Project 2 NLA: PageRank computations report

Arnau Escapa

December 11, 2017

The goal of this project is to work on the numerical algebra ideas behind the PageRank computation. We will consider the following problem.

Given a webpage network with n webpages and a link matrix G (defining a direct graph), we define the PageRank (PR) score x_k of the page k as

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j} \tag{1}$$

where $L_k = \{\text{webpages with link to page } k\}$ and $n_j =$ number of outgoing links from page j. The value x is a rank value used to sort webpages by importance. The relation (1) can be rewritten with a system $Ax = x$.

Task 1

Exercise 1. Suppose the people who own page 3 in the web of Figure 1 are infuriated by the fact that its importance score, computed using formula (1), is lower than the score of page 1. In an attempt to boost page 3's score, they create a page 5 that links to page 3; page 3 also links to page 5. Does this boost page 3's score above that of page 1?

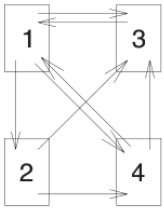

Figure 1: Original web of the exercise 1.

The system of the score probem of the web of Figure 1 is given by $Ax = x$

where $x = (x_1, x_2, x_3, x_4)^T$ and

$$A = \begin{pmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

The solution of $Ax = x$ is given by the eigenvectors with eigenvalue equal to 1. Since the web in question does not have gangling nodes and it is connected we know that $\dim(E_1) = 1$. We choose the eigenvector $x \in E_1$ such that the sum of its components equals to one.

We compute the eigenvalue using the function `numpy.linalg.eig` on A. We obtain

$$x = \frac{1}{31}(12, 4, 9, 6) \approx (0.387, 0.129, 0.290, 0.194) \tag{2}$$

Observe that the best well ranked page is 1 followed by 3. Let us repeat the procedure on the extended webpage. Now we have an extra webpage 5 with a double directed link to 3.

The matrix of the extended system $Ax = x$ is given by

$$A = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} & 1 \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

Again the graph is strongly connected so we just have one eigenvalue with value 1. Its eigenvalue is

$$x = (0.245, 0.082, 0.367, 0.122, 0.184) \tag{3}$$

Observe that now page 3 has a better rank than 1 meaning that the attempt to boost its rank by adding a new page to the web has been achived.

Exercise 4. In the web of Figure 1, remove the link from page 3 to page 1. In the resulting web page 3 is now a dangling node. Set up the corresponding substochastic matrix and find its largest positive (Perron) eigenvalue. Find a non-negative Perron eigenvector for this eigenvalue, and scale the vector so that components sum to one. Does the resulting ranking seem reasonable?

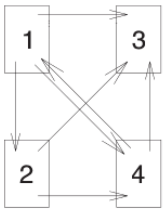

Figure 2: Web of exercise 4

Observe that 3 is a dangling node because it dos not have outlinks. The system $x = Ax$ is given by

$$A = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

This matrix is not column stochastic because it has a column of zeros. So it may not have 1 as a eigenvalue. We compute its eigenvalues. We obtain the trivial eigenvalue 0, 0.561 and a pair of conjugated complex eigenvalues. The Perron eigenvalue is 0.561 and its normalized eigenvector is given by

$$x = (0.207, 0.123, 0.439, 0.232) \tag{4}$$

Altough the values are different this PR value is similar to the one obtained from the original system (2) because the sorting of the pages is the same: $(3, 1, 4, 2)$. This makes sence because the webs are the same except for one link. Taking this into account we could say that we have indices to think that the Parrot eigenvalue gives a reasonable value for the PR.

Exercise 10. In this exercise we will proof that if A is the link matrix of a strongly connected web of n pages then $\dim(V_1(A)) = 1$ by following the steps of the exercise 10. Let A^k_{ij} denote the (i, j)-entry of A^k .

If A is a strongly connected link matrix then A is column-stochastic because all nodes must have in and out links. The sum of the columns is one by definition of the link matrix. Note that if $A_{ij} > 0$ there exists a link from j to i hence page i can be reached from page j exacly with one step. It is also true that if $A^k_{ij} > 0$ then i can be reached from j with exacly k steps. To show this we must just write down the matrix product.

$$A^k_{ij} = \sum_{i_1,\ldots,i_k}^{n} A_{i,i_1} \cdots A_{i_k,j}$$

Observe now that $A^k_{ij} > 0$ if and only if exists $i_1, \ldots, i_k$ such that $A_{i,i_1} \cdots A_{i_k,j} > 0$. And $A_{i,i_1} \cdots A_{i_k,j} > 0$ if and only if $A_{i,i_1}, \ldots, A_{i_k,j} > 0$ hence i can be reached form j by the k-lenght path $(i_k, \ldots, i_1)$.

