Università della Svizzera italiana | Faculty of Informatics | Institute of Computing CI

Numerical Computing **2022**

Students: Albert Cerfeda - Alessandro Gobbetti

Solution for Project 1

Due date: Wednesday, 12 October 2022, 23:59 AM

Numerical Computing 2022 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Julia). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

The purpose of this assignment[1] is to learn the importance of numerical linear algebra algorithms to solve fundamental linear algebra problems that occur in search engines.

PageRank Algorithm

1. Theory [20 points]

(a) **What are an eigenvector, an eigenvalue and an eigenbasis?**
An eigenvector is a nonzero vector that, when involved in a linear transformation gets stretched and scaled but stays on the same span.

An eigenvalue, denoted with symbol λ represents the factor of how much the eigenvector gets stretched along his direction.
The eigenvalue can be negative, as it means the resulting vectors inverts its direction after the linear transformation.

An eigenbasis is a diagonal matrix that has eigenvectors as its columns, and eigenvalues along its diagonal.
Given a matrix A and a linear transformation T, if there exists a vector v such that $Tv = \lambda v$, then v is an eigenvector of A and λ is an eigenvalue of A. If there exists a set of vectors V such that $T(V) = \lambda V$, then V is an eigenbasis of A.

[1]This document is originally based on a SIAM book chapter from *Numerical Computing with Matlab* from Clever B. Moler.

(b) **What assumptions should be made to guarantee convergence of the power method?**
We need to assume that the eigenvalue to which the power method converges is the dominant eigenvalue, and we also need to assume that the randomly-chosen initial vector has a component in the same direction as the eigenvector.
Also, to guarantee a faster convergence, λ_1 and λ_2 have to be distant, as the asymptotic error constant is $|\frac{\lambda_1}{\lambda_2}|$ meaning convergence is extremely slow if the two eigenvalues are close to each other.

(c) **What is the shift and invert approach?**
As mentioned previously, if the eigenvalues are close to each other convergence can be extremely slow.
The inverse iteration uses the shift and invert tecnique to speed up the convergence.

If the eigenvalues of A are λ_j, the eigenvalues of $A - \alpha I$ are $\lambda_j - \alpha$, and the eigenvalues of $B = (A - \alpha I)^{-1}$ are $\mu_j = \frac{1}{\lambda_j - \alpha}$

If we apply the power method to B we get an improve rate of $|\frac{\mu_2}{\mu_1}| = |\frac{\frac{1}{\lambda_2-\alpha}}{\frac{1}{\lambda_1-\alpha}}| = |\frac{\lambda_1-\alpha}{\lambda_2-\alpha}|$.
It is therefore in our best interests to find an α as close as possible to any eigenvalue.
The shift and invert approach is to choose α such that the improve rate is as small as possible.

(d) **What is the difference in cost of a single iteration of the power method, compared to the inverse iteration?**
Using the power method, each iteration involves a *matrix-vector multiplication*, while the *inverse iteration* requires solving a linear system. Solving a linear system is way more computationally expensive than a matrix-vector multiplication, having a fast convergence for inverse iteration is necessary in order for it to to be effective
this would be impossible when tackling larger issues such as Internet searching. Inverse iteration must be used for smaller problems or problems with special structure that enables fast direct methods.

(e) **What is a Rayleigh quotient and how can it be used for eigenvalue computations?**
The Rayleigh quotient is an improvement over the inverse iteration method for finding eigenvalues.
It guarantees very fast convergence, in most cases even cubically. To guarantee fast convergence, we change the value of α dynamically to be the Rayleigh quotient in each iteration.
Even though Rayleigh quotient iteration is more computationally expensive as it requires to refactor the matrix in each iteration, the cubic convergence makes it worthwhile.

2. Other webgraphs [10 points]

In this part of the assignment, the pageranks of three different subsets of the Web (Chess.com, RSI, and StackOverflow) has been computed. In each section we show the connectivity graph of the webgraph, the pagerank graph and the top 10 pages with the highest pagerank.

The experiments are all done by calling the same function `surfer` that access the desired home-page and then generates a 500-by-500 test case.

2.1. Webgraph 1: Chess.com

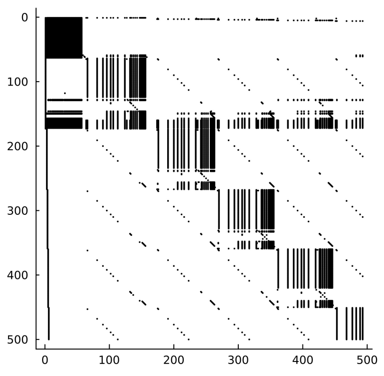

Figure 1: Spy plot of the Chess.com webgraph.

