

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE



Linear Algebra (MT1007) - Project 12

Matrix eigenvalues and the Google's PageRank Algorithm

Lecturer: Đậu Thế Phiệt
Group 7: Nguyễn Trọng Cường - 1951124
Nguyễn Bình Nguyên - 2252545
Trần Anh Tuấn - 2252877
Phạm Nguyễn Hoàng Yến - 2353368
Nguyễn Thanh Thảo - 2353114
Vũ Song Anh - 2252048
Nguyễn Trịnh Bảo Trung - 2252857

HO CHI MINH CITY, AUGUST 17



Contents

1	Introduction	2
2	Theory	2
2.1	Background on Markov chain	2
2.1.1	Definition and basic concepts	2
2.1.2	Application in Web Navigation	3
2.2	Theoretical Basis of PageRank	3
2.2.1	Matrix representation	3
2.2.2	Transition Matrix (S Matrix)	3
2.2.3	The Google Matrix	4
3	MATLAB code, results and explanations	4
3.1	Workspace	4
3.2	MATLAB code	5
3.3	Explanations and results	7
4	A small example	14

1 Introduction

The PageRank algorithm is a groundbreaking link analysis algorithm developed at Stanford University by Larry Page, after whom the algorithm is named, and Sergey Brin. Initially conceived in 1996 as part of a research project, the algorithm laid the foundation for what would later become Google, officially launched in 1998. PageRank was inspired by the work of Eugene Garfield on Citation Analysis from the 1950s, which focused on the importance of academic papers based on the frequency of their citations. Similarly, PageRank evaluates the importance of web pages based on the quantity and quality of hyperlinks pointing to them.

At its core, PageRank is built on the principles of graph theory and linear algebra, particularly utilizing Markov chains and eigenvalues to rank web pages. Each web page is represented as a node in a directed graph, with hyperlinks between pages forming the edges. The algorithm assigns a numerical value, the PageRank, to each web page, which reflects its relative importance within the network. This ranking is not just a simple count of inbound links but also considers the rank of the linking pages, making it a recursive and iterative process.

PageRank revolutionized web search by introducing a method to prioritize search results based on the structure of the web, rather than just keyword matching. This approach helped Google differentiate itself from other search engines of the time, offering more relevant and authoritative search results. While PageRank is only one of many factors that determine search engine rankings today, it remains a fundamental concept in understanding how modern search engines function.

The algorithm's reliance on the mathematical concept of eigenvalues and eigenvectors makes it particularly robust. The PageRank value of a page is derived from solving a system of linear equations, where the solution corresponds to the principal eigenvector of the stochastic matrix representing the web graph. The stability of this solution, known as the equilibrium distribution, ensures that the PageRank values converge to a unique set of scores, regardless of the initial distribution.

This report delves into the theoretical foundations of PageRank, exploring the role of Markov chains in modeling web navigation, the mathematical formulation of the PageRank algorithm, and the practical implications of its application. By understanding the underpinnings of PageRank, we can gain insight into the broader field of network analysis and its applications in various domains, including social networks, recommendation systems, and more.

2 Theory

2.1 Background on Markov chain

2.1.1 Definition and basic concepts

Markov Chains are mathematical systems that undergo transitions from one state to another within a finite set of states. The key characteristic of a Markov Chain is that the probability of transitioning to the next state depends only on the current state and not on the sequence of events that preceded it. This property is known as the "memoryless" property or the Markov property.

- **State Space:** In a Markov Chain, the set of all possible states is called the state space. Each state represents a possible condition or situation the system can be in. For example, in a simple board game, the squares on the board can be considered the state space.
- **Transition Probabilities:** The movement from one state to another is governed by transition probabilities. These probabilities indicate the likelihood of transitioning from one state to another within the system. For example, if we consider a weather system with states like "Sunny," "Rainy," and

"Cloudy," the transition probabilities would tell us the likelihood of moving from "Sunny" to "Rainy" or from "Rainy" to "Cloudy" on any given day.

2.1.2 Application in Web Navigation

In the context of web navigation, Markov Chains can be used to model the behavior of users as they move from one web page to another. Here, each web page represents a state, and the hyperlinks between pages represent the transition probabilities. The probability that a user moves from one page to another depends on the number of links on the current page and how likely the user is to click on a specific link.

