remains static.

As such, the best way to calculate the importance factor, or probability of traversing to a specific webpage is to conduct a Markov chain using a stochastic matrix. Stochastic matrices are matrices containing fixed probabilities between 0 and 1, modeling the different probabilistic states that arise as time changes.[6] Markov chains take each time as they arise, taking only the current state information and multiplying this information into the stochastic matrix, of fixed probabilities, to get the probabilities of a new state. The process repeats, until ideally, the probabilities for each node converge.[7]

For PageRank, the changing time conditions refer to the traversal from webpage to webpage (node to node). Starting with a vector holding an equal probability to traveling to each of these webpages, the vector will be multiplied into the stochastic Google Matrix, the probabilities of traveling to each of the webpages will be updated, and then the process continues. As such, a Markov Chain is conducted. This process continues until the individual states (probability of going to a webpage at a given time) no longer changes, with the probability values converging, thereby telling the relative importance of each webpage.[8]

Termination of Iteration:

However, key to the problem of determining the importance of webpages, by using any Markov Chain including PageRank, is determining when to stop the iteration. To make this determination, the following steps were taken to see if a pattern been proposed:

a. A tolerance factor variable was created. This factor equals 1.0 e^{-n}, where n is the number of digits to the right of the decimal that is expected to be precise.
b. Then an error variable was created.
c. Next, the PageRank Markov Chain is run, in a while loop. The condition of the while loop allows for Markov chain to run while the error is greater than the tolerance.
d. After each iteration is conducted, the error is calculated. The error is calculated by:
    1. Calculating the absolute value of the current result minus one previous result, column wise.
    2. Summing the result of this difference for each column.
e. If the error is greater than the current tolerance factor, the loop continues, and the next iteration begins. Otherwise, the PageRank process terminates, and the matrix of probability results is produced, where each row corresponds to an updated state, while each column corresponds to the importance of each webpage.

[6] Larson, R (2013), Elementary Linear Algebra, Seventh Edition, Brooks/Cole, Cengage: Boston, pp. 84. Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ, pp. 286-287.
[7] Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ, pp. 286-287.
[8] Davey, B Pitkethly, J (2013), "Math Delivers! Google PageRank", Australian Mathematical Science Institute, Available at: https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Pagerank5.pdf. Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ, pp. 498-500.

This process was conducted with the following conditions:

a. The initial vector conditions were set to equal probability. Given 10 nodes, each probability was .1
b. The damping factor was set to .85 to calculate the Google Matrix
c. The following tolerance factors were used: 2-8 digits. Below is the outcome of each test:

Tolerance Factor (Number of Digits)	Number of Iterations Needed to Find Convergence (including the v(0) as the first iteration)
2	8
3	11
4	13
5	16
6	19
7	21
8	23

The number of iterations includes the initial value vector.

Using the process yielded a far fewer required number of iterations to find convergency. According to the PageRank documentation, the number of iterations should be found through the following formula:[9]

Number of iterations = $\frac{m}{log_{10}(1/d)}$, where m is number of digits of accuracy (the tolerance factor) and d is the damping factor.

Google's formula for finding the number of iterations for convergency requires substantially more iterations than the method proposed in this report. In fact, comparing the findings:

Number of Digits, right of decimal	Number of Iterations Needed to Find Convergence (Google Method) (including the v(0) as the first iteration)	Number of Iterations Needed to Find Convergence (Report Method) (including the v(0) as the first iteration)
2	28	8
3	43	11
4	56	13
5	71	16
6	85	19
7	99	21
8	113	23

[9] Davey, B Pitkethly, J (2013), "Math Delivers! Google PageRank", Australian Mathematical Science Institute, Available at: https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Pagerank5.pdf, pp.15

Note: in the above table, the damping factor is .85, the initial vector's values are all set to .1, as mentioned in the earlier table. Number of iterations includes initial value vector. The results presented in both tables in Section C can be found by running the function presented in Section D of this report.

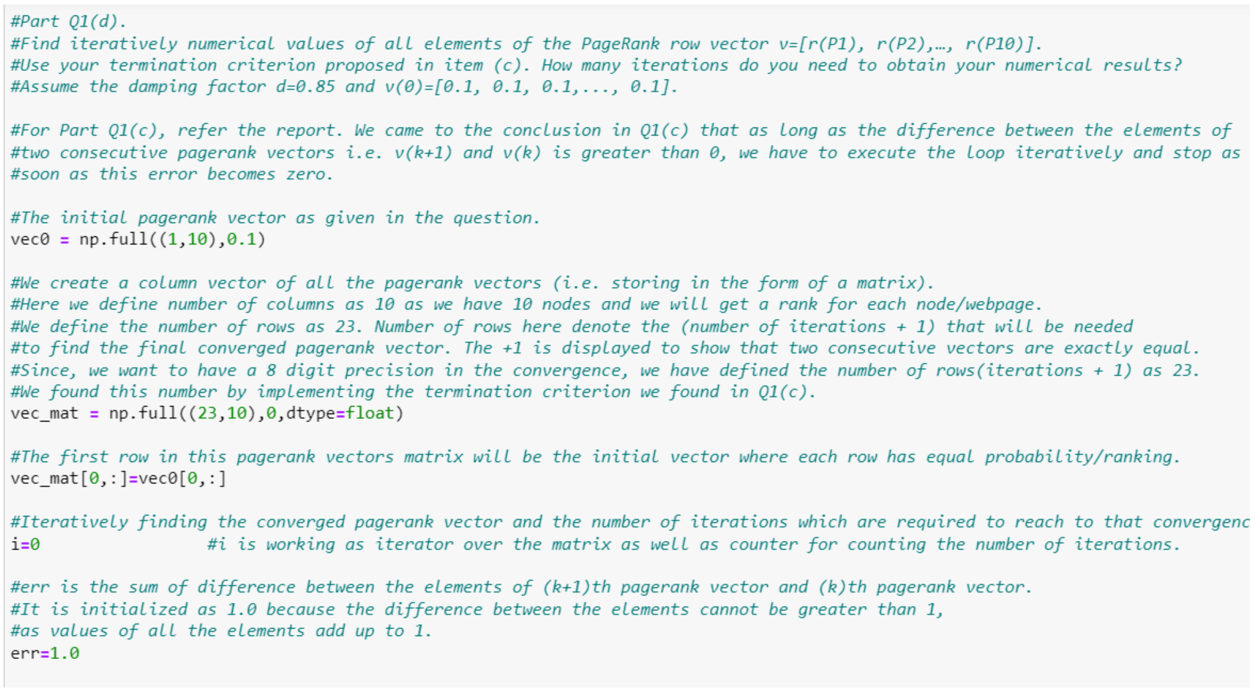
Method used in this report:

As such, calculating an error rate by first, taking the absolute value of the difference of one row vector and the previous row vector and then summing up these differences, followed by comparing this error rate to a tolerance factor and terminating the iterations when the error is less than tolerance rate, is the optimal way of calculating the number of iterations.

It should also be noted that the method proposed in this report for stopping iterations, fits the process of a Markov Chain. Since Markov chains are non-information gain chains, whose current probability is based on the previous state, and whose future probability is based only on the current state, comparing the current state probability to that of one previous state (in order to calculate the error) fits the definition and overall idea of a Markov Chain.[10]

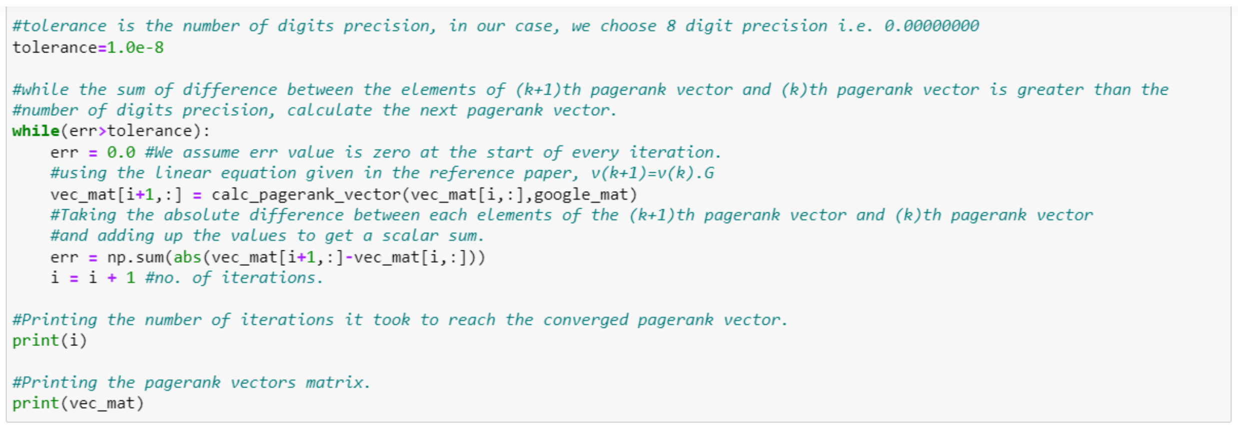
(d) Find iteratively numerical values of all elements of the PageRank row vector v=[r(P1), r(P2),…, r(P10)]. Use your termination criterion proposed in item (c). How many iterations do you need to obtain your numerical results? Assume the damping factor d=0.85 and v (0)=[0.1, 0.1, 0.1,…, 0.1]. Implement this iterative algorithm by yourself (absolutely no Excel or program libraries).

Using the stopping criteria stated in Section C, the following python code was run, and at each state a vector of probabilities for visiting a page in the graph was recorded, using a .85 damping factor and an initial vector of equal probabilities (each entry was .1). Below is the code and results:

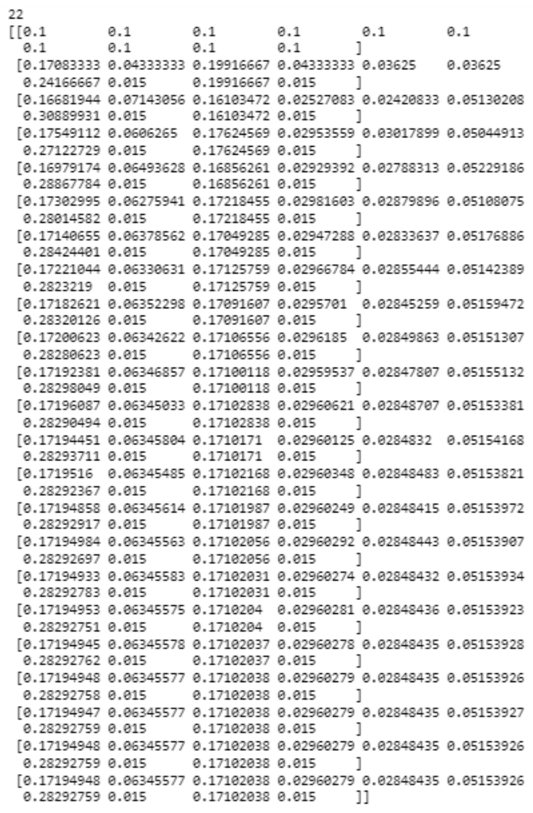


```
#Part Q1(d).
#Find iteratively numerical values of all elements of the PageRank row vector v=[r(P1), r(P2),…, r(P10)].
#Use your termination criterion proposed in item (c). How many iterations do you need to obtain your numerical results?
#Assume the damping factor d=0.85 and v(0)=[0.1, 0.1, 0.1,..., 0.1].

#For Part Q1(c), refer the report. We came to the conclusion in Q1(c) that as long as the difference between the elements of
#two consecutive pagerank vectors i.e. v(k+1) and v(k) is greater than 0, we have to execute the loop iteratively and stop as
#soon as this error becomes zero.

#The initial pagerank vector as given in the question.
vec0 = np.full((1,10),0.1)

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 23. Number of rows here denote the (number of iterations + 1) that will be needed
#to find the final converged pagerank vector. The +1 is displayed to show that two consecutive vectors are exactly equal.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations + 1) as 23.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat = np.full((23,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
vec_mat[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergenc
i=0                 #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0
```

[10] Larson, R (2013), Elementary Linear Algebra, Seventh Edition, Brooks/Cole, Cengage: Boston, pp. 84. Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ, pp. 286-287.



```
#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat[i+1,:] = calc_pagerank_vector(vec_mat[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat[i+1,:]-vec_mat[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print(i)

#Printing the pagerank vectors matrix.
print(vec_mat)
```



```
22
[[0.1        0.1        0.1        0.1        0.1        0.1
  0.1        0.1        0.1        0.1       ]
 [0.17083333 0.04333333 0.19916667 0.04333333 0.03625    0.03625
  0.24166667 0.015      0.19916667 0.015     ]
 [0.16681944 0.07143056 0.16103472 0.02527083 0.02420833 0.05130208
  0.30889931 0.015      0.16103472 0.015     ]
 [0.17549112 0.0606265  0.17624569 0.02953559 0.03017899 0.05044913
  0.27122729 0.015      0.17624569 0.015     ]
 [0.16979174 0.06493628 0.16856261 0.02929392 0.02788313 0.05229186
  0.28867784 0.015      0.16856261 0.015     ]
 [0.17302995 0.06275941 0.17218455 0.02981603 0.02879896 0.05108075
  0.28014582 0.015      0.17218455 0.015     ]
 [0.17140655 0.06378562 0.17049285 0.02947288 0.02833637 0.05176886
  0.28424401 0.015      0.17049285 0.015     ]
 [0.17221044 0.06330631 0.17125759 0.02966784 0.02855444 0.05142389
  0.2823219  0.015      0.17125759 0.015     ]
 [0.17182621 0.06352298 0.17091607 0.0295701  0.02845259 0.05159472
  0.28320126 0.015      0.17091607 0.015     ]
 [0.17200623 0.06342622 0.17106556 0.0296185  0.02849863 0.05151307
  0.28280623 0.015      0.17106556 0.015     ]
 [0.17192381 0.06346857 0.17100118 0.02959537 0.02847807 0.05155132
  0.28298049 0.015      0.17100118 0.015     ]
 [0.17196087 0.06345033 0.17102838 0.02960621 0.02848707 0.05153381
  0.28290494 0.015      0.17102838 0.015     ]
 [0.17194451 0.06345804 0.1710171  0.02960125 0.0284832  0.05154168
  0.28293711 0.015      0.1710171  0.015     ]
 [0.1719516  0.06345485 0.17102168 0.02960348 0.02848483 0.05153821
  0.28292367 0.015      0.17102168 0.015     ]
 [0.17194858 0.06345614 0.17101987 0.02960249 0.02848415 0.05153972
  0.28292917 0.015      0.17101987 0.015     ]
 [0.17194984 0.06345563 0.17102056 0.02960292 0.02848443 0.05153907
  0.28292697 0.015      0.17102056 0.015     ]
 [0.17194933 0.06345583 0.17102031 0.02960274 0.02848432 0.05153934
  0.28292783 0.015      0.17102031 0.015     ]
 [0.17194953 0.06345575 0.1710204  0.02960281 0.02848436 0.05153923
  0.28292751 0.015      0.1710204  0.015     ]
 [0.17194945 0.06345578 0.17102037 0.02960278 0.02848435 0.05153928
  0.28292762 0.015      0.17102037 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292758 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]
```

Since the pagerank vector we found is as follows:

v = [0.17194948,0.06345577,0.17102038,0.02960279,0.02848435,0.05153926,0.28292759,0.015,0.17102038,0.015]

