Numerical Linear Algebra Project 2

Peter Weber

December 10, 2017

1 Task 1

1.1 Exercise 1

In exercise 1 we have to calculate the eigenvector to the eigenvalue $\lambda = 1$ when adding a 5th webpage in Fig. 2.1 (in reference [1]). Page 5 links to page 3 and vice versa. The goal is to find out whether the importance score of page 3 has been boosted above that of page 1, by this additional link.
We have to solve for the eigenvector $\vec{x}$ of the linear system

$$\boldsymbol{M}\vec{x} = \vec{x}$$

where

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 & 1 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 \end{bmatrix} \tag{1.1}$$

is the link matrix of the example web. By setting

$$\boldsymbol{M} - \boldsymbol{I} = 0$$

and solving the linear system one finds the eigenvector $\vec{x}^T = (4/3, 4/9, 2, 2/3, 1)$ for $\lambda = 1$, which still has to be scaled such that $\sum_i x_i = 1$. This leads to the importance score eigenvector

$$\vec{x} = \begin{pmatrix} 0.2449 \\ 0.0816 \\ 0.3673 \\ 0.1224 \\ 0.1837 \end{pmatrix}$$

So, indeed the importance score of page 3 has been boosted above that of page 1. I have calculated the eigenvector by hand, but to verify my result there is an implementation in the attached Python file.

1.2 Exercise 4

In the previous exercise the link matrix is column stochastic, meaning that all entries are non-negative, it has 1 as an eigenvalue, and the columns sum up to 1. In this exercise we are dealing with the web given in Fig. 2.1 (of [1]), but the link from page 3 to page 1 is removed, so that page 3 is now a dangling node. The substochastic link matrix for this problem is

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0. \end{bmatrix} \tag{1.2}$$

We are asked to find the Perron eigenvector (the eigenvector for the largest eigenvalue ≤ 1, and having only non-negative entries). The characteristic polynomial resulting from the equation

$$det(\boldsymbol{M} - \lambda \boldsymbol{I}) = 0$$

is

$$\lambda^4 - \frac{1}{6}\lambda^2 - \frac{1}{12}\lambda = 0.$$

The largest positive and real solution to this polynomial is $\lambda \approx 0.5614$.
The resulting eigenvector is

$$\vec{v} = \begin{pmatrix} 0.2066 \\ 0.1227 \\ 0.4386 \\ 0.2320 \end{pmatrix}$$

With page 3 having by far the highest importance score, the ranking does not seem reasonable. Both page 3 and page 4 have two backlinks whereas one of them is from page 1. The second backlink of page 3 comes from page 4 and the second backlink of page 4 comes from page 2. Page 2 should have higher weight, because it links only to one page, while page 4 links to two pages. Page 1 and page 2 have only one backlink, and therefore have lower importance score.

1.3 Exercise 10

Claim: if $\boldsymbol{A}$ is the link matrix of a strongly connected web, then the dimension of the eigenspace of eigenvalue 1 is $\dim(V_1(\boldsymbol{A})) = 1$.
Proof: let us assume we can reach page i from page j in one step. This is equivalent to $A_{ij} > 0$, meaning that there is a link from page j to page i.
If one can reach page i from page j in exactly two steps then $\exists k \neq i, j$ such that $A_{ik}A_{kj} > 0$, meaning that there is a link from page j to page k and from page k to page i. From this it follows