- **Equilibrium Distribution:** Over time, as users continue to navigate the web, the system may reach an equilibrium distribution, where the probability of being on any particular page stabilizes. This equilibrium distribution is crucial in the PageRank algorithm, as it represents the steady-state probabilities that determine the rank of each page.

The concept of equilibrium distribution in Markov Chains is directly related to how PageRank is calculated. The PageRank of a web page is analogous to the long-term probability of a Markov Chain being in the state corresponding to that page. By using the transition matrix, which includes the link structure of the web, the PageRank algorithm determines the equilibrium distribution, providing a ranking of web pages based on their importance.

2.2 Theoretical Basis of PageRank

2.2.1 Matrix representation

The PageRank algorithm is rooted in linear algebra, where matrices play a central role in representing and analyzing the web's structure. Each web page is treated as a node in a directed graph, and the hyperlinks between them form the edges. This graph can be represented by an adjacency matrix, which is a square matrix used to indicate whether pairs of nodes are adjacent (i.e., directly connected).

- **Adjacency Matrix:** For a web of N page, the adjacency matrix A is an $N \times N$ matrix where the entry $A[i][j]$ is 1 if there is a link from page i to page j , and 0 otherwise. This matrix helps us understand the direct connections between pages, but it does not provide information about the probability of moving between them, which is where the transition matrix comes in.

2.2.2 Transition Matrix (S Matrix)

Construction of the S matrix

The transition matrix, often denoted as S , extends the concept of the adjacency matrix by incorporating probabilities. Each entry $S[i][j]$ in this matrix represents the probability of moving from page i to page j by following a hyperlink. The transition probability is calculated by dividing 1 by the number of outgoing links from page i . If page i has k outgoing links, then each link has a probability of $1/k$ to be followed. Thus, if there is a link from page i to page j , then $S[i][j] = 1/k$; otherwise, $S[i][j] = 0$.

Challenges with the Transition Matrix

While the transition matrix is a powerful tool, it presents some challenges in practice:

- **Dangling nodes:** Some pages, known as dangling nodes, have no outbound links. These pages correspond to rows in the transition matrix that consist entirely of zeros, which can disrupt the PageRank calculations because they fail to distribute any rank to other pages.

- **Disconnected components:** The web is not always fully connected; there may be groups of pages that are isolated from each other. In such cases, the transition matrix may consist of multiple disconnected components, which can lead to rank calculation issues where certain pages are not reachable from others.

2.2.3 The Google Matrix

To address the challenges associated with the transition matrix, the PageRank algorithm introduces the Google matrix, denoted as G . This matrix is a modified version of the transition matrix S and is designed to ensure that the rank calculation process is both stable and accurate, even in the presence of dangling nodes and disconnected components. A critical component of the Google matrix is the **Damping factor**, typically set to $\alpha = 0.85$. The damping factor represents the probability that a user will continue following links on a web page, as opposed to jumping to a random page. This factor ensures that the PageRank algorithm accounts for both the link structure and the possibility of random jumps across the web.

Mathematical Formation of the Google Matrix

The Google matrix G is constructed by combining the transition matrix S with a matrix E , where E is a matrix with all elements equal to $1/N$, where N is the total number of pages. The Google matrix is defined as:

$$G = \alpha S + (1 - \alpha)E$$

Here, αS represents the part of the rank distribution that comes from following links, while $(1 - \alpha)E$ accounts for the random jumps to any page in the web.

3 MATLAB code, results and explanations

For this part, we will implement the PageRank Algorithm on the given matrix provided in the file "AdjMatrix.mat" by using MATLAB programming platform

3.1 Workspace

Workspace			
Name	Value	Size	Class
AdjMatrix	8297x8297 sparse double	8297x8297	double (sparse)
AdjMatrixSmall	500x500 sparse double	500x500	double (sparse)
GoogleMatrix	500x500 double	500x500	double
MaxLinks	24.8783	1x1	double
MaxRank	0.0333	1x1	double
MostLinks	1x500 double	1x500	double
NumLinks	500x1 sparse double	500x1	double (sparse)
NumNetwork	500	1x1	double
PageMaxLinks	28	1x1	double
PageMaxRank	28	1x1	double
SumGoogleMatrix	500x1 double	500x1	double
alpha	0.1500	1x1	double
coordinates	500x2 double	500x2	double
deltaw	1x500 double	1x500	double
entriesSum	1.0000	1x1	double
i	500	1x1	double
idx	1	1x1	double
j	500	1x1	double