On observation, it can be said that the ranking order of nodes is as follows:

P7, P1, P3, P9, P2, P6, P4, P5, P8, P10 as the ranking is based on the respective final converged pagerank vector values, which is

0.28292759, 0.17194948, 0.17102038, 0.17102038, 0.06345577, 0.05153926, 0.02960279, 0.02848435, 0.015, 0.015

Using the probabilities found, the output of probabilities in descending order of rank score is:

Page	Probability
P7	.28292759
P1	.17194948
P3	.17102038

P9	.17102038
P2	.06345577
P6	.05153926
P4	.02960279
P5	.02848435
P8	.015
P10	.015

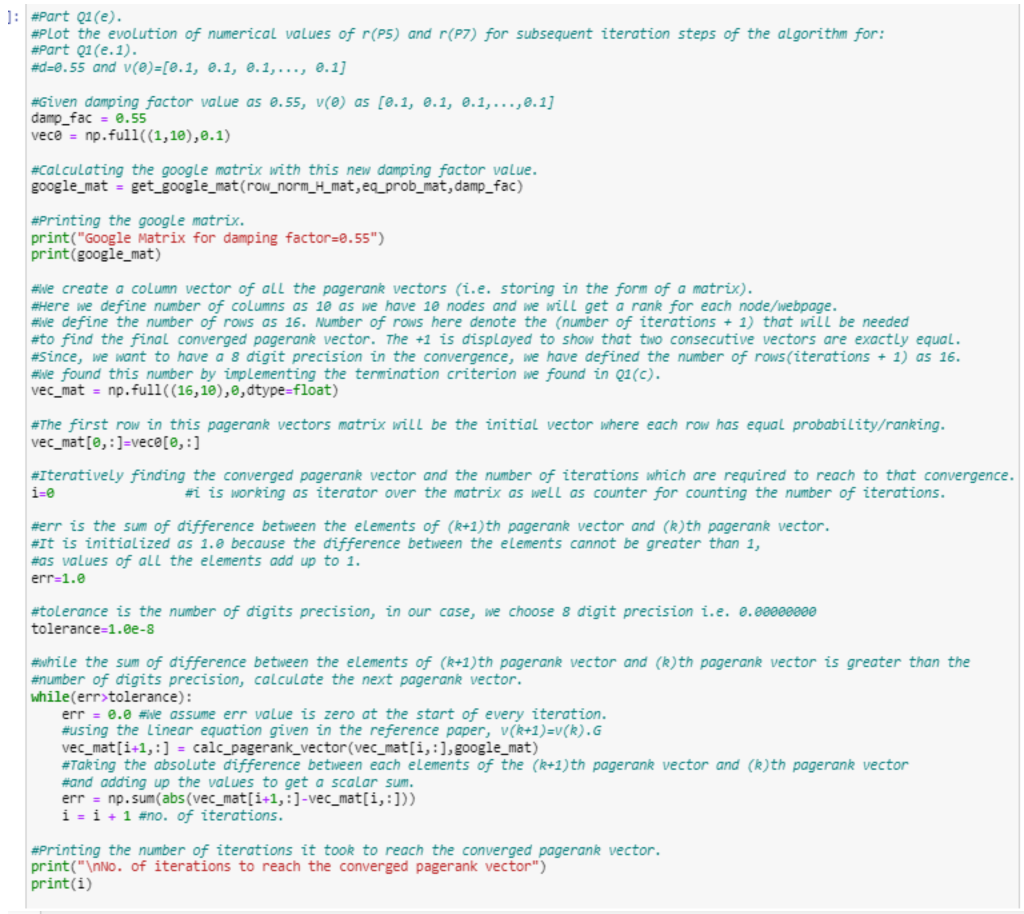
The number of iterations:

Using a tolerance of 8 digits, the number of iterations to find this convergence was 23 (including the initial value vector).

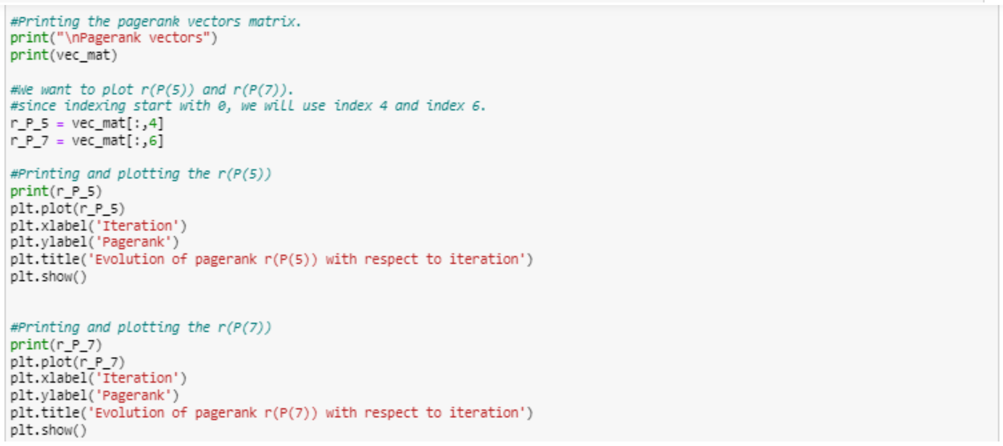
(e) Plot the evolution of numerical values of r(P5) and r(P7) for subsequent iteration steps of the algorithm for: (e.1) d=0.55 and v (0)=[0.1, 0.1, 0.1,…, 0.1] (e.2) d=0.85 and v (0)=[0.1, 0.1, 0.1,…, 0.1]

Presented below is the code and plots for the iteration steps for r(P5) and r(P7) using the PageRank algorithm using two damping factors, d=.55 and d=.85, with both using an initial vector of equal probability (entries equal at .1):

Damping Factor .55 (Code and Results):

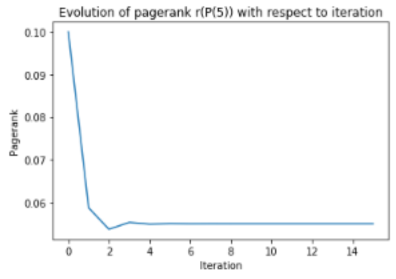


```
]: #Part Q1(e).
   #Plot the evolution of numerical values of r(P5) and r(P7) for subsequent iteration steps of the algorithm for:
   #Part Q1(e.1).
   #d=0.55 and v(0)=[0.1, 0.1, 0.1,..., 0.1]

   #Given damping factor value as 0.55, v(0) as [0.1, 0.1, 0.1,...,0.1]
   damp_fac = 0.55
   vec0 = np.full((1,10),0.1)

   #Calculating the google matrix with this new damping factor value.
   google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

   #Printing the google matrix.
   print("Google Matrix for damping factor=0.55")
   print(google_mat)

   #We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
   #Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
   #We define the number of rows as 16. Number of rows here denote the (number of iterations + 1) that will be needed
   #to find the final converged pagerank vector. The +1 is displayed to show that two consecutive vectors are exactly equal.
   #Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations + 1) as 16.
   #We found this number by implementing the termination criterion we found in Q1(c).
   vec_mat = np.full((16,10),0,dtype=float)

   #The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
   vec_mat[0,:]=vec0[0,:]

   #Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergence.
   i=0                #i is working as iterator over the matrix as well as counter for counting the number of iterations.

   #err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
   #It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
   #as values of all the elements add up to 1.
   err=1.0

   #tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
   tolerance=1.0e-8

   #While the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
   #number of digits precision, calculate the next pagerank vector.
   while(err>tolerance):
       err = 0.0 #We assume err value is zero at the start of every iteration.
       #using the linear equation given in the reference paper, v(k+1)=v(k).G
       vec_mat[i+1,:] = calc_pagerank_vector(vec_mat[i,:],google_mat)
       #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
       #and adding up the values to get a scalar sum.
       err = np.sum(abs(vec_mat[i+1,:]-vec_mat[i,:]))
       i = i + 1 #no. of iterations.

   #Printing the number of iterations it took to reach the converged pagerank vector.
   print("\nNo. of iterations to reach the converged pagerank vector")
   print(i)
```

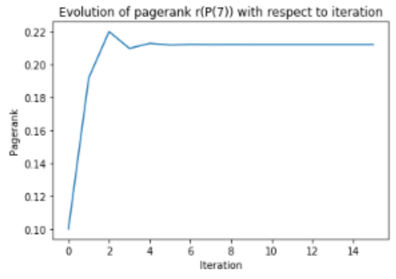


```
#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat)

#We want to plot r(P(5)) and r(P(7)).
#since indexing start with 0, we will use index 4 and index 6.
r_P_5 = vec_mat[:,4]
r_P_7 = vec_mat[:,6]

#Printing and plotting the r(P(5))
print(r_P_5)
plt.plot(r_P_5)
plt.xlabel('Iteration')
plt.ylabel('Pagerank')
plt.title('Evolution of pagerank r(P(5)) with respect to iteration')
plt.show()


#Printing and plotting the r(P(7))
print(r_P_7)
plt.plot(r_P_7)
plt.xlabel('Iteration')
plt.ylabel('Pagerank')
plt.title('Evolution of pagerank r(P(7)) with respect to iteration')
plt.show()
```

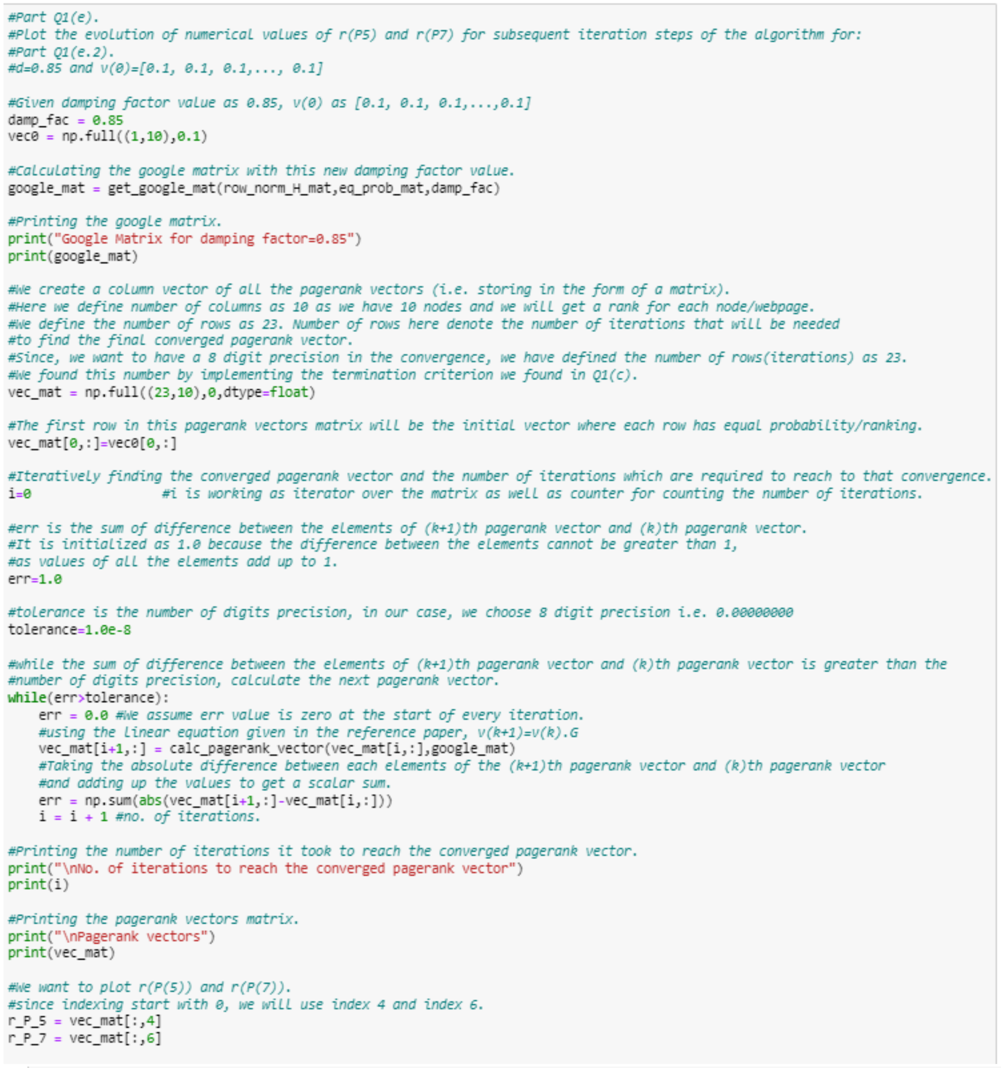
```
Google Matrix for damping factor=0.55
[[0.045      0.045      0.1825     0.045      0.045      0.1825
  0.1825     0.045      0.1825     0.045     ]
 [0.045      0.045      0.1825     0.045      0.1825     0.045
  0.1825     0.045      0.1825     0.045     ]
 [0.22833333 0.22833333 0.045      0.045      0.045      0.045
  0.22833333 0.045      0.045      0.045     ]
 [0.1825     0.045      0.1825     0.045      0.045      0.045
  0.1825     0.045      0.1825     0.045     ]
 [0.045      0.045      0.32       0.045      0.045      0.045
  0.045      0.045      0.32       0.045     ]
 [0.22833333 0.045      0.045      0.22833333 0.045      0.045
  0.22833333 0.045      0.045      0.045     ]
 [0.22833333 0.045      0.22833333 0.045      0.045      0.045
  0.045      0.045      0.22833333 0.045     ]
 [0.22833333 0.045      0.22833333 0.045      0.045      0.045
  0.045      0.045      0.22833333 0.045     ]
 [0.045      0.045      0.045      0.045      0.045      0.045
  0.595      0.045      0.045      0.045     ]
 [0.1825     0.045      0.1825     0.045      0.045      0.045
  0.1825     0.045      0.1825     0.045     ]]

No. of iterations to reach the converged pagerank vector
15

Pagerank vectors
[[0.1        0.1        0.1        0.1        0.1        0.1
  0.1        0.1        0.1        0.1       ]
 [0.14583333 0.06333333 0.16416667 0.06333333 0.05875    0.05875
  0.19166667 0.045      0.16416667 0.045     ]
 [0.14415278 0.07509722 0.14820139 0.05577083 0.05370833 0.06505208
  0.21981597 0.045      0.14820139 0.045     ]
 [0.14650205 0.07217025 0.15232225 0.05692622 0.05532587 0.06482101
  0.2096101  0.045      0.15232225 0.045     ]
 [0.14550297 0.07292575 0.15097543 0.05688385 0.05492341 0.06514403
  0.21266913 0.045      0.15097543 0.045     ]
 [0.14587027 0.07267883 0.15138626 0.05694307 0.05502729 0.06500666
  0.21170137 0.045      0.15138626 0.045     ]
 [0.14575112 0.07275415 0.1512621  0.05691789 0.05499334 0.06505716
  0.21200215 0.045      0.1512621  0.045     ]
 [0.1457893  0.07273138 0.15129841 0.05692715 0.0550037  0.06504078
  0.21191087 0.045      0.15129841 0.045     ]
 [0.14577749 0.07273804 0.15128792 0.05692414 0.05500057 0.06504603
  0.21193789 0.045      0.15128792 0.045     ]
 [0.14578107 0.07273612 0.15129089 0.05692511 0.05500148 0.06504441
  0.21193003 0.045      0.15129089 0.045     ]
 [0.14578001 0.07273666 0.15129006 0.05692481 0.05500122 0.0650449
  0.21193228 0.045      0.15129006 0.045     ]
 [0.14578032 0.07273651 0.15129029 0.0569249  0.05500129 0.06504475
  0.21193165 0.045      0.15129029 0.045     ]
 [0.14578023 0.07273655 0.15129023 0.05692487 0.05500127 0.06504479
  0.21193182 0.045      0.15129023 0.045     ]
 [0.14578026 0.07273654 0.15129024 0.05692488 0.05500128 0.06504478
  0.21193177 0.045      0.15129024 0.045     ]
 [0.14578025 0.07273654 0.15129024 0.05692488 0.05500127 0.06504479
  0.21193179 0.045      0.15129024 0.045     ]
 [0.14578025 0.07273654 0.15129024 0.05692488 0.05500127 0.06504478
  0.21193178 0.045      0.15129024 0.045     ]]
[0.1        0.05875    0.05370833 0.05532587 0.05492341 0.05502729
 0.05499334 0.0550037  0.05500057 0.05500148 0.05500122 0.05500129
 0.05500127 0.05500128 0.05500127 0.05500127]
```



