Università della Svizzera italiana

Institute of Computing CI

**Numerical Computing** – Fall Semester 2024
**Lecturer:** Dr. Edoardo Vecchi
**Assistants:** Gianmarco De Vita, Samuele Pasini

# Project 1 – The PageRank Algorithm

Due date: Wednesday, 2 October 2024, 11:59 PM

The purpose of this project[1] is to learn the importance of numerical algorithms in the solution of fundamental linear algebra problems frequently occurring in search engines.

## 1. Linear Systems and the PageRank Algorithm

One of the reasons why Google$^{\text{TM}}$ is such an effective search engine is the PageRank$^{\text{TM}}$ algorithm developed by Google's founders, Larry Page and Sergey Brin, when they were graduate students at Stanford University. PageRank is determined entirely by the link structure of the World Wide Web. It is recomputed about once a month and does not involve the actual content of any Web pages or individual queries. Then, for any particular query, Google finds the pages on the Web that match that query and lists those pages sorting them according to the value of their PageRank. Surfing the Web—going from page to page by randomly choosing an outgoing link from one page to get to the next—can sometimes lead to dead ends (i.e., pages with no outgoing links) or to cycles around cliques of interconnected pages. To tackle this issue, the algorithm chooses, according to a predefined probability, a random page from the Web as next destination of the surfing process. This theoretical random walk is known as a *Markov chain* or *Markov process*. The limiting probability that an infinitely dedicated random surfer visits any particular page is its PageRank. Thus, a certain page has a high PageRank if other pages with high PageRank link to it.

Let $W$ be the set of Web pages that can be reached by following a chain of hyperlinks starting at some root page and let n be the number of pages in $W$. For Google, the set $W$ highly fluctuates over time and is not exactly known, but it was estimated to be about 30 billion in 2016.[2] Let $G$ be the $n$-by-$n$ connectivity matrix of a portion of the Web, that is $g_{ij} = 1$ if there is a hyperlink to page $i$ from page $j$ and zero otherwise. The matrix $G$ can be huge, but it is very sparse. Its $j$-th column shows the links on the $j$-th page. The number of nonzeros in $G$ is the total number of hyperlinks in $W$. Let $r_i$ and $c_j$ be the row and column sums of $G$:

$$r_i = \sum_j g_{ij} \quad \text{and} \quad c_j = \sum_i g_{ij}. \tag{1}$$

The quantities $r_j$ and $c_j$ are the *in-degree* and *out-degree* of the $j$-th page. Let $p$ be the probability that the random walk follows a link. A typical value is $p = 0.85$. Then $1 - p$ is the probability that some arbitrary page is chosen and $\delta = (1-p)/n$ is the probability that a particular random page is chosen. Let $A$ be a $n$-by-$n$ matrix with elements

$$a_{ij} = \begin{cases} p\; g_{ij}/c_j + \delta & c_j \neq 0, \\ 1/n & c_j = 0. \end{cases} \tag{2}$$

Notice that $A$ comes from scaling the connectivity matrix by its column sums. The $j$-th column is the probability of jumping from the $j$-th page to the other pages on the Web. If the $j$-th page is a dead end – i.e., it has no out-links – then we assign a uniform probability of $1/n$ to all the elements in its column. Most of the elements of $A$ are equal to $\delta$, the probability of jumping from one page to another without following a link. If $n = 4 \cdot 10^9$ and $p = 0.85$, then $\delta = 3.75 \cdot 10^{-11}$. Matrix $A$ is the transition probability matrix of the Markov chain. Its elements are all strictly between zero and one and its column sums are all equal to one. An important result from matrix theory, known as the *Perron-Frobenius theorem*, applies to such matrices: a nonzero solution of the equation

$$x = Ax \tag{3}$$

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

[2] van den Bosch, A., et a. (2016). Estimating search engine index size variability: a 9-year longitudinal study. *Scientometrics*, 107, 839-856.

