Homework of Eigenvalues and Eigenvectors Approximation

Computational Linear Algebra for Large Scale Problems
Politecnico di Torino

Hajali Bayramov - S288874
Hafez Ghaemi - S289963

June 5, 2021

Abstract

In this homework, we will delve into the topic of eigenvalues and eigenvectors approximation, and implement and analyze the Google's PageRank algorithm on a graph representing a small version of the world-wide web. In the following paragraphs, we will answer to the questions given in the homework instructions and present our results using *python* programming language. A file named *INSTRUCTIONS.txt* containing the explanations how to run the codes and reproduce the results presented in this report, is attached.

1 Problem Overview

The problem of ranking a large number of elements can be addressed using tools from numerical linear algebra, among which are numerical approximations of eigenvalues and eigenvectors. In particular, we will focus on the ranking problem related to Google's search engine. The algorithm that lies at the basis of this well-known web search engine is the Google's PageRank algorithm and it was defined by Cleve Moler as "The World's Largest Matrix Computation". The PageRank algorithm finds a vector that ranks the webpages according to the number of in and out degrees.

2 Methodology

In the following paragraphs, we will answer to the questions given in the homework instructions and present our results.

$*$ 1. Construct the adjacency matrix $\mathbf{G}$ of the given graph in a sparse storage format and graphically visualize its sparsity pattern.

The function *adj_mat* from *utils.py* receives as its input the path to the two-column graph connectivity data, and will return the adjacency matrix and its sparse format where:

$$g_{ij} := \begin{cases} 1 & \text{if there is an edge from the vertex j to the vertex i;} \\ 0 & \text{otherwise.} \end{cases}$$

The sparsity pattern is visualized using the *spy* method of *matplotlib* in figure 1.

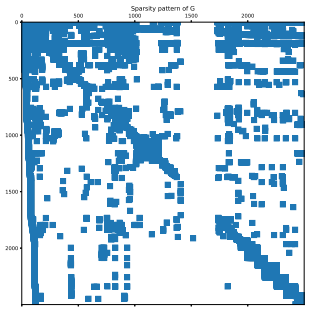

Figure 1: The sparsity pattern of $\mathbf{G}$

$*$ 2. What does the j-th column of $\mathbf{G}$ represent? What does the number of nonzeros of $\mathbf{G}$ represent?

The j-the column of $\mathbf{G}$ represents the out-links of node j. In other words, if the element $G_{ij} = 1$, there is a link from node j to node i, and if it is zero, there is no link from j to i. Based on this definition, the number of non-zeros in $\mathbf{G}$ represents the total number of edges in this directed graph (the total number of links in our web).

$*$ 3. Starting from the matrix $\mathbf{G}$, compute the in-degree and the out-degree of each web page. What do they represent?

The in-degrees and out-degrees of $\mathbf{G}$ are calculated by summing $\mathbf{G}$ over its columns and rows, and storing the results in vectors r (in-degs), and c (out-degs) respectively (using the function *in_out_degs*). The in-degree of a node represent the number of links to this page in the web, and the out-degree of that node represents the number of links in this page to the other pages of the web.

$**$ 4. Show that if the transition probability matrix is primitive than the Perron-Frobenius Theorem guarantees the existence and the uniqueness of the stationary probability distribution vector.

To prove this statement, we will first prove the following lemma:

Lemma 2.1. *The largest eigenvalue of a stochastic matrix is 1.*

Proof. First, if $\mathbf{A}$ is a stochastic matrix, then $A\vec{1} = \vec{1}$ (where $\vec{1}$ is the vector with all elements equal to 1). This is trivial because each row (or column) of $\mathbf{A}$ sums to 1. This proves that 1 is an eigenvalue of $\mathbf{A}$ (The definition of eigenvalue is $Av = \lambda v$). Second, suppose there exists an eigenvalue where $|\lambda| > 1$ and a non-zero eigenvector x such that $Ax = \lambda x$. Let x_i be the largest element of x. Since any scalar multiple of x will also satisfy this equation we can assume, without loss of generality, that $x_i > 0$. Since the rows of $\mathbf{A}$ are non-negative and sum to 1, each entry in x

is a convex combination (all coefficients are non-negative and they sum to one) of the elements of x. Thus, no entry in λx can be larger than x_i. But since $\lambda > 1$, we will have that $x_i > x_i$ which is a contradiction. Therefore, we have proved that the largest eigenvalue (or its absolute value) of A is 1. □

