Introduction

The importance of a website is a very subjective case, this generally depends on the reader's interests, knowledge and opinion. PageRank is a method for rating the importance of websites.

Google PageRank is one of the methods Google uses/used to determine a page's relevance or importance. PageRank has been named after one of the founders of Google, Larry Page. The PageRank algorithm estimates the importance of a website by counting the number and quality of links to that page. The assumption for this algorithm is that more important websites are likely to receive more links from other websites.

The output of the algorithm is a probability distribution which can be used to represent the likelihood that a random surfer by clicking on links will arrive at a certain page. Because more links go to the important websites, the surfer is more likely to end up there.

This behaviour can be described as a Markov process. A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

Syntax

This function uses three input arguments. First, a $(n \times n)$ sparse connection matrix A, furthermore it is optional to give a list of associated websites (a cell array with n strings). It is also optional to give the probability p, which is the probability that you follow an arbitrary link on the current page, $1-p$ represents the probability that you jump to a completely arbitrary page, e.g. by typing in that URL in the search bar. The default value of p is set to 0.85.

The output is a vector with the page rankings, also a bar plot is shown, which displays the number of links to the pages, see Figure 2 for an example. If the optional list of websites is given as input, the output also produces a list of the top sites with their respective page rank.

Description of the Method

First of all, the probability of following a link is set to the default Google PageRank value, $p = 0.85$, if no value is assigned to this input variable. The $n \times n$ (n is the number of pages in the set W of Web pages that can be reached by following a chain of hyperlinks) connection matrix A displays the number of hyperlinks. $A_{ij} = 1$ if there is a hyperlink to page i from page j and $A_{ij} = 0$ otherwise. The number of nonzeros in A is the total number of hyperlinks in W. As $p = 0.85$ equals the probability that the random walk follows a link, $1-p$ equals the probability that some arbitrary

page is chosen. $\frac{1-p}{n}$ is the probability that a particular random page is chosen. The Markov process is described by the matrix:

$$B_{ij} = \begin{cases} \frac{p}{c_j} A_{ij} + \frac{1-p}{n} & \text{if } c_j \neq 0 \\ \frac{1}{n} & \text{if } c_j = 0 \end{cases}$$

The matrix A is the transition probability matrix of the Markov chain. If we solve $x = Bx$ and this is scaled by $\sum_i x_i = 1$ then x is the state vector of the Markov chain and also the page rank. The solution to this is $(I - B)x = 0$.

We can write B as $B = pAD + f$. B is formed by scaling the connection matrix by its column sums. The jth column is the probability of jumping from the jth page to other pages. If the jth page has no out-links, then all elements in its column are assigned the value of $\frac{1}{n}$. All elements in B are between zero and one, and its column sums are equal to one. Because n can be very large, matrix B is never actually formed.

In this implementation, the sum of each column of the connection matrix G is calculated. This is followed by finding the linear indices corresponding to the nonzero entries. Then a sparse matrix D is created, the diagonal matrix formed by the reciprocals of the out-degrees:

$$D_{jj} = \begin{cases} \frac{1}{c_j} & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases}$$

An $n \times 1$ vector of ones f is created to account for the random choices of websites which do not follow links. The equation $x = Bx$ is then solved and rescaled to create the page rank vector.

Example and Illustration

We were given six different datasets to try Google page rank on, with sizes 100, 500, 2,000, 10,000, 50,000 and 200,000 respectively. With a lot of URLs, the graph of the page rank is rather cluttered or too small to distinguish different page ranks per URL. This is why plotting is off by default.

The function input requires an $n \times n$ sparse matrix for relations and an $n \times 1$ sparse matrix for URLs. Dataset `sp100` and `sp500` were given in the correct format, however, the other ones were all $1 \times n$. This is easily solved by transposing the matrix, however, this is purposely not accounted for in the function. The function requires an $n \times 1$ sparse matrix for URLs.

Below you can find the output of `pagerank(sp100,url100,true)`. The page rank of the smallest data set, with a result that includes URLs and shows a bar plot. The full result can be found below.

```
>> pagerank(sp100,url100,true)

ans =

  5×2 cell array

    {[0.0684]}    {'http://www.regering.nl'                          }
    {[0.0655]}    {'http://www.government.nl'                        }
    {[0.0599]}    {'http://www.digid.nl'                             }
    {[0.0523]}    {'http://search.regering.nl/government/search...'}
    {[0.0499]}    {'http://www.postbus51.nl'                         }
```

Figure 1: Function Output of `pagerank(sp100,url100,true)`.