left_eigenvalue	500x1 complex double	500x1	double (complex)
left_eigenvector	500x1 double	500x1	double
m	8297	1x1	double
n	8297	1x1	double
nnzAdjMatrix	0.1506	1x1	double
right_eigenvalue	500x1 complex double	500x1	double (complex)
right_eigenvector	500x1 double	500x1	double
tolerance	1.0000e-10	1x1	double
u1	500x1 double	500x1	double
w0	1x500 double	1x500	double
w1	1x500 double	1x500	double
w10	1x500 double	1x500	double
w2	1x500 double	1x500	double
w3	1x500 double	1x500	double
w5	1x500 double	1x500	double

3.2 MATLAB code

```

1      % Google PageRank algorithm on the example of random
      network
2
3  %% Load the network data
4  load("AdjMatrix.mat");
5  %% Check ratio of non-zero elements
6  nnzAdjMatrix = (nnz(AdjMatrix) / numel(AdjMatrix)) * 100;
7  %% Dimensions of the matrix AdjMatrix
8  [m,n] = size(AdjMatrix);
9  %% Display a small amount of network
10 NumNetwork=500;
11 AdjMatrixSmall=AdjMatrix(1:NumNetwork,1:NumNetwork);
12
13 for j=1:NumNetwork
14     coordinates(j,1)=NumNetwork*rand;
15     coordinates(j,2)=NumNetwork*rand;
16 end;
17
18 gplot(AdjMatrixSmall,coordinates,'k-*');
19
20 %% Check the amount of links originating from each webpage
21 NumLinks=sum(AdjMatrixSmall,2);
22
23 %% Create a matrix of probabilities (Google matrix)
24 % Element (i,j) of the matrix shows the probability of moving from
    i-th
25 % page of the network to jth page. It is assumed that the user can
    follow
26 % any link on the page with a total probability of 85% (all
    hyperlinks are
27 % equal), and jump (teleport) to any other page in the network
    with a total
28 % probability of 15% (again, all pages are equal).
29

```

```
30 alpha=0.15;
31 GoogleMatrix=zeros(NumNetwork,NumNetwork);
32 for i=1:NumNetwork
33     if NumLinks(i)~=0
34         GoogleMatrix(i,:)=AdjMatrixSmall(i,:)./NumLinks(i);
35     else
36         GoogleMatrix(i,:)=1./NumNetwork;
37     end;
38 end;
39
40 GoogleMatrix=(1-alpha)*GoogleMatrix+alpha*ones(NumNetwork,
    NumNetwork)./NumNetwork;
41
42 %% Check that all the rows in the GoogleMatrix matrix sum to 1
43 SumGoogleMatrix=sum(GoogleMatrix,2);
44
45 %% Finding an eigenvector corresponding to 1 (why is there such an
    eigenvector)?
46 w0=ones(1,NumNetwork)./NumNetwork;
47 w1=w0*GoogleMatrix;
48 w2=w1*GoogleMatrix;
49 w3=w2*GoogleMatrix;
50 w5=w0*(GoogleMatrix^5);
51 w10=w0*(GoogleMatrix^10);
52
53 % Check the difference between v30 and v20. Observe that it is
    pretty
54 % small
55 deltaw=w10-w5;
56 %% Compute the eigenvalues and the right eigenvectors
57 [right_eigenvector, right_eigenvalue] = eig(GoogleMatrix);
58 % Explain the result
59 right_eigenvalue = diag(right_eigenvalue);
60 %%find eigenvalue 1
61 tolerance = 1e-10;
62 idx = find(abs(real(right_eigenvalue) - 1) < tolerance & abs(imag(
    right_eigenvalue)) < tolerance, 1);
63
64 % Check if an eigenvalue 1 was found
65 if isempty(idx)
66     disp('No eigenvalue 1 found. ');
67 else
68     right_eigenvector = right_eigenvector(:, idx);
69 end
70 %% Compute the eigenvalues and the left eigenvectors
71 [left_eigenvector, left_eigenvalue] = eig(GoogleMatrix');
72 % Explain the result
73 left_eigenvalue = diag(left_eigenvalue);
74 %%find eigenvalue 1
```

```
75 tolerance = 1e-10;
76 idx = find(abs(real(left_eigenvalue) - 1) < tolerance & abs(imag(
    left_eigenvalue)) < tolerance, 1);
77
78 % Check if an eigenvalue 1 was found
79 if isempty(idx)
80     disp('No eigenvalue 1 found. ');
81 else
82     left_eigenvector = left_eigenvector(:, idx);
83 end
84 %% Separate the eigenvector corresponding to the eigenvalue 1 and
    scale it
85 u1 = left_eigenvector;
86 u1=abs(u1)/norm(u1,1);
87 entriesSum=sum(u1);
88 %% Select the maximum element and the corresponding element.
89 %Which page is the most important in the network?
90 [MaxRank, PageMaxRank]=max(u1);
91 %% Check if it's the most popular (most linked to page):
92 MostLinks=sum(GoogleMatrix, 1);
93 [MaxLinks, PageMaxLinks]=max(MostLinks);
94
```

