Project 12: Matrix eigenvalues and the Google's PageRank algorithm

Goals: To apply matrix eigenvalues and eigenvectors to ranking of webpages in the World Wide Web.

To get started:

- Download the file `lab12.m` and put it in your working directory[15].
- Download the file `AdjMatrix.mat` which contains the adjacency matrix of a so-called "wiki-vote" network with 8297 nodes [16] [9].

Matlab commands used: `load, size, numel, nnz, for... end, gplot`

What you have to submit: The file `lab12.m` which you will modify during the lab session.

INTRODUCTION

According to social polls, the majority of users only look at the first few results of the online search and very few users look past the first page of results. Hence, it is crucially important to rank the pages in the "right" order so that the most respectable and relevant results will come first. The simplest way to determine the rank of a webpage in a network is to look at how many times it has been referred to by other webpages. This simple ranking method leaves a lot to be desired. In particular, it can be easily manipulated by referring to a certain webpage from a lot of "junk" webpages. The quality of the webpages referring to the page we are trying to rank should matter too. This is the main idea behind the Google PageRank algorithm.

Find the "right" result

The Google PageRank algorithm is the oldest algorithm used by Google to rank the web pages which are preranked offline. The PageRank scores of the webpages are recomputed each time Google crawls the web. Let us look at the theory behind the algorithm. As it turns out it, it is based on the theorems of linear algebra!

The main assumption of the algorithm is that if you are located on any webpage then with equal probability you can follow any of the hyperlinks from that page to another page. This allows to represent a webpage network as a directed graph with the webpages being the nodes, and the edges being the hyperlinks between the webpages. The adjacency matrix of such a network is built in the following way: the (i, j)th element of this matrix is equal to 1 if there is a hyperlink from the webpage i to the webpage j and is equal to 0 otherwise. Then the row sums of this matrix will represent numbers of hyperlinks from each webpage and the column sums will represent numbers of times each webpage has been referred to by other webpages.

1 page lead to other pages

can be represented by a graph

Even further, we can generate a matrix of probabilities $\mathbf{S}$ such that the (i, j)th element of this matrix is equal to the probability of traveling from ith webpage to jth webpage in the network. This probability is equal to zero if there is no hyperlink from ith page to jth page and is equal to $1/N_i$ if there is a hyperlink from ith page to jth page, where N_i is the total number of hyperlinks from ith page. For instance, consider a sample network of only four webpages shown on the Fig. 17. The matrix $\mathbf{S}$ for this network can be written as:

[15] The printout of the file `lab12.m` is given in the appendices of this book.

[16] The original network data is available here: `https://snap.stanford.edu/data/`.

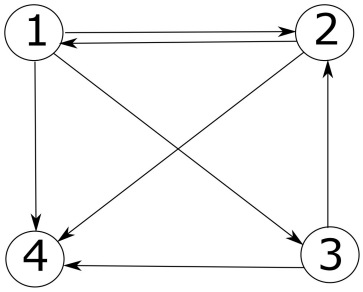

Figure 17: Sample network of four webpages

$$\mathbf{S} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{1}$$

links from columns to rows

node 4 is a dangling node

There are several issues which make working with the matrix $\mathbf{S}$ inconvenient. First of all, there are webpages that do not have any hyperlinks - the so-called "dangling nodes" (such as the node 4 in Fig. 17). These nodes will correspond to zero rows of the matrix $\mathbf{S}$. Moreover, the webpages in the network may not be connected to each other and the graph of the network may consist of several disconnected components. These possibilities lead to undesirable properties of the matrix $\mathbf{S}$ which make computations with it complicated and not even always possible.