exists and is unique to within a scaling factor. If this scaling factor is chosen so that

$$\sum_i x_i = 1, \tag{4}$$

then $x$ is the *state vector* of the Markov chain and corresponds to *Google's PageRank*. The elements of $x$ are all positive and less than one. Vector $x$ is the solution to the singular, homogeneous linear system

$$(I - A)x = 0 \tag{5}$$

For modest $n$, an easy way to compute $x$ in Matlab is to start with some approximate solution, such as the PageRanks from the previous month, or

```
x = ones(n, 1) / n
```

Then simply repeat the assignment statement

```
x = A * x
```

until the difference between successive vectors is within a specified tolerance. This is known as the power method and is about the only possible approach for very large $n$. In practice, the matrices $G$ and $A$ are never actually formed. One step of the power method would be done by one pass over a database of Web pages, updating weighted reference counts generated by the hyperlinks between pages. The best way to compute PageRank in Matlab is to take advantage of the particular structure of the Markov matrix. Here is an approach that preserves the sparsity of $G$. The transition matrix can be written

$$A = pGD + ez^{\mathsf{T}}, \tag{6}$$

where $D$ is the diagonal matrix formed from the reciprocals of the out-degrees,

$$d_{jj} = \begin{cases} 1/c_j & c_j \neq 0 \\ 0 & c_j = 0. \end{cases} \tag{7}$$

$e$ is the $n$-vector of all ones, and $z$ is the vector with components

$$z_j = \begin{cases} \delta & c_j \neq 0 \\ 1/n & c_j = 0. \end{cases} \tag{8}$$

The rank-one matrix $ez^{\mathsf{T}}$ accounts for the random choices of Web pages that do not follow links. The equation

$$x = Ax \tag{9}$$

can be written as

$$(I - pGD)x = \gamma e \tag{10}$$

where

$$\gamma = z^{\mathsf{T}} x. \tag{11}$$

We do not know the value of $\gamma$ because it depends on the unknown vector $x$, but we can temporarily take $\gamma = 1$. As long as $p$ is strictly less than one, the coefficient matrix $I - pGD$ is nonsingular and the equation

$$(I - pGD)x = e \tag{12}$$

can be solved for $x$. Then the resulting $x$ can be rescaled so that

$$\sum_i x_i = 1. \tag{13}$$

Notice that vector $z$ is not involved in this calculation. The following Matlab statements implement this approach.

```
c = sum(G,1);
k = find(c~=0);
D = sparse(k,k,1./c(k),n,n);
e = ones(n,1);
I = speye(n,n);
x = (I - p*G*D)\e;
x = x/norm(x,1);
```

The **power method** [1, 2] can also be implemented in a way that does not actually form the Markov matrix and so preserves sparsity. Compute

```
G = p*G*D;
z = ((1-p)*(c~=0) + (c==0))/n;
```

Start with

```
x = e / n;
```

Then repeat the statement

```
x = G*x + e*(z*x);
```

until $x$ settles down to several decimal places.
It is also possible to use an algorithm known as **inverse iteration** [1, 3] where one defines

```
x = e/n
A = p*G*D + e*z
```

and then the following statement is repeated until convergence given a suitable value $\alpha$.

```
x = (alpha*I - A)\x
x = x/norm(x,1)
```

