Università della Svizzera italiana
Institute of Computing CI

Numerical Computing **2024**

Student: Lorenzo Galli **Discussed with:** Costanza Rogriduez Gavazzi, Gianmarco De Vita, Lorenzo Varese, Alessandro Della Flora, Prof. Edoardo Vecchi

---

Solution for Project 1 **Due date:** Wednesday, 2 October 2024, 11:59 PM

---

1. Theoretical questions [15 points]

(a) **What are an eigenvector, an eigenvalue and an eigenbasis?**

Let V be a vector space over a field K (for example R of real numbers) and let the linear application f: V $\rightarrow$ V be an endomorphism. We will say that v $\in$ V is an eigenvector and λ its eigenvalue when f(v) = λv. In other words the linear application does not change the direction and λv is proportional to v. The set of all eigenvectors of V corresponding to the same eigenvalue, together with the zero vector, is called the eigenspace. If a set of eigenvectors of V forms a basis of the domain of V, then this basis is called an eigenbasis.

(b) **What assumptions should be made to guarantee convergence of the power method?**

The power iteration algorithm starts with a vector b_0, which may be an approximation to the dominant eigenvector or a random vector. At every iteration, the vector b_k is multiplied by the matrix A and normalized. If we assume A has an eigenvalue that is strictly greater in magnitude than its other eigenvalues and the starting vector b_0 has a nonzero component in the direction of an eigenvector associated with the dominant eigenvalue, then a subsequence (b_k) converges to an eigenvector associated with the dominant eigenvalue. Without the two assumptions above, the sequence (b_k) does not necessarily converge.
The Theorem for the Convergence of the Power Method tells us that a sufficient condition for convergence of the power method is that the matrix A be diagonalizable (and have a dominant eigenvalue).

(c) **What is the shift and invert approach?**

The shift-invert method is based on a transformation to access interior eigenvalues of matrices. It's a technique used to solve eigenvalue problems more efficiently, particularly, as just said, when one is interested in finding a few eigenvalues of a large matrix near a specific target value.

(d) **What is the difference in cost of a single iteration of the power method, compared to the inverse iteration?**

The steps of a single iteration of the Power Method are the matrix - vector multiplication and the normalization of the resulting vector. These two operations in the worst case cost respectively O(n^2) and O(n), so the total cost will be: O(n^2).

Differentially, the Inverse Iteration Method involves solving a system of linear equations and normalizing the resulting vector at each iteration. The cost of factorization for an n×n matrix is $O(n^3)$ while, as we have seen before, the cost of normalization is O(n). That said, the total cost will be: $O(n^3)$.

(e) **What is a Rayleigh quotient and how can it be used for eigenvalue computations?**

We understand that the power method is used to approximate a dominant eigenvector of a certain matrix A. Let's assume that we don't know the dominant eigenvalue of A. The Rayleigh Theorem provides a formula, called the Rayleigh quotient, for determining the eigenvalue corresponding to a given eigenvector.
If x is an eigenvector of a matrix A, then its corresponding eigenvalue is given by the following formula: λ = Ax x / x x

The Rayleigh quotient can be used to improve eigenvalue approximations in iterative methods, such as the power method or the inverse iteration method. For this reason it's frequently used for eigenvalue problems primarily for two purposes: perfecting eigenvalue estimates and accelerating convergence in iterative methods.

2. Other webgraphs [5 points]

I chose two websites about athletics to compute PageRanks and analyze the results.
The websites are respectively www.fidal.it and www.atletica.me .

In order to give a better understanding of what the behaviour of those websites' PageRank will be, let me write a couple lines about what are they used for. Fidal.it is the website of the Italian Athletics Federation, and there you can find competition calendar, results, news, press releases, photos, rankings and much more. Atletica.me is a website fully dedicated to athletes results, where each athlete has his own profile displaying all the progress and results.

I analyzed the PageRank of 200 pages of each website, and the results are pretty interesting in their difference. Let's get started with www.fidal.it :