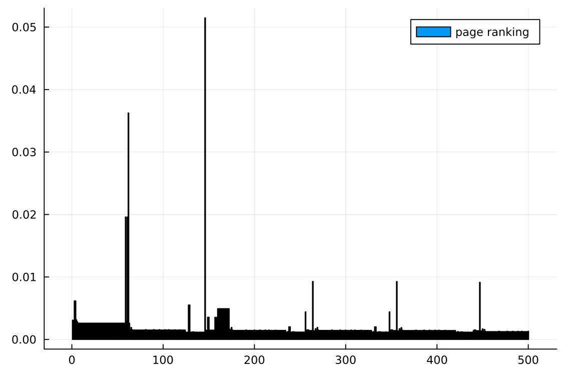

Figure 2: PageRank of Chess.com webgraph.

Index	Page-rank	In	Out	Url
146	0.05148	96	1	https://support.chess.com
62	0.0362528	58	1	https://www.instagram.com/wwwchesscom/
59	0.0196081	80	1	https://twitter.com/chesscom
61	0.0196081	80	1	https://www.twitch.tv/chess
264	0.00928716	19	1	https://twitter.com/chesscom_es
356	0.00927139	19	1	https://twitter.com/chesscom_fr
447	0.00915483	19	1	https://twitter.com/chesscom_de
3	0.0061748	84	174	https://www.chess.com/es
4	0.00616691	84	173	https://www.chess.com/fr
128	0.00551868	103	2	https://support.chess.com/collection/136-community-safety

Table 1: The ten most important entries in the PageRank of the Chess.com webgraph.

2.2. Webgraph 2: Radiotelevisione svizzera di lingua italiana (RSI)

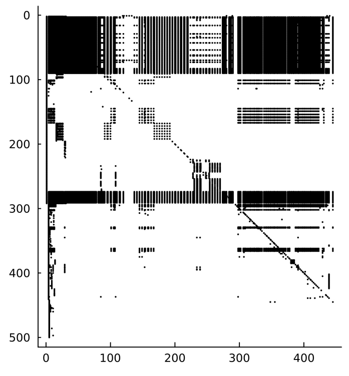

Figure 3: Spy plot of the RSI webgraph.

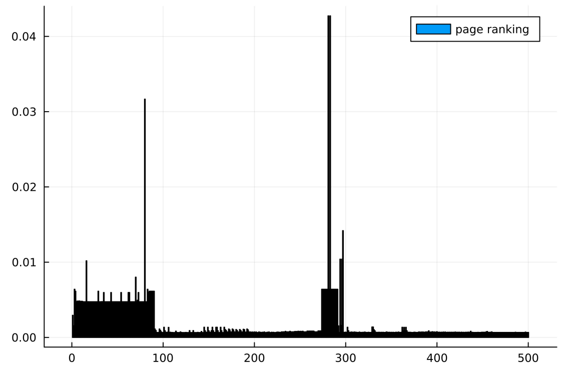

Figure 4: PageRank of RSI webgraph.

Index	Page-rank	In	Out	Url
281	0.0427305	242	1	`https://www.srgssr.ch/it/chi-siamo/organizzazione/piattaforma-per-segnalazioni-di-illeciti`
283	0.0427305	242	1	`https://www.corsi-rsi.ch/`
80	0.0316679	202	1	`https://boutique.rsi.ch/`
297	0.0141953	110	1	`https://www.instagram.com/rsinews/`
294	0.0104083	109	1	`https://www.youtube.com/channel/UC6UUxoC7DGUUlUHcJvulsfQ`
296	0.0104083	109	1	`https://twitter.com/rsinews`
16	0.0101878	241	139	`https://www.rsi.ch/sport/`
70	0.0080161	247	107	`https://www.rsi.ch/meteo/`
288	0.00640957	241	37	`https://www.rsi.ch/audio/`
284	0.00640957	241	106	`https://www.rsi.ch/la-rsi/Dichiarazione-sulla-protezione-dei-dati-10499633.html`

Table 2: The ten most important entries in the PageRank of the RSI webgraph.