Now it is easy to see that that

$$(I + A + A^2 + \cdots + A^p)_{ij} > 0 \tag{5}$$

if and only if page i can be reached from page j in p or fewer steps. The identity I follows from the choice $p = 0$ which is a legitimate choice because any page can be reached from itself in zero steps.

If there is exists a $t < p$ such that i can be reached from page j in t steps then $A^t_{ij} > 0$ and $\sum_k^p A^k_{ij} > 0$. If such a t does not exist we have that page i cannot be reached from page j in p or fewer steps and $\sum_k^p A^k_{ij} = 0$.

We will proof now that if we have a strongly connected web with n nodes

$$(I + A + A^2 + \cdots + A^{n-1})_{ij} > 0 \text{ for all } i, j \tag{6}$$

To fulfill (6) is enough to see that exists a path with $n-1$ or less steps from j to i for all i, j. Since the web is strongly connected there exists a path $p_{ji} = (j, \ldots, i)$ for all j, i. If the path has $n-1$ steps or less we are done. If the path has n steps or more means that at least one web k is visited twice. This is $p_{ji} = (j, \ldots, k, k_1, \ldots, k, k_2, \ldots, i)$. Consider the shorter version of the path, $\hat{p_{ji}} = (j, \ldots, k, k_2, \ldots, i)$. If $\hat{p}$ does not visit any web twice it has $n-1$ steps or less and we are done. If it does visit a web twice we find a shorter version of $\hat{p}$ as we just have done. Since the dimension of the web is finite we can eliminate all loops from the original path so there exists a path with $n-1$ or less steps from j to i for all i, j.

Let us now consider

$$B = \frac{1}{n}(I + A + A^2 + \cdots + A^{n-1})$$

B is positive by (6). Observe that since A^k is product of column-stochastic it also column-stochastic:

$$\sum_i A^k_{i,j} = \sum_{i,i_1,\ldots,i_k}^{n} A_{i,i_1} \cdots A_{i_k,j} = \sum_{i_k}^{n} A_{i_k,j} \cdots \sum_{i_1}^{n} A_{i_1,i_2} \sum_{i}^{n} A_{i,i_1} = 1$$

Then

$$\sum_i B_{i,j} = \sum_i \sum_{k=0}^{n-1} \frac{1}{n} A^k_{i,j} = \frac{1}{n} \sum_{k=0}^{n-1} \sum_i A^k_{i,j} = \frac{1}{n} \sum_{k=0}^{n-1} 1 = 1 \text{ for all } j$$

We have seen that B is positive and column-stochastic hence by Lemma 3.2, $\dim(V_1(B)) = 1$.

We will show now that $V_1(A) \subset V_1(B)$. Observe first that if $x \neq 0$ such that $Ax = x$ then $A^k x = A^{k-1} x = x$ for all $k \geq 0$. Let $x \in V_1(A)$, then

$$Bx = \frac{1}{n}(I+A+A^2+\cdots+A^{n-1})x = \frac{1}{n}(Ix+Ax+A^2x+\cdots+A^{n-1}x) = \frac{1}{n}nx = x$$

Hence $x \in V_1(B)$, and $\dim(V_1(A)) \leq \dim(V_1(B)) = 1$. Since $V_1(A) \neq \emptyset$ because A is column-stochastic we have that $\dim(V_1(A)) = 1$. □

Task 2

In this task we implement three different strategies to find the PR vector by computing the eigenvector with eigenvalue 1 of the matrix

$$M_m = (1-m)A + mS,$$

where $0 \leq m \leq 1$ is a damping factor ($m = 0.15$ by default) . The matrix S is defined as follows. Consider $e = (1, \ldots, 1)^t$, and consider $z = (z_1, \ldots, z_n)^t$ to be the vector given by $z_j = \frac{m}{n}$ if the column j of the matrix A contains non-zero elements, $z_j = 1/n$ otherwise. Then $mS = ez^t$.