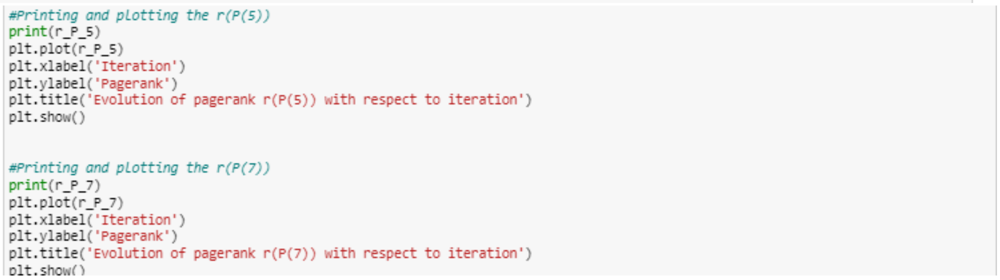
```
[0.1        0.19166667 0.21981597 0.2096101  0.21266913 0.21170137
 0.21200215 0.21191087 0.21193789 0.21193003 0.21193228 0.21193165
 0.21193182 0.21193177 0.21193179 0.21193178]
```



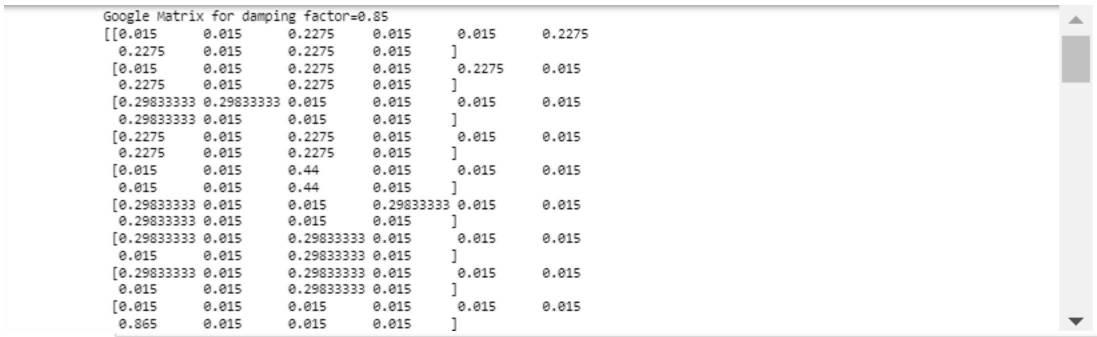
Damping Factor .85:



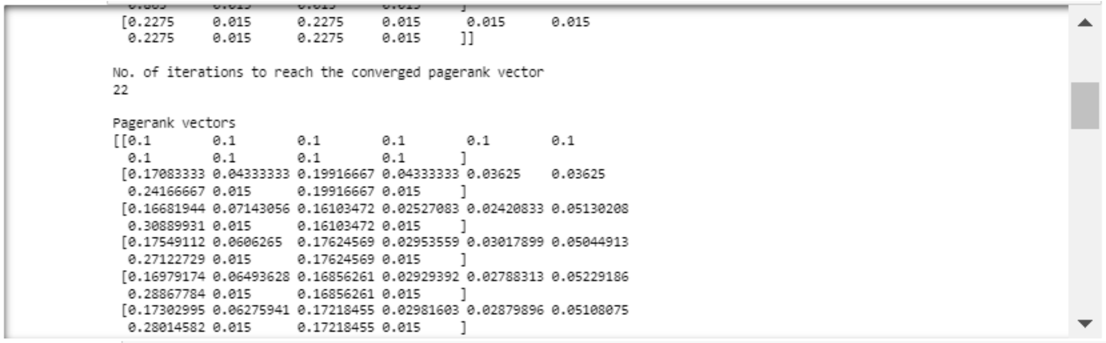
```
#Part Q1(e).
#Plot the evolution of numerical values of r(P5) and r(P7) for subsequent iteration steps of the algorithm for:
#Part Q1(e.2).
#d=0.85 and v(0)=[0.1, 0.1, 0.1,..., 0.1]

#Given damping factor value as 0.85, v(0) as [0.1, 0.1, 0.1,...,0.1]
damp_fac = 0.85
vec0 = np.full((1,10),0.1)

#Calculating the google matrix with this new damping factor value.
google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

#Printing the google matrix.
print("Google Matrix for damping factor=0.85")
print(google_mat)

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 23. Number of rows here denote the number of iterations that will be needed
#to find the final converged pagerank vector.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 23.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat = np.full((23,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
vec_mat[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergence.
i=0              #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0

#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#While the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat[i+1,:] = calc_pagerank_vector(vec_mat[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat[i+1,:]-vec_mat[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print("\nNo. of iterations to reach the converged pagerank vector")
print(i)

#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat)

#We want to plot r(P(5)) and r(P(7)).
#since indexing start with 0, we will use index 4 and index 6.
r_P_5 = vec_mat[:,4]
r_P_7 = vec_mat[:,6]
```



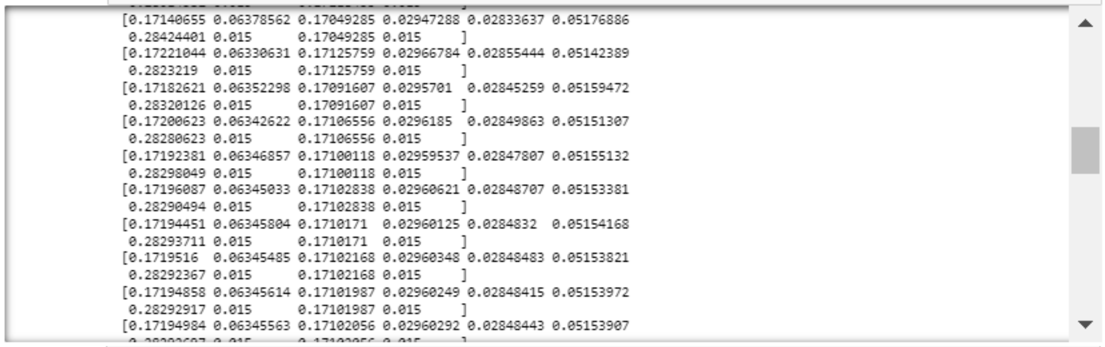
```
#Printing and plotting the r(P(5))
print(r_P_5)
plt.plot(r_P_5)
plt.xlabel('Iteration')
plt.ylabel('Pagerank')
plt.title('Evolution of pagerank r(P(5)) with respect to iteration')
plt.show()


#Printing and plotting the r(P(7))
print(r_P_7)
plt.plot(r_P_7)
plt.xlabel('Iteration')
plt.ylabel('Pagerank')
plt.title('Evolution of pagerank r(P(7)) with respect to iteration')
plt.show()
```



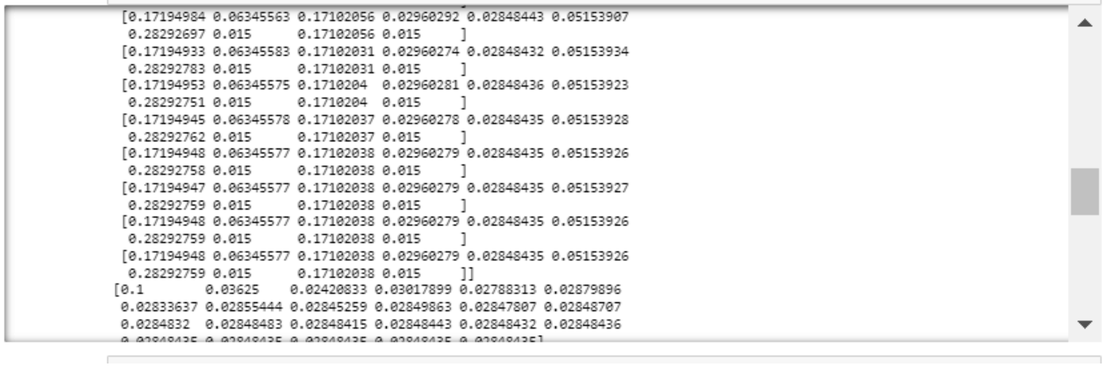
```
Google Matrix for damping factor=0.85
[[0.015      0.015      0.2275     0.015      0.015      0.2275
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.2275     0.015      0.2275     0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.29833333 0.29833333 0.015      0.015      0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.44       0.015      0.015      0.015
  0.015      0.015      0.44       0.015     ]
 [0.29833333 0.015      0.015      0.29833333 0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.015      0.015      0.015      0.015      0.015      0.015
  0.865      0.015      0.015      0.015     ]
```



```
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]]

No. of iterations to reach the converged pagerank vector
22

Pagerank vectors
[[0.1        0.1        0.1        0.1        0.1        0.1
  0.1        0.1        0.1        0.1       ]
 [0.17083333 0.04333333 0.19916667 0.04333333 0.03625    0.03625
  0.24166667 0.015      0.19916667 0.015     ]
 [0.16681944 0.07143056 0.16103472 0.02527083 0.02420833 0.05130208
  0.30889931 0.015      0.16103472 0.015     ]
 [0.17549112 0.0606265  0.17624569 0.02953559 0.03017899 0.05044913
  0.27122729 0.015      0.17624569 0.015     ]
 [0.16979174 0.06493628 0.16856261 0.02929392 0.02788313 0.05229186
  0.28867784 0.015      0.16856261 0.015     ]
 [0.17302995 0.06275941 0.17218455 0.02981603 0.02879896 0.05108075
  0.28014582 0.015      0.17218455 0.015     ]
```



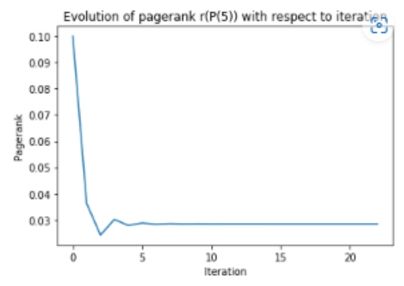
```
 [0.17140655 0.06378562 0.17049285 0.02947288 0.02833637 0.05176886
  0.28424401 0.015      0.17049285 0.015     ]
 [0.17221044 0.06330631 0.17125759 0.02966784 0.02855444 0.05142389
  0.2823219  0.015      0.17125759 0.015     ]
 [0.17182621 0.06352298 0.17091607 0.0295701  0.02845259 0.05159472
  0.28320126 0.015      0.17091607 0.015     ]
 [0.17200623 0.06342622 0.17106556 0.0296185  0.02849863 0.05151307
  0.28280623 0.015      0.17106556 0.015     ]
 [0.17192381 0.06346857 0.17100118 0.02959537 0.02847807 0.05155132
  0.28298049 0.015      0.17100118 0.015     ]
 [0.17196087 0.06345033 0.17102838 0.02960621 0.02848707 0.05153381
  0.28290494 0.015      0.17102838 0.015     ]
 [0.17194451 0.06345804 0.1710171  0.02960125 0.0284832  0.05154168
  0.28293711 0.015      0.1710171  0.015     ]
 [0.1719516  0.06345485 0.17102168 0.02960348 0.02848483 0.05153821
  0.28292367 0.015      0.17102168 0.015     ]
 [0.17194858 0.06345614 0.17101987 0.02960249 0.02848415 0.05153972
  0.28292917 0.015      0.17101987 0.015     ]
 [0.17194984 0.06345563 0.17102056 0.02960292 0.02848443 0.05153907
```



```
 [0.17194984 0.06345563 0.17102056 0.02960292 0.02848443 0.05153907
  0.28292697 0.015      0.17102056 0.015     ]
 [0.17194933 0.06345583 0.17102031 0.02960274 0.02848432 0.05153934
  0.28292783 0.015      0.17102031 0.015     ]
 [0.17194953 0.06345575 0.1710204  0.02960281 0.02848436 0.05153923
  0.28292751 0.015      0.1710204  0.015     ]
 [0.17194945 0.06345578 0.17102037 0.02960278 0.02848435 0.05153928
  0.28292762 0.015      0.17102037 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292758 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]
[0.1        0.03625    0.02420833 0.03017899 0.02788313 0.02879896
 0.02833637 0.02855444 0.02845259 0.02849863 0.02847807 0.02848707
 0.0284832  0.02848483 0.02848415 0.02848443 0.02848432 0.02848436
 0.02848435 0.02848435 0.02848435 0.02848435 0.02848435]
```

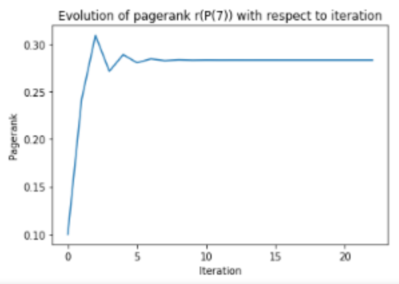
```
plt.show()
```

```
0.02848435 0.02848435 0.02848435 0.02848435 0.02848435]
```



```
[0.1        0.24166667 0.30889931 0.27122729 0.28867784 0.28014582
 0.28424401 0.2823219  0.28320126 0.28280623 0.28298049 0.28290494
 0.28293711 0.28292367 0.28292917 0.28292697 0.28292783 0.28292751
 0.28292762 0.28292758 0.28292759 0.28292759 0.28292759]
```

```
 0.28293711 0.28292367 0.28292917 0.28292697 0.28292783 0.28292751
 0.28292762 0.28292758 0.28292759 0.28292759 0.28292759]
```



(f) Assume d=0.85 and run your algorithm for three different initial PageRank row vectors v1 (0) , v2 (0) and v3 (0). What are the final numerical values of v1, v2 and v3 ? Are they identical or different?