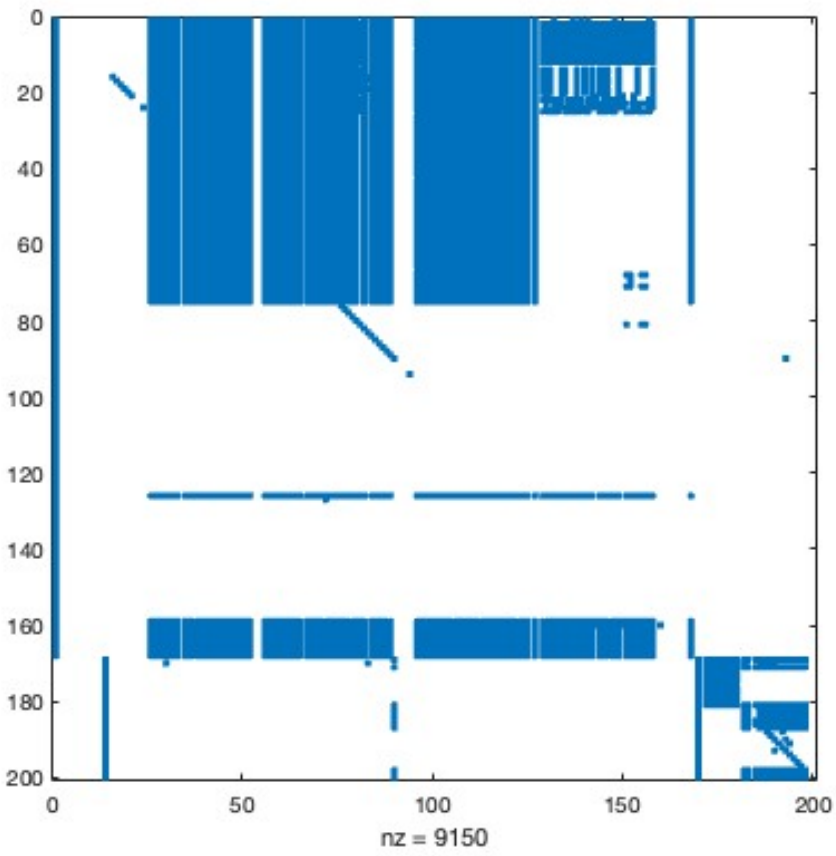


Figure 1: www.fidal.it Page Connections Graph

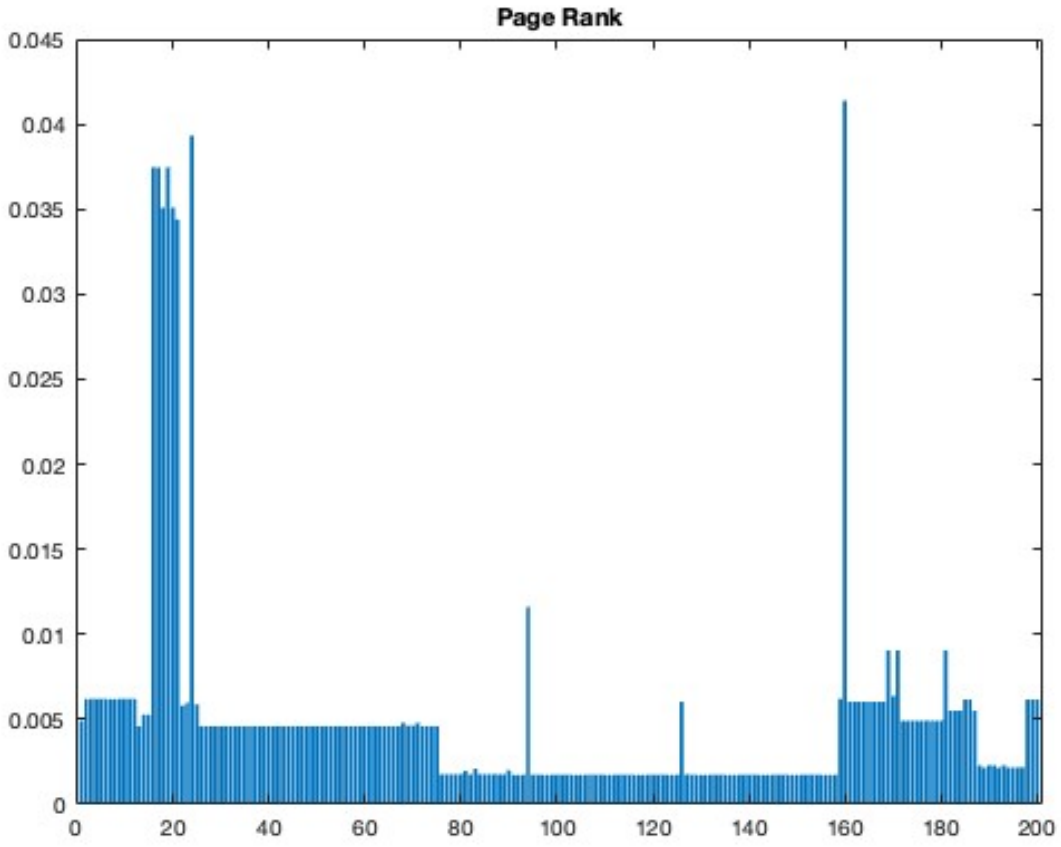


Figure 2: www.fidal.it URLs PageRank Distribution Graph

Rank	Scanning position	PageRank	In	Out	Url
1	160	**0.0414**	122	1	https://www.iubenda.com/privacy-policy/252652/cookie-policy
2	24	**0.0393**	116	1	https://www.instagram.com/atleticaitaliana
3	16	**0.0375**	107	1	https://www.uliveto.it
4	17	**0.0375**	107	1	https://www.felicetti.it/it
5	19	**0.0375**	107	1	https://www.corrieredellosport.it
6	18	**0.0351**	106	1	https://www.conica.com/it
7	20	**0.0351**	106	1	https://www.tuttosport.com
8	21	**0.0344**	102	1	https://www.facebook.com/fidal.it
9	94	**0.0116**	2	1	https://www.sportesalute.eu/bandi-e-avvisi/bandi-altri-enti.html
10	181	**0.0091**	28	0	https://www.joma-sport.net

It's very interesting to analyze the results we have obtained. The first graph shows a very big shape of cliques, and that is easily explained by the fact that there are big groups of pages that link to each other. We can also see some pages less connected, but we can say that the website has a lot of well linked subgraphs.

Now, let's take a look at the second graph. It shows how the PageRanks are distributed and shows us how big is the difference between high and low pageranks. We notice that most of the pages with the highest PageRank are about the ones that were analyzed before. However, the page with the highest PageRank seems to be the 160th analyzed page. Apart from this, the most important thing that we can see is that there is a huge difference between the pages with high PageRank and the rest of the pages.

In conclusion, the table shows the top 10 URLs ordered by the highest PageRank. We can easily notice that a high PageRank is given by a high number of "In" pages, which are the pages that link to that URL. It's also interesting to notice that the website with the highest PageRank is the privacy policy, and that is because we can find the privacy policy in the footer of every page. The second website in the rank is the instagram channel of fidal.it, which appears in the header, the footer and in the content of almost every page. The other URLs are mainly sponsorships of the Federation, that obviously must appear very often because they pay for it.

Let's now take a look at the analysis of www.atletica.me :