2.3. Webgraph 3: StackOverflow

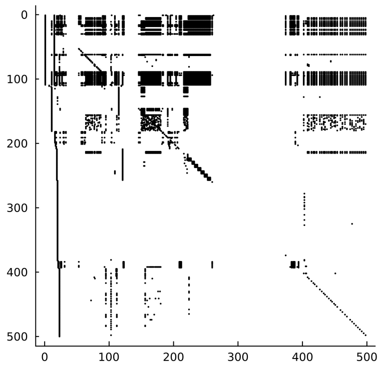

Figure 5: Spy plot of the StackOverflow webgraph.

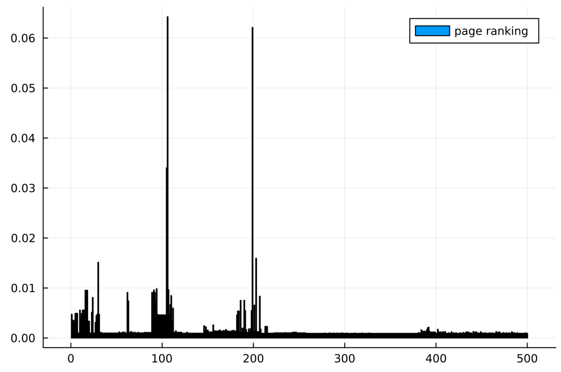

Figure 6: PageRank of chess.com webgraph.

Index	Page-rank	In	Out	Url
106	0.0642491	167	1	`https://twitter.com/stackoverflow`
199	0.0621094	28	1	`https://www.instagram.com/thestackoverflow/`
105	0.0340524	144	1	`https://www.facebook.com/officialstackoverflow/`
203	0.015951	4	1	`https://talent.stackoverflow.com/users/login`
30	0.0151374	183	23	`https://stackoverflow.co/teams`
94	0.0098377	184	47	`https://stackoverflow.com/legal/cookie-policy`
107	0.00963737	166	0	`https://linkedin.com/company/stack-overflow`
91	0.00959788	175	47	`https://stackoverflow.com/legal/privacy-policy`
16	0.00954374	182	36	`https://stackoverflow.co/`
17	0.00954374	182	26	`https://stackoverflow.co/talent`

Table 3: The ten most important entries in the PageRank of the StackOverflow webgraph.

2.4. Conclusions

As we can see, in every connectivity matrix there are many areas that are not connected to the rest of the webgraph and areas that are very dense.
Matematically speaking, there are many cliques.
This happens because communities or similar links that are really close and they all reference each other, whereas many web pages are not related to each other and therefore are not linked.

Taking the `Chess.com` connectivity graph as an example (1), we can see that the first 60 pages are strongly connected to each other, these are just the different homepages depending on the language of the user (e.g. `https://www.chess.com/es`). These pages just redirect to the main page and as it can be noticed in the pagerank table 1 are even more highly ranked than the main page.

3. Connectivity matrix and subcliques [5 points]

As we can see from the plot, there are several patches belonging to different organization.

Range	Dominant organization
73-100 range	baug.ethz.ch organization
113-130 range	math.ethz.ch
164-182 range	mavt.ethz.ch
198-220 range	biol.ethz.ch
221-263 range	chab.ethz.ch
264-315 range	math.ethz.ch
319-348 range	erdw.ethz.ch
350-356 range	hest.ethz.ch
359-373 / 385-393 range	usys.ethz.ch
396-416 / 426-431 range	mtec.ethz.c
436-461 range	gess.ethz.ch
488-500 range	bilanz.ch

4. Connectivity matrix and disjoint subgraphs [10 points]

Let us model and execute the PageRank algorithm for the six-node subset depicted in Figure 5:

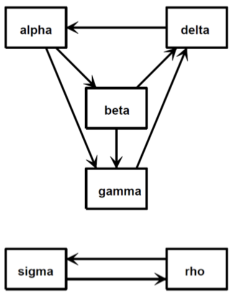

Figure 5: Another tiny Web.

```
n = 6;
i = [ 1 2 3 3 4 4 5 6 ]
j = [ 4 1 1 2 2 3 6 5 ]
G = sparse(vec(i), vec(j), vec(ones(Bool, 1, 8)));
U = ["alpha", "beta", "gamma", "delta", "rho", "sigma"];
p = .85;
function pagerank(U, G, p)
    // ...
end
pagerank(U,G,p)
```

4.1. What is the connectivity matrix G? Which are its entries?

A connectivity matrix is an $n \times n$ sparse logical matrix where n is the number of web pages. The position $(a, b) = 1$ if there is an outgoing link from b to a and 0 otherwise.
The connectivity matrix G for our six-node webgraph is:

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

where the columns from left-to-right and the rows top-to-bottom belong to the following in-order nodes: *"alpha"*, *"beta"*, *"gamma"*, *"delta"*, *"rho"*, *"sigma"*

4.2. What are the PageRanks if the hyperlink transition probability p assumes the default value of 0.85?

By running the page rank algorithm we get the following results:

index **i**	page-rank **x**	in **r**	out **c**	url **U**
4	0.203694	2	1	delta
1	0.19814	1	2	alpha
5	0.166667	1	1	rho
6	0.166667	1	1	sigma
3	0.155623	2	1	gamma
2	0.109209	1	2	beta

We notice how page *delta* is the highest-ranking one.

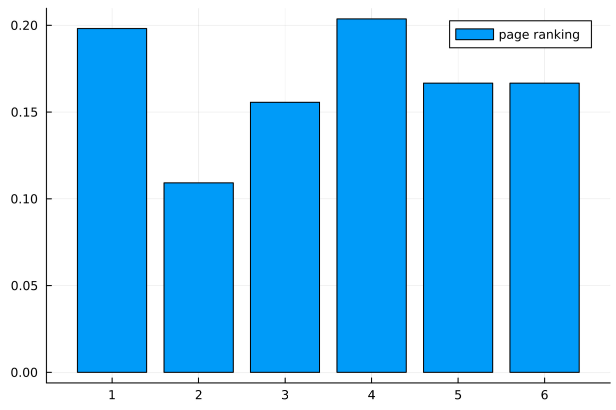

Figure 7: PageRank of 4.

4.3. Describe what happens with this example to both the definition of PageRank and the computation done by pagerank in the limit $p \to 1$

$p = .9999999$

index **i**	page-rank **x**	in **r**	out **c**	url **U**
4	0.20513	2	1	delta
1	0.20513	1	2	alpha
5	0.166664	1	1	rho
6	0.166664	1	1	sigma
3	0.153847	2	1	gamma
2	0.102565	1	2	beta

p is the probability that a random walk follows a link.
When it is set to 0.85, the number of incoming links has a higher weight when determining the

PageRank of any given page.
So, by setting it to 0, we notice how pages that have the same amount of in+out links have the same PageRank, as the weight of incoming links and outgoing links is basically the same.

5. PageRanks by solving a sparse linear system [40 points]

5.1. Power method

```julia
function pagerank1(U, G)
    # ...
    precision = 1e-16;
    prev_x = zeros(n);
    while any(x->abs(x)>=precision, prev_x-x)
        prev_x = x;
        x = G * x + e * (z * x);
    end
    # ...
end
```

At every iteration we compare the distances between elements of the newly-computed x and the previously-computed x to see if any of them are greater than the *precision* (i.e they did not converge yet).

5.2. Inverse iteration

```julia
function pagerank2(U, G)
    # ...
    precision = 1e-16;
    prev_x = zeros(n);
    while any(x->abs(x)>=precision, prev_x-x)
        prev_x = x;
        x = (alpha * I - A) \ x;
        x = x / sum(x);
    end
    # ...
end
```

Running the algorithms with precision 10^{-16} yields the following results:

	`pagerank1.jl`	`pagerank2.jl`
iterations	69	10

5.3. Impact of α on the inverse iteration

Running `pagerank2` with precision 10^{-16} yields the following results:

	$\alpha = 1.00$	$\alpha = .95$	$\alpha = .90$	$\alpha = .80$
iterations	10	72	*no convergence*	*no convergence*

5.4. PageRanks for three selected graphs

By running the algorithms with precision 10^{-16} and $alpha = 0.99$ on the three datasets and by timing them with package `BenchmarkTests` we get the following results (keep in mind the computing time is hardware-dependent):

iterations / computing time	`pagerank1.jl`	`pagerank2.jl`
Chess.com	168 / 4.199ms	14 / 40.936ms
RSI	169 / 6.162ms	14 / 40.648ms
StackOverflow	160 / 3.555ms	13 / 37.491ms

We notice that `pagerank1` which is using the *power method* performs several more iterations but, since the algorithm involves a computationally-cheaper matrix multiplication, the computing time is very short.
On the other hand, `pagerank2` which is using the *inverse iteration* method performs the least iterations but the computing time is longer, as it involves solving a linear system.
The effectiveness of the *inverse iteration* relies on how effectively we choose α.