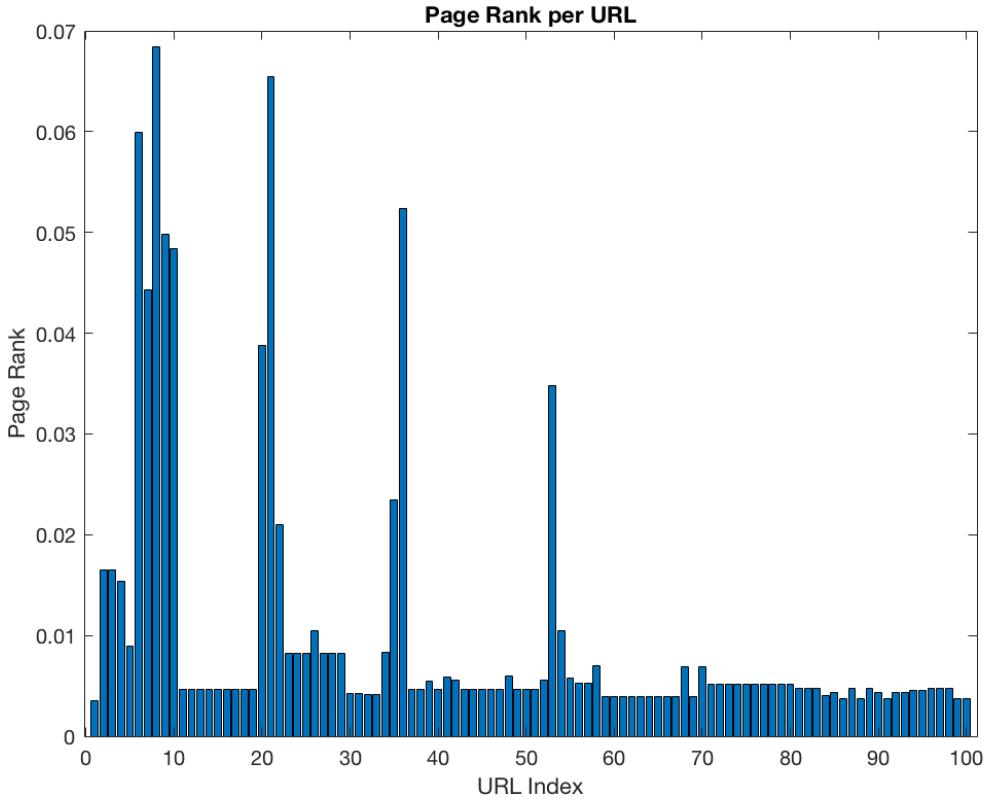

Figure 2: Page Rank per URL with `sp100` as Sparse Input and `url100` as URL Input.

Advantages and Drawbacks

Some of the advantages of using the PageRank algorithm as a ranking measure are, its robustness against spam, because it is not easy for a webpage owner to add inlinks to their website from other important pages. Furthermore, it is a global measure and query independent. PageRank pre computes the rank score, causing it to be fast and less time-consuming.

However, it also has some drawbacks, first of all, it favors the older pages because a new page will not have many links, unless its part of an existing website. Also, website owners can 'buy' their links in purpose of raising their PageRank. However, Google has publicly warned if this is discovered, the links are to be ignored in the future, or maybe even taken out of Google's index.

Performance

As the function only uses matrix operations, the operation is rather swiftly. The six datasets were tested on a laptop with an Intel® Core™ i5-7360U@2.30 GHz processor (Hyper-Threading disabled) running MATLAB R2019a on macOS 10.13.6 (17G7024). The operation timings were calculated using the functions `tic` and `toc`.

URLs	Seconds to Complete
100	0.001299
500	0.001817
2,000	0.375720
10,000	2.673912
50,000	146.608815
200,000	?

Table 1: Function Timings of the Six Given Datasets.

As you can see with small datasets the funtion completes rather instantly. However, if the size linearly increases, the completion time seems to go up exponentially. Due to this we were not able to find the timings of the largest dataset on the current test setup.

Appendix I: Full Function Implementation

```
function [] = pagerank(relations,urls,show_plot,top_max,p)
%This function determines the Google page rankings using the power method
%   PAGERANK(RELATIONS,URLS,SHOW_PLOT,TOP_MAX,P)
%   tries to find the Google page rank of the sparse input matrix RELATIONS
%   using the power method with probability P. Default is 85% for Google
%   page rank. The result will be a nx1 cell array containing the top n
%   pageranks. The default value for n is 5, but can be changed by setting
%   TOP_MAX. It is also optional to show a plot of the page rank
%   distribution by setting SHOW_PLOT to true. The pageranks can be
%   connected to a URL input by giving an mx1 cell array as input for the
%   URLS parameter. The output will then change to an nx2 cell array, with
%   the second column the most popular URLs.

% Check if parameters are present, otherwise set defaults.
if ~exist('show_plot')
    show_plot = false;
end
if ~exist('top_max')
    top_max = 5;
end
if ~exist('p')
    p = 0.85;
end

% Get amount of websites.
n = length(relations);
% Calculate the out-degree c_j.
c = sum(relations, 1);
% Find all URLs with out degree.
c_non_zero = find(c~=0);
% Calculate 1/cj when c is non zero and create sparse matrix.
D = sparse(c_non_zero,c_non_zero,1./c(c_non_zero),n,n)
% Create identity matrix.
I = speye(n,n);
% Solve x = Bx, account for URLs that do not follow links with f.
x = (I-p*relations*D)\ones(n,1);
% Rescale to get page ranks.
pageranks = x/sum(x);

% Sort the found page ranks with the highest on top, then save it as a
% vector. Also save indices as a vector.
[topPageRanks, topIndices] = sort(pageranks, 'descend');

% Get only the top n values of the page rank and URLs for display. Also
% convert the vector to cell array.
topPageRanks = num2cell(topPageRanks(1:top_max));
urls         = urls(topIndices(1:top_max));

% If we have URL input, show the top n URLs, otherwise only show page ranks.
```

```
50 if exist('urls')
51     [topPageRanks urls]
52 else
53     [topPageRanks]
54 end
55 
56 % If plot wanted, show it.
57 if show_plot
58     % Bring figure to front.
59     figure();
60     bar(pageranks);
61     title(['Page Rank per URL']);
62     xlabel('URL Index');
63     ylabel('Page Rank');
64 end
65 
66 end
```