Now, we can easily use the lemma 2.1, to prove the statement. Because our transition probability matrix is primitive, based on the Perron-Frobenius theorem, the largest eigenvalue (which according to the lemma is 1) is unique. Also, this theorem says that the only eigenvector with all-positive components is the one corresponding to this largest eigenvalue. Because of that, we can normalize this eigenvector (v) such that its components sum to one, and all of them be between zero and one. This way, the vector v will become a probability distribution, that is unique and is the eigenvector corresponding to the eigenvalue $\lambda = 1$ for the matrix $\mathbf{P}$. This completes the proof, because by the definition of eigenvalue we can write: $P - 1 \times Iv = 0$ which is equivalent with $Pv = v$, and we have shown the existence and uniqueness of v.

∗ 5. Construct the hyperlink matrix $\mathbf{M}$ of the given graph in a sparse storage format and graphically visualize the sparsity pattern of the matrix $\mathbf{M}$ and compare it with that of $\mathbf{G}$.

The hyperlink matrix M defined as:

$$m_{ij} := \begin{cases} \frac{g_{ij}}{c_j} & \text{if } c_j \neq 0, \\ 0 & \text{if } c_j = 0. \end{cases}$$

is constructed using *hyperlink_mat* function. The sparsity pattern of M is shown in figure 2.

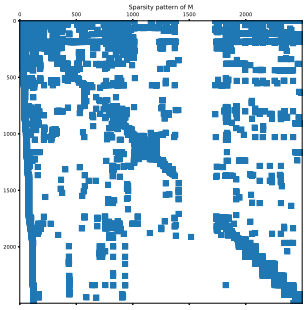

Figure 2: The sparsity pattern of $\mathbf{M}$

It can be seen that the sparsity pattern of $\mathbf{M}$ is the same as $\mathbf{G}$ which is reasonable since based on the definition of $\mathbf{M}$, exactly the same elements of $\mathbf{M}$ and $\mathbf{G}$ are zero. This matrix divides equally the weight of each node between its outgoing links, and gives no value to the other nodes.

∗ 6. Is our hyperlink matrix a stochastic matrix? Do you think that in general, given a dataset W describing a portion of the Web, the hyperlink matrix will be stochastic? Motivate your answer. Do you think that this kind of matrix could be suitable for describing our Markov chain?

No, this hyperlink matrix is not stochastic. By definition, each column (or row) of a stochastic matrix should represent a probability distribution (they should be between zero and one and they should sum to one). However, as a counterexample, if we have a node with no outgoing links, the whole i-th column will be zero and cannot be a probability distribution. Furthermore, if we do not include a random "teleport" probability in our web, these nodes will become dead-ends. So, because our hyperlink matrix is not stochastic, it is not suitable for describing a Markov chain.

∗ 7. Construct the modified hyperlink matrix $\tilde{\mathbf{M}}$ in a sparse storage format.

The matrix $\tilde{\mathbf{M}}$ is constructed as follows by the function *mod_hyperlink_mat*:

$$\tilde{m}_{ij} := \begin{cases} \frac{g_{ij}}{c_j} & \text{if } c_j \neq 0, \\ \frac{1}{n} & \text{if } c_j = 0. \end{cases}$$

The sparsity pattern of this matrix is shown in figure 3:

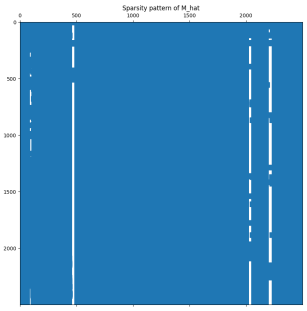

Figure 3: The sparsity pattern of $\tilde{\mathbf{M}}$

It can be seen that this modified matrix is not sparse anymore, because we have filled the zero values of $\mathbf{M}$ with $\frac{1}{n}$.