3.3 Explanations and results

Loading the Network Data and checking the number of non-zero elements

```
1 %% Load the network data
2 load("AdjMatrix.mat");
3 %% Check ratio of non-zero elements
4 nnzAdjMatrix = (nnz(AdjMatrix) / numel(AdjMatrix)) * 100;
5 disp('Ratio of non-zero elements')
6 disp(nnzAdjMatrix);
7
```

The program loads an adjacency matrix from the file "AdjMatrix.mat", which represents the structure of a network. Each element in this matrix indicates whether there is a link between two web pages. The code also calculates and displays the percentage of non-zero elements in the adjacency matrix which is saved in the "nnzAdjMatrix" variable. This ratio gives an idea of the density of connections (links) in the network.

Command Window

```
>> lab12  
Ratio of non-zero elements  
0.1506
```

Hình 1: The percentage of non-zero elements

In this case, about 0.1516% of the matrix AdjMatrix are non-zero.

Displaying a small Subnetwork with 500 nodes

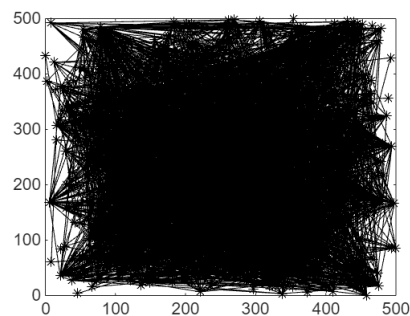
```
1 %% Dimensions of the matrix AdjMatrix  
2 [m,n] = size(AdjMatrix);  
3 disp(m);  
4 disp(n);  
5 %% Display a small amount of network  
6 NumNetwork=500;  
7 AdjMatrixSmall=AdjMatrix(1:NumNetwork,1:NumNetwork);  
8 for j=1:NumNetwork  
9     coordinates(j,1)=NumNetwork*rand;  
10    coordinates(j,2)=NumNetwork*rand;  
11 end;  
12 gplot(AdjMatrixSmall,coordinates,'k-*');
```

In this part, the size() function is used to get the two dimensions of the matrix AdjMatrix and saved to 2 variables m and n. These two numbers should be equal to ensure the matrix is squared. The program considers a smaller portion of the network (500 nodes) for visualization and analysis. Random coordinates are assigned to each node for plotting purposes. The gplot function is used to visualize the small network, where nodes are connected based on the links in AdjMatrixSmall.

Command Window

```
>> lab12  
8297  
  
8297  
  
>>
```

Hình 2: Dimensions of the AdjMatrix



Hình 3: The graph of 500 nodes and their connections

The first figure illustrates a matrix with dimensions of 8297 by 8297. The second figure presents a graph with 500 nodes and their respective connections. **Due to the limited size of the plotting area and the large number of nodes and links, it is challenging to distinguish the individual connections.**

