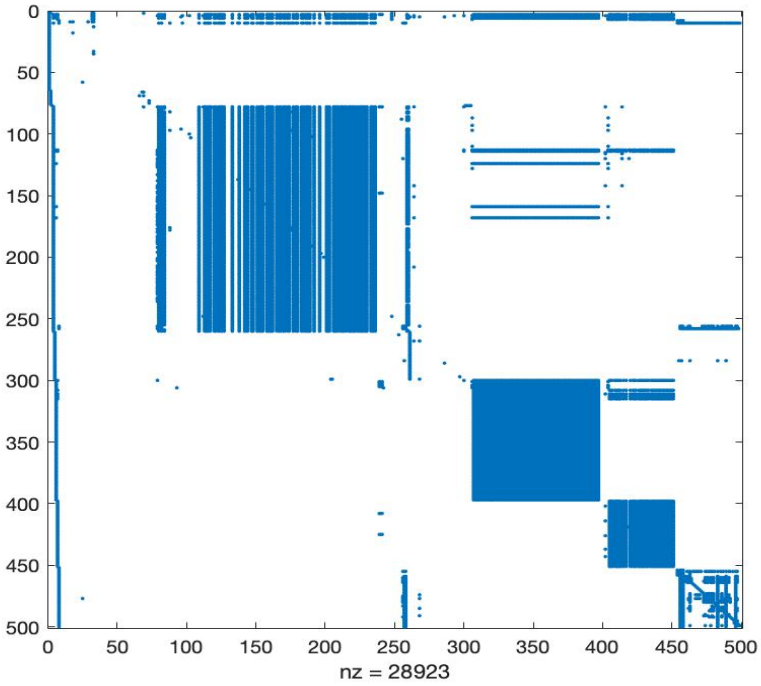
Introduction to Scientific Computing
Report Practical : Markov Chains
Denis Garabajiu S4142551
Constantin Cainarean S4142152
Feb 23, 2021

PART 1 PageRank I

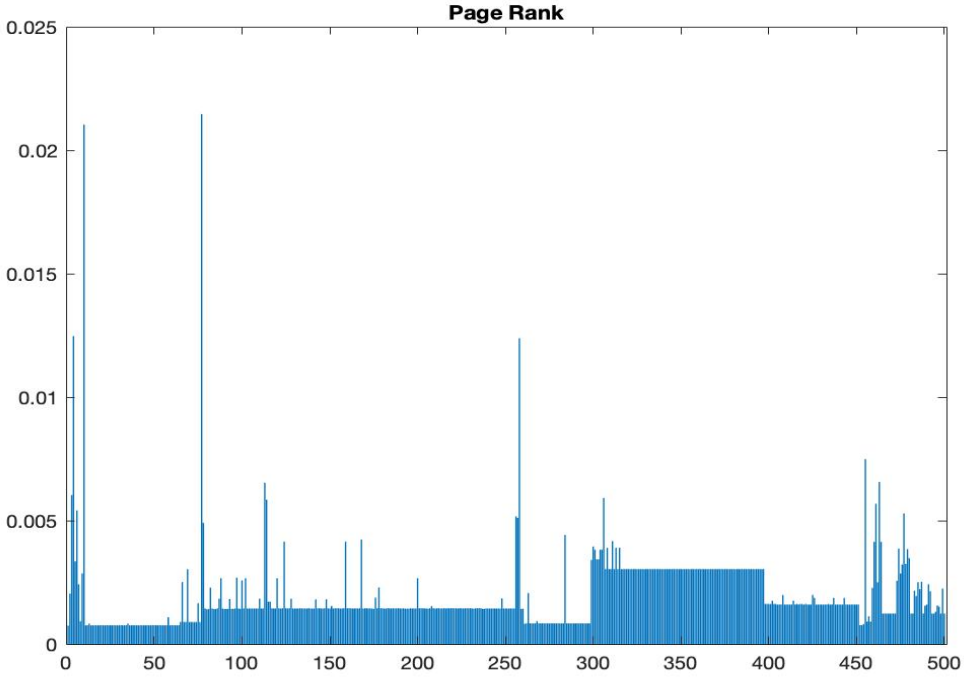
a. We have began the assignment by loading the *harvard500-2018* dataset, then we have used the function *spy* to generate a spy plot that helps us checking the nonzero structure of the connectivity, and we also used the function *pageranks* from EXM toolbox to compute the PageRanks of the items present in dataset, producing a bar graph and printing the dominant URLs. The top 10 dominant urls can be seen in the table below :

	page-rank	in	out	url
77	0.0215	6	0	http://dublincore.org/documents/abstract-model/
10	0.0210	134	0	http://schema.org
4	0.0125	230	186	http://xmlns.com/foaf/0.1
258	0.0124	126	46	http://schema.org/CreativeWork
455	0.0075	36	2	http://schema.org/Text
463	0.0066	29	26	http://schema.org/Event
113	0.0065	218	186	http://xmlns.com/foaf/0.1/Agent
3	0.0060	221	1	http://purl.org/dc/terms
306	0.0059	95	16	http://rdfs.org/sioc/spec
114	0.0058	217	186	http://xmlns.com/foaf/0.1/Document