The solution of the problem (1) is given by the eigenvector with eigenvalue one of the matrix A. But when the web is not connected the value may not be unique and when there are dangling nodes the solution may not exist. For this reason we consider M_m instead of A. M_m has $\dim(V_1(M_m)) = 1$ also for the problematic cases.

All three strategies are implemented in the file `lib.py` with the following prototype

```
strX(G, m=0.15, tol=1e-6)
```

where `X` is the number of the strategy.

Strategy one finds the solution by solving the linear system

$$(Id - (1-m)GD)x = e$$

where $D = \text{diag}(d_1, \ldots, d_n)$ with $d_j = 1/\sum_i g_{ij}$ if $\sum_i g_{ij} \neq 0$ and $d_j = 0$ otherwise, and then scaling the vector so that $\sum x_i = 1$.

We implemented this taking advantatge of the sparse structure of the matrix and using the function `scipy.sparse.linalg.spsolve` to solve the system.

Strategy two finds the solution by appling the power method taking advantatge of the sparse of M_m. This is, in every iterate we compute

$$x_{k+1} = (1-m)GDx_k + ez^t x_k$$

Observe that it is enought to compute the real number ez^t just once and without need to construct e. To compute the left term we take advantatge of

the diagonal structure of D. We compute the product term by term between $\text{diag}(D)$ and x_k and then the dot product between G and $\text{diag}(D)x_k$. Since in each iteration we compute a matrix vector product we chose the CSR sparse format to save G for this strategy because CSR is fast reading rows.

Strategy three uses the power method without storing M_m. We are given a code that computes one iteration of this implementation of the power method. In Task 4 we explain why this code works. We use it to implement the `str3()` function. We also write a variation of the given code `str3bis()` that improves the computational time of the strategy by performing more efficient loops. After some empirical expiriments we observe that CSR performs better that CSC for this method too.

In order to check that the functions work properly we applied them on the webs of exercises one and two. We compared the PR score with the scaled eigenvector of eigenvalue 1 of the matrix A or the Parrot eigenvector. The obtained results are satisfactory. We observe that the ranking list of webpages is always the same, however the PR score is quite different. This is because we compute the eigenvector of M_m instead of A. When the web is not one of the problematic cases (i.e disconnected web or dangling nodes) we can set $m = 0$ to obtain a most accurated solution. The routine `test.py` prints via terminal the results of this tests.

Task 3

In this task we use the methods on the matrix that can be find in the file `Data/p2p-Gnutella30.mtx`. It is a link matrix of a web with 36682 pages and 88328 links. The routine `Gnutella.py` applies the different previous strategies on the matrix to compute the PR score, prints via terminal the computation time of strategy 1 and stores rellevant information about the performance of the strategies 2 and 3 in the files `nutella_str2.txt` and `nutella_str3.txt` that can be found inside the folder `Results`.

Since the PR value is a ranking, an interesting question may be how the tolerance choise affects on the list of indices that makes the PR score a storted vector. In other words how the ranking of websites change from the real ranking depending on the tolerance. One way of quantifing this is checking how many values of a ranking given by a method are in the same position than the real ranking. We call this number accuracy.

Each line of the file `nutella_str2.txt` corresponds to the following values seperated by space obtained by applying `str2(G,tol=TOL)` for different values of `TOL`:

- Tolerance used
- Computacional time

- Norm of the difference between the obtained PR and exact solution 1
- Accuracy respect exact solution 1
- Norm of the difference between the obtained PR and exact solution 2
- Accuracy respect exact solution 2

where we take as exact solutions 1 and 2 the obtained by strategy 1 and strategy 2 with `tol=1e-16`. The file `nutella_str3.txt` is analog but using strategy 3. We use this files to plot the results using `gnuplot`.