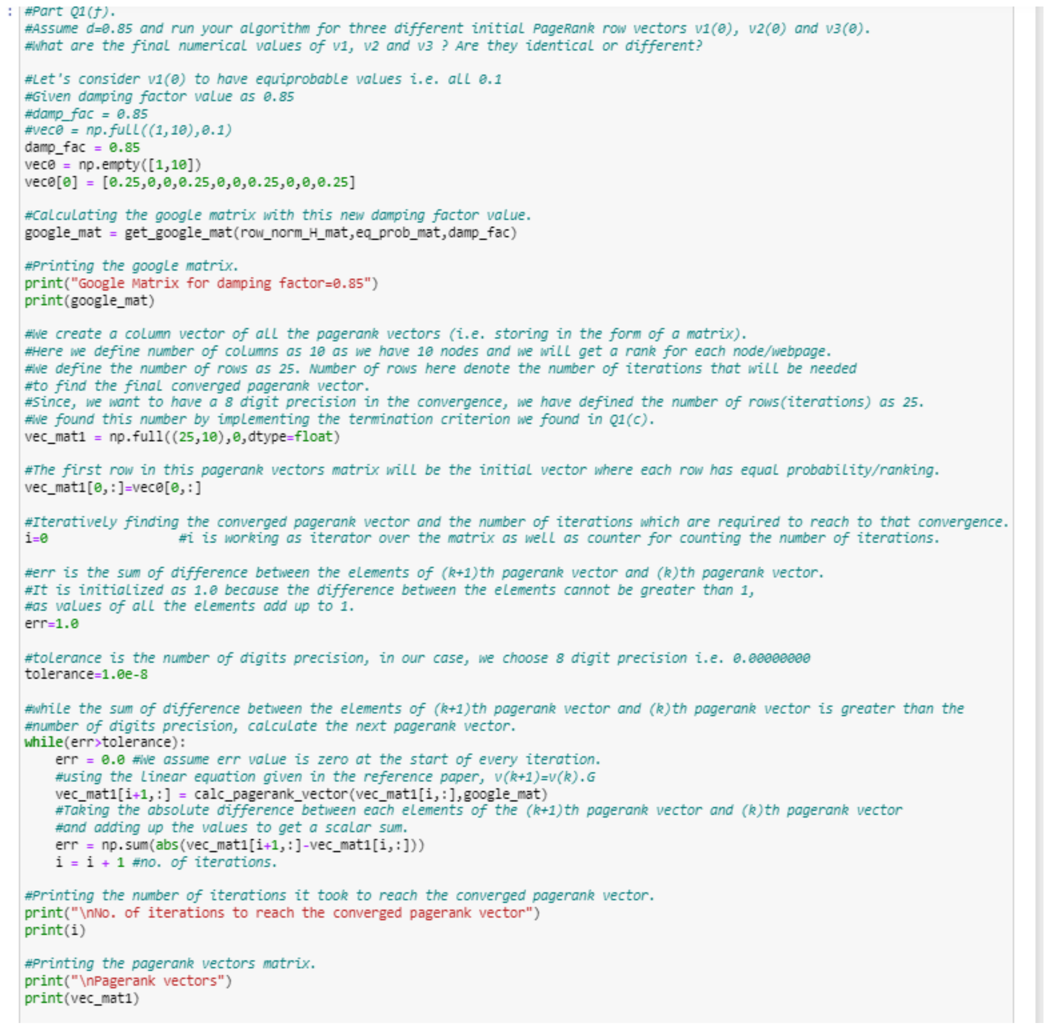
For this section the following initial vectors were used:

v1(0)=<.25, 0, 0, .25, 0, 0, .25, 0, 0, .25>
v2(0)=<.2, 0, .2, 0, .2, 0, .2, 0, .2, 0>
v3(0)=<.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, .5>

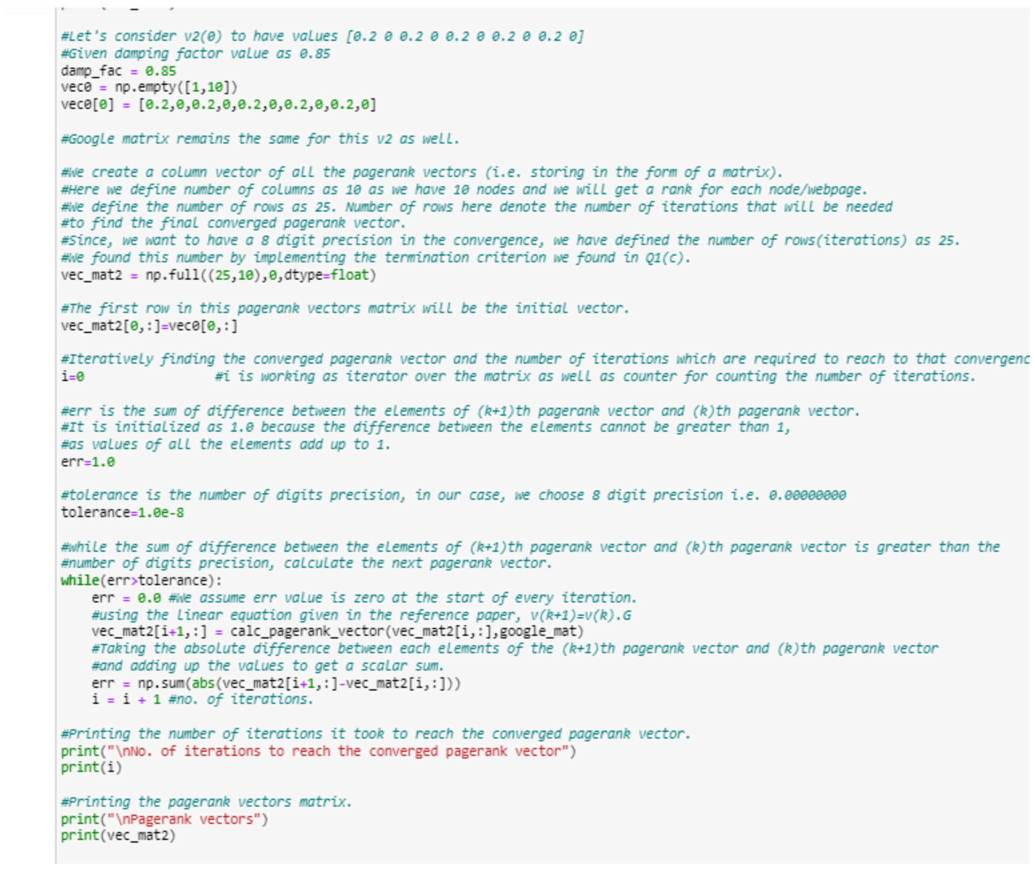
With each of these row vectors summing to 1.

Next, the following steps were taken:

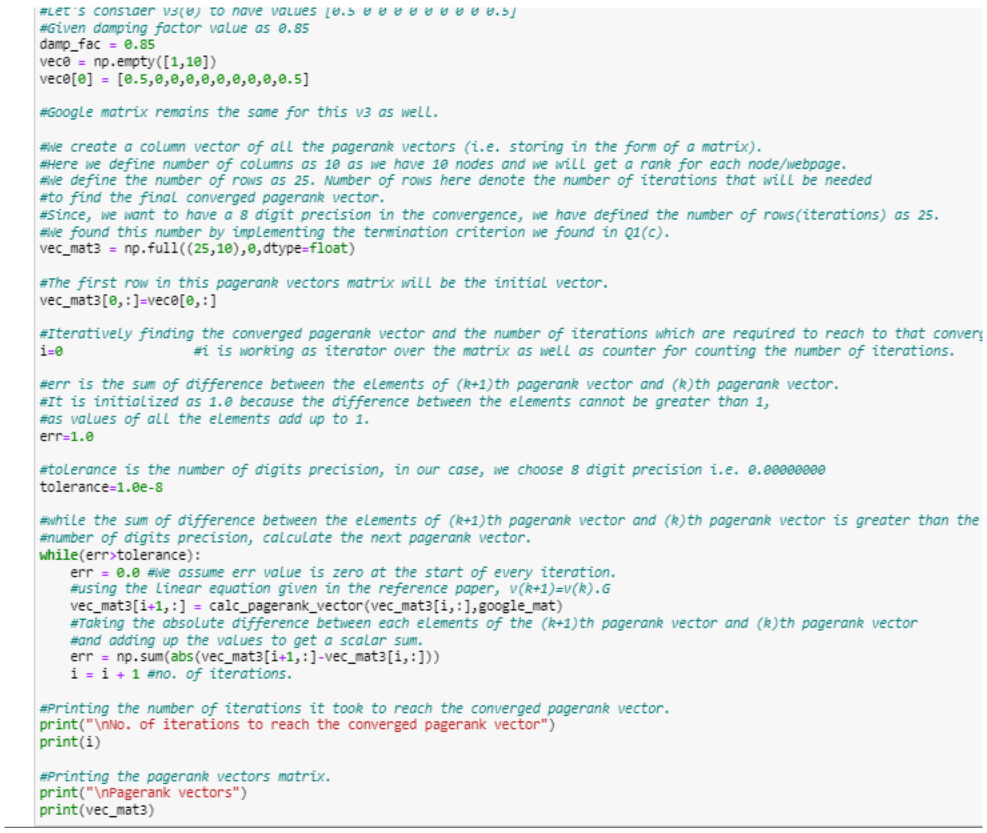
1. The Google Matrix was found using the damping factor of .85
2. Next each one of these initial vectors v1-v3 was run iteratively through the PageRank algorithms. Below is the code and results:



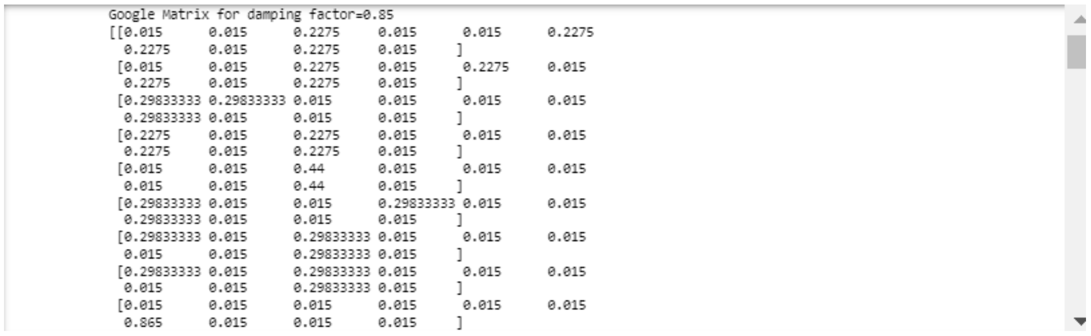
```
#Part Q1(f).
#Assume d=0.85 and run your algorithm for three different initial PageRank row vectors v1(0), v2(0) and v3(0).
#What are the final numerical values of v1, v2 and v3 ? Are they identical or different?

#Let's consider v1(0) to have equiprobable values i.e. all 0.1
#Given damping factor value as 0.85
#damp_fac = 0.85
#vec0 = np.full((1,10),0.1)
damp_fac = 0.85
vec0 = np.empty([1,10])
vec0[0] = [0.25,0,0,0.25,0,0,0.25,0,0,0.25]

#Calculating the google matrix with this new damping factor value.
google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

#Printing the google matrix.
print("Google Matrix for damping factor=0.85")
print(google_mat)

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 25. Number of rows here denote the number of iterations that will be needed
#to find the final converged pagerank vector.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 25.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat1 = np.full((25,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
vec_mat1[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergence.
i=0                 #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0

#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat1[i+1,:] = calc_pagerank_vector(vec_mat1[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat1[i+1,:]-vec_mat1[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print("\nNo. of iterations to reach the converged pagerank vector")
print(i)

#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat1)
```



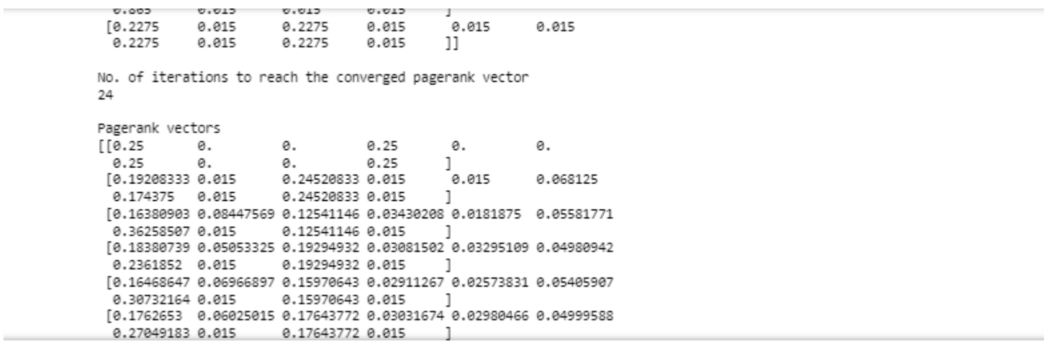
```python
#Let's consider v2(0) to have values [0.2 0 0.2 0 0.2 0 0.2 0 0.2 0]
#Given damping factor value as 0.85
damp_fac = 0.85
vec0 = np.empty([1,10])
vec0[0] = [0.2,0,0.2,0,0.2,0,0.2,0,0.2,0]

#Google matrix remains the same for this v2 as well.

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 25. Number of rows here denote the number of iterations that will be needed
#to find the final converged pagerank vector.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 25.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat2 = np.full((25,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector.
vec_mat2[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergenc
i=0                 #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0

#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat2[i+1,:] = calc_pagerank_vector(vec_mat2[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat2[i+1,:]-vec_mat2[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print("\nNo. of iterations to reach the converged pagerank vector")
print(i)

#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat2)
```



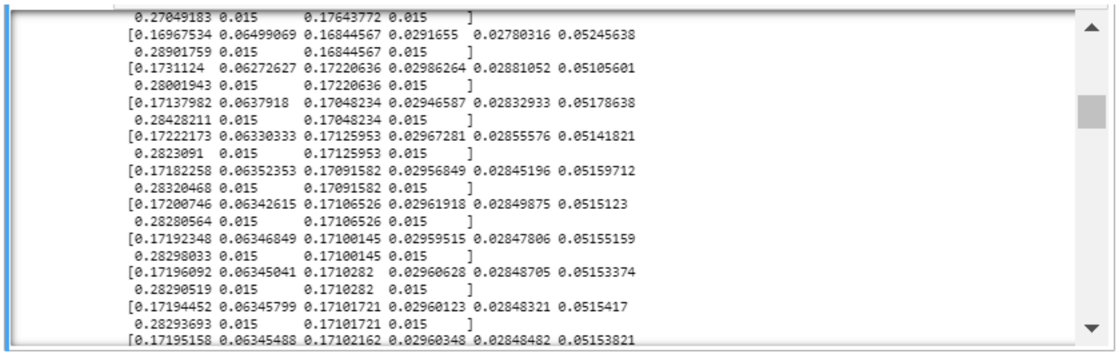
```python
#Let's consider v3(0) to have values [0.5 0 0 0 0 0 0 0 0 0.5]
#Given damping factor value as 0.85
damp_fac = 0.85
vec0 = np.empty([1,10])
vec0[0] = [0.5,0,0,0,0,0,0,0,0,0.5]

#Google matrix remains the same for this v3 as well.

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 25. Number of rows here denote the number of iterations that will be needed
#to find the final converged pagerank vector.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 25.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat3 = np.full((25,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector.
vec_mat3[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that conver
i=0                 #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0

#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat3[i+1,:] = calc_pagerank_vector(vec_mat3[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat3[i+1,:]-vec_mat3[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print("\nNo. of iterations to reach the converged pagerank vector")
print(i)

#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat3)
```