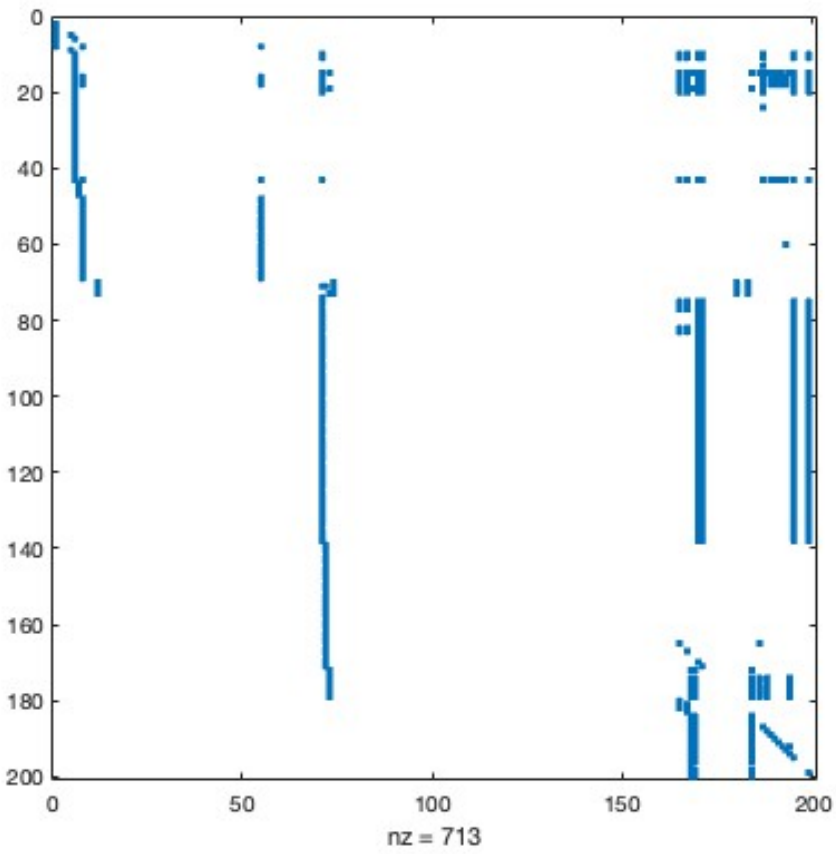


Figure 3: www.atletica.me Page Connections Graph

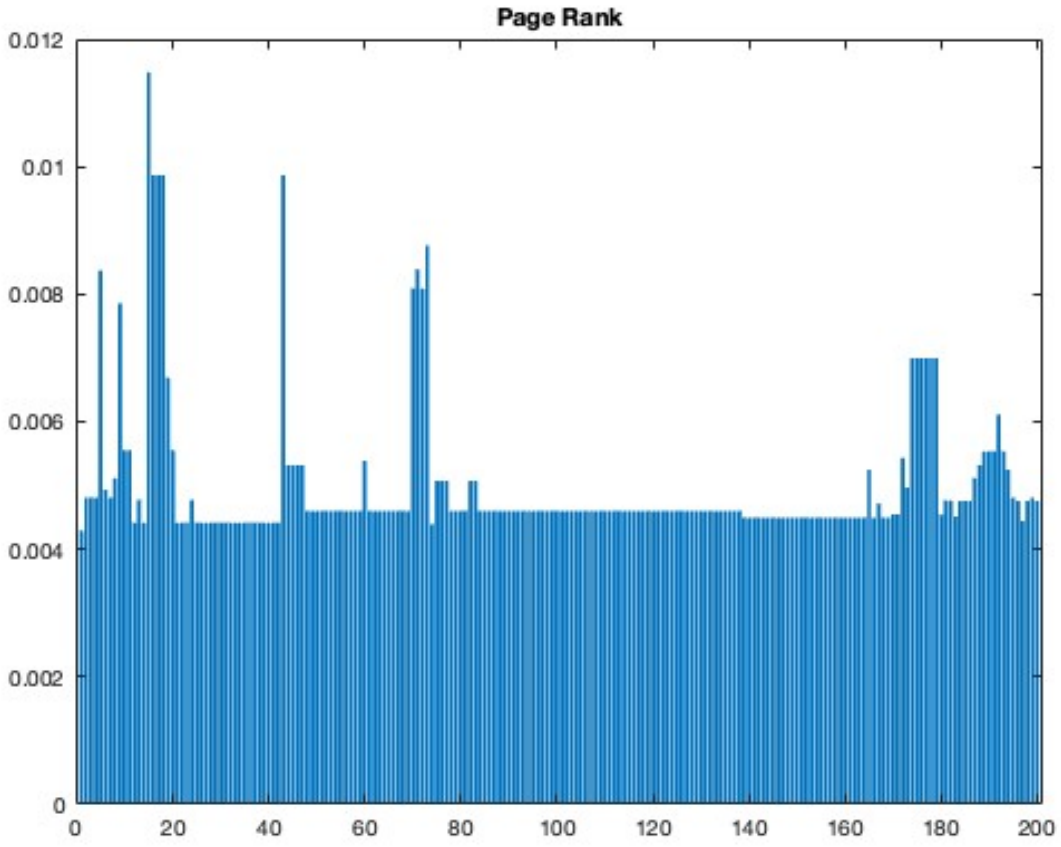


Figure 4: www.atletica.me URLs PageRank Distribution Graph

Rank	Scanning position	PageRank	In	Out	Url
1	15	**0.0115**	20	0	https://static.xx.fbcdn.net
2	16	**0.0099**	16	0	https://www.facebook.com/help/787040499275067
3	17	**0.0099**	16	0	https://www.facebook.com/help/2579891418969617
4	18	**0.0099**	16	0	https://www.facebook.com/help/contact/419859403337390
5	43	**0.0099**	16	0	https://edge-chat.facebook.com/mqtt/pull
6	73	**0.0088**	5	11	https://www.facebook.com/settings/ads
7	71	**0.0084**	6	75	https://www.facebook.com/help/1561485474074139
8	5	**0.0084**	2	2	https://4clubs.atletica.me
9	70	**0.0081**	4	0	https://static.xx.fbcdn.net/rsrc.php/v3/yQ/r/6ri-SA-YXh1.png
10	72	**0.0081**	4	34	https://mbasic.facebook.com/privacy/policies/cookies/printable

Clearly, here the situation is all different. The top 10 URLs table shows us that the URLs with the highest PageRank are almost all Facebook redirects. That means that basically every athlete profile has an outgoing link to Facebook. For this reason, as we can see, a lot of pages have almost the same PageRank.

Now, let's take a look at the first graph. Completely different from before, it shows a pattern of lines and little squares, and that means that the pages are not really linked with each other. In fact, usually an athlete profile has some outgoing links but not necessarily to other athlete profiles.

Finally, we have the graph that shows how the PageRanks are distributed and this is very interesting. For the other website we had big high and lows, but here we notice that a great amount of pages share almost the same PageRank. Of course there are some exceptions, but in big terms this means that a huge amount of pages have the same amount of ingoing links.

3. Connectivity matrix and subcliques [5 points]

The sky plot for the ETH500 data set has various submatrices that produce dense patches near the diagonal of the graph. For example, the first submatrix has indices around 80.

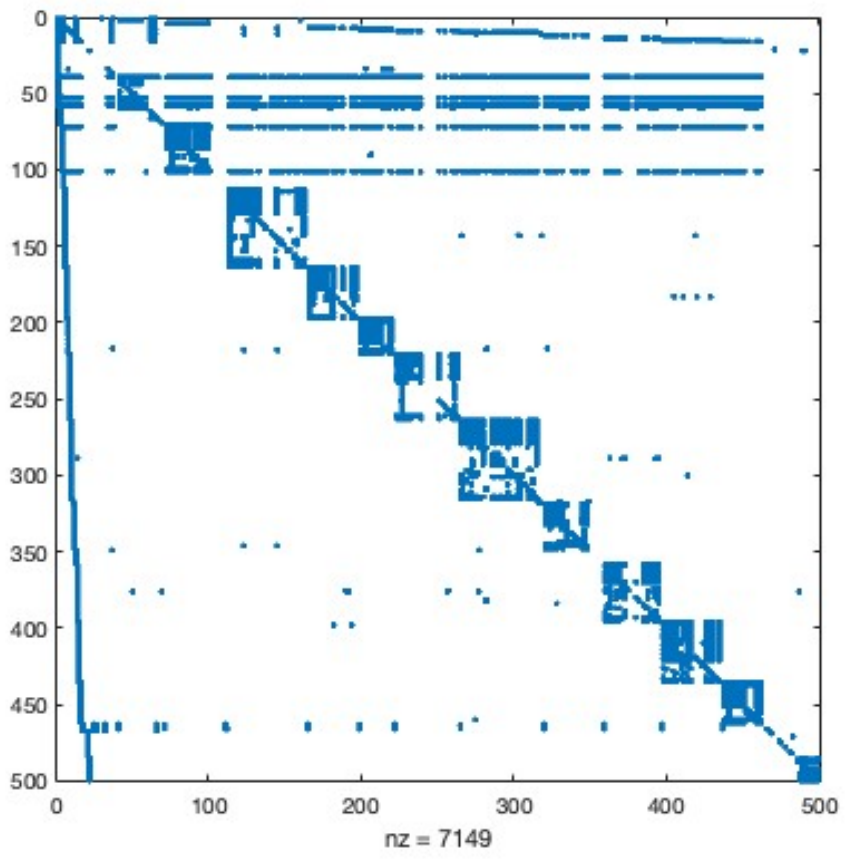


Figure 5: Spy plot of the Department of Computer Science (ETH Zurich) web graph

Let's understand that mathematically, a graph where all nodes are connected to each other is known as a clique.

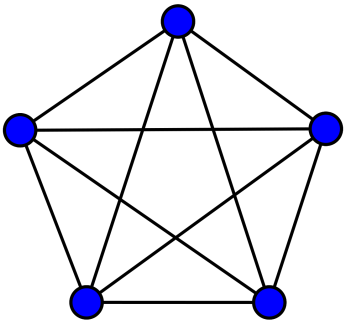


Figure 6: Example of a clique

Now, we want to identify the organizations within the ETH community that are responsible for these near cliques. To make the things easier, we will analyze the indices from 75 to 80, in order to understand the behaviour.

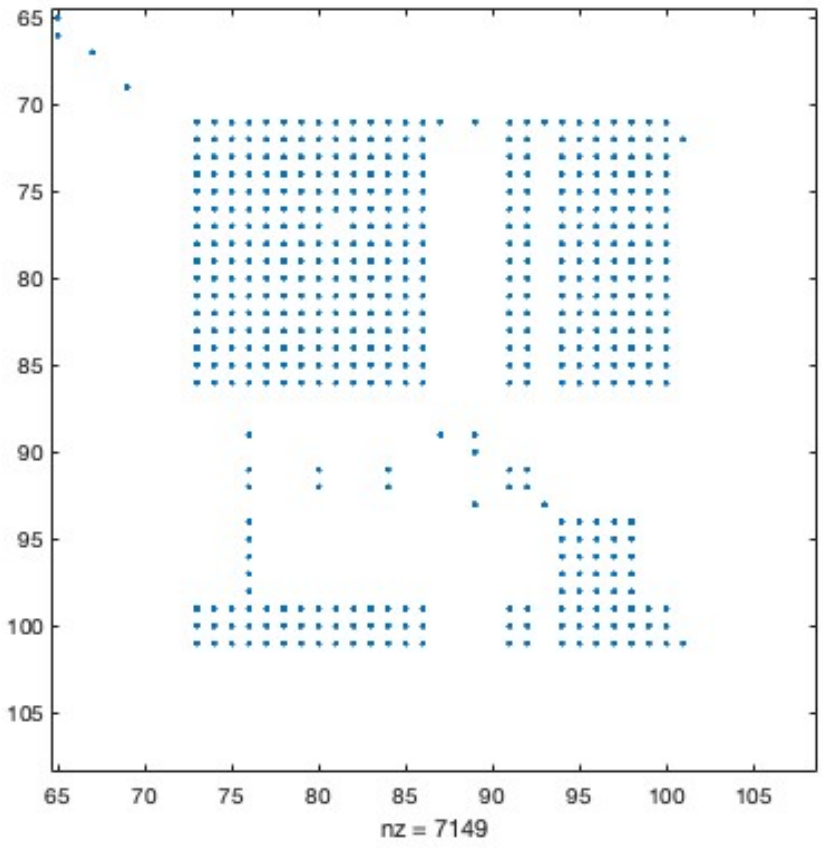


Figure 7: ETH500 Clique for indices around 80

I extracted the graph details of pages from 75 to 80, and I created a table to understand what happens. The results are as follows:

Pages that links to 80th	**Pages that links to 81st**	**Pages that links to 82nd**
(80,80)	(80,81)	(80,82)
(81,80)	(81,81)	(81,82)
(82,80)	(82,81)	(82,82)
(83,80)	(83,81)	(83,82)
(84,80)	(84,81)	(84,82)
(85,80)	(85,81)	(85,82)

Table 1: Links to the 80th, 81st, and 82nd page

Pages that links to 83rd	**Pages that links to 84th**	**Pages that links to 85th**
(80,83)	(80,84)	(80,85)
(81,83)	(81,84)	(81,85)
(82,83)	(82,84)	(82,85)
(83,83)	(83,84)	(83,85)
(84,83)	(84,84)	(84,85)
(85,83)	(85,84)	(85,85)

Table 2: Links to the 83rd, 84th, and 85th page

As we can easily notice, these 5 pages make a clique because they are all connected with each other. Let's now analyze what type of pages are we talking about:

Page Graph Number	**URL**
80	http://www.baug.ethz.ch/education/index$_E N$
81	http://www.baug.ethz.ch/services/index$_E N$
82	http://www.baug.ethz.ch/documents/index$_E N$
83	http://www.baug.ethz.ch/research/institutes/index$_E N$
84	http://www.baug.ethz.ch/education/diploma$_s tudies/index_E N$
85	http://www.baug.ethz.ch/infrastructure/index$_E N$

Table 3: Page 75-80 URLs

In order to answer the question, let's analyze these website pages that toghether create a clique. We can see that all of them correspond to www.baug.ethz.ch, which is the Departement Bau, Umwelt und Geomatik of the ETH Zurich. As we thought, these pages are various redirects from the home page of that website, and apparently they all have a link to each other.

4. Connectivity matrix and disjoint subgraphs [10 points]

The following image reports the graph of a six-node sub-graph of the Web, composed by two disjoint subgraphs.

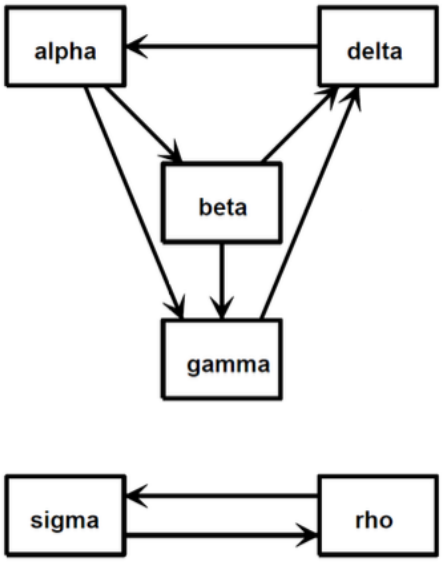


Figure 8: Web Graph

Let's consider U = {https://www.alpha.com, https://www.beta.com, https://www.gamma.com, https://www.delta.com, https://www.rho.com, https://www.sigma.com}.

In order to access the string contained in a cell we can simply write U(k), with k = 1, ..., 6 in this small web graph. For example, U(1) would correspond to the string "https://www.alpha.com".

We can generate the connectivity matrix by finding the pairs of indices (i,j) of the nonzero elements. For example, there is a link to beta.com from alpha.com and the (2,1) element of G is nonzero. Let's keep in mind that (i,j) is 1 if there is a link from page j to page i, which might seems confusing.

The eight connections are described by:

i = [2 3 3 4 4 1 6 5]
j = [1 1 2 2 3 4 5 6]

The sparse representation of the matrix is given by the command full(G):

$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

We can now compute the operations x = $\frac{(I-p*G*D)}{e}$ and x = $\frac{x}{norm(x,1)}$ to discover the PageRank of this tiny web.

First of all, we want to know the PageRank with the standard probability p = 0.85.

After calculations, we can finally obtain the PageRank graph and the URLs ranking:

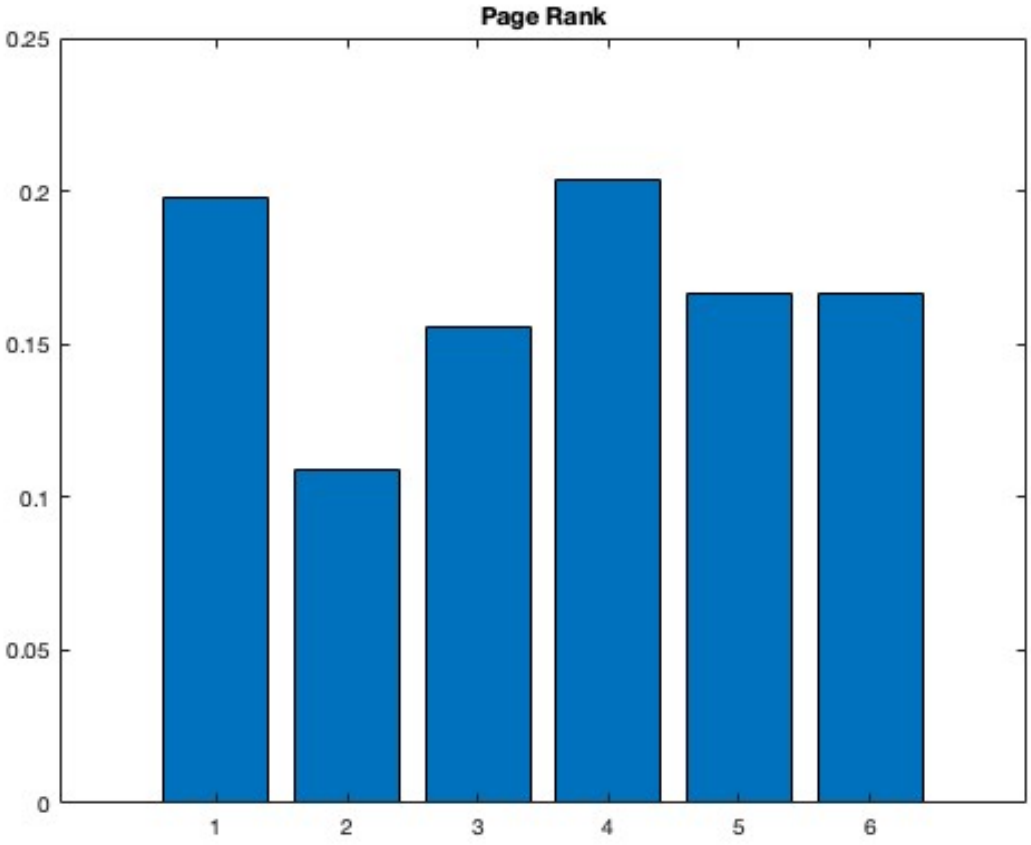


Figure 9: PageRank of the tiny web graph with p = 0.85

Rank	Scanning position	PageRank	In	Out	Url
1	4	**0.2037**	2	1	`https://www.delta.com`
2	1	**0.1981**	1	2	`https://www.alpha.com`
3	5	**0.1667**	1	1	`https://www.rho.com`
4	6	**0.1667**	1	1	`https://www.sigma.com`
5	3	**0.1556**	2	1	`https://www.gamma.com`
6	2	**0.1092**	1	2	`https://www.beta.com`

As we can see, www.delta.com is the page with the highest PageRank, even if it has the same ingoing and outgoing links of page www.gamma.com.