no links between nodes

The problem of the dangling nodes can be solved by assigning all elements of the matrix $\mathbf{S}$ in the rows corresponding to the dangling nodes equal probabilities $1/N$, where N is the number of the nodes in the network. This can be understood in the following way: if we are at the dangling node we can with equal probability jump to any other page in the network. To solve the potential disconnectedness problem, we assume that a user can follow hyperlinks on any page with a probability $1-\alpha$ and can jump (or "teleport") to any other page in the network with a probability α. The number α is called a damping factor. The value of $\alpha = 0.15$ is usually taken in practical applications. The "teleport" surfing of the network can be interpreted as a user manually typing the webpage address in the browser or using a saved hyperlink from their bookmarks to move from one page onto another. The introduction of the damping factor allows us to obtain the Google matrix $\mathbf{G}$ in the form:

the last row becomes 1/4 for all elements (solving the dangling problem)

$$\mathbf{G} = (1-\alpha)\mathbf{S} + \alpha\mathbf{E},$$

where $\mathbf{E}$ is a matrix with all the elements equal to $1/N$, where N is a number of webpages in the network.

The matrix $\mathbf{G}$ has nice properties. In particular, it has only positive entries and all of its rows sum up to 1. In mathematical language, this matrix is *stochastic* and *irreducible* (you can look up the precise definitions of these terms if you are interested). The matrix $\mathbf{G}$ satisfies the following **Perron-Frobenius theorem**:

Theorem 1 (Perron-Frobenius) *Every square matrix with positive entries has a unique unit eigenvector with all positive entries. The eigenvalue corresponding to this eigenvector is real and positive. Moreover, this eigenvalue is simple and is the largest in absolute value among all the eigenvalues of this matrix.*

Let us apply this theorem to the matrix $\mathbf{G}$. First of all, observe that the row sums of the

matrix $\mathbf{G}$ are equal to 1. Consider the vector $\mathbf{v}_1 = (1, 1, ..., 1)^T/N$. It is easy to see that

$$\mathbf{G}\mathbf{v}_1 = \mathbf{v}_1.$$

But then it follows that $\mathbf{v}_1$ is the unique eigenvector with all positive components, and, therefore, by the Perron-Frobenius theorem, $\lambda_1 = 1$ is the largest eigenvalue!

We are interested in the left eigenvector for the eigenvalue $\lambda_1 = 1$:

$$\mathbf{u}_1^T\mathbf{G} = \mathbf{u}_1^T.$$

Again, by the Perron-Frobenius theorem, the vector $\mathbf{u}_1$ is the unique unit eigenvector with all positive components corresponding to the largest in absolute value eigenvalue $\lambda_1 = 1$. We will use the components of this vector for the ranking of webpages in the network.

Let us look at the justification behind this algorithm. We have already established that the vector $\mathbf{u}_1$ exists. Consider the following iterative process. Assume that at the beginning a user can be on any webpage in the network with equal probability:

$$\mathbf{w}_0 = (1/N, 1/N, ..., 1/N).$$

After 1 step (one move from one webpage to another using hyperlinks or teleporting), the probability vector of being on the ith webpage is determined by the ith component of the vector

$$\mathbf{w}_1 = \mathbf{w}_0\mathbf{G}.$$

After two moves the vector of probabilities becomes

$$\mathbf{w}_2 = \mathbf{w}_1\mathbf{G} = \mathbf{w}_0\mathbf{G}^2,$$

and so on.

We hope that after a large number of steps n, the vector $\mathbf{w}_n = \mathbf{w}_0\mathbf{G}^n$ starts approaching some kind of limit vector $\mathbf{w}_*$, $\mathbf{w}_n \to \mathbf{w}_*$. It turns out that due to the properties of the matrix $\mathbf{G}$ this limit vector $\mathbf{w}_*$ indeed exists and it is exactly the eigenvector corresponding to the largest eigenvalue, namely, $\lambda_1 = 1$. Moreover, numerical computation of matrix eigenvalues is actually based on taking the powers of the matrix (it is called the Power method) and not on solving the characteristic equation!

Let us assume that the vector $\mathbf{w}_*$ is a non-negative vector whose entries sum to 1. Then the components of this vector represent the probabilities of being on each webpage in the network after a very large number of moves along the hyperlinks. Thus, it is perfectly reasonable to take these probabilities as ranking of the webpages in the network.