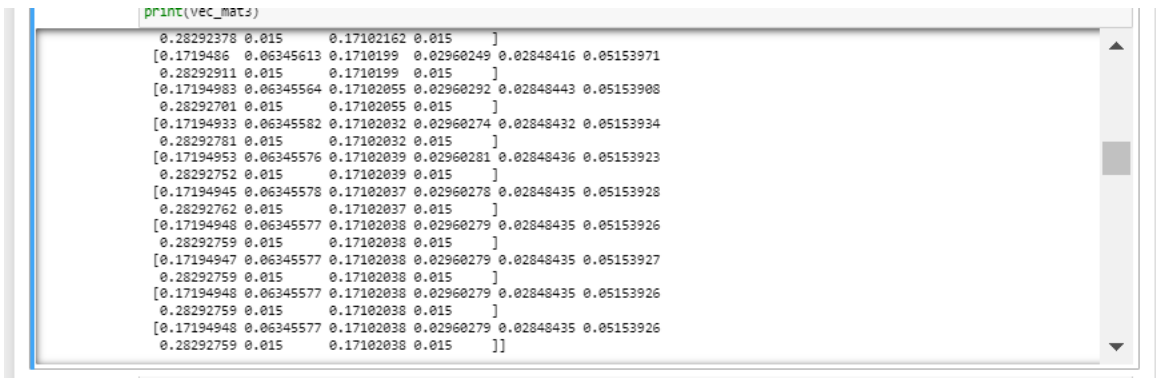
```
Google Matrix for damping factor=0.85
[[0.015      0.015      0.2275     0.015      0.015      0.2275
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.2275     0.015      0.2275     0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.29833333 0.29833333 0.015      0.015      0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.44       0.015      0.015      0.015
  0.015      0.015      0.44       0.015     ]
 [0.29833333 0.015      0.015      0.29833333 0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.015      0.015      0.015      0.015      0.015      0.015
  0.865      0.015      0.015      0.015     ]
```



```
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]]

No. of iterations to reach the converged pagerank vector
24

Pagerank vectors
[[0.25       0.         0.         0.25       0.         0.
  0.25       0.         0.         0.25      ]
 [0.19208333 0.015      0.24520833 0.015      0.015      0.068125
  0.174375   0.015      0.24520833 0.015     ]
 [0.16380903 0.08447569 0.12541146 0.03430208 0.0181875  0.05581771
  0.36258507 0.015      0.12541146 0.015     ]
 [0.18380739 0.05053325 0.19294932 0.03081502 0.03295109 0.04980942
  0.2361852  0.015      0.19294932 0.015     ]
 [0.16468647 0.06966897 0.15970643 0.02911267 0.02573831 0.05405907
  0.30732164 0.015      0.15970643 0.015     ]
 [0.1762653  0.06025015 0.17643772 0.03031674 0.02980466 0.04999588
  0.27049183 0.015      0.17643772 0.015     ]
```

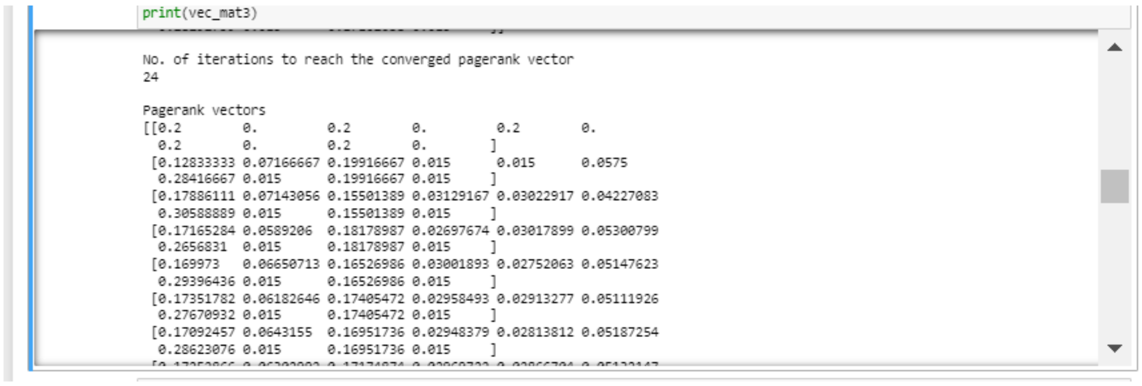


```
  0.27049183 0.015      0.17643772 0.015     ]
 [0.16967534 0.06499069 0.16844567 0.0291655  0.02780316 0.05245638
  0.28901759 0.015      0.16844567 0.015     ]
 [0.1731124  0.06272627 0.17220636 0.02986264 0.02881052 0.05105601
  0.28001943 0.015      0.17220636 0.015     ]
 [0.17137982 0.0637918  0.17048234 0.02946587 0.02832933 0.05178638
  0.28428211 0.015      0.17048234 0.015     ]
 [0.17222173 0.06330333 0.17125953 0.02967281 0.02855576 0.05141821
  0.2823091  0.015      0.17125953 0.015     ]
 [0.17182258 0.06352353 0.17091582 0.02956849 0.02845196 0.05159712
  0.28320468 0.015      0.17091582 0.015     ]
 [0.17200746 0.06342615 0.17106526 0.02961918 0.02849875 0.0515123
  0.28280564 0.015      0.17106526 0.015     ]
 [0.17192348 0.06346849 0.17100145 0.02959515 0.02847806 0.05155159
  0.28298033 0.015      0.17100145 0.015     ]
 [0.17196092 0.06345041 0.1710282  0.02960628 0.02848705 0.05153374
  0.28290519 0.015      0.1710282  0.015     ]
 [0.17194452 0.06345799 0.17101721 0.02960123 0.02848321 0.0515417
  0.28293693 0.015      0.17101721 0.015     ]
 [0.17195158 0.06345488 0.17102162 0.02960348 0.02848482 0.05153821
```



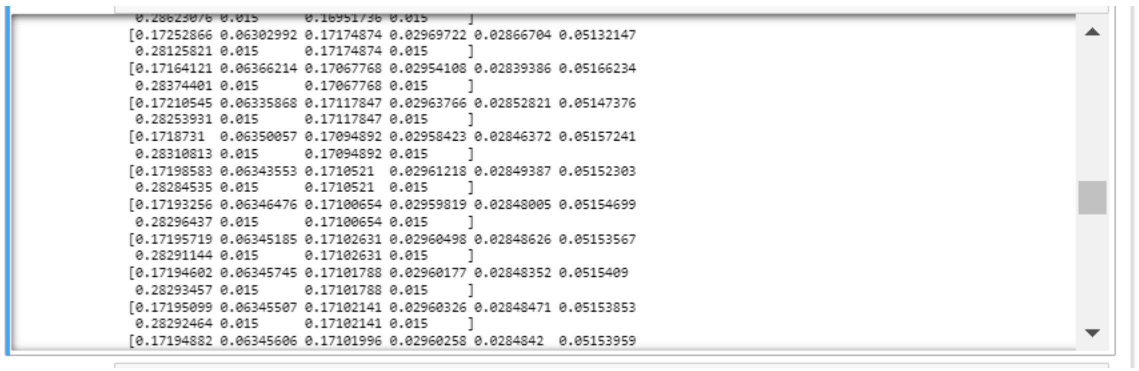
```
print(vec_mat3)
```

```
  0.28292378 0.015      0.17102162 0.015     ]
 [0.1719486  0.06345613 0.1710199  0.02960249 0.02848416 0.05153971
  0.28292911 0.015      0.1710199  0.015     ]
 [0.17194983 0.06345564 0.17102055 0.02960292 0.02848443 0.05153908
  0.28292701 0.015      0.17102055 0.015     ]
 [0.17194933 0.06345582 0.17102032 0.02960274 0.02848432 0.05153934
  0.28292781 0.015      0.17102032 0.015     ]
 [0.17194953 0.06345576 0.17102039 0.02960281 0.02848436 0.05153923
  0.28292752 0.015      0.17102039 0.015     ]
 [0.17194945 0.06345578 0.17102037 0.02960278 0.02848435 0.05153928
  0.28292762 0.015      0.17102037 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]
```

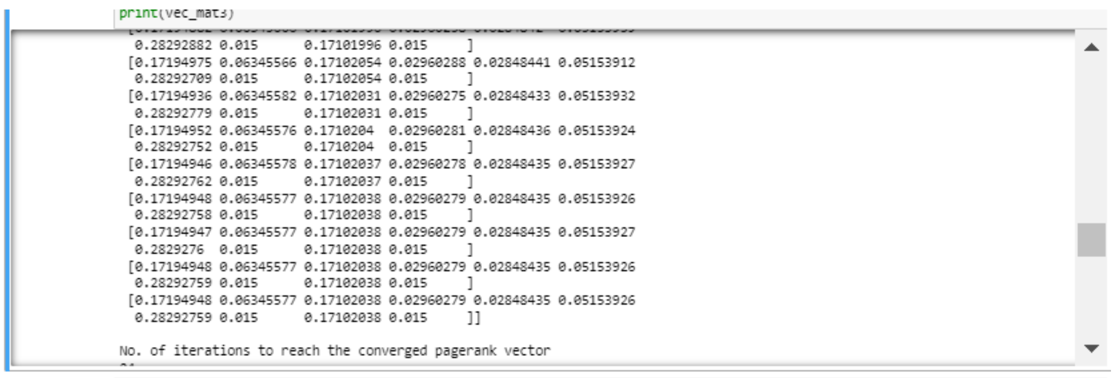


```
print(vec_mat3)
```

```
No. of iterations to reach the converged pagerank vector
24

Pagerank vectors
[[0.2        0.         0.2        0.         0.2        0.
  0.2        0.         0.2        0.        ]
 [0.12833333 0.07166667 0.19916667 0.015      0.015      0.0575
  0.28416667 0.015      0.19916667 0.015     ]
 [0.17886111 0.07143056 0.15501389 0.03129167 0.03022917 0.04227083
  0.30588889 0.015      0.15501389 0.015     ]
 [0.17165284 0.0589206  0.18178987 0.02697674 0.03017899 0.05300799
  0.2656831  0.015      0.18178987 0.015     ]
 [0.169973   0.06650713 0.16526986 0.03001893 0.02752063 0.05147623
  0.29396436 0.015      0.16526986 0.015     ]
 [0.17351782 0.06182646 0.17405472 0.02958493 0.02913277 0.05111926
  0.27670932 0.015      0.17405472 0.015     ]
 [0.17092457 0.0643155  0.16951736 0.02948379 0.02813812 0.05187254
  0.28623076 0.015      0.16951736 0.015     ]
```

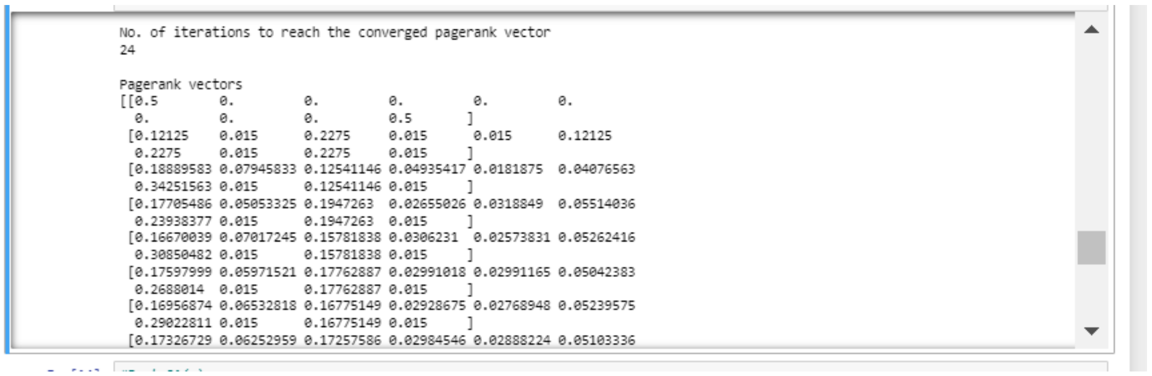


```
 [0.17252866 0.06302992 0.17174874 0.02969722 0.02866704 0.05132147
  0.28125821 0.015      0.17174874 0.015     ]
 [0.17164121 0.06366214 0.17067768 0.02954108 0.02839386 0.05166234
  0.28374401 0.015      0.17067768 0.015     ]
 [0.17210545 0.06335868 0.17117847 0.02963766 0.02852821 0.05147376
  0.28253931 0.015      0.17117847 0.015     ]
 [0.1718731  0.06350057 0.17094892 0.02958423 0.02846372 0.05157241
  0.28310813 0.015      0.17094892 0.015     ]
 [0.17198583 0.06343553 0.1710521  0.02961218 0.02849387 0.05152303
  0.28284535 0.015      0.1710521  0.015     ]
 [0.17193256 0.06346476 0.17100654 0.02959819 0.02848005 0.05154699
  0.28296437 0.015      0.17100654 0.015     ]
 [0.17195719 0.06345185 0.17102631 0.02960498 0.02848626 0.05153567
  0.28291144 0.015      0.17102631 0.015     ]
 [0.17194602 0.06345745 0.17101788 0.02960177 0.02848352 0.0515409
  0.28293457 0.015      0.17101788 0.015     ]
 [0.17195099 0.06345507 0.17102141 0.02960326 0.02848471 0.05153853
  0.28292464 0.015      0.17102141 0.015     ]
 [0.17194882 0.06345606 0.17101996 0.02960258 0.0284842  0.05153959
```