For details of when and why this method works, please refer to the references cited above. However, we would like to briefly observe what happens when we take $\alpha = 1$ in the expression $(\alpha I - A)$. Since the resulting matrix $(I - A)$ is theoretically singular, with exact computation some diagonal elements of the upper triangular factor of $(I - A)$ would, in principle, be zero and this computation should fail. But with roundoff error, the computed matrix $(I - A)$ is probably not exactly singular. Even if it is singular, roundoff during Gaussian elimination will most likely prevent any exact zero diagonal elements. We know that Gaussian elimination with partial pivoting always produces a solution with a small residual, relative to the computed solution, even if the matrix is badly conditioned. The vector obtained with the backslash operation, $(I - A)\backslash x$, usually has very large components. If it is rescaled by its sum, the residual is scaled by the same factor and becomes very small. Consequently, the two vectors $x$ and $Ax$ equal each other to within roundoff error. In this setting, solving the singular system with Gaussian elimination blows up, but it blows up in exactly the right direction.
Figure 1 is the graph for a tiny example, with $n = 6$ instead of $n = 4 \cdot 10^9$. The pages on the Web are identified by strings known as uniform resource locators, or URLs. Most URLs begin with `https` because they use the hypertext transfer protocol. In Matlab, we can store the URLs as an array of strings in a cell array. The example in Figure 1 can be written as follows by using a 6-by-1 cell array:

```
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 $k = 1, \ldots, 6$ in this small web graph. Thus, $U(1)$ would, e.g., correspond to the string ”https://www.alpha.com”.
We can generate the connectivity matrix by specifying the pairs of indices $(i, j)$ of the nonzero elements. Because there is a link to beta.com from alpha.com, the $(2, 1)$ element of $G$ is nonzero. The nine connections are described by

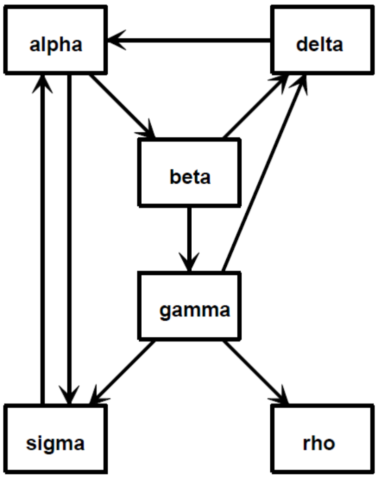

Figure 1: A tiny web graph.

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

A sparse matrix is stored in a data structure that requires memory only for the nonzero elements and their indices. This is hardly necessary for a 6-by-6 matrix with only 27 zero entries, but it becomes crucially important for larger problems. The statements

```
n = 6
G = sparse(i,j,1,n,n);
full(G)
```

generate the sparse representation of an $n$-by-$n$ matrix with ones in the positions specified by the vectors $i$ and $j$ and display its full representation.

```
     0     0     0     1     0     1
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     1     1     0     0     0
     0     0     1     0     0     0
     1     0     1     0     0     0
```

The statement

```
c = full(sum(G))
```

computes the column sums

```
 2     2     3     1     0     1
```

Notice that $c(5) = 0$ because the 5th page, labeled $rho$, has no out-links. The statements

```
x = (I - p*G*D)\e
x = x/norm(x,1)
```

solve the sparse linear system to produce

```
x =
0.3210
0.1705
0.1066
0.1368
0.0643
0.2007
```

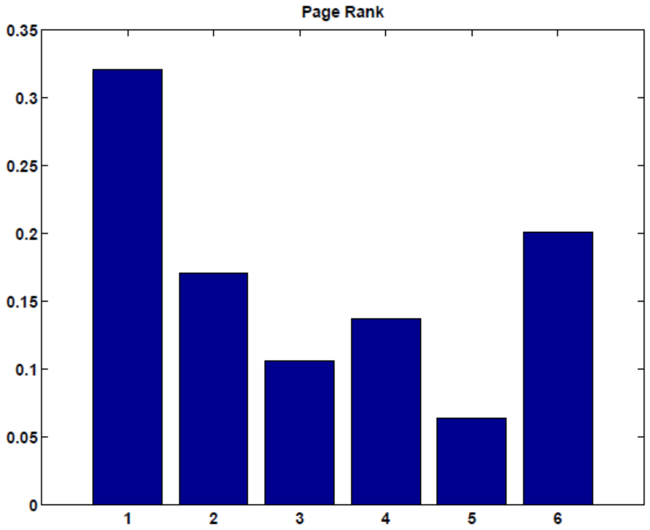

Figure 2: Page Rank for the tiny web graph

The bar graph of $x$ is shown in Figure 2. If the URLs are sorted in PageRank order and listed along with their in- and out-degrees, the result is

```
   page-rank   in      out    url
1  0.3210       2       2     https://www.alpha.com
6  0.2007       2       1     https://www.sigma.com
2  0.1705       1       2     https://www.beta.com
4  0.1368       2       1     https://www.delta.com
3  0.1066       1       3     https://www.gamma.com
5  0.0643       1       0     https://www.rho.com
```

We see that `alpha` has a higher PageRank than `delta` or `sigma`, even though they all have the same number of in-links. A random surfer will visit `alpha` over 32% of the time and `rho` only about 6% of the time.
For this tiny example with $p = .85$, the smallest element of the Markov transition matrix is $\delta = .15/6 = .0250$.