∗ 8. Is the modified hyperlink matrix $\tilde{\mathbf{M}}$ a stochastic matrix?

Yes, this matrix is stochastic. Specifically, we are modifying $\mathbf{M}$ in a way that $\tilde{\mathbf{M}}$ becomes stochastic. To do so, we have set each zero element of $\mathbf{M}$ to $\frac{1}{n}$. This way, $\mathbf{M}$ is rescaled so that each element is between 0 and 1 and each column of $\tilde{\mathbf{M}}$ will sum to 1. To illustrate this process, consider the counterexample we offered in question 6. By setting the zero elements of $\mathbf{M}$ to $\frac{1}{n}$ we have added an equal "invisible" connection weight from a node to all of the nodes which it does not have a link to, and augmented the out-link weights of each node so that they sum to one. The sum of column j is $s = c_{ij} \times \frac{1}{c_{ij}} = 1$ if c_j is not zero, and it is $s = n \times \frac{1}{n} = 1$ if c_j is zero.

Therefore, $\tilde{\mathbf{M}}$ will now be a square matrix with each column representing a probability distribution, and this is the definition of a stochastic matrix.

** 9. Prove theoretically the above result.

The proof was given in the lemma 2.1. However, we will provide it again:

Proof. First, if $\mathbf{A}$ is a stochastic matrix, then $A\vec{1} = \vec{1}$ (where $\vec{1}$ is the vector with all elements equal to 1). This is trivial because each row (or column) of $\mathbf{A}$ sums to 1. This proves that 1 is an eigenvalue of $\mathbf{A}$ (The definition of eigenvalue is $Av = \lambda v$). Second, suppose there exists an eigenvalue where $|\lambda| > 1$ and a non-zero eigenvector x such that $Ax = \lambda x$. Let x_i be the largest element of x. Since any scalar multiple of x will also satisfy this equation we can assume, without loss of generality, that $x_i > 0$. Since the rows of $\mathbf{A}$ are non-negative and sum to 1, each entry in x is a convex combination (all coefficients are non-negative and they sum to one) of the elements of x. Thus, no entry in λx can be larger than x_i. But since $\lambda > 1$, we will have that $x_i > x_i$ which is a contradiction. Therefore, we have proved that the largest eigenvalue (or its absolute value) of A is 1. □

* 10. Write a function that implement a suitable numerical method among those studied during the course (you can also use/combine more methods) to approximate the dominant eigenvalue of the modified hyperlink matrix $\tilde{\mathbf{M}}$. Check the result comparing it with the ones obtained using the Matlab/Python functions to numerically approximate eigenvalues. Compute the relative error. Discuss the results.

The function *power_method*, implements the power method for approximating the dominant eigenvalue of the modified hyperlink matrix. The convergence tolerance has been set to 10^{-16} and it measures the difference between the eigenvalues in two consecutive iterations of the algorithm. Also, the maximum number of iterations is set to 5000. The results of this algorithm are given below (note that the eigenvector is normalized so that its elements sum to one):

- The dominant eigenvalue using the power method: 1.00
- The dominant eigenvalue using the direct method (numpy): 1.00
- The convergence rate: $\left|\frac{\lambda_{2,actual}}{\lambda_{1,actual}}\right| = 1$
- Number of iterations needed for convergence: 3158
- The relative error of the approximated eigenvalue: $\frac{|\lambda_{1,actual} - \lambda_{2,appx}|}{\lambda_{1,actual}} = 4.05 \times 10^{-14}$

The small relative error between the eigenvalues and eigenvectors computed using the implemented method and "numpy", shows that the power method has successfully converged, and found the dominant eigenvalue which is 1, as expected.

* 11. Prove that the $\mathbf{A}$ is both stochastic and primitive so that the PageRank vector exists and it is unique.