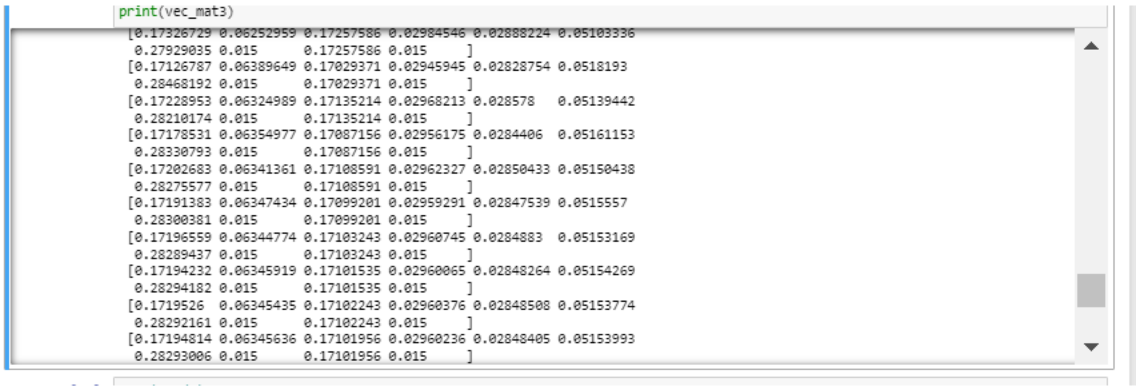


```
print(vec_mat3)
```

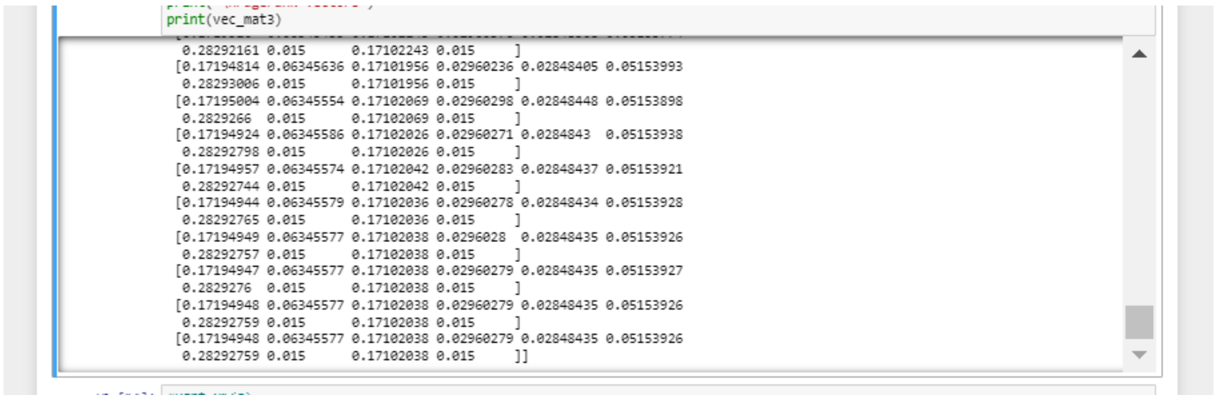
```
  0.28292882 0.015      0.17101996 0.015     ]
 [0.17194975 0.06345566 0.17102054 0.02960288 0.02848441 0.05153912
  0.28292709 0.015      0.17102054 0.015     ]
 [0.17194936 0.06345582 0.17102031 0.02960275 0.02848433 0.05153932
  0.28292779 0.015      0.17102031 0.015     ]
 [0.17194952 0.06345576 0.1710204  0.02960281 0.02848436 0.05153924
  0.28292752 0.015      0.1710204  0.015     ]
 [0.17194946 0.06345578 0.17102037 0.02960278 0.02848435 0.05153927
  0.28292762 0.015      0.17102037 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292758 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.2829276  0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]

No. of iterations to reach the converged pagerank vector
```



```
No. of iterations to reach the converged pagerank vector
24

Pagerank vectors
[[0.5        0.         0.         0.         0.         0.
  0.         0.         0.         0.5       ]
 [0.12125    0.015      0.2275     0.015      0.015      0.12125
  0.2275     0.015      0.2275     0.015     ]
 [0.18889583 0.07945833 0.12541146 0.04935417 0.0181875  0.04076563
  0.34251563 0.015      0.12541146 0.015     ]
 [0.17705486 0.05053325 0.1947263  0.02655026 0.0318849  0.05514036
  0.23938377 0.015      0.1947263  0.015     ]
 [0.16670039 0.07017245 0.15781838 0.0306231  0.02573831 0.05262416
  0.30850482 0.015      0.15781838 0.015     ]
 [0.17597999 0.05971521 0.17762887 0.02991018 0.02991165 0.05042383
  0.2688014  0.015      0.17762887 0.015     ]
 [0.16956874 0.06532818 0.16775149 0.02928675 0.02768948 0.05239575
  0.29022811 0.015      0.16775149 0.015     ]
 [0.17326729 0.06252959 0.17257586 0.02984546 0.02888224 0.05103336
```



```
print(vec_mat3)
 [0.17326729 0.06252959 0.17257586 0.02984546 0.02888224 0.05103336
  0.27929035 0.015      0.17257586 0.015     ]
 [0.17126787 0.06389649 0.17029371 0.02945945 0.02828754 0.0518193
  0.28468192 0.015      0.17029371 0.015     ]
 [0.17228953 0.06324989 0.17135214 0.02968213 0.028578   0.05139442
  0.28210174 0.015      0.17135214 0.015     ]
 [0.17178531 0.06354977 0.17087156 0.02956175 0.0284406  0.05161153
  0.28330793 0.015      0.17087156 0.015     ]
 [0.17202683 0.06341361 0.17108591 0.02962327 0.02850433 0.05150438
  0.28275577 0.015      0.17108591 0.015     ]
 [0.17191383 0.06347434 0.17099201 0.02959291 0.02847539 0.0515557
  0.28300381 0.015      0.17099201 0.015     ]
 [0.17196559 0.06344774 0.17103243 0.02960745 0.0284883  0.05153169
  0.28289437 0.015      0.17103243 0.015     ]
 [0.17194232 0.06345919 0.17101535 0.02960065 0.02848264 0.05154269
  0.28294182 0.015      0.17101535 0.015     ]
 [0.1719526  0.06345435 0.17102243 0.02960376 0.02848508 0.05153774
  0.28292161 0.015      0.17102243 0.015     ]
 [0.17194814 0.06345636 0.17101956 0.02960236 0.02848405 0.05153993
  0.28293006 0.015      0.17101956 0.015     ]
```



```
print(vec_mat3)
  0.28292161 0.015      0.17102243 0.015     ]
 [0.17194814 0.06345636 0.17101956 0.02960236 0.02848405 0.05153993
  0.28293006 0.015      0.17101956 0.015     ]
 [0.17195004 0.06345554 0.17102069 0.02960298 0.02848448 0.05153898
  0.2829266  0.015      0.17102069 0.015     ]
 [0.17194924 0.06345586 0.17102026 0.02960271 0.0284843  0.05153938
  0.28292798 0.015      0.17102026 0.015     ]
 [0.17194957 0.06345574 0.17102042 0.02960283 0.02848437 0.05153921
  0.28292744 0.015      0.17102042 0.015     ]
 [0.17194944 0.06345579 0.17102036 0.02960278 0.02848434 0.05153928
  0.28292765 0.015      0.17102036 0.015     ]
 [0.17194949 0.06345577 0.17102038 0.0296028  0.02848435 0.05153926
  0.28292757 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.2829276  0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]
```

Note:

For all 3 iterations using different initial value vectors, the same converged values were found (see above). They were:

[0.17194948, 0.06345577, 0.17102038, 0.02960279, 0.02848435, 0.05153926, 0.28292759, 0.015, 0.17102038, 0.015]

Discussion of Findings:

In terms of findings, the following should be noted.

1. The probability that a person will go to a webpage did not change, based on the initial vector. Conducting this algorithm using the initial conditions of vectors v1(0), v2(0), v3(0) and the initial condition of equal selection—see part D—always yielded the same probability results.
2. The number of iterations did fluctuate slightly. The initial condition vector using equal probabilities (.1 for every entry—see section D--), required 22 iterations (not including the initial condition, 23 with), while each of the three initial vectors used above required 24 iterations (not including the initial condition vector, 25 with).

Discussion:

To understand why the probabilities of visiting webpages always converge to the same values, it is important to understand the concept of a steady state Markov Chain vector. A steady state vector is the vector for which values converge as the number of iterations of the Markov chain goes towards infinity. As such a steady state vector is a vector that holds the probability values for which this Markov chain converges. This condition arises provided the stochastic matrix--one is multiplying the state vectors into--is regular, that is to say it holds all positive values. In addition, this steady state vector will arise regardless of the initial condition vector selected, though the number of iterations may vary.[11] The number of iterations to convergence may change because a steady state vector arises when the current state vector and the dominant eigenvector are the same, which based on the initial condition vector may result in the number of iterations being different.[12]

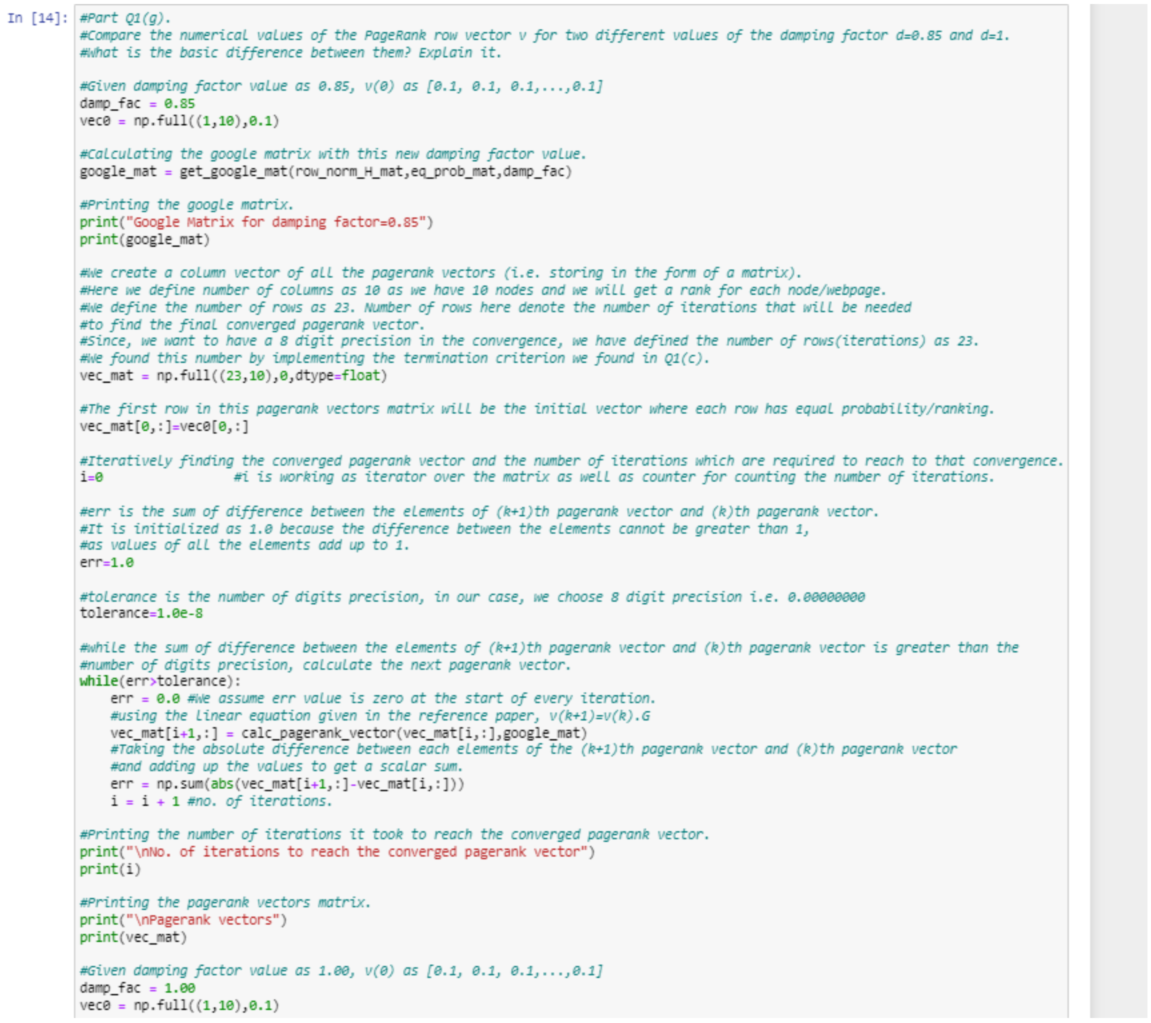
The Google Matrix used in this assignment is the stochastic matrix, and has values that are all positive, thus it is a regular matrix. This matrix guarantees that one will find a unique steady state vector regardless of initial conditions. Since each of the vectors' initial conditions were tested against the same Google Matrix it can be expected that the converged results will be the same though the number of iterations may be different. This outcome is exactly what was found.

(g) Compare the numerical values of the PageRank row vector v for two different values of the damping factor d=0.85 and d=1. What is the basic difference between them? Explain it.

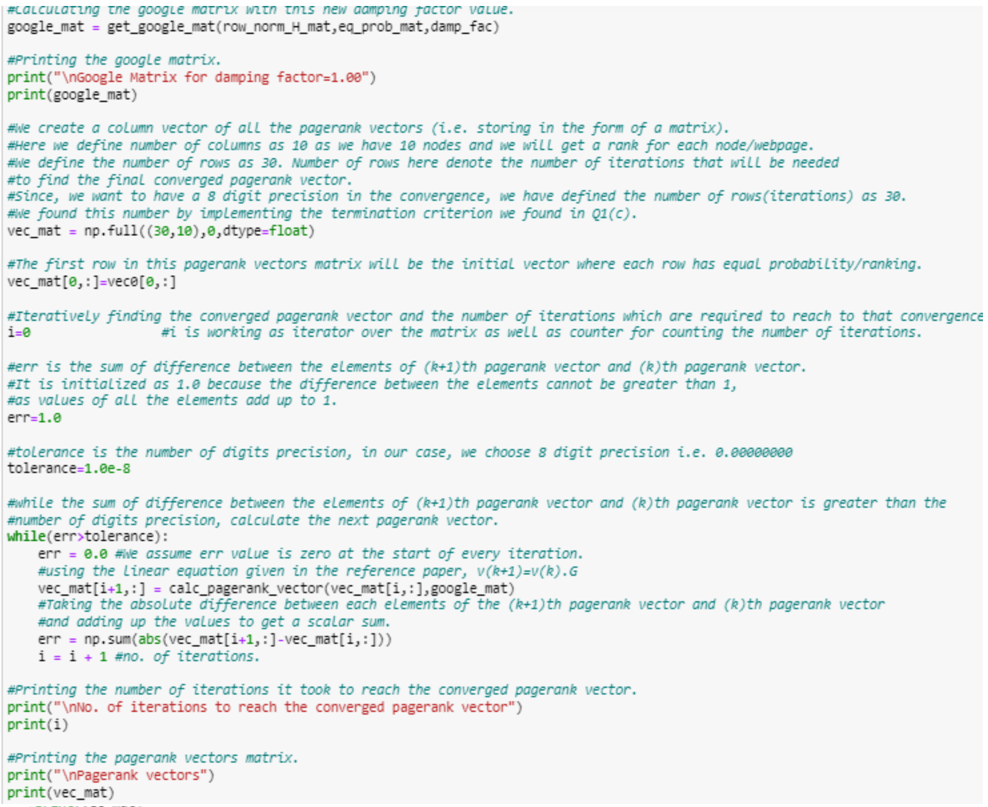
Below is the code and output of the PageRank, using the same initial value vector of equal probabilities (all entries are .1), but with differing damping factors (.85 and .1 were used). Below is the code and the outputs from this run of the algorithm:

[11] Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ, pp. 286-287.
[12] Ibid., 498.