The reason of that is because the pages that links to www.delta.com have an highest PageRank than the other ones.

Let's now see what happens if we change the dumping factor p with values approaching 1. We need to remember that the damping factor p models the behavior of a typical web user, who is more likely to follow links but occasionally might choose to visit a random page. The random jump component $(1 - p)$ ensures that the algorithm does not get stuck in "dead ends" or "spider traps" (pages or sets of pages with no outgoing links). If p = 1, the random jump component is 0 and the algorithm could get stuck in some pages.

If p = 0 the PageRank would ignore the web's link structure completely, and every page would have the same rank, regardless of how many links it has.
If p = 1 the PageRank becomes entirely dependent on the link structure, without any random jump component.

In order to see the difference, we want to set values for p really close to 1.
Let's set, for example, p = 0.9:

x = [0.2005 0.1069 0.1550 0.2043 0.1667 0.1667]

We notice that the PageRank of the page www.alpha.com, for example, is increasing its value.
Let's now try with p = 0.99999:

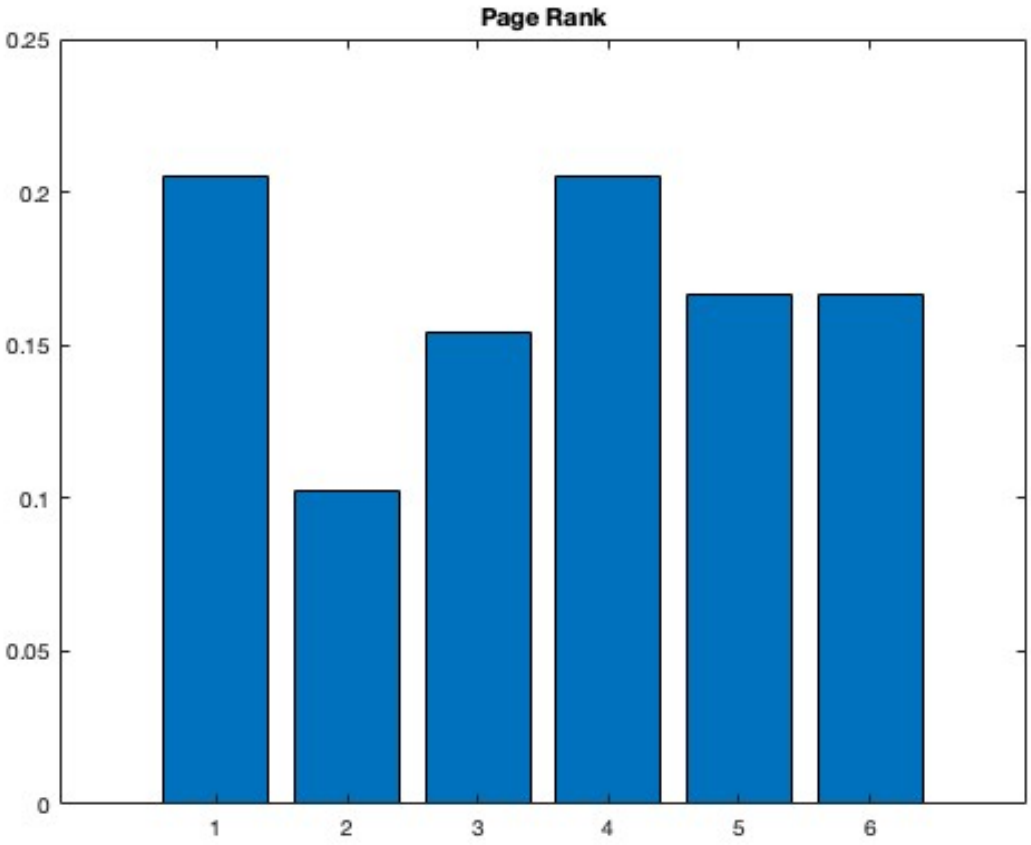


Figure 10: PageRank of the tiny web graph with p = 0.99999

Rank	Scanning position	PageRank	In	Out	Url
1	4	**0.2051**	2	1	`https://www.delta.com`
2	1	**0.2051**	1	2	`https://www.alpha.com`
3	5	**0.1667**	1	1	`https://www.rho.com`
4	6	**0.1667**	1	1	`https://www.sigma.com`
5	3	**0.1538**	2	1	`https://www.gamma.com`
6	2	**0.1026**	1	2	`https://www.beta.com`

As we can see, the PageRank of www.delta.com is now equal to the PageRank of www.alpha.com. What changed?

Basically, increasing the value of p close to 1, the PageRank becomes almost entirely dependent on the link structure, without any random jump component. For this reason, even if www.alpha.com has only one ingoing link, the page that links to it is www.delta.com, which is one of the page with the highest PageRank. Let's remember that the PageRank of a page depends on the number of pages that links to it and on the importance (PageRank) of those pages.

5. PageRanks by solving a sparse linear system [25 points]

The function pagerank(U,G) computes PageRanks by solving a sparse linear system. It then plots a bar graph and prints the dominant URLs. Now, we want to use other method to calculate the PageRank, such as the Power Method and the Inverse Iteration.

I created pagerank1.m by modifying pagerank.m to use the power method instead of solving the sparse linear system:

First of all, I added two key statements: G = p * G * D and z = ((1-p)*(c~=0) + (c==0))/n.

Then, I scripted the while loop:

```
disp('Using the Power Method implementation for PageRank calculation');

% Initialize the previous PageRank vector for comparison
previous_x = zeros(n,1);

% While loop for the power method iteration
while norm(x - previous_x, 2) > tol
    previous_x = x;  % Update the old vector
    x = G * x + e * (z * previous_x);  % Power iteration update
end
```

Figure 11: Modified part of pagerank1.m

An appropriate test for terminate the power iteration is to set a certain tolerance, in this case I chose a small number as tolerance (tol) = 1e-6 because this number is a good choice that helps to ensure convergence of the PageRank algorithm with a good balance of accuracy and efficiency. This value allows the iterative method to stop once the changes in the PageRank vector become sufficiently small.

In particular, norm(x - previous_x, 2) > 0 represents a condition used to check if the difference between the two vectors exceeds the chosen tolerance. We can say that norm(x - previous_x, 2) is the Euclidean distance between x and previous_x and, if the norm of the difference between x and previous_x is greater than the tolerance, the condition returns true, meaning that the difference is still significant and we have to continue.

After that, I created the script pagerank2.m, to use the inverse iteration. I added two key statements: A = p*G*D + e*z and alpha = 0.99, as requested, and then I coded the new loop, as follows:

```
disp('Using Inverse Iteration implementation for PageRank calculation');

% Initial settings
previous_x = zeros(n,1);
iterations = 0;

while norm(x - previous_x, 2) > tol && iterations < max_iter
    % Update the old vector
    previous_x = x;
    % Solve the linear system
    x = (alpha * I - A) \ previous_x;
    % Normalize the PageRank vector
    x = abs((x/norm(x,1)));
    iterations = iterations + 1;
end

fprintf('Number of iterations: %d\n', iterations);
```

Figure 12: Modified part of pagerank2.m

Once I have done both pagerank1.m and pagerank2.m, I calculated the PageRank of the tiny web graph with all the three different methods. The results are the following:

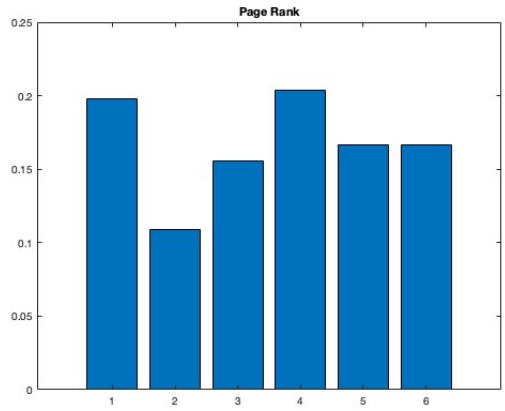


Figure 13: Tiny Web Plot Backslash Division

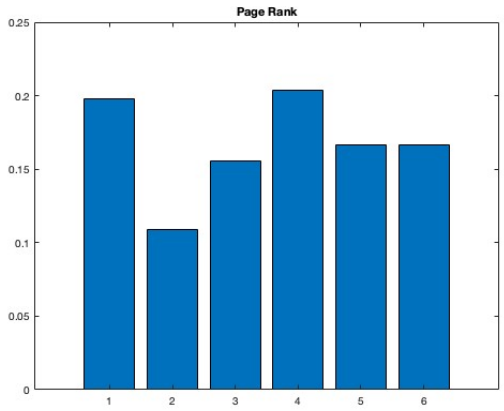


Figure 14: Tiny Web Plot Power Method

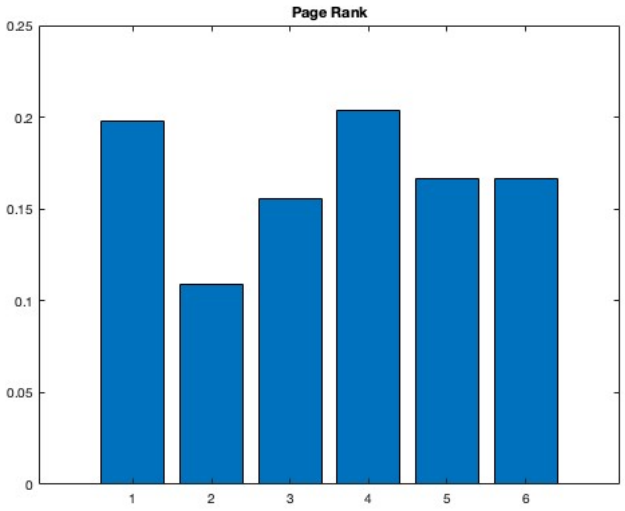


Figure 15: Tiny Web Plot Inverse Iteration

Those three graphs show the result of the PageRank of the tiny graph calculated with pagerank.m, pagerank1.m and pagerank2.m.

As we can see, the results are exactly identical, and that's a great thing because it means that we have done everything properly.

We now want to analyse the impact of alpha on the inverse iteration. Using the ETH500.mat example, I set alpha equal to 0.8, 0.9, 0.95 and 1 to see the different number of iterations that the four cases take until convergence.

Notice that I set 1e-6 as tolerance and 1000 as the maximum number of iterations:

Alpha	**Number of Iterations until Convergence**
0.8	6
0.9	1000 (reached max_iter)
0.95	13
1	2

As we can see, the inverse itaration is faster with the shift alpha = 1, and we can notice that in a weird way, it doesn't converge for alpha = 0,9, since it reaches the maximum number of iterations. As we clarified in class, there are some numerical problems with the newest versions of MatLab that make the iteration values going hugely negative sometimes. In order to avoid this problem, we had to calculate the absolute value of x, which works but it might cause some irregular results. Actually, the negative results could be fine, as long as the iterations converge to the closest eigenvalue, but in this case, with 0.9, it doesn't because probably it keeps oscillating between two eigenvalues.

However, I analyzed the PageRank value of the first 5 pages for each value of alpha, and we can see some interesting things:

Node Number	PageRank	In	Out	URL
57	0.1458	292	1	http://www.zope.org
466	0.0500	19	1	http://purl.org/dc/elements/1.1
39	0.0288	311	0	http://www.ethz.ch
58	0.0219	291	0	http://www.infrae.com
53	0.0212	288	0	http://www.cd.ethz.ch/services/web

Table 4: Top 5 PageRanks for Alpha = 1

Node Number	PageRank	In	Out	URL
57	0.1458	292	1	http://www.zope.org
466	0.0500	19	1	http://purl.org/dc/elements/1.1
39	0.0288	311	0	http://www.ethz.ch
58	0.0219	291	0	http://www.infrae.com
53	0.0212	288	0	http://www.cd.ethz.ch/services/web

Table 5: Top 5 PageRanks for Alpha = 0.95

Node Number	PageRank	In	Out	URL
482	0.2377	2	1	http://www.sunnysidegames.ch/portfolio/thefirm
466	0.0997	19	1	http://purl.org/dc/elements/1.1
57	0.0838	292	1	http://www.zope.org
39	0.0208	311	0	http://www.ethz.ch
58	0.0159	291	0	http://www.infrae.com

Table 6: Top 5 PageRanks for Alpha = 0.9

Node Number	PageRank	In	Out	URL
57	0.3165	292	1	http://www.zope.org
466	0.1030	19	1	http://purl.org/dc/elements/1.1
39	0.0223	311	0	http://www.ethz.ch
58	0.0174	291	0	http://www.infrae.com
53	0.0169	288	0	http://www.cd.ethz.ch/services/web

Table 7: Top 5 PageRanks for Alpha = 0.8

There are a lot of things to talk about. We can see that for alpha = 1 and alpha = 0.95 the result is identical, and that makes sense because they are pretty close shifts and we can assume that they are close to the dominant eigenvalue as well, which means that they converge to the same eigenvalue 1, which is the dominant one. That's why with the shift alpha = 1 we have the smallest number of iterations.

For alpha = 0.9, some weird things happened: because of the problem with convergency and absolute values, we notice that there are two websites with high PageRank that probably shouldn't be there, because of their low number of incoming and outcoming links. We can assume that using the absolute value made positive a very big negative value, and that created confusion. However, we can think that with alpha = 0.9 the iteration doesn't converge to any eigenvalue because it keeps oscillating between two eigenvalues, which are 0.8031, 0.8500 and 1.0000.

For alpha = 0.8, we can see that the PageRanks are heavily increased, but their ranking still make sense if we look at the links. This is probably because this shift alpha is further from the eigenvalue and in fact, it converges towards the eigenvalue 0.8031.

Lastly, I used my functions pagerank1.m and pagerank2.m (alpha = 0.99) to compute the PageRanks of web1.mat, web2.mat and web3.mat. The results on the convergence of the two methods for these subgraphs are interesting:

	Pagerank1.mat	**Pagerank2.mat**
Web1.mat	46 iterations 0.0281 0.0274 0.0235	5 iterations 0.0281 0.0274 0.0235
Web2.mat	40 iterations 0.0493 0.0267 0.0235	5 iterations 0.0493 0.0267 0.0235
Web3.mat	47 iterations 0.0350 0.0276 0.0262	5 iterations 0.0350 0.0276 0.0262

As we can see, the Inverse Iteration method takes a way less iterations to compute the same result. In the table I put the top 3 of PageRank higher values to make it shorter, but the whole PageRank plot is the same.

We understand that the inverse iteration method often converges faster than the power method, especially for finding the dominant eigenvector, which is the case of PageRank calculations. Also, if the shift (a constant added to the diagonal elements of the matrix) alpha is pretty close to the eigenvalue, it will take even less time. This allows the method to target specific eigenvalues more efficiently and faster.

A disadvantage of the power method is that not always the largest eigenvalue is significantly larger than the other eigenvalues (for rapid convergence). If there are close eigenvalues, the convergence can slow down significantly. On the other side, inverse iteration doesn't depend on that difference between eigenvalues as heavily. If we choose a good shift, it can converge quickly to the desired eigenvector, even if the eigenvalues are close in value.

On the other hand, the power method is simpler to implement than inverse iteration. It requires only matrix-vector multiplications instead of solving a system of equations at each iteration, and that makes it computationally less costly especially when the matrix is large. In contrast, inverse iteration often requires solving a linear system for each iteration, which can be computationally more expensive.

6. Graph perturbations [15 points]

Each node in the web graph has its role in the computation of the PageRank. However, some nodes have more weight than others in this process, and their presence or absence may impact in a more or less significant way the final result. The key aspect that needs to be taken into account for assessing the influence of a node is the presence of links and connections. Indeed, influential nodes are expected to have plenty of incoming and outgoing links, and the larger the number of pages that link to a node, the higher its influence is expected to be.