$(\boldsymbol{A}^2)_{ij} = \sum_k A_{ik}A_{kj} > 0$ since $A_{ij} \geq 0 \; \forall i,j$ (because $\boldsymbol{A}$ is column stochastic). The backward direction reads as follows. Knowing that $(\boldsymbol{A}^2)_{ij} > 0$ we have $\sum_k A_{ik}A_{kj} > 0$, meaning that there exists a k so that $A_{ik} > 0$ and $A_{kj} > 0$ (again because $A_{ij} \geq 0 \; \forall i,j$).
Reaching page i from page j in exactly p steps we have $\exists k_1, ..., k_{p-1}$ (equivalently there are $k_1, ..., k_{p-1}$ pages that connect page j with page i) such that $A_{ik_1}A_{k_1k_2}...A_{k_{p-1}j} > 0$, and as before $(A^p)_{ij} = \sum_{k_1}\sum_{k_2}...\sum_{k_{p-1}} A_{ik_1}A_{k_1k_2}...A_{k_{p-1}j} > 0$. The other direction ($(\boldsymbol{A}^p)_{ij} > 0 \rightarrow$ page i can be reached from page j in exactly p steps) is done by induction. For $p = 0$ we have the identity matrix, meaning that page i is reached from page $i = j$ in 0 steps. For $p = 1$ we have $A_{ij} > 0$ and for $p = 2$ we have shown that it holds above. The induction hypothesis is $(\boldsymbol{A}^{p-1})_{ij} > 0$ implies page i is reached from page j in $p-1$ steps. Then we obtain $(\boldsymbol{A}^p)_{ij} = \sum_k A_{ik}(\boldsymbol{A}^{p-1})_{kj} > 0$. It follows $\exists k$ so that both parts in the sum are larger zero. I conclude, that i can be reached from j in p steps.
Having proved the previous point it is trivial that $B_{ij} = I_{ij} + A_{ij} + (A^2)_{ij} + ... + (A^p)_{ij} > 0$ is equivalent to the fact that i can be reached from j in at most p steps, since there will be a $k \in \{0, ..., p\}$ so that one of the summands $(\boldsymbol{A}^k)_{ij} > 0$.
Additionally, in a strongly connected web, every entry $(\boldsymbol{B})_{ij} = (\boldsymbol{I}+\boldsymbol{A}+\boldsymbol{A}^2+...+\boldsymbol{A}^{n-1})_{ij} > 0 \; \forall i,j$, because for $\forall i \; \exists j$ such that page i can be reached in $1, 2, ..., n-1$ steps.
Redefining $\boldsymbol{B}$ as $\boldsymbol{B} = \frac{1}{n}(\boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 + ... + \boldsymbol{A}^{n-1})$ we have that

$$\boldsymbol{B}\vec{x} = \vec{x}$$

provided

$$\boldsymbol{A}\vec{x} = \vec{x}, \tag{1.3}$$

because of two reasons. First, from the latter equation it follows $\boldsymbol{A}^n\vec{x} = \vec{x} \; \forall n \in \mathbb{N}$ due to the fact that $\boldsymbol{A}^{n-1}(\boldsymbol{A}\vec{x}) = \boldsymbol{A}^{n-1}\vec{x} = \vec{x} \; \forall n \in \mathbb{N}$. For $n = 0$ this is trivial since $\boldsymbol{A}^0 = \boldsymbol{I}$. Second,

$$(\boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 + ... + \boldsymbol{A}^{n-1})\vec{x} = n \cdot \boldsymbol{A}\vec{x} = n \cdot \vec{x}.$$

In order to prove that $\boldsymbol{B}$ is column stochastic, one has to show that $\boldsymbol{A}^n \forall n$ is column stochastic. I do this by induction. The case $n = 0$ is trivial and $n = 1$ is precisely the condition for a link matrix, as it is defined in [1]. Specifically, if $\boldsymbol{A}$ is column stochastic we have $\sum_i A_{ij} = 1$. For the case $n = 2$ one has to show that $\sum_i \sum_k A_{ik}A_{kj} = 1$, which follows by flipping the summation order

$$\sum_i \sum_k A_{ik}A_{kj} = \sum_k \sum_i A_{kj}A_{ik} = \sum_k A_{kj} \sum_i A_{ik} = 1$$

By induction, it follows for all n and by Lemma 3.2 $dim(V_1(\boldsymbol{B})) = 1$. $\square$
To recap, we have shown that if $\boldsymbol{A}\vec{x} = \vec{x}$ then $\boldsymbol{B}\vec{x} = \vec{x}$, where $\boldsymbol{B} = \frac{1}{n}(\boldsymbol{I}+\boldsymbol{A}+\boldsymbol{A}^2+...+\boldsymbol{A}^{n-1})$, which is equivalent to

$$x \in V_1(\boldsymbol{A}) \rightarrow x \in V_1(\boldsymbol{B}). \tag{1.4}$$

Now assume $\vec{x}_1, \vec{x}_2 \in V_1(\boldsymbol{A})$ where $\vec{x}_1$ and $\vec{x}_2$ are linearly independent. In this case, we have $\vec{x}_1, \vec{x}_2 \in V_1(\boldsymbol{B})$ from Eq. 1.4, which contradicts $dim(V_1(\boldsymbol{B})) = 1$. $\square$