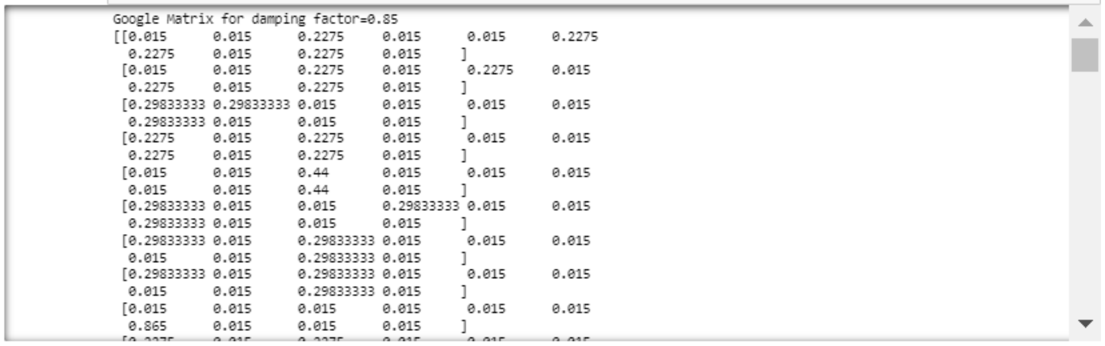
Damping Factor .85:



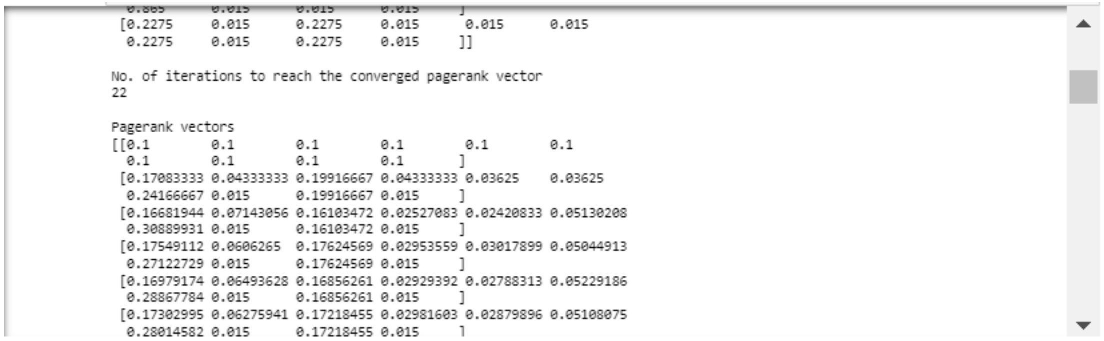
```
In [14]: #Part Q1(g).
         #Compare the numerical values of the PageRank row vector v for two different values of the damping factor d=0.85 and d=1.
         #What is the basic difference between them? Explain it.

         #Given damping factor value as 0.85, v(0) as [0.1, 0.1, 0.1,...,0.1]
         damp_fac = 0.85
         vec0 = np.full((1,10),0.1)

         #Calculating the google matrix with this new damping factor value.
         google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

         #Printing the google matrix.
         print("Google Matrix for damping factor=0.85")
         print(google_mat)

         #We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
         #Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
         #We define the number of rows as 23. Number of rows here denote the number of iterations that will be needed
         #to find the final converged pagerank vector.
         #Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 23.
         #We found this number by implementing the termination criterion we found in Q1(c).
         vec_mat = np.full((23,10),0,dtype=float)

         #The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
         vec_mat[0,:]=vec0[0,:]

         #Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergence.
         i=0                #i is working as iterator over the matrix as well as counter for counting the number of iterations.

         #err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
         #It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
         #as values of all the elements add up to 1.
         err=1.0

         #tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
         tolerance=1.0e-8

         #while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
         #number of digits precision, calculate the next pagerank vector.
         while(err>tolerance):
             err = 0.0 #We assume err value is zero at the start of every iteration.
             #using the linear equation given in the reference paper, v(k+1)=v(k).G
             vec_mat[i+1,:] = calc_pagerank_vector(vec_mat[i,:],google_mat)
             #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
             #and adding up the values to get a scalar sum.
             err = np.sum(abs(vec_mat[i+1,:]-vec_mat[i,:]))
             i = i + 1 #no. of iterations.

         #Printing the number of iterations it took to reach the converged pagerank vector.
         print("\nNo. of iterations to reach the converged pagerank vector")
         print(i)

         #Printing the pagerank vectors matrix.
         print("\nPagerank vectors")
         print(vec_mat)

         #Given damping factor value as 1.00, v(0) as [0.1, 0.1, 0.1,...,0.1]
         damp_fac = 1.00
         vec0 = np.full((1,10),0.1)
```



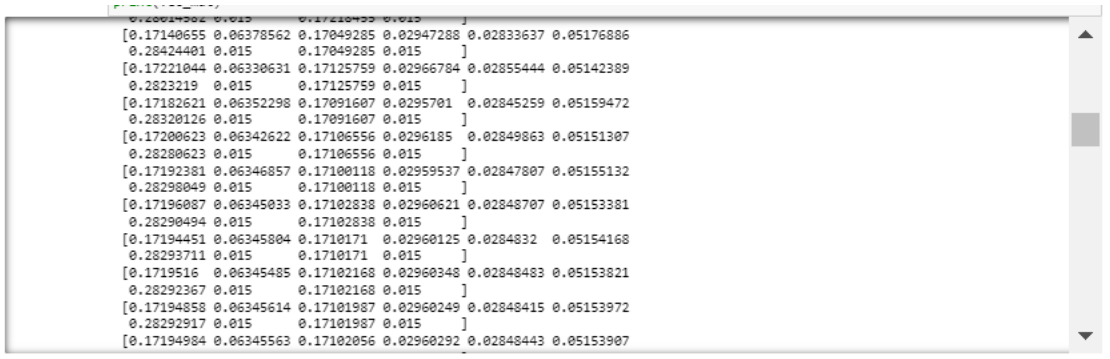
```
#calculating the google matrix with this new damping factor value.
google_mat = get_google_mat(row_norm_H_mat,eq_prob_mat,damp_fac)

#Printing the google matrix.
print("\nGoogle Matrix for damping factor=1.00")
print(google_mat)

#We create a column vector of all the pagerank vectors (i.e. storing in the form of a matrix).
#Here we define number of columns as 10 as we have 10 nodes and we will get a rank for each node/webpage.
#We define the number of rows as 30. Number of rows here denote the number of iterations that will be needed
#to find the final converged pagerank vector.
#Since, we want to have a 8 digit precision in the convergence, we have defined the number of rows(iterations) as 30.
#We found this number by implementing the termination criterion we found in Q1(c).
vec_mat = np.full((30,10),0,dtype=float)

#The first row in this pagerank vectors matrix will be the initial vector where each row has equal probability/ranking.
vec_mat[0,:]=vec0[0,:]

#Iteratively finding the converged pagerank vector and the number of iterations which are required to reach to that convergence
i=0                 #i is working as iterator over the matrix as well as counter for counting the number of iterations.

#err is the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector.
#It is initialized as 1.0 because the difference between the elements cannot be greater than 1,
#as values of all the elements add up to 1.
err=1.0

#tolerance is the number of digits precision, in our case, we choose 8 digit precision i.e. 0.00000000
tolerance=1.0e-8

#while the sum of difference between the elements of (k+1)th pagerank vector and (k)th pagerank vector is greater than the
#number of digits precision, calculate the next pagerank vector.
while(err>tolerance):
    err = 0.0 #We assume err value is zero at the start of every iteration.
    #using the linear equation given in the reference paper, v(k+1)=v(k).G
    vec_mat[i+1,:] = calc_pagerank_vector(vec_mat[i,:],google_mat)
    #Taking the absolute difference between each elements of the (k+1)th pagerank vector and (k)th pagerank vector
    #and adding up the values to get a scalar sum.
    err = np.sum(abs(vec_mat[i+1,:]-vec_mat[i,:]))
    i = i + 1 #no. of iterations.

#Printing the number of iterations it took to reach the converged pagerank vector.
print("\nNo. of iterations to reach the converged pagerank vector")
print(i)

#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat)
```



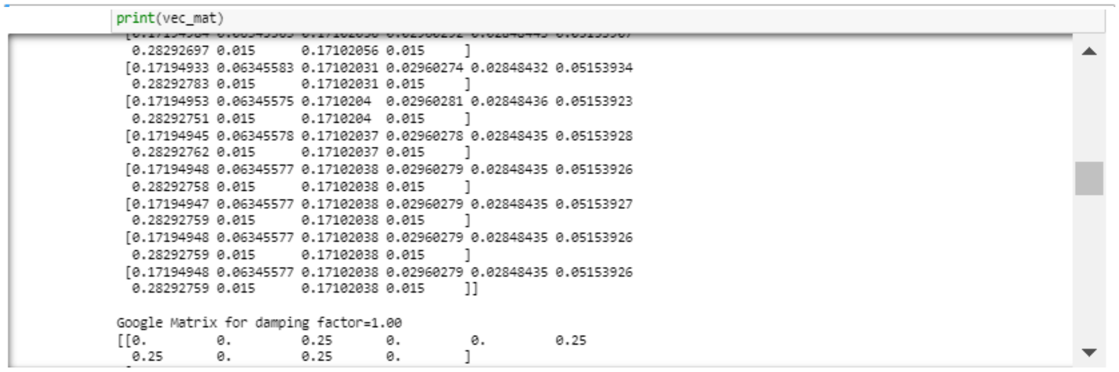
```
Google Matrix for damping factor=0.85
[[0.015      0.015      0.2275     0.015      0.015      0.2275
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.2275     0.015      0.2275     0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.29833333 0.29833333 0.015      0.015      0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]
 [0.015      0.015      0.44       0.015      0.015      0.015
  0.015      0.015      0.44       0.015     ]
 [0.29833333 0.015      0.015      0.29833333 0.015      0.015
  0.29833333 0.015      0.015      0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.29833333 0.015      0.29833333 0.015      0.015      0.015
  0.015      0.015      0.29833333 0.015     ]
 [0.015      0.015      0.015      0.015      0.015      0.015
  0.865      0.015      0.015      0.015     ]
```



```
  0.865      0.015      0.015      0.015     ]
 [0.2275     0.015      0.2275     0.015      0.015      0.015
  0.2275     0.015      0.2275     0.015     ]]

No. of iterations to reach the converged pagerank vector
22

Pagerank vectors
[[0.1        0.1        0.1        0.1        0.1        0.1
  0.1        0.1        0.1        0.1       ]
 [0.17083333 0.04333333 0.19916667 0.04333333 0.03625    0.03625
  0.24166667 0.015      0.19916667 0.015     ]
 [0.16681944 0.07143056 0.16103472 0.02527083 0.02420833 0.05130208
  0.30889931 0.015      0.16103472 0.015     ]
 [0.17549112 0.0606265  0.17624569 0.02953559 0.03017899 0.05044913
  0.27122729 0.015      0.17624569 0.015     ]
 [0.16979174 0.06493628 0.16856261 0.02929392 0.02788313 0.05229186
  0.28867784 0.015      0.16856261 0.015     ]
 [0.17302995 0.06275941 0.17218455 0.02981603 0.02879896 0.05108075
  0.28014582 0.015      0.17218455 0.015     ]
```



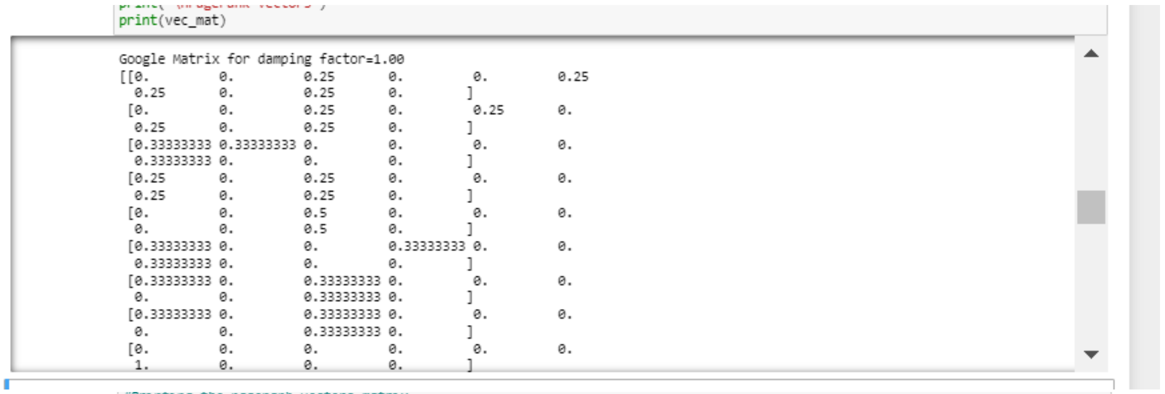
```
 [0.17140655 0.06378562 0.17049285 0.02947288 0.02833637 0.05176886
  0.28424401 0.015      0.17049285 0.015     ]
 [0.17221044 0.06330631 0.17125759 0.02966784 0.02855444 0.05142389
  0.2823219  0.015      0.17125759 0.015     ]
 [0.17182621 0.06352298 0.17091607 0.0295701  0.02845259 0.05159472
  0.28320126 0.015      0.17091607 0.015     ]
 [0.17200623 0.06342622 0.17106556 0.0296185  0.02849863 0.05151307
  0.28280623 0.015      0.17106556 0.015     ]
 [0.17192381 0.06346857 0.17100118 0.02959537 0.02847807 0.05155132
  0.28298049 0.015      0.17100118 0.015     ]
 [0.17196087 0.06345033 0.17102838 0.02960621 0.02848707 0.05153381
  0.28290494 0.015      0.17102838 0.015     ]
 [0.17194451 0.06345804 0.1710171  0.02960125 0.0284832  0.05154168
  0.28293711 0.015      0.1710171  0.015     ]
 [0.1719516  0.06345485 0.17102168 0.02960348 0.02848483 0.05153821
  0.28292367 0.015      0.17102168 0.015     ]
 [0.17194858 0.06345614 0.17101987 0.02960249 0.02848415 0.05153972
  0.28292917 0.015      0.17101987 0.015     ]
 [0.17194984 0.06345563 0.17102056 0.02960292 0.02848443 0.05153907
```



```
print(vec_mat)
```

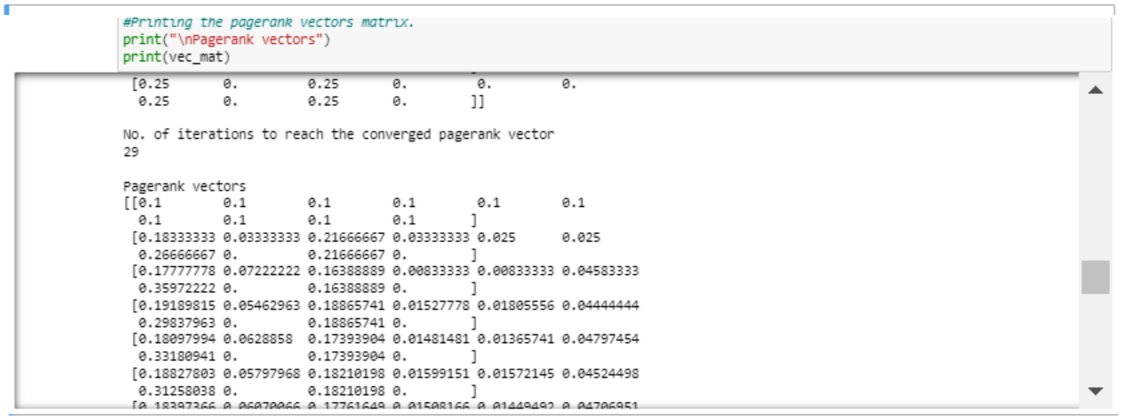
```
  0.28292697 0.015      0.17102056 0.015     ]
 [0.17194933 0.06345583 0.17102031 0.02960274 0.02848432 0.05153934
  0.28292783 0.015      0.17102031 0.015     ]
 [0.17194953 0.06345575 0.1710204  0.02960281 0.02848436 0.05153923
  0.28292751 0.015      0.1710204  0.015     ]
 [0.17194945 0.06345578 0.17102037 0.02960278 0.02848435 0.05153928
  0.28292762 0.015      0.17102037 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292758 0.015      0.17102038 0.015     ]
 [0.17194947 0.06345577 0.17102038 0.02960279 0.02848435 0.05153927
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]
 [0.17194948 0.06345577 0.17102038 0.02960279 0.02848435 0.05153926
  0.28292759 0.015      0.17102038 0.015     ]]

Google Matrix for damping factor=1.00
[[0.         0.         0.25       0.         0.         0.25
  0.25       0.         0.25       0.        ]
```