Creating Google Matrix

```
1 alpha=0.15;
2 GoogleMatrix=zeros(NumNetwork,NumNetwork);
3 for i=1:NumNetwork
4     if NumLinks(i)~=0
5         GoogleMatrix(i,:)=AdjMatrixSmall(i,:)./NumLinks(i);
6     else
7         GoogleMatrix(i,:)=1./NumNetwork;
8     end;
9 end;
10
11 GoogleMatrix=(1-alpha)*GoogleMatrix+alpha*ones(NumNetwork,
12     NumNetwork)./NumNetwork;
13 %% Check that all the rows in the GoogleMatrix matrix sum to 1
14 SumGoogleMatrix=sum(GoogleMatrix,2);
15
```

In this case, we will assume that the user can follow any link on the page with a total probability of 85% (all hyperlinks are equal), and jump (teleport) to any other page in the network with a total probability of 15% (again, all pages are equal). Therefore the variable "alpha" is set to 0.15. The Google Matrix is constructed based on the adjacency matrix, where the probability of moving from one page to another is calculated. If a page has no outbound links, equal probability is assigned to all pages. The matrix is adjusted with the teleportation factor to account for random jumps, making it a valid Markov matrix with each row summing to 1.

At the end, the sum() function is used to ensure that all rows in the Google Matrix sum to 1, confirming it represents a valid probability distribution.

Computing the difference between w_{10} and w_5

```
1 w0=ones(1,NumNetwork)./NumNetwork;
2 w1=w0*GoogleMatrix;
3 w2=w1*GoogleMatrix;
4 w3=w2*GoogleMatrix;
5 w5=w0*(GoogleMatrix^5);
6 w10=w0*(GoogleMatrix^10);
7 %% Check the difference between v30 and v20. Observe that it is
8 %% pretty
9 %% small
10 deltax=w10-w5;
```

The code introduce the vector $w_0 = (1, 1, \dots, 1)/NumNetwork$, and compute the consequence vectors $w_1 = w_0 G$, $w_2 = w_1 G$, $w_3 = w_2 G$, $w_5 = w_0 G^5$, $w_{10} = w_0 G^{10}$ (G is the Google matrix) and calculate the

difference $\delta w = w_{10} - w_5$ w_0 is the initial rank vector with equal probability for each page, the program iterates the Google Matrix multiple times (w_1, w_2 , etc.) to see how the ranks evolve, and δw is the difference between the rank vectors after 10 and 5 iterations, checking for convergence towards a steady state.

Computing eigenvalues, left and right eigenvectors of the Google Matrix

Eigenvalues and Right Eigenvectors

```
1 [right_eigenvector, right_eigenvalue] = eig(GoogleMatrix);
2 right_eigenvalue = diag(right_eigenvalue);
3
```

Firstly, we calculate the corresponding eigenvalue and the right eigenvector of the Google matrix using the `eig()` function. The `diag()` function extracts the diagonal elements of the eigenvalues and store them in a column vector.

The next step is to find the right eigenvector corresponding to the eigenvalue $\lambda_1 = 1$. To do so, we need to find an eigenvalue that is equal to 1 and extract the corresponding right eigenvector. However, in this case, such value does not exist. Instead, there is only eigenvalue that is **very close to 1**.

```
1 tolerance = 1e-10;
2 idx = find(abs(real(right_eigenvalue) - 1) < tolerance & abs(imag(
    right_eigenvalue)) < tolerance, 1);
```

In this segment, the variable *tolerance* equal to $1e - 10$ (very very small) defines how close the eigenvalue must be to 1 to be considered equal to 1. The function *real(right_eigenvalue)* extracts the real part of the eigenvalue, while the function *imag(right_eigenvalue)* extracts the imaginary part of the eigenvalues. Then the *find()* function locates the index (*idx*) of the first eigenvalue that satisfies:

- The real part is close to 1 within the specified tolerance.
- The imaginary part is close to 0 within the specified tolerance.