2 TASK 2

In the attached Python file I implemented the three algorithms described in the task and checked that they work properly on the examples from the paper. In particular, as examples, I chose both the link matrices of Fig. 2.1 and Fig. 2.2 with $m = 0.15$ (in [1]) as well as the link matrices of task 1 exercise 1 (Eq. 1.1) and task 1 exercise 4 (Eq. 1.2), both with $m = 0$.

2.1 SOLVING A LINEAR SYSTEM

```
TASK 2

PART 1: SOLVING THE LINEAR SYSTEM

By solving the linear system, the solution vector of the
link matrix of Fig. 2.1 is (m = 0.15):
[ 0.36815068  0.14180936  0.28796163  0.20207834]

By solving the linear system, the solution vector of the
link matrix in task 1 exercise 1 is (m = 0):
[ 0.24489796  0.08163265  0.36734694  0.12244898  0.18367347]

By solving the linear system, the solution vector of the
link matrix in task 1 exercise 4 is (m = 0):
[ 0.21649485  0.16494845  0.37113402  0.24742268]

By solving the linear system, the solution vector of the
link matrix of Fig. 2.2 is (m = 0.15):
[ 0.2    0.2    0.285  0.285  0.03 ]
```

Figure 2.1: Page rank eigenvectors of the examples when solving the linear system.

The implementation of this part can be found in the functions $get\_G\_and\_D()$ and $solve\_linear\_system()$. The former performs the decomposition $\boldsymbol{A} = \boldsymbol{GD}$ of a page rank matrix $\boldsymbol{A}$ into a link matrix $\boldsymbol{G} = (g_{ij})$ ($g_{ij} = 0$ or $g_{ij} = 1$ according to the existence of a link) and a diagonal matrix $\boldsymbol{D}$, where $d_{jj} = 1/c_j$ if $c_j \neq 0$ and $d_{jj} = 0$ otherwise. Here, $c_j = \sum_i g_{ij}$. It is important to note, that both $\boldsymbol{G}$ and $\boldsymbol{D}$ are returned in sparse csc_matrix format. The latter of the two previously mentioned functions solves the equation given in the assignment sheet

$$(\boldsymbol{Id} - (1-m)\boldsymbol{GD})\vec{x} = v\vec{e}$$

where $v = 1$ and $\vec{e} = (1, ..., 1)^T$ and the solution $\vec{x}$ is normalized such that $\sum_i x_i = 1$. I am using scipy's sparse solver $spsolve$ in this case. The verification of the correct implementation on the examples is shown in Fig. 2.1.

2.2 POWER METHOD

The function implementing this part is $solve\_power\_method$. Further, I am again using the decomposition $\boldsymbol{A} = \boldsymbol{GD}$. As mentioned in the assignment, the key to good performance when working with large matrices is the structure of the summands in $\boldsymbol{M} = (1-m)A + m\boldsymbol{S}$. In

particular, the second summand is a vector times a scalar and the first summand is sparse so that the iteration becomes

$$x^{(k+1)} = (1-m)\boldsymbol{GD}x^{(k)} + e(z^T x^{(k)}).$$

See Fig. 2.2 for verification of the solutions on the examples.

```
PART 2: POWER METHOD

Link matrix of Fig. 2.1 (m = 0.15).
The power method needed 31 iterations to reach tolerance 1e-10
The solution of the power method is:
 [ 0.36815068  0.14180936  0.28796163  0.20207834]

Link matrix of task 1 exercies 1 (m = 0).
The power method needed 69 iterations to reach tolerance 1e-10
The solution of the power method is:
 [ 0.24489796  0.08163265  0.36734694  0.12244898  0.18367347]

Link matrix of task 1 exercise 4 (m = 0).
The power method needed 26 iterations to reach tolerance 1e-10
The solution of the power method is:
 [ 0.21649485  0.16494845  0.37113402  0.24742268]

Link matrix of Fig. 2.2 (m = 0.15)
The power method needed 137 iterations to reach tolerance 1e-10
The solution of the power method is:
 [ 0.2    0.2    0.285  0.285  0.03 ]
```

Figure 2.2: Page rank eigenvectors of the examples when solving with the power method.