Let's consider the web graph in file WIKI450.mat, we now want to find the 5 nodes with the highest degree centrality, including them in an ordered list, and showing the node index, the link of the web-page and the degree centrality. The script is the following:

```
load('WIKI450.mat')

c = sum(G,1);
r = sum(G,2);

% Finding the self loops
x = diag(G);

% c' (the transpose of c) converts the row vector c into a column vector so that it can be added to r.
degc = c' + r - x;

% Sort the degree centrality in descending order
[sortedDegc, sortedIdx] = sort(degc, 'descend');

% Display the sorted degree centralities and corresponding node indices
disp('Sorted degree centralities:');
disp(sortedDegc);
disp('Corresponding node indices:');
disp(sortedIdx);
```

And after running it, I get the following results, which I stored in a table to make it clearer:

Node Index	Degree Centrality	URL
3	743	https://www.wikidata.org/wiki/Special:EntityPage/Q5296
10	743	https://species.wikimedia.org/wiki/Main$_{Page}$
7	742	https://meta.wikimedia.org/wiki/Main$_{Page}$
12	741	https://www.wikidata.org/wiki/Wikidata:Main$_{Page}$
5	737	https://foundation.wikimedia.org/wiki/Home

Table 8: Node Index and Degree Centrality with Corresponding URLs

At this point, I created a copy WIKI450_ISO.mat of matrix WIKI450.mat such that the five nodes with the higher degree centrality do not have any incident edge. Here's the script:

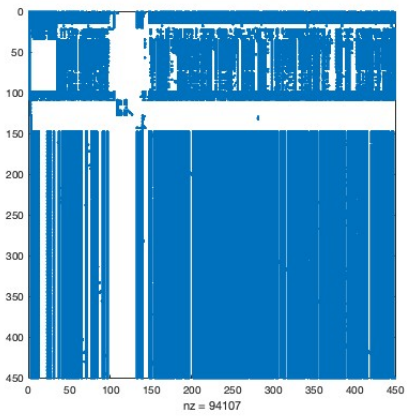
```
load('WIKI450_ISO.mat')

n = [3, 10, 7, 12, 5];

for i = 1:length(n)
    G(n(i), :) = 0;  % Remove all outgoing edges from node n(i)
    G(:, n(i)) = 0;  % Remove all incoming edges to node n(i)
end

spy(G);
```

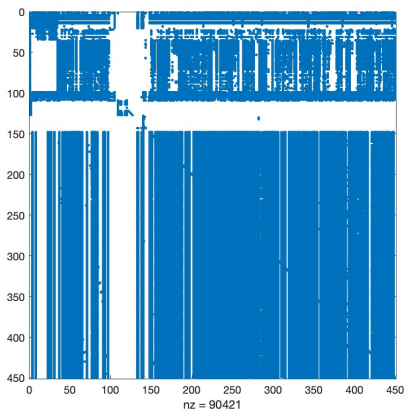
Let's see what is their degree centrality now:

Node Index	Degree Centrality	URL
3	0	https://www.wikidata.org/wiki/Special:EntityPage/Q5296
10	0	https://species.wikimedia.org/wiki/Main$_{Page}$
7	0	https://meta.wikimedia.org/wiki/Main$_{Page}$
12	0	https://www.wikidata.org/wiki/Wikidata:Main$_{Page}$
5	0	https://foundation.wikimedia.org/wiki/Home

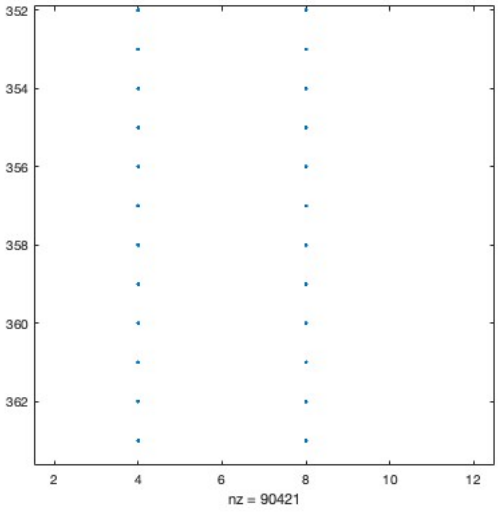
Obviously, after removing their edges, they appear as the last five nodes in the degree centrality list, sorted in a descending order. Actually, the program prints 445 nodes instead of 450 because those 5 nodes are kind of "unconnected", since we removed all the edges by changing the matrix values to 0. Naturally, their degree centrality is now 0.

To conclude this analysis, let's see something interesting. Even if we remove all the edges of the nodes with the higher degree centrality, we are not going to see any significant difference in the sky plot (but of course the nz changes). This is caused by the fact that the plot is huge and there is a very large number of connections, and removing some nodes will basically just remove a vertical line in the plot.





However, if we zoom in into the plot, we can see that there are no more vertical lines from the nodes 3, 10, 7, 12, 5.



Now, I compute the PageRank on the data in WIKI450.mat to compare the results with those obtained from the data in WIKI450_ISO.mat:

Node Number	PageRank	In	Out	URL
125	0.0218	9	1	https://www.facebook.com/creativecommons
109	0.0173	137	20	https://creativecommons.org/licenses/by-sa/4.0
127	0.0167	8	1	https://fontawesome.com
105	0.0165	411	28	https://www.mediawiki.org
104	0.0164	408	3	https://wikimediafoundation.org
101	0.0148	404	3	https://developer.wikimedia.org
122	0.0130	7	1	https://mail.creativecommons.org/subscribe
124	0.0130	7	1	https://mastodon.social/@creativecommons
100	0.0097	401	7	https://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Universal$_{C}ode_{o}f_{C}onduct$
4	0.0088	404	333	https://commons.wikimedia.org/wiki/Main$_{P}age$

Table 9: Top 10 PageRanks for WIKI500.mat

Node Number	PageRank	In	Out	URL
125	0.0247	9	1	https://www.facebook.com/creativecommons
109	0.0202	134	20	https://creativecommons.org/licenses/by-sa/4.0
105	0.0193	406	23	https://www.mediawiki.org
104	0.0191	403	3	https://wikimediafoundation.org
127	0.0189	8	1	https://fontawesome.com
101	0.0170	399	3	https://developer.wikimedia.org
122	0.0147	7	1	https://mail.creativecommons.org/subscribe
124	0.0147	7	1	https://mastodon.social/@creativecommons
100	0.0110	396	7	https://foundation.wikimedia.org/wiki/Special:MyLanguage/Policy:Universal$_{C}ode_{o}f_{C}onduct$
4	0.0104	399	328	https://commons.wikimedia.org/wiki/Main$_{P}age$

Table 10: Top 10 PageRanks for WIKI500_ISO.mat

As we can see in the result, the PageRank is similar but it's absolutely not the same.
For example, in the second table, we can notice that all the PageRank values are higher.
Even more interesting is to notice that a lot of pages have less ingoing and outgoing links, and that is explained by the fact that probably the edges we removed were linking to them.
However, the websites with higher PageRanks doesn't change, and that's explained by the fact that removing the nodes with higher degree centrality doesn't necessarily means that they have an high PageRank, because a lot of links might be from and to websites with relative low importance.

7. Introduction to HPC: Algorithmic performance [10 points]

In this section, we consider the 6 files included in the benchmark folder. They consist of sparse symmetric matrices of the same density (i.e., the proportion of non-zero elements is the same) of increasing size.