Google matrix $\mathbf{A}$ has no negative elements and therefore, A^k for $k \geq 2$ is always positive. So, the Google Matrix is primitive. Furthermore, by construction, the Google matrix is stochastic, because each of its columns sum to one, and all of the elements are between zero and one, which means that each column constitutes a probability distribution. To show that each column's elements sum to one, we consider two cases. First, if $c_j = 0$, then we have n elements equal to $\frac{1}{n}$ which sum to 1. Furthermore, if $c_j \neq 0$, then because we have c_j elements with $g_{ij} = 1$ in column j, the sum will be equal to $s = c_j \times (\alpha \frac{1}{c_j}) + n \times \frac{1-\alpha}{n} = \alpha + (1 - \alpha) = 1$.

We have proved that the Google matrix is stochastic and primitive. Therefore, the PageRank vector exists, and it is unique.

$*$ 12. Assemble the Google matrix $\mathbf{A}$.

Using the function *google_mat*, we have assembled the Google matrix $\mathbf{A}$ defined as below:

$$a_{ij} := \begin{cases} \alpha \frac{g_{ij}}{c_j} + \delta & \text{if } c_j \neq 0, \\ \frac{1}{n} & \text{if } c_j = 0, \end{cases}$$

where α is scalar from 0 to 1 and $\delta := \frac{1-\alpha}{n}$. $(1 - \alpha)$ is denoting the damping factor and the probability that the surfer will dump the current page and teleport to a random page. This matrix models the random web surfer who will follow the hyperlinks in a webpage with probability alpha, but also has a small probability chance (the damping factor) of jumping to a random webpage. A typical value for α is 0.85.

$**$ 13. Consider the sequence $\mathbf{At}, \mathbf{A^2t}, ..., \mathbf{A^kt}, ...$ Towards what do you think it will converge?

To answer this question, we have simulated this series, and calculated the converged vector (using the function *explore_web*). Later, we will see that this converged vector is the very same PageRank vector. This result is also intuitive since we know that $\mathbf{A^kt}$, gives the probability vector that each node will be visited after k steps, and it is intuitive that if we let our web surfer start with an equiprobable probability distribution where each node has a $\frac{1}{n}$ chance to be visited, he/she will converge to the optimal PageRank vector following the in and out-degrees pattern after enough exploration. In our case, the convergence was achieved after 180 number of iterations for tolerance equal to 10^{-16}.

$**$ 14. Recalling the method for approximation of eigenvalues and eigenvectors studied during the course choose the most suitable method to compute the PageRank vector. Implement and apply the selected method directly to the Google matrix $\mathbf{A}$ defined above. Motivate your choice. Check the result comparing it with the one obtained using the Matlab/Python functions to numerically approximate eigenvalues and eigenvectors. Compute the relative error *(N.B. Remember to normalize the computed eigenvector so that its components sum to 1).*

Since the Google matrix $\mathbf{A}$ is a transition probability matrix related to our random walk through the web, the stationary probability distribution vector p, where $Ap = p$ is bound to exist and be unique based on

Perron-Frobenius theorem. Also, based on this theorem this vector is the eigenvector of $\mathbf{A}$ corresponding to the dominant eigenvalue (in this case 1, because $\mathbf{A}$ is a Markov matrix) or the Perron-Frobenius eigenvector. So, we can use our implemented *power_method* function to calculate this eigenvector (tolerance of 10^{-16} and maximum iterations of 5000). As expected, the dominant eigenvalue is one, and the corresponding eigenvector is the pagerank vector which is equal to the one calculated in question 13. The results are given below:

- The dominant eigenvalue using the power method: 1.00
- The dominant eigenvalue using the direct method (numpy): 1.00
- The convergence rate: $\left|\frac{\lambda_{2,actual}}{\lambda_{1,actual}}\right| = 0.843 \approx 0.845 = \alpha$
- Number of iterations needed for convergence: 179
- The relative error of the approximated eigenvalue: $\frac{|\lambda_{1,actual} - \lambda_{2,appx}|}{\lambda_{1,actual}} = 2.664 \times 10^{-15}$
- The norm of difference between the actual dominant eigenvector and the approximate one: $\|v_{1,actual} - v_{1,appx}\| = 4.788 \times 10^{-16}$
- The norm of difference between the actual dominant eigenvector (PageRank vector) and the result of question 13 (iterative exploration of the web): $\|v_{1,actual} - t_{explore}\| = 4.404 \times 10^{-16}$

The relative error between the eigenvalues computed using the implemented method and *numpy*, shows that the power method has successfully converged, and found the dominant eigenvalue which is 1, as expected. Also, the norm of the difference between the exact dominant eigenvector and the approximated eigenvector shows the convergence of the power method. Lastly, the norm of the difference between the exact dominant eigenvector and the converged vector from the iterations in question 13, confirms that the iterative exploration of the web will also result in finding the PageRank vector.