The spy graph :



The PageRanks graph :



b. For this subpoint we have created a function *ISClinks* that takes as parameters the URLs of the nodes and an n-by-n sparse connectivity matrix with G(i,j) = 1 if node j is linked to node i and G(i,j)=0 otherwise. Our task is to search pages for the harvard dataset that

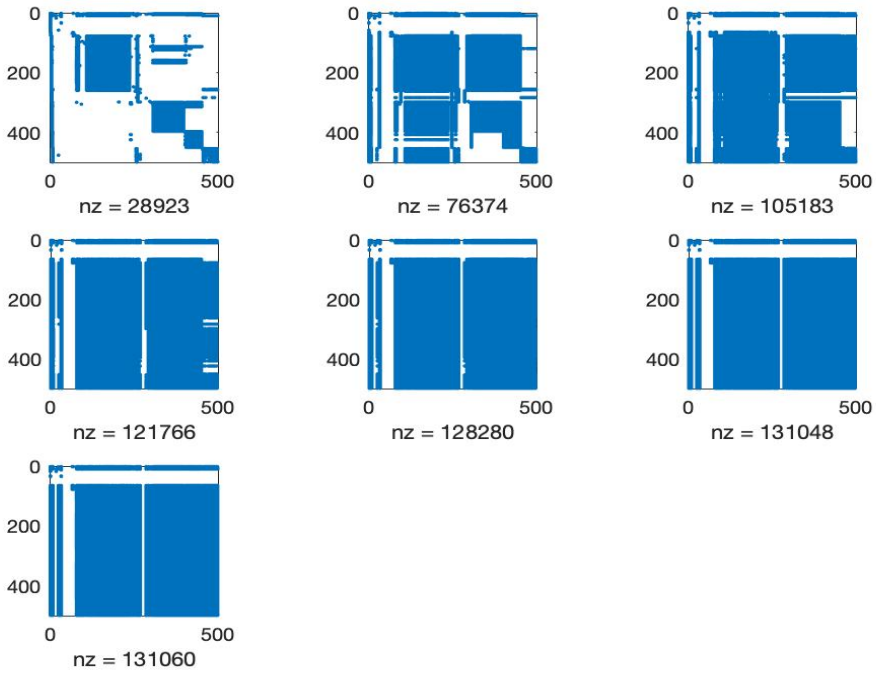
only have one outgoing and incoming node (excluding self-referential links). We have selected the following approach : for each of the matrix G columns, and rows we will compute the sum. As we know that outgoing connections are located on columns, and incoming on the rows, also we will compute the diagonal to be sure that we exclude self-referential links. Therefore we have an if condition where we will check: if an URL has 2 outgoing links and 2 incoming links and if one of each of them is a self-link or if an URL has an incoming link and an outgoing link and does not have self-link then this URL falls under the conditions from the exercise. Our function can be found in the file under the name *ISClinks .m*.

c. For this part of the assigment we work with the matrix G raised at the power p.The matrix power G^p shows the number of paths of length p to page i from page j.

- In the first subpoint of this part we had to compute the power p, where the number of nonzeros stops increasing. In order to make our algorithm more efficient, we have used a single for loop that would check if the current number of nonzeros for the G raised at the power p is equal to the number of nonzeros for the G raised at the power p + 1, in that case we conclude that the number stops increasing. To compute the numbers of nonzeros we have used the function *nnz*. After running the algorithm we computed that G raised to the power 7 has equal number of nonzeros with G raised to the power 8, therefore when p = 7, the number of nonzeros of G^p stops increasing.

- In order to find out what fraction of the entries in G^p are nonzero, we have computed the number of nonzeros of G^p where p is 7, and we have divided that by the number of the total elements in the matrix G (500 * 500 = 25000), we multiplied the result by 100 in order to get the percentage. The result was : **52.424%**

- In this subpoint we have used the function *spy* to generate a spy plot that helps us checking the nonzero structure of the connectivity in the successive powers of the matrix G. Below we have attached the figure with the results of this operation :



The first thing we notice is that when we increase the power that we raise G on, the number of nonzero values is also increasing. As we have concluded before, after G^7 there is no increase in the number of nonzeros. So this algorithm constructs the paths from point A to point B by traversing multiple URLs, something similar we learned at Discrete Structures(Prim's and Kruskal's algorithms). By multiplying G by G multiple times, we obtain all possible paths that can take a user from point A to point B using paths of length p. The algorithm stops when no new paths of higher length are found. From spy graphs above we can see that there are some links that can't be accessed from other links.