2.3 Power method without storing

The algorithm is implemented using the functions $get\_L\_and\_c$, $get\_x\_without\_storing$, and $iterate\_without\_storing$. The first function decomposes a page rank ($\boldsymbol{A}$) or a link matrix ($\boldsymbol{G}$) into the set L_j denoting the indices of non-zero elements in column j of the link matrix ($\forall j$), and into $c_j = \sum_i g_{ij}$, which is the length of L_j. The second function implements the code provided in the assignment sheet, and the third function performs the iteration. For verification of results see Fig. 2.3.

3 Task 3

Fig. 3.1 shows the eigenvector for the $p2p-Gnutella30.mtx$ matrix for the three methods. Note, that my code reads the matrix from the same directory as the Python file and I have not changed the filename of the matrix. Solving the linear system is by far the slowest method with more than one minute execution time. The fastest method is the power method adapted to PR computation with less than one second exectution time until it reaches tolerance 10^{-12}, whereas the power method without storing matrices needs about 15 s for the same tolerance. Although the power method without storing is slower than the power method it still has the

```
PART 3: POWER METHOD WITHOUT STORING

Link matrix of Fig. 2.1 (m = 0.15).
The power method without storing reached tolerance 1e-10 after 123 iterations!!!
The solution is:
 [ 0.36815068  0.14180936  0.28796163  0.20207834]

Link Matrix of task 1 exercise 1 (m = 0).
The power method without storing reached tolerance 1e-10 after 70 iterations!!!
The solution is:
 [ 0.24489796  0.08163265  0.36734694  0.12244898  0.18367347]

Link matrix of task 1 exercise 4 (m = 0).
The power method without storing reached tolerance 1e-10 after 26 iterations!!!
The solution is:
 [ 0.21649485  0.16494845  0.37113402  0.24742268]

Link matrix of Fig. 2.2 (m = 0.15).
The power method without storing reached tolerance 1e-10 after 133 iterations!!!
The solution is:
 [ 0.2    0.2    0.285  0.285  0.03 ]
```

Figure 2.3: Page rank eigenvectors of the examples when solving with the power method without storing.

advantage that it uses less memory for the computation, as it is not storing the matrix. I also implemented the power method without storing with only one loop. To be found in $get\_x\_without\_storing$ under $vectorized$. Here, I added a normalization of the iterate x, which gave about a factor 2 in speed. In order to execute this part of the script set the $vectorize$ parameter in line 163 to $True$.

3.1 Investigate the role of tolerance

The dependence of the elapsed time until tolerance is reached and the number of iterations is shown in Fig. 3.2 for the power method, and in Fig. 3.3 for the power method without storing. The qualitative behaviour is obvious, higher tolerance results in less iterations and less exectution time. However, I can not obtain any conclusion from the exact functional form of the graphs.
In order to investigate the influence of the tolerance on the components, I first check the the number of unique elements in the page rank vectors given by the three methods. Strangely, this is around 9200 although the vector has more than 30000 elements. This is the case for all the methods that I have used, independent of the value chosen for the tolerance, which implies that some of the pages have ambigious importance score.

3.2 Larger Matrices

The fastest solver I have found for sparse matrices is the $scipy.sparse.linalg.bicg\_stab$. This solver finds the solution of the linear system in about 0.1 s using a tolerance of 10^{-16} and it takes about half the time for a tolerance of 10^{-6}. Intuitively, the tolerance should be chosen according to the number of pages contained in the web. For instance, for the

```
To solve the linear system it took 99.51 s!!!
The page rank eigenvector is:

[  1.32691841e-04   4.42088107e-06   6.58669412e-05 ...,   4.42088107e-06
   4.42088107e-06   4.42088107e-06]
The power method needed 73 iterations to reach tolerance 1e-12
The solution of the power method is:
 [  1.32691841e-04   4.42088107e-06   6.58669412e-05 ...,   4.42088107e-06
   4.42088107e-06   4.42088107e-06]
It took: 0.32 s to reach tolerance!
The power method without storing reached tolerance 1e-12 after 183 iterations!!!
The solution is:
 [  1.32691841e-04   4.42088106e-06   6.58669411e-05 ...,   4.42088106e-06
   4.42088106e-06   4.42088106e-06]
It took 18.91 s to reach tolerance!
```

Figure 3.1: Page rank vector for the $p2p-Gnutella30$ matrix using three different methods: Solving linear system, power method, and power method without storing matrices.