```
A =
  0.0250 0.0250  0.0250  0.8750  0.1667  0.8750
  0.4500 0.0250  0.0250  0.0250  0.1667  0.0250
  0.0250 0.4500  0.0250  0.0250  0.1667  0.0250
  0.0250 0.4500  0.3083  0.0250  0.1667  0.0250
  0.0250 0.0250  0.3083  0.0250  0.1667  0.0250
  0.4500 0.0250  0.3083  0.0250  0.1667  0.0250
```

We can notice that all the columns of matrix $A$ sum to one, thus agreeing with the definition of left stochastic matrix.
This mini-project includes the Matlab file `surfer.m`. A statement like

```
[U,G] = surfer('https://www.xxx.zzz',n)
```

starts at a specified URL and tries to surf the Web until it has visited $n$ pages. If successful, it returns an $n$-by-1 cell array of URLs and an n-by-n sparse connectivity matrix. Surfing the Web automatically is a dangerous undertaking and this function must be used with care. Some URLs contain typographical errors and illegal characters. There is a list of URLs to avoid that includes .gif files and Web sites known to cause difficulties. Most importantly, surfer can get completely bogged down trying to read a page from a site that appears to be responding, but that never delivers the complete page. When this happens, it may be necessary to have the computer's operating system ruthlessly terminate Matlab. With these precautions in mind, you can use surfer to generate your own PageRank examples. The statement

```
[U, G] = surfer('https://inf.ethz.ch/',500);
```

accesses the home page of the Department of Computer Science at the Swiss Federal Institute of Technology in Zurich (ETH) and generates a 500-by-500 test case. The file `ETH500.mat` included in the code folder was generated by using this function in September 2014. In the following, it is indicated as the *ETH500 data set*. The statement

```
spy(G)
```

produces a spy plot (Figure 3) that shows the nonzero structure of the connectivity matrix. The statement

```
pagerank(U, G)
```

computes the page ranks, produces a bar graph (Figure 4) of the ranks, and prints the most highly ranked URLs in PageRank order. The highly ranked pages are