PART 2 PageRank II

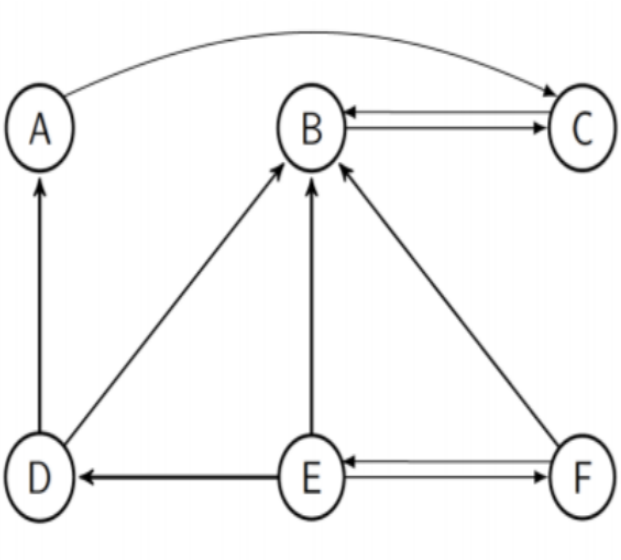


Figure 1: A tiny web.

a. The task of the first subpoint of the part 2 of the practical was to compute the connectivity matrix of the given figure (found above). In the figure, nodes represents web pages and edges represent links between web pages. To compute the connectivity matrix we have used the following principles : The number of nodes in the network will represent the size of the matrix, in our case 6x6. Each cell representing a connection between two nodes receives a value of 1 (e.g. Cell A – C). Otherwise, if there is no connection the value is 0. This have resulted us in the following matrix :

	A	B	C	D	E	F
A	0	0	0	1	0	0
B	0	0	1	1	1	1
C	1	1	0	0	0	0
D	0	0	0	0	1	0
E	0	0	0	0	0	1
F	0	0	0	0	1	0

b. For this subpoint our task was to write a function that computes the Markov matrix of a network G with surfing probability p. It is obvious that the function should take as parameters the value p and the network G. By looking at the definition of the Markov matrix we came up with the following algorithm :
We will iterate over the rows and columns of the network G, and we will check if the

value of the sum of the current column is 0 then we will use the formula $1/n$, otherwise if the sum of the column is different from 0, we use the other formula. Our function can be found in the file *ISCmarkov_matrix.m*.

c. In order to find the equilibrium distribution of the Markov chain in full form we have used the power method. We have created a function *[x,iter] = ISCmarkov(A,tol,maxiter)* that takes as the parameters the Markov matrix, the value of tolerance and the number of maxiteration. The algorithm is described in the *Practical2.mlx* file, we have implemented it following the described steps. To check that our algorithm implementation works properly we have used the Markov matrix from the lecture : $\begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix}$ and the values of tol = **0.0001** and maxiter = **100**. This has given us the correct result(up to 4 decimals as well), as in the lecture (0.4500 and 0.5500).

d. We added the necessary functionality, it can be found in the file *ISCpagerank_test1.c*. We have ordered the output values in descending order with the MATLAB code : x = *sort(x,'descend');*. The setting of tol and maxiter are the previous used values : tol = **0.0001** and maxiter = **100**.

e. We modiefied the Matlab function pagerank(U,G,p) from the EXM toolbox so that it just computes the PageRanks, but does not do any plot-ting or printing. The comments helped us understand the algorithm so removing unused functionality was not a problem for us. Next we created a script file textitISCpagerank_test2.c in which we call our modified function with p = 0.85 and with G from the point a. The results of test 1 and test 2 agree, so we can say that our algorithm was implemented correctly. Below are the screenshots of the computations:

```
>> ISCpagerank_test1
    0.4252
    0.4209
    0.0405
    0.0405
    0.0365
    0.0365
```

```
>> ISCpagerank_test2
    0.4252
    0.4208
    0.0405
    0.0405
    0.0365
    0.0365
```

Appendix: the skeleton of our scripts : `ISClinks.m`

```
load harvard500-2018;
n = size(G,2);

% Function PART 1 point b)
function A = ISClinks(U, G)
    colsum = sum(G);
    rowsum = sum(G, 2);
    diagonal = diag(G);
    A = [];
    x_index = 1;
    for i = 1 : n
        if ((colsum(i) == 2 && rowsum(i) == 2  && diagonal(i) == 1) || (colsum(i) == 1
            A(x_index) = U(i);
            x_index = x_index + 1;
        end
    end
end
```

part1.m

```
load harvard500-2018;

n = size(G,2);
total_elements = n * n;

% PART 1 Q3 - a)
for p = 1 : 50
    if ( nnz(G ^ p) == nnz ( G ^ (p + 1) ) )
        fprintf('The power p where the number of non-zeros stops increasing is %d\n',
        break
    end
end

% PART 1 Q3 - b)
fraction = nnz(G ^ p) / total_elements * 100;

% PART 1 Q3 - c )
for i = 1:7
    subplot(3,3,i)
    spy(G^i)
end
```

ISCmarkov.m

```
% PART 2 Q3
```

```
function [x, iter] = ISCmarkov(A, tol, maxiter)
    n = size(A,2);
    x = zeros(n,1);

    for dist = 1 : n
        x(dist) = 1 / n;
    end

    new_x = x;

    for iter = 1 : maxiter
        new_x = A * new_x;

        if (norm(abs(x - new_x))) <= tol
            break
        end

        x = new_x;
    end

end
```

Other files are included in the archive, Overleaf had an error reading them. Sorry for the inconvenience