Using the implementations of the PageRank algorithm, I want to evaluate the performance of the three different approaches: backslash division, power method, and inverse iteration method. In particular, let's evaluate the time performance and the number of iterations required by each implementation to converge:

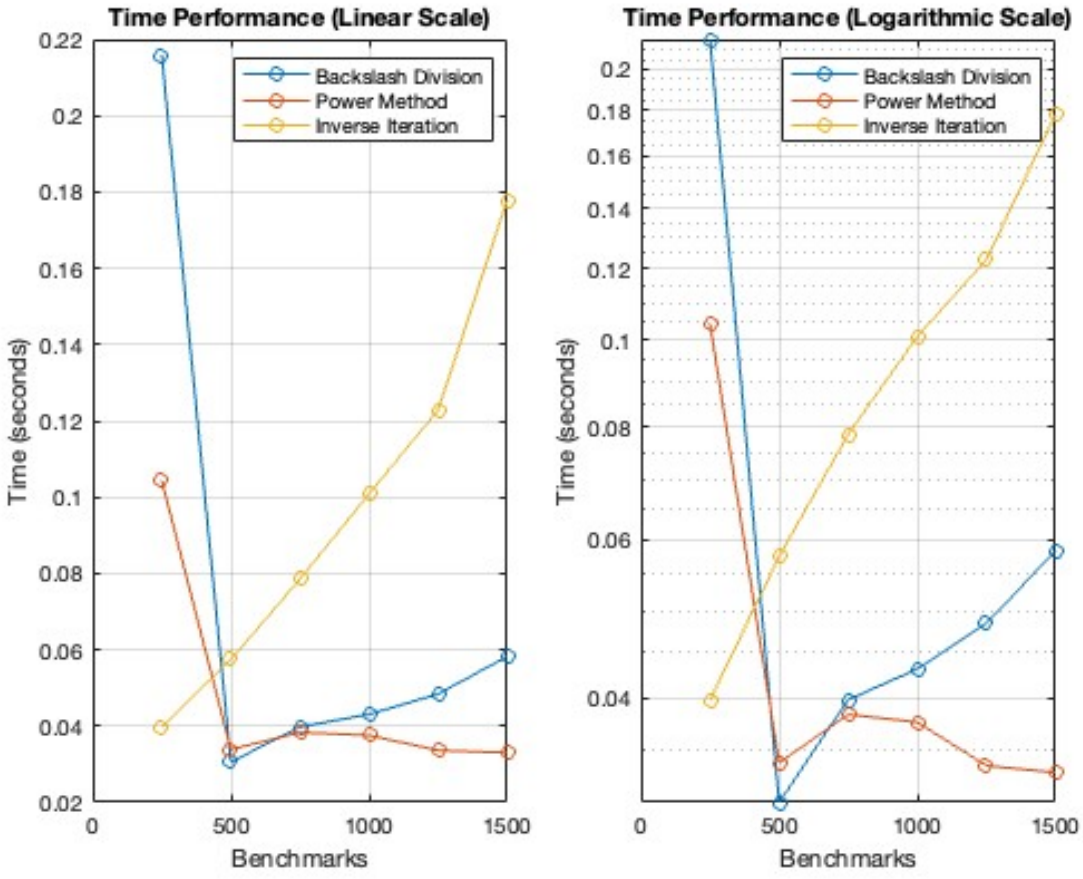


Figure 16: Time Performance of the three methods in two different scales

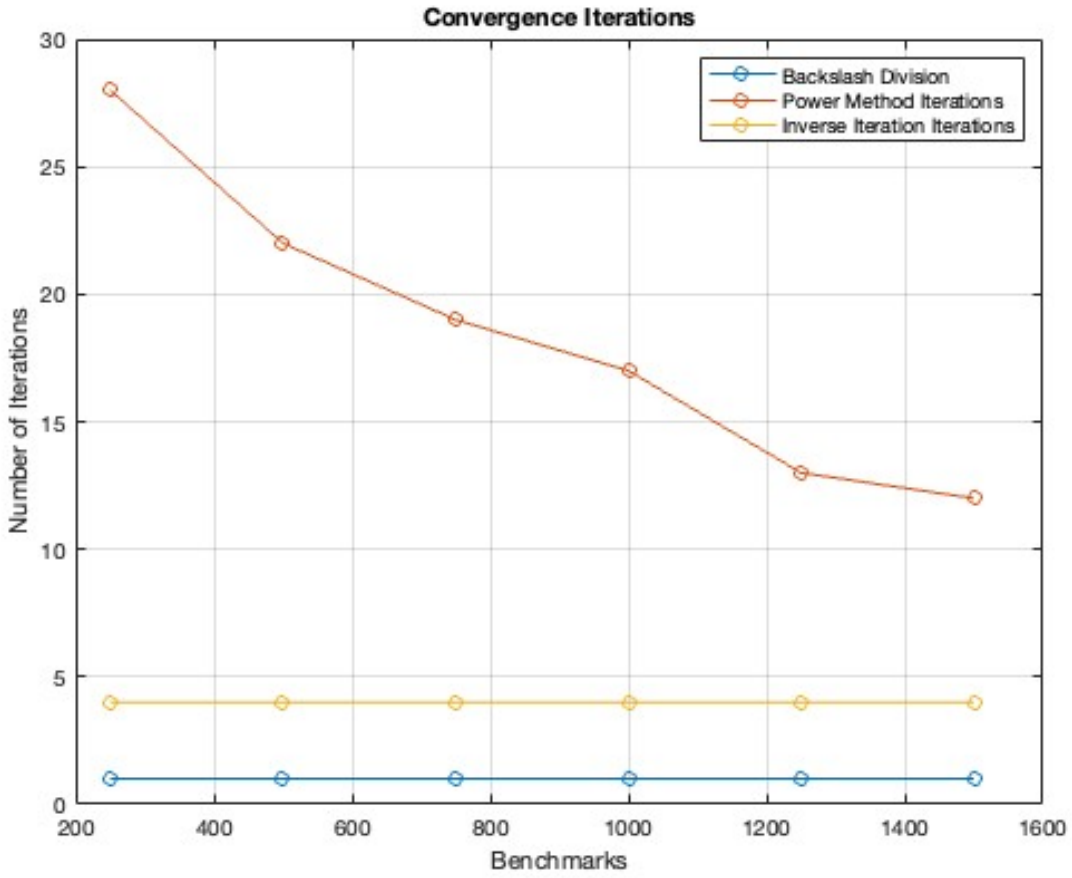


Figure 17: Iterations needed to reach Convergence for the three methods

The results are very interesting. We instantly notice that at the beginning the backslash division and the power method take more time to compute the PageRank and that is explained by the fact that these type of algorithms usually stop iterating once the vector stops changing from a very small threshold. If the tolerance is set pretty high, the algorithm may perform more iterations on smaller graphs to converge. In fact, in smaller graphs, small changes in the vector can lead to a larger number of iterations before reaching the stop condition. Also, in larger graphs, minor changes in individual node might become insignificant more quickly due to the greater number of nodes, leading to fewer iterations to converge.

Having said that, we now notice that larger the graph is, higher is the time required by the backslash division but more significantly the inverse iteration. This method takes more time to compute with larger graphs, despite having the same number of iterations. This happens because it involves solving a linear system of equations in each iteration. As the size of the matrix increases, the complexity of these operations grows.

On the other hand, the power method takes less time to compute when operating on larger graphs. How is that possible? We have to consider the difference between the largest eigenvalue (which is typically 1 for PageRank) and the second largest eigenvalue of the matrix. The larger is this difference, the faster the convergence happens. In PageRank matrices, as the size of the matrix increases, this difference might become larger. This means that the algorithm will converge faster, requiring fewer iterations to reach the solution. In fact, a larger difference increases the dominance of the principal eigenvector, and that makes the convergence quicker because the smaller eigenvalues contribute less and kind of "disappear" faster in the next iterations.

Sources:

Linear Algebra Notes - Università di Pisa

Power Method, Convergence Theorem, Rayleigh Quotient, Complexity

Basic assumptions for the Convergence of the Power Method

Inverse Iteration, Complexity