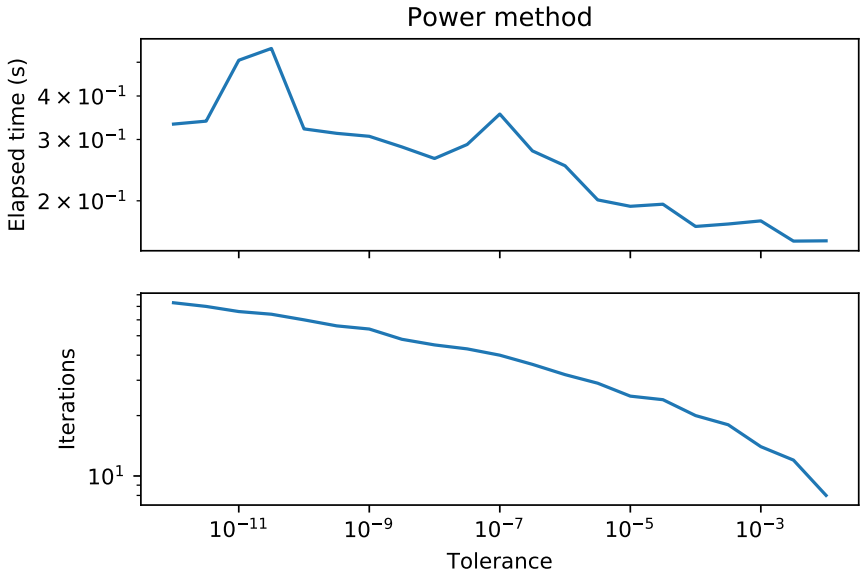

Figure 3.2:

$web-google.mtx$ matrix, which has about 1 million pages, it should be sufficient to choose a tolerance of 10^{-6}. With this tolerance the $bicg\_stab$-solver should not take more than a few seconds.

4 Task 4

The goal of the Python code provided in the assignment sheet is to compute $\boldsymbol{M}x$ in order to iterate $x^{(k+1)} = \boldsymbol{M}x^{(k)}$ without considering the matrix $\boldsymbol{M}$. Here $\boldsymbol{M} = (1-m)\boldsymbol{A} + m\boldsymbol{S}$ where $\boldsymbol{A} =$

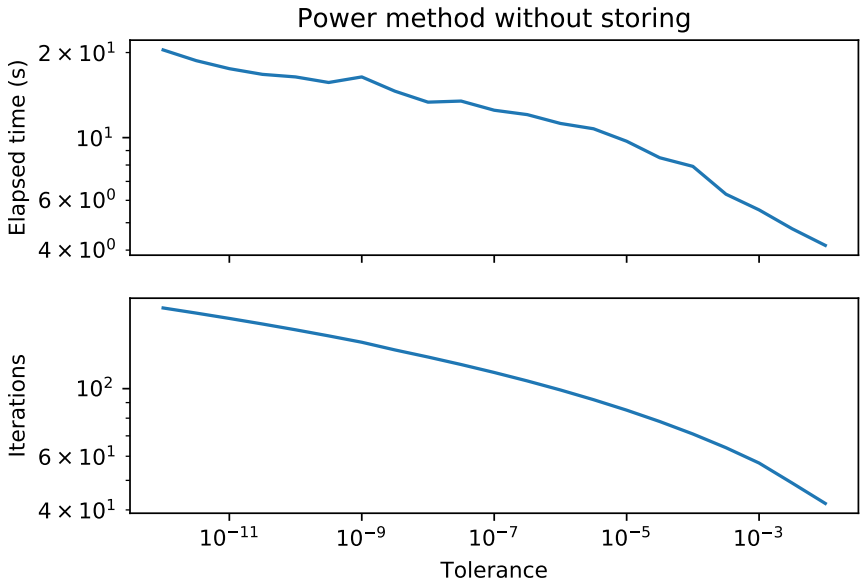

Figure 3.3:

$\boldsymbol{GD}$ defined as in task 2 (as in the assignment sheet) and $m\boldsymbol{S} = ez^T$. We consider $e = (1,...,1)^T$ and $z = (z_1,...,z_n)^T$ with $z_j = m/n$ if column j of $\boldsymbol{A}$ contains non-zero elements and $z_j = 1/n$ otherwise. Instead of storing the matrix $\boldsymbol{M}$ we are storing the sets L_j of non-zero row indices in column j $\forall\, j \in 1,...,n$ of the matrix $\boldsymbol{A}$, and c_j, which is the length of L_j.
Let us first consider the case in which all elements of $\boldsymbol{A}$ in column j are zero. This is equivalent to $c_j = \sum_i g_{ij} = 0$ and the j-th column of the matrix reduces to $\boldsymbol{M}_{ij} = 1/n\ \forall i \in \{1,...,n\}$. The i-th element of the multiplication $\boldsymbol{M}x$ then becomes

$$\frac{1}{n} \sum_{j|c_j=0} x_j.$$

In the provided code, this is precisely what is done inside the "$if(c[j] == 0)$" condition together with the outer foor loop.
Let us now consider the remaining columns j of $\boldsymbol{A}$ for which $c_j \neq 0$. This is equivalent to $A_{ij} = 0$ if $g_{ij} = 0$ and $A_{ij} = 1/c_j$ if $g_{ij} = 1$. In other terms, this can be formulated as $g_{ij} = 0$ if, and only if, $i \notin L_j$. The inner loop, in the "else" part of the code, loops over every non-zero row i in column j, and multiplies the j-th element of x with $1/c_j$. It therefore computes the matrix vector product of $\boldsymbol{A}x$ separately for every column j. The outer loop then adds up the elements of x. This implementation of the matrix vector multiplication is an alternative to the conventional multiplication (in which the inner loop corresponds to looping over columns and the outer loop over rows). Let us summarize the case $c_j \neq 0$ in mathematical notation for the i-th element

$$\sum_{j|c_j\neq 0|g_{ij}=1} \frac{x_j}{c_j}.$$

The last line of the provided code multiplies the sum of the two parts, that I have just described, with $1-m$ and adds m/n. To give the full picture of what the provided code does I am denoting x_i as the i-th element of the vector $\sum_j M_{ij} x_j$. Then we have

$$x_i = (1-m) \sum_{j|c_j \neq 0 | g_{ij}=1} \frac{x_j}{c_j} + (1-m)\frac{1}{n} \sum_{j|c_j=0} x_j + \frac{m}{n} \tag{4.1}$$

where we can write the last two summands as

$$\frac{1}{n} \sum_{j|c_j=0} x_j - \frac{m}{n} \sum_{j|c_j=0} x_j + \frac{m}{n} = \frac{1}{n} \sum_{j|c_j=0} x_j + \frac{m}{n}(1 - \sum_{j|c_j=0} x_j) = \frac{1}{n} \sum_{j|c_j=0} x_j + \frac{m}{n} \sum_{j|c_j \neq 0} x_j.$$

The last equal sign holds, because

$$\sum_j^n x_j = \sum_{j|c_j \neq 0} + \sum_{j|c_j=0} = 1.$$

Plugging this back into Eq. 4.1 we obtain

$$x_i = (1-m) \sum_{j|c_j \neq 0 | g_{ij}=1} \frac{x_j}{c_j} + \frac{1}{n} \sum_{j|c_j=0} x_j + \frac{m}{n} \sum_{j|c_j \neq 0} x_j$$

where the first summand represents the multiplication $(1-m)\boldsymbol{A}$, the second summand $m\boldsymbol{S}$ for $c_j = 0$ and the third summand $m\boldsymbol{S}$ for $c_j \neq 0$, which is precisely

$$x^{(k+1)} = (1-m)\boldsymbol{A}x^{(k)} + m\boldsymbol{S}x^{(k)} = \boldsymbol{M}x^{(k)}. \quad \square$$

REFERENCES

[1] K. Bryan, T. Leise. *The 25 Billion dollar Eigenvector: the linear algebra behind Google,* 2006.