** 15. Modify the implementation of the previous method preserving the sparsity.

We can write the matrix $\mathbf{A}$ as $\mathbf{A} = \alpha\mathbf{GD} + ez^T$, where D is the diagonal matrix formed from the reciprocal of the outdegrees:

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

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

$$z_j := \begin{cases} \delta & \text{if } c_j \neq 0, \\ \frac{1}{n} & \text{if } c_j = 0, \end{cases}$$

By using the function *matrix_sparser* we can return $\mathbf{A}$ as the sum of two: sparse $\alpha\mathbf{GD}$ plus ez^T

∗ ∗ ∗ 16. Modify again the implementation of the method that you have used before. This time implement a version of the numerical method that does not do any matrix operation. To do that you need to use only the link structure of the adjacency matrix $\mathbf{G}$ *(Hint: think at the multiplication of a matrix times a vector as a linear combination of the columns of the matrix by the entries of the vector. Distinguish what happens for* $c_j = 0$ *and* $c_j \neq 0$*).* Why this way of implementing the algorithm can be useful in practical applications? Explain.

This question is optional.

∗∗ 17. Check that $\left|\lambda_2(\tilde{\mathbf{M}})\right| \approx 1$ for our Web. Write a function that implement a suitable numerical method among those studied during the course (you can also use/combine more methods) to approximate $\lambda_2(\tilde{\mathbf{M}})$. Motivate your choice. Check the result comparing it with the ones obtained using the Matlab/Python functions to numerically approximate eigenvalues. Discuss the results.

To approximate the second highest eigenvalue of a matrix we have implemented the power method with shift using *shifted_power_method* function, which uses the dominant eigenvalue and calculates λ_2 using the fact that $A - \lambda_1 I$ will have the same eigenvalues as A but shifted to the left by the value of 1. This method is the most straight forward method for calculating all eigenvalues of a large matrix. However, it is only plausible for a few extreme eigenvalues because after each shift, a complete set of power iterations are needed. Therefore, because we only needed the largest eigenvalues, we have used this method.

The results and the comparison with the results using *numpy* are given below:

- The dominant eigenvalue using the shifted power method: 0.9795
- The dominant eigenvalue using the direct method (numpy): 1.00
- The relative error of the approximated eigenvalue: $\frac{|\lambda_{1,actual} - \lambda_{2,appx}|}{\lambda_{1,actual}} = 0.020$

These results show that $\left|\lambda_2(\tilde{M})\right| \approx 1$ for our web and relative error between the eigenvalues computed using the implemented method and *numpy*, shows that the power method with shift has successfully converged, and found the second dominant eigenvalue.

∗∗ 18. How can you exploit this information to study the asymptotic rate of convergence of the numerical method that you have used to approximate the stationary probability distribution vector?

We have used power method to approximate the eigenvector associated with the largest eigenvalue, which is the stationary probability distribution vector. From theory, we know that the rate of convergence in power method is $\left|\frac{\lambda_2}{\lambda_1}\right|$ meaning that the distance between the approximated vector and the real eigenvector decreases by this factor in each iteration. Because $\lambda_2(\tilde{M}) \approx 1$, the convergence rate for the modified hyperlink matrix $\tilde{\mathbf{M}}$ using the power method is slow. $\tilde{\mathbf{M}}$ is equivalent to the Google