Damping Factor 1:



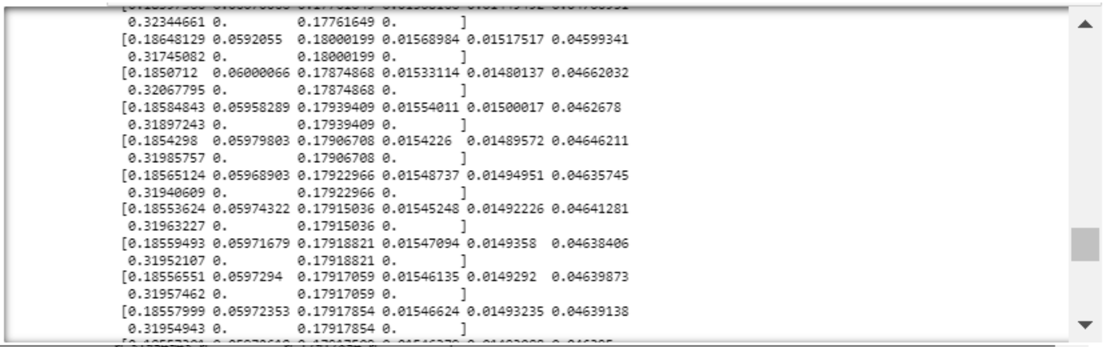
```
print(vec_mat)
```

```
Google Matrix for damping factor=1.00
[[0.         0.         0.25       0.         0.         0.25
  0.25       0.         0.25       0.        ]
 [0.         0.         0.25       0.         0.25       0.
  0.25       0.         0.25       0.        ]
 [0.33333333 0.33333333 0.         0.         0.         0.
  0.33333333 0.         0.         0.        ]
 [0.25       0.         0.25       0.         0.         0.
  0.25       0.         0.25       0.        ]
 [0.         0.         0.5        0.         0.         0.
  0.         0.         0.5        0.        ]
 [0.33333333 0.         0.         0.33333333 0.         0.
  0.33333333 0.         0.         0.        ]
 [0.33333333 0.         0.33333333 0.         0.         0.
  0.         0.         0.33333333 0.        ]
 [0.33333333 0.         0.33333333 0.         0.         0.
  0.         0.         0.33333333 0.        ]
 [0.         0.         0.         0.         0.         0.
  1.         0.         0.         0.        ]
```

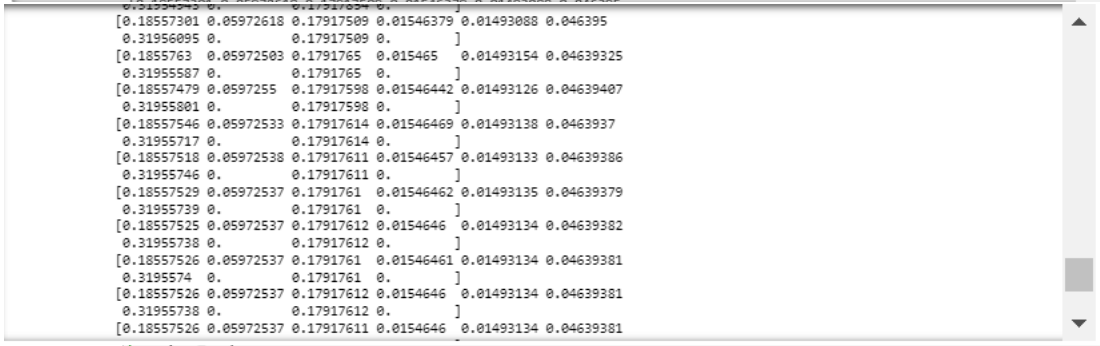


```
#Printing the pagerank vectors matrix.
print("\nPagerank vectors")
print(vec_mat)
```

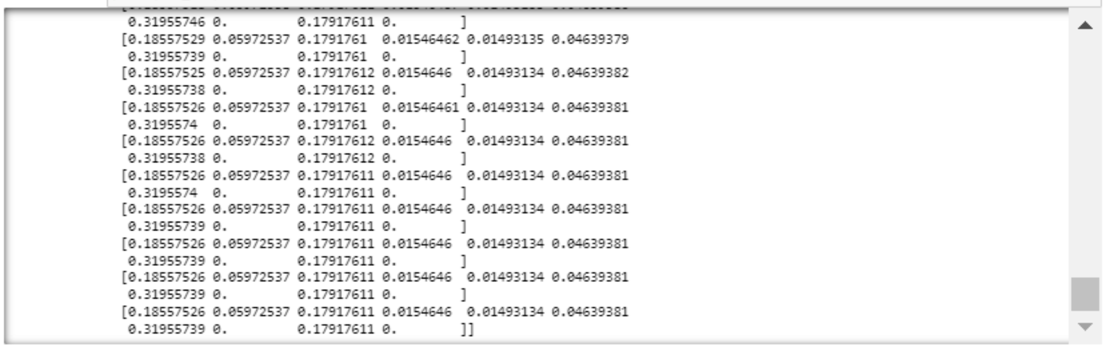
```
 [0.25       0.         0.25       0.         0.         0.
  0.25       0.         0.25       0.        ]]

No. of iterations to reach the converged pagerank vector
29

Pagerank vectors
[[0.1        0.1        0.1        0.1        0.1        0.1
  0.1        0.1        0.1        0.1       ]
 [0.18333333 0.03333333 0.21666667 0.03333333 0.025      0.025
  0.26666667 0.         0.21666667 0.        ]
 [0.17777778 0.07222222 0.16388889 0.00833333 0.00833333 0.04583333
  0.35972222 0.         0.16388889 0.        ]
 [0.19189815 0.05462963 0.18865741 0.01527778 0.01805556 0.04444444
  0.29837963 0.         0.18865741 0.        ]
 [0.18097994 0.0628858  0.17393904 0.01481481 0.01365741 0.04797454
  0.33180941 0.         0.17393904 0.        ]
 [0.18827803 0.05797968 0.18210198 0.01599151 0.01572145 0.04524498
  0.31258038 0.         0.18210198 0.        ]
 [0.18397366 0.06070066 0.17761649 0.01508166 0.01449492 0.04706951
```



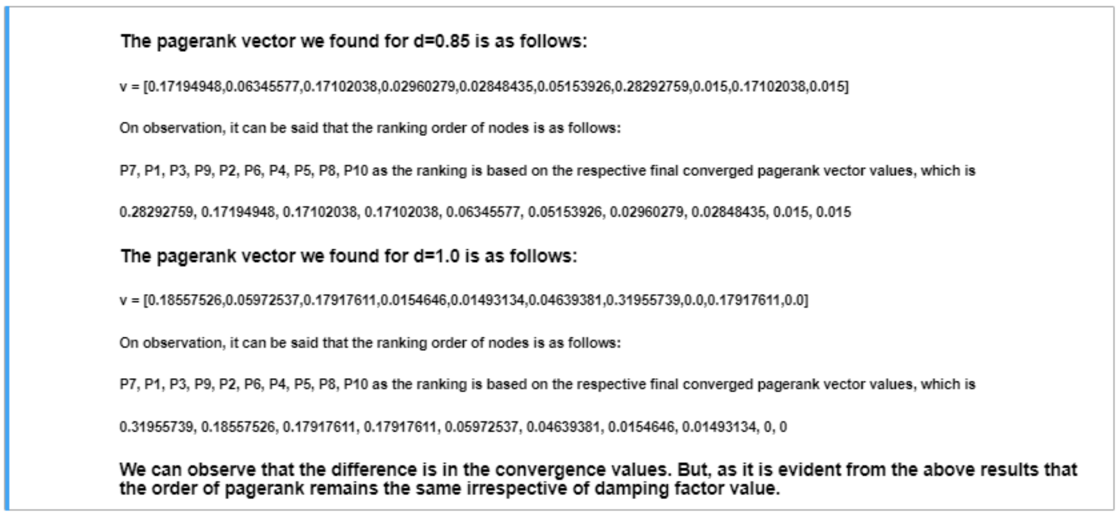
```
  0.32344661 0.         0.17761649 0.         ]
 [0.18648129 0.0592055  0.18000199 0.01568984 0.01517517 0.04599341
  0.31745082 0.         0.18000199 0.         ]
 [0.1850712  0.06000066 0.17874868 0.01533114 0.01480137 0.04662032
  0.32067795 0.         0.17874868 0.         ]
 [0.18584843 0.05958289 0.17939409 0.01554011 0.01500017 0.0462678
  0.31897243 0.         0.17939409 0.         ]
 [0.1854298  0.05979803 0.17906708 0.0154226  0.01489572 0.04646211
  0.31985757 0.         0.17906708 0.         ]
 [0.18565124 0.05968903 0.17922966 0.01548737 0.01494951 0.04635745
  0.31940609 0.         0.17922966 0.         ]
 [0.18553624 0.05974322 0.17915036 0.01545248 0.01492226 0.04641281
  0.31963227 0.         0.17915036 0.         ]
 [0.18559493 0.05971679 0.17918821 0.01547094 0.0149358  0.04638406
  0.31952107 0.         0.17918821 0.         ]
 [0.18556551 0.0597294  0.17917059 0.01546135 0.0149292  0.04639873
  0.31957462 0.         0.17917059 0.         ]
 [0.18557999 0.05972353 0.17917854 0.01546624 0.01493235 0.04639138
  0.31954943 0.         0.17917854 0.         ]
```



```
 [0.18557301 0.05972618 0.17917509 0.01546379 0.01493088 0.046395
  0.31956095 0.         0.17917509 0.         ]
 [0.1855763  0.05972503 0.1791765  0.015465   0.01493154 0.04639325
  0.31955587 0.         0.1791765  0.         ]
 [0.18557479 0.0597255  0.17917598 0.01546442 0.01493126 0.04639407
  0.31955801 0.         0.17917598 0.         ]
 [0.18557546 0.05972533 0.17917614 0.01546469 0.01493138 0.0463937
  0.31955717 0.         0.17917614 0.         ]
 [0.18557518 0.05972538 0.17917611 0.01546457 0.01493133 0.04639386
  0.31955746 0.         0.17917611 0.         ]
 [0.18557529 0.05972537 0.1791761  0.01546462 0.01493135 0.04639379
  0.31955739 0.         0.1791761  0.         ]
 [0.18557525 0.05972537 0.17917612 0.0154646  0.01493134 0.04639382
  0.31955738 0.         0.17917612 0.         ]
 [0.18557526 0.05972537 0.1791761  0.01546461 0.01493134 0.04639381
  0.3195574  0.         0.1791761  0.         ]
 [0.18557526 0.05972537 0.17917612 0.0154646  0.01493134 0.04639381
  0.31955738 0.         0.17917612 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
```



```
  0.31955746 0.         0.17917611 0.         ]
 [0.18557529 0.05972537 0.1791761  0.01546462 0.01493135 0.04639379
  0.31955739 0.         0.1791761  0.         ]
 [0.18557525 0.05972537 0.17917612 0.0154646  0.01493134 0.04639382
  0.31955738 0.         0.17917612 0.         ]
 [0.18557526 0.05972537 0.1791761  0.01546461 0.01493134 0.04639381
  0.3195574  0.         0.1791761  0.         ]
 [0.18557526 0.05972537 0.17917612 0.0154646  0.01493134 0.04639381
  0.31955738 0.         0.17917612 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
  0.3195574  0.         0.17917611 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
  0.31955739 0.         0.17917611 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
  0.31955739 0.         0.17917611 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
  0.31955739 0.         0.17917611 0.         ]
 [0.18557526 0.05972537 0.17917611 0.0154646  0.01493134 0.04639381
  0.31955739 0.         0.17917611 0.         ]]
```



The pagerank vector we found for d=0.85 is as follows:

v = [0.17194948,0.06345577,0.17102038,0.02960279,0.02848435,0.05153926,0.28292759,0.015,0.17102038,0.015]

On observation, it can be said that the ranking order of nodes is as follows:

P7, P1, P3, P9, P2, P6, P4, P5, P8, P10 as the ranking is based on the respective final converged pagerank vector values, which is

0.28292759, 0.17194948, 0.17102038, 0.17102038, 0.06345577, 0.05153926, 0.02960279, 0.02848435, 0.015, 0.015

The pagerank vector we found for d=1.0 is as follows:

v = [0.18557526,0.05972537,0.17917611,0.0154646,0.01493134,0.04639381,0.31955739,0.0,0.17917611,0.0]

On observation, it can be said that the ranking order of nodes is as follows:

P7, P1, P3, P9, P2, P6, P4, P5, P8, P10 as the ranking is based on the respective final converged pagerank vector values, which is

0.31955739, 0.18557526, 0.17917611, 0.17917611, 0.05972537, 0.04639381, 0.0154646, 0.01493134, 0, 0

We can observe that the difference is in the convergence values. But, as it is evident from the above results that the order of pagerank remains the same irrespective of damping factor value.

Analysis of Findings:

The following differences and similarities between the two runs of PageRank (using damping factor .85 and damping factor 1) are:

1. The PageRank rank of pages' importance—ranked from greatest probability to least—did not change based on the damping factor.
2. The number of iterations increased from 22 to 29 (excluding the initial condition vector, 23 to 30 with).
3. However, probabilities did change. Namely, nodes representing the webpages for 8 and 10's probabilities were both reduced to 0, while the probabilities for webpages P7, P1, P3, and P9 increased slightly, while the probabilities for webpages P2, P6, P4, and P5 decreased slightly.

Comments:

The slight changes in probabilities of visiting a webpage—see bullet 2—was due to increasing the damping factor to 1. As noted in the section c of this report, the damping factor is the proportion of time a web user visits various webpages using only weblinks, while its complement (1- damping factor), is the proportion of time a user visits any page not using direct weblinks. As such, if the damping factor is set to 1, then a web user can only visit webpages using weblinks. Looking at the graph of weblinks and nodes, along with the H matrix (both presented in section a), it should be noted that webpages 8 and 10 only have outbound links. As such, when the damping factor is 1, unless a user starts their web-browsing on one of those pages, it will become impossible for them to visit Page 8 or 10, given that these webpages have no inbound links.

Division of work:

1. Matt and Bhavesh worked on figuring out the terminating condition in part (c).
2. Bhavesh programmed the python script for this HW.
3. Girish and Muhammed helped in writing the initial draft of the report.
4. Matt wrote the finalized version of the report.
5. Everyone participated in the HW, and it was a team effort.

References:

Anton, H Rorres C (2010), Elementary Linear Algebra: Applications Version, Tenth Edition, Wiley: Hoboken, NJ

Davey, B Pitkethly, J (2013), "Math Delivers! Google PageRank", Australian Mathematical Science Institute, Available at: https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Pagerank5.pdf

Larson, R (2013), Elementary Linear Algebra, Seventh Edition, Brooks/Cole, Cengage: Boston