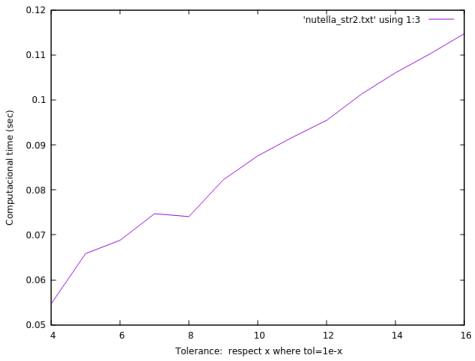

Figure 3: Computacional time respect tolerance strategy 2

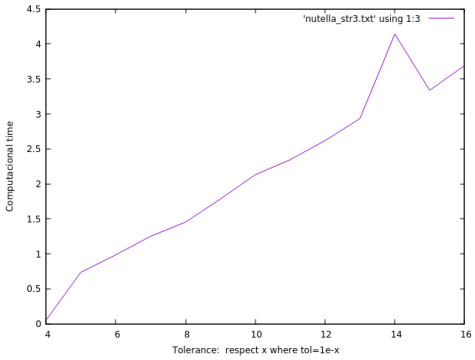

Figure 4: Computacional time respect tolerance strategy 3

We observe a linear behavior of the computacional time respect the tolerance. Strategy 1 takes 49.72 seconds to compute the PR. Strategy 2 with tolerance 10^{-16} takes just 0.114 seconds. Strategy 3 with tolerance 10^{-16} without the improvement needs 8.47 seconds to compute the PR . With the improved version it takes 3.87 seconds.

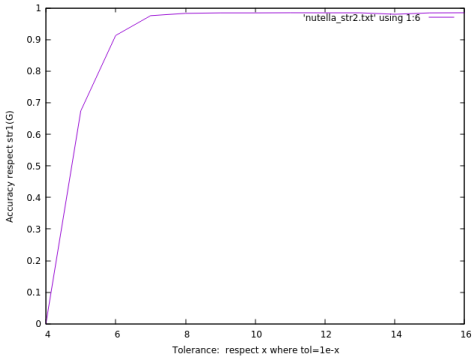

Figure 5: Accuracy respect tolerance taking as exact solution `str1(G)`.

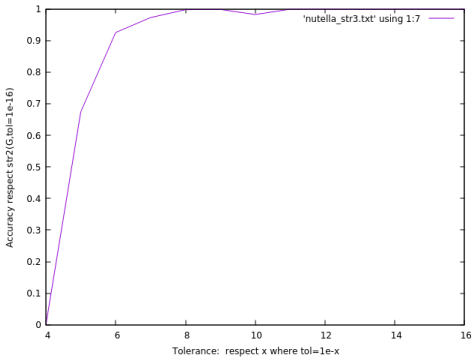

Figure 6: Accuracy taking as exact solution `str2(G,tol=1e-16)`.

We observe that we do not need to impose a very small tolernace in order to get a good ranking. With `tol=1e-8` we get an accuracy of 0.994 but with a too small tolerance (i.e `tol=1e-4`) we get a totally wrong ranking .

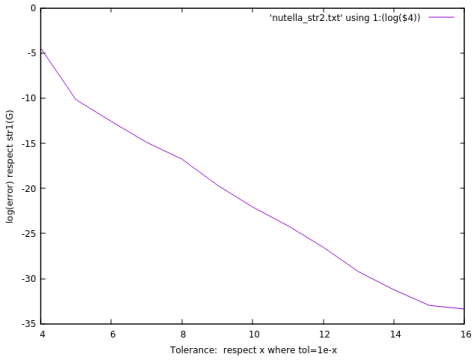

Figure 7: Error taking as exact solution `str1(G)`.

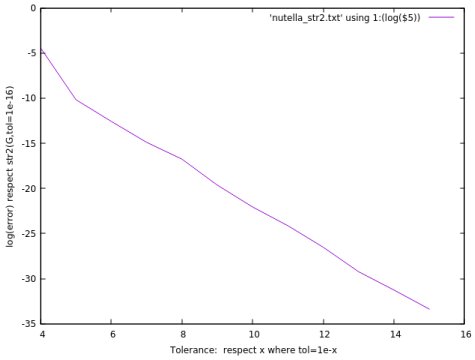

Figure 8: Error taking as exact solution `str2(G,tol=1e-16)`.

Observe that we used log scale. The error is the same order of the tolerance used as expected.

Let us now check the order of convergence of the implemented power method. Recall that the power method convergences linealry with ratio of convergence equal to $|\frac{\lambda_2}{\lambda_1}|$ where the eigenvalues are sorted by their module (i.e. $|\lambda_1| > |\lambda_2| > \cdots$). In our case we know that $\lambda_1 = 1$ and $\lambda_2 \leq 1 - m = 0.85$ (see Google paper). Hence we may expect a rate of confergence bounded by 0.85.