```
>> pagerank(U,G)
        page-rank   in  out  url
57        0.145755   292  1   http://www.zope.org
466       0.049996   19   1   http://purl.org/dc/elements/1.1
39        0.028766   311  0   http://www.ethz.ch
58        0.021863   291  0   http://www.infrae.com
53        0.021202   288  0   http://www.cd.ethz.ch/services/web
101       0.016601   234  6   http://www.hk.ethz.ch/index_EN
72        0.016473   235  0   http://www.ethz.ch/index_EN
486       0.007374   14   2   http://www.handelszeitung.ch
463       0.006896   17   0   http://ns.adobe.com/xap/1.0
```

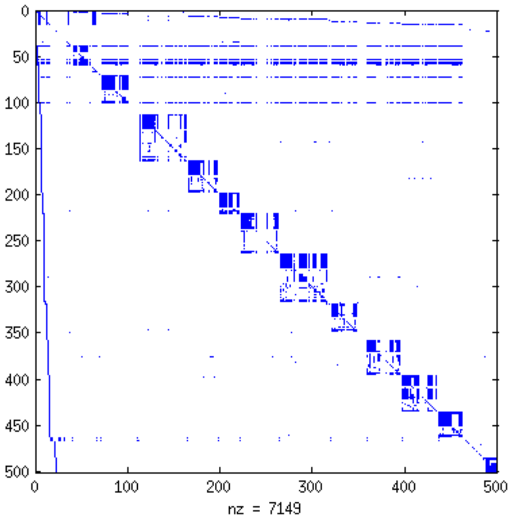

Figure 3: Spy plot of the Department of Computer Science (ETH Zurich) web graph.

## Complete the following tasks [85 points]:

### 0. Preliminary: Read "A First Course on Numerical Methods"

Read chapter 8 from the textbook [1], in order to gain a better understanding of the topic summarized above.

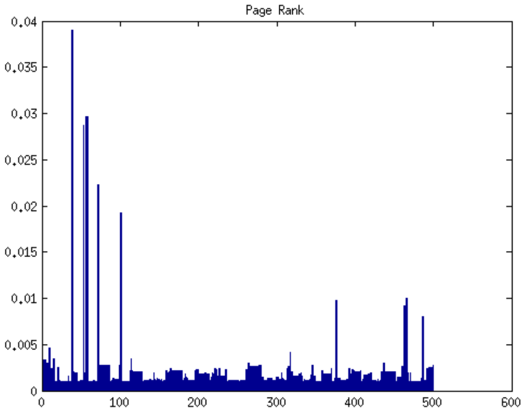

Figure 4: PageRank of the Department of Computer Science (ETH Zurich) web graph.

## 1. Theoretical questions [15 points]

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

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

(c) What is the shift and invert approach?

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

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

## 2. Other webgraphs [5 points]

Use `surfer.m` and `pagerank.m` to compute PageRanks for some subset of the Web of your choice. Do you see any interesting structure in the results (e.g., cliques, see next question)? Report on two PageRanks for different webgraphs by showing the connectivity matrix, the PageRanks, and the ten most important entries in the graph.

## 3. Connectivity matrix and subcliques [5 points]

The connectivity matrix for the ETH500 data set (`ETH500.mat`) has various small, almost entirely nonzero, submatrices that produce dense patches near the diagonal of the spy plot. You can use the zoom button to find their indices. The first submatrix has, e.g., indices around 80. Mathematically, a graph where all nodes are connected to each other is known as a clique. Identify the organizations within the ETH community that are responsible for these near cliques.

## 4. Connectivity matrix and disjoint subgraphs [10 points]

Figure 5 reports the graph of a tiny six-node subset of the Web. Unlike the example reported in Figure 1, in this case there are two disjoint subgraphs.

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

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

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

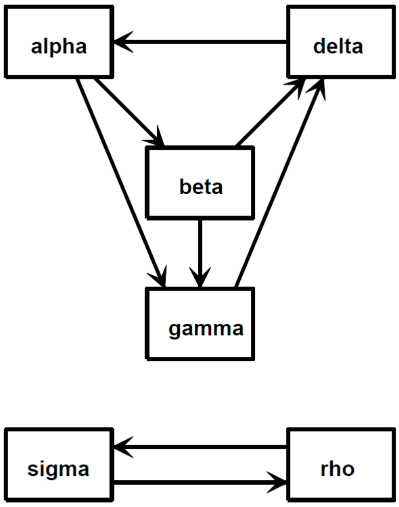

Figure 5: Another tiny Web.

## 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.

1. Create `pagerank1.m` by modifying `pagerank.m` to use the power method instead of solving the sparse linear system. The key statements are

   ```
   G = p*G*D
   z = ((1-p)*(c~=0) + (c==0))/n;
   while termination_test
     x = G*x + e*(z*x)
   end
   ```

   What is an appropriate test for terminating the power iteration?

2. Create `pagerank2.m` by modifying `pagerank.m` to use the inverse iteration. The key statements are

   ```
   while termination_test
     x = (alpha*I - A)\x
     x = x/norm(x,1)
   end
   ```

   Use your functions `pagerank1.m` and `pagerank2.m` (set $\alpha = 0.99$) to compute the PageRanks of the six-node example presented in Figure 1. Make sure you get the same result from each of your three functions.

3. We now want to analyse the impact of $\alpha$ on the inverse iteration. Using the ETH500 example, set $\alpha$ equal to $0.8, 0.9, 0.95$ and $1$. Comment on the different number of iterations the four cases take until convergence. Analyse your results and explain what you observe.
   *Hint:* Check your solution $x$ for all 4 cases. Are they always the same?

4. Use your functions `pagerank1.m` and `pagerank2.m` (set $\alpha = 0.99$) to compute the PageRanks of three selected graphs (`web1.mat`, `web2.mat` and `web3.mat`). Report on the convergence of the two methods for these subgraphs and summarize the advantages and disadvantages of the power method implemented in `pagerank1.m` against the inverse iteration in `pagerank2.m`.