After that, we need to check if such eigenvalue was found by using the following code:

```
1 if isempty(idx)
2     disp('No eigenvalue 1 found. ');
3 else
4     right_eigenvector = right_eigenvector(:, idx);
5 end
6
```

The *isempty()* function checks if the *idx* variable is empty, meaning that no eigenvalue equal to 1 was found. In this situation, the message **'No eigenvalue 1 found'** is displayed. If an eigenvalue of 1 is found, the corresponding eigenvector is extracted from the *right_eigenvector* matrix. This eigenvector is important because, in the context of PageRank, it represents the steady-state distribution of the web pages.

right_eigenvector ×	
500x1 double	
	1
1	-0.0447
2	-0.0447
3	-0.0447
4	-0.0447
5	-0.0447
6	-0.0447
7	-0.0447
8	-0.0447
9	-0.0447
10	-0.0447
11	-0.0447
12	-0.0447
13	-0.0447
14	-0.0447
15	-0.0447

Hình 4: the right eigenvector corresponding to the eigenvalue $\lambda_1 = 1$

In this case, all entries of this vector equal to -0.0447 , meaning that this right eigenvector of the Google Matrix is proportional to the vector $v_1 = (1, 1, \dots, 1)^T$

Eigenvalues and Left Eigenvectors

```

1 %% Compute the eigenvalues and the left eigenvectors
2 [left_eigenvector, left_eigenvalue] = eig(GoogleMatrix');
3 % Explain the result
4 left_eigenvalue = diag(left_eigenvalue);
5 %%find eigenvalue 1
6 tolerance = 1e-10;
7 idx = find(abs(real(left_eigenvalue) - 1) < tolerance & abs(imag(
    left_eigenvalue)) < tolerance, 1);
8
9 % Check if an eigenvalue 1 was found
10 if isempty(idx)
11     disp('No eigenvalue 1 found. ');
12 else
13     left_eigenvector = left_eigenvector(:, idx);
14     u1 = left_eigenvector;

```

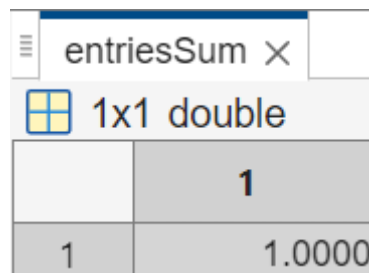
15 `end`

This block of code is used to find the eigenvalues, left eigenvectors and the left eigenvector corresponding to the eigenvalue $\lambda_1 = 1$. The method is the same as finding right eigenvalues and vector but in this case, we perform the function `eig()` on the **transposed** Google Matrix. After that, we denote u_1 as the left eigenvector corresponding to eigenvalue 1.

Normalizing the u_1 vector

```
1 u1= abs(u1)/norm(u1,1);
2 entriesSum=sum(u1);
```

By default, the vector u_1 is not scaled to have all positive components. Using the code will create a vector with all positive components whose entries sum to 1 (called a probability vector). We check this statement by calculating the sum of all entries of u_1 and save it as the `entriesSum` variable.



entriesSum ×	
1x1 double	
	1
1	1.0000

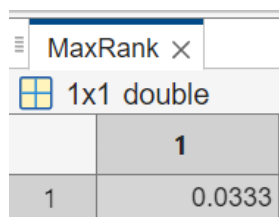
Hình 5: The sum of all entries of vector u_1

After executing the code, the `entriesSum` variable equal to 1.0000, which satisfies the requirement

Selecting the most ranked page

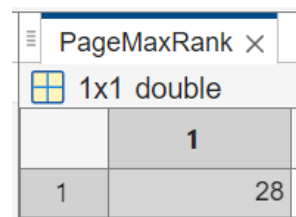
```
1 %Which page is the most important in the network?
2 [MaxRank, PageMaxRank]=max(u1);
```

Using the `max()` function, the code identifies the highest-ranked page in the PageRank algorithm by finding the maximum element in the vector u_1 . The `MaxRank` variable stores the value of this maximum rank, while the `PageMaxRank` variable indicates the position (or index) of the page with this highest rank.



MaxRank ×	
1x1 double	
	1
1	0.0333

Hình 6: Rank of the highest-ranked page



PageMaxRank ×	
1x1 double	
	1
1	28

Hình 7: Location of highest-ranked page

u1 ×	
500x1 double	
	1
23	0.0027
24	0.0007
25	0.0007
26	0.6966
27	0.0007
28	0.0333
29	0.0058
30	0.0028
31	0.0007
32	0.0029

Hình 8: Location and rank of highest-ranked page in vector u_1

Identifying the most popular page (most linked to page)

```
1 MostLinks=sum(GoogleMatrix, 1);
2 [MaxLinks, PageMaxLinks]=max(MostLinks);
```

The code calculates the sum of all the elements of the Google Matrix along the column by using the `sum()` function and return the result as a row vector called *MostLinks*, where each element of this vector represents the total number of outbound links (links from a given page to other pages) for the corresponding page.

```
1 [MaxLinks, PageMaxLinks] = max(MostLinks);
```

The `max(MostLinks)` function finds the maximum value in the *MostLinks* vectors. The *MaxLinks* variable stores this maximum value, which represents the highest number of outbound links from any page, while *PageMaxLinks* is the index of this maximum value, indicating the page that has the most outbound links.