TASKS

1. Open the file `lab12.m`. In the code cell titled `%%Load the network data` load the data from the file `AdjMatrix.mat` into Matlab by using the `load` command. Save the resulting matrix as `AdjMatrix`. Observe that the adjacency matrices of real networks are likely to be very large (may contain millions of nodes or more) and sparse. Check the sparsity of the matrix `AdjMatrix` using the functions `numel` and `nnz`. Denote the ratio of non-zero elements as `nnzAdjMatrix`. If you did everything correctly you should obtain that only 0.15% of the elements of the matrix `AdjMatrix` are non-zero.
   Variables: `AdjMatrix`, `nnzAdjMatrix`

2. Check the dimensions of the matrix `AdjMatrix` using the `size` function. Save the dimensions as new variables `m` and `n`.
   Variables: `m`, `n`

3. Observe that while the network described by the matrix `AdjMatrix` is not large at all from the viewpoint of practical applications, computations with this matrix may still take a noticeable amount of time. To save time, we will cut a subset out of this network and use it to illustrate the Google PageRank algorithm. Introduce a new variable `NumNetwork` and set its value to 500. Then cut a submatrix `AdjMatrixSmall` out of the matrix `AdjMatrix` and plot the graph represented by the matrix `AdjMatrixSmall` by running the following code cell:

```
%% Display a small amount of network
NumNetwork=500;
AdjMatrixSmall=AdjMatrix(1:NumNetwork,1:NumNetwork);
for j=1:NumNetwork
    coordinates(j,1)=NumNetwork*rand;
    coordinates(j,2)=NumNetwork*rand;
end;
gplot(AdjMatrixSmall,coordinates,'k-*');
```

   This will plot the subgraph of the first 500 nodes in the network with random locations of the nodes. Notice the use of the function `gplot` to produce this graph. Observe that Matlab has special functions `graph` and `digraph` for working with graphs, but those functions are a part of the special package "Graph and Network Algorithms" which may not be immediately available. Simpler methods, as shown above, will be sufficient for our purposes.
   Variables: `AdjMatrixSmall`, `coordinates`, `NumNetwork`

4. Set the parameter $\alpha = 0.15$. Introduce the vector w_0`=(1,1,...,1)/NumNetwork`, and compute the consequent vectors $w_1 = w_0G$, $w_2 = w_1G$, $w_3 = w_2G$, $w_5 = w_0G^5$, $w_{10} = w_0G^{10}$. Compute the difference $\delta w = w_{10} - w_5$. Observe that the sequence w_n converges to a certain limit vector w_* very fast.
   Variables: `w0`, `w1`, `w2`, `w3`, `w5`, `w10`, `deltaw`

5. Compute the eigenvalues and the left and the right eigenvectors of the matrix G using the function `eig`. Observe that the right eigenvector corresponding to the eigenvalue $\lambda_1 = 1$ is proportional to the vector $v_1 = (1, 1, ..., 1)$. To compute the left eigenvectors, use the function `eig` on the matrix `G'`. Select the left eigenvector corresponding to the eigenvalue $\lambda_1 = 1$ and denote it as `u1`.
   Variables: `u1`

6. Observe that by default the vector `u1` is not scaled to have all positive components (even though all the components of the vector $\mathbf{u}_1$ will have the same sign). Normalize this vector by using the code:

```
u1=abs(u1)/norm(u1,1);
```

   This will create a vector with all positive components whose entries sum to 1 (called a probability vector).

7. Use the function `max` to select the maximal element and its index in the array.
   Variables: `MaxRank`, `PageMaxRank`

8. Find out whether the highest ranking webpage is the same as the page with the most hyperlinks pointed to it. To do so, create the vector of column sums of the matrix `G` and save it as `MostLinks`. Use the function `max` again to select the element with the maximal number of links.
   Variables: `MostLinks`, `MaxLinks`, `PageMaxLinks`

9. Compare if `MaxRank` and `MaxLinks` are the same.
   Q1: What is the number of hyperlinks pointing to the webpage `MaxRank`?
   (look at the vector `MostLinks` to find out). **24.8783**