### 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.

**Degree centrality** Given the web graph in file `WIKI450.mat`, find the 5 nodes with the highest degree centrality.[3] Include them in an ordered list, and show the node index, the link of the web-page and the degree centrality.

**Isolate the nodes** Now, take the top 5 nodes with the highest degree centrality found before, and create a copy `WIKI450_ISO.mat` of matrix `WIKI450.mat` such that the five nodes do not have any incident edge. What is their degree centrality now? Visualize side by side the original matrix and the one you modified. Is there any visible change in the spy plot?

**Impact of the nodes** Compute the PageRank on the data in `WIKI450.mat` and compare the results with those obtained from the data in `WIKI450_ISO.mat`. Include the results of the top 10 pages by rank in the report. Do you see any difference? Which could be the reasons behind this new result? Briefly describe them.

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

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 your own implementations of the PageRank algorithm, evaluate the performance of the three different approaches, namely backslash division, power method, and inverse iteration method. In particular, we are interested in evaluating the time performance, i.e., the time elapsed until the PageRank calculation converges to a solution (for this purpose, check out the MATLAB functions `tic()` and `toc()`), as well as the number of iterations required by each implementation to converge. Plot the results, side by side on a standard scale and on a logarithmic scale. How does the curve grow? Did you expect this behavior? What factors do you think make the two graphs look different? Is there any advantage in using the linear scale or the logarithmic scale? When might the logarithmic scale be useful for plotting the result? Explain your reasoning and justify your answer, including references and links to any source you checked.

## Quality of the code and of the report [15 Points]

The highest possible score for each project is 85 points. and up to 15 additional points can be awarded based on the quality of your report and code (maximum possible grade: 100 points). Your report should be a coherent document, structured according to the template provided on iCorsi. If there are theoretical questions, provide a complete and detailed answer. All figures must have a caption and must be of sufficient quality (include them either as .eps or .pdf). If you made a particular choice in your implementation that might be out of the ordinary, clearly explain it in the report. The code you submit must be self-contained and executable, and must include the set-up for all the results that you obtained and listed in your report. It has to be readable and, if particularly complicated, well-commented.

[3] In graph theory and network analysis, centrality refers to indicators which identify the most important vertices within a graph. Potential applications include identifying the most influential person(s) in a social network, key infrastructure nodes in the Internet or urban networks, and super spreaders of a given disease. Here we are interested in the **degree centrality**, which is defined as the number of links incident upon a node (i.e., the number of edges that a node has). The degree centrality of a vertex $v$, for a given graph $G := (V, E)$ with $|V|$ vertices and $|E|$ edges, is defined as the numbers of edges of vertex $v$.

## Additional notes and submission details

Summarize your results and experiments for all exercises by writing an extended LaTeX report, by using the template provided on iCorsi. Submit your gzipped archive file (tar, zip, etc.) **on iCorsi strictly before the deadline** in the dedicated section and use the following standard naming: `project_1_lastname_firstname.zip` (or `tgz`). Submission by email or through any other channel will not be considered. Late submissions will not be graded and will result in a score of 0 points for that project. You are allowed to discuss all questions with anyone you like, but: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently. Please remember that plagiarism will result in a harsh penalization (0 points) for all involved parties and that the usage of generative AI, even for rephrasing, is strictly forbidden.

## Mastery check

After the submission, you are required to take part in a mastery check scheduled in the following week. For details concerning the procedure, please check the course organization.

## In-class assistance

If you experience difficulties in solving the problems above, you can receive in-class assistance either on Tuesdays (13:30-15:00, in room D1.14) or on Wednesdays (13:30-15:00, in room D0.03). Please refer to this schedule for any eventual change in the allocated room.

## References

[1] The power method and variants, Chapter 8: Eigenvalues and Singular Values, SIAM Book "A First Course on Numerical Methods", C. Greif, U. Ascher, pp. 219-229.

[2] Power iteration, `http://en.wikipedia.org/wiki/Power_iteration`

[3] Inverse iteration, `http://en.wikipedia.org/wiki/Inverse_iteration`