Matrix $\mathbf{A}$ when the damping factor $\alpha = 1$. If we use a damping factor, the probability distribution vector we need to estimate is the eigenvector corresponding to $\lambda_1(A)$. By using a damping factor, if $\tilde{\mathbf{M}}$ has eigenvalues $\{1, \lambda_2, \lambda_3, ..., \lambda_n\}$, $\mathbf{A}$ will have $\{1, \alpha\lambda_2, \alpha\lambda_3, ..., \alpha\lambda_n\}$ as its eigenvalues. So, if $\lambda(\tilde{M}) \approx 1$, $\lambda_2(A) \approx \alpha$, and the convergence rate of the power method used on $\mathbf{A}$ will be $\frac{1}{\alpha}$ which is 0.85 for $\alpha = 0.85$ that is faster than $\tilde{\mathbf{M}}$ (using a web without damping factor). So, the use of damping factor, not only makes our web model more plausible, but also, it speeds up the convergence rate of the PageRank vector.

** 19. Using the computed PageRank vector produce a bar graph of all the ranks of the different pages. Moreover, find the dozen most highly ranked web pages and produce a table whose columns contain: the number associated to each one of these web page, their rank (in decreasing order), their related in-degrees and their out-degrees. Discuss the results.

The bar graph of all of the ranks of the different pages is plotted in figure 4.

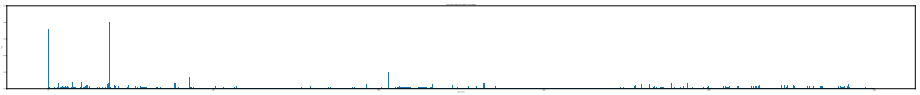

Figure 4: The bar graph of all the ranks

Furthermore, figure 5 and table 1 visualize the 12 highest PageRanks in our miniature web along with their corresponding in and out-degrees. These results make sense, because the nodes with the highest rank, have a high number of ingoing links. However, there is no clear correlation between the number of outgoing links and the rank of a node.

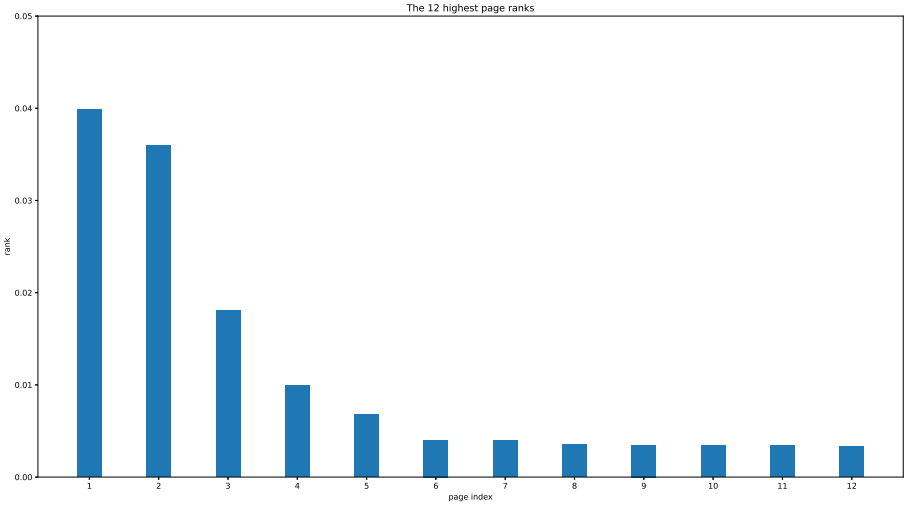

Figure 5: The bar graph of top 12 ranks

Table 1: 12 highest PageRanks with in and out degrees

	186	0	187	1029	427	75
rank	0.0398	0.0359	0.0181	0.0100	0.0068	0.0040
IN degree	1112	1109	283	29	70	38
OUT degree	13	179	3	0	1	31
	100	381	185	1319	1318	384
rank	0.0040	0.0035	0.0035	0.0034	0.0034	0.0033
IN degree	16	1600	35	30	30	41
OUT degree	14	0	1	1	1	1

3 Conclusion

In this homework, we implemented and analyzed the PageRank algorithm for ranking a simplified version of the world-wide web. We also inspected the mathematical backgrounds behind this algorithm, and found out what the PageRank vector is, why it exists and is unique, and what connections it has to the concept of eigenvalues and eigenvectors. Finally, we have visualized the PageRank vector using to bar graphs and a table.