MaxLinks ×	
1x1 double	
	1
1	24.8783

Hình 9: Dimensions of the AdjMatrix

PageMaxLinks ×	
1x1 double	
	1
1	28

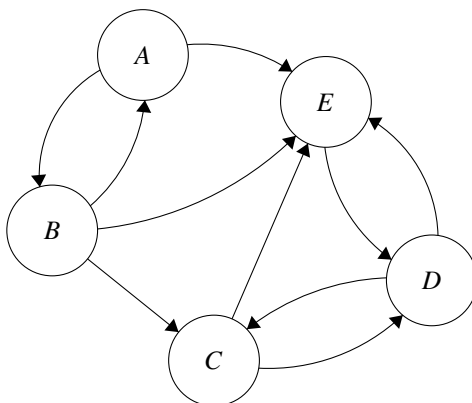
Hình 10: Location of highest-ranked page

MostLinks ×					
1x500 double					
	27	28	29	30	
1	0.2962	24.8783	2.1414	2.2356	

Hình 11: Location and number of linked to page of most-linked page

4 A small example

Given a small graph which represent 5 webpages A, B, C, D, E and their hyperlinks to each other:



- **Step 1:** From the graph, we can generate the probability matrix S :

$$S = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1/2 & 1/3 & 1 & 1/2 & 0 \end{bmatrix}$$

- **Step 2:** Generate the Google Matrix using the equation: $G = \alpha S + (1 - \alpha)E$ (given $\alpha = 0.15$):

$$\begin{aligned}
 G &= (1 - 0.15) \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1/2 & 1/3 & 1 & 1/2 & 0 \end{bmatrix} + 0.15 \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix} \\
 &= \begin{bmatrix} 3/100 & 47/150 & 3/100 & 3/100 & 3/100 \\ 91/200 & 3/100 & 3/100 & 3/100 & 3/100 \\ 3/100 & 47/150 & 3/100 & 91/200 & 3/100 \\ 3/100 & 3/100 & 3/100 & 3/100 & 22/25 \\ 91/200 & 47/150 & 22/25 & 91/200 & 3/100 \end{bmatrix}.
 \end{aligned}$$

From the matrix G , we can see that G is irreducible (no traps), column stochastic (columns sum to 1), with non-negative entries, therefore, it satisfies the **Perron-Frobenius theorem**, meaning that its largest eigenvalue is 1, and its eigenvector corresponding to eigenvalue 1 $p = [p_1 \ p_2 \ \dots \ p_5]^T$ satisfies: $p_i > 0, \sum_i p_i = 1$.

- **Step 3:** Introducing the initial probability vector $u_1 = 1/5 [1 \ 1 \ 1 \ 1 \ 1]^T$, we will multiply it by matrix G until it reaches **steady-state distribution**:

$$Gu_1 = \begin{bmatrix} 0.0886 \\ 0.115 \\ 0.172 \\ 0.2 \\ 0.427 \end{bmatrix}, G^2u_1 = \begin{bmatrix} 0.063 \\ 0.067 \\ 0.148 \\ 0.393 \\ 0.330 \end{bmatrix}, G^3u_1 = \begin{bmatrix} 0.049 \\ 0.057 \\ 0.216 \\ 0.311 \\ 0.368 \end{bmatrix}, G^4u_1 = \begin{bmatrix} 0.046 \\ 0.051 \\ 0.178 \\ 0.343 \\ 0.382 \end{bmatrix}, \dots, G^{10}u_1 = \begin{bmatrix} 0.044 \\ 0.049 \\ 0.190 \\ 0.347 \\ 0.371 \end{bmatrix}$$

At iteration 10th, we can see that 0.371 is the highest rank among 5 webpages, therefore, webpage E is the highest ranked webpage according to the PageRank Algorithm.