One can compute an approximation of the order of convergence using the following idea.

Let us denote $\varepsilon_k = x_k - x$ where x is the exact solution. Recall that if a method converges linearly the ratio of convergence is defined as the $mu \in (0, 1)$such that

$$\lim_{k\to\infty} \frac{|\varepsilon_{k+1}|}{|\varepsilon_k|} = \mu$$

The routine `conv.py` implements an approximation of μ by computing the average of the tail of the serie $\frac{|\varepsilon_{k+1}|}{|\varepsilon_k|}$. We compute μ using as exact solution the obtained from `str1(G)` and `str2(g,tol=1e-16)`. We obtained rates of convergence of 0.7014637 and 0.7014635 respectively which are congruent values with the expectations.

When we face with webs of higher dimension like for example the `web-Google.mtw` which has $n = 875713$ webs and 5105039 links we must take into account the efficiencie of the different strategies. Strategy 1 must solve a system of dimension n which is not computacionally viable with methods like LU, QR. Alternatively one may use recursive methods that take advantatge of the sparse structure to solve the system. One of this methods is the function `bicgstab` implemented in the `scipy.sparse` library. We use this function in the routine `google.py` which computes the

PR score for the `web-Google.mtw` matrix using this routine and also strategy 2 and 3 and prints via terminal the computacional time for each strategy. We obtained the following results.

```
Strategy bicgstab:
    Tol: 1e-8
    Time: 5.9187009999999995 sec
Strategy 2:
    Tol: 1e-8
    Time: 4.521743999999998 sec
Strategy 3:
    Tol: 1e-8
    Time: 164.09870 sec
```

The file `web-Google.mtw` it is not included in the folder `Data` because of its size. It seems that strategy 2 is the best option we have.

Task 4

In this exercise we explain how to compute the iteration of the power method

$$x_{k+1} = M_m x_k$$

without storing the matrix M_m by the code

```
xc=x
x=numpy.zeros(n)
for j in range (0,n):
    if(c[j]==0):
        x=x+xc[j]/n
    else:
        for i in L[j]:
            x[i]=x[i]+xc[j]/c[j]
 x=(1-m)*x+m/n
```

The only data we need to store are the set of indices L_j corresponding to pages having a link with page j and c_j which are given by the lenght of L_j for $j = 1, \ldots, n$.

Remember that M_m is defined as

$$(M_m)_{ij} = (1-m)A_{ij} + mS_{ij} \tag{7}$$

$$mS_{ij} = \begin{cases} 1/n & \text{for } j \mid c_j = 0 \\ m/n & \text{for } j \mid c_j \neq 0 \end{cases}$$

$$A_{ij} = \begin{cases} 1/c_j & \text{if } i \in L_j \\ 0 & \text{otherwise} \end{cases}$$

We will denote $M_m = M$, $x_k = x$ and $x_{k+1} = x^{k+1}$.

Let us write down what the code does in each iteration. With the first `for` it computes

$$x_i^{k+1} = \sum_{j|c_j=0} \frac{x_j}{n} + \sum_{i|i\in L_j} \frac{x_j}{c_j} \tag{8}$$

With the last line of the code the final computation of an iterate is

$$x_i^{k+1} = (1-m) \sum_{j|c_j=0} \frac{x_j}{n} + (1-m) \sum_{i|i\in L_j} x_j c_j + \frac{m}{n} \tag{9}$$

We just have to check that

$$x_i^{k+1} = (Mx)_i$$

$$x_i^{k+1} = \sum_{j}^{n} M_{ij} x_j = \sum_{j|c_j=0} M_{ij} x_j + \sum_{j|c_j\neq 0} M_{ij} x_j$$

We use (7) and that if $c_j = 0$ the column j of A is a column of zeros.

$$x_i^{k+1} = \sum_{j|c_j=0} m S_{ij} x_j + (1-m) \sum_{j|c_j\neq 0} A_{ij} x_j + \sum_{j|c_j\neq 0} m S_{ij} x_j = \tag{10}$$

$$= \sum_{j|c_j=0} \frac{x_j}{n} + \frac{m}{n} \sum_{j|c_j\neq 0} x_j + (1-m) \sum_{j|c_j\neq 0} A_{ij} x_j \tag{11}$$

Let us now check a property that we will use to finish the prove.

$$\sum_{i}^{n} x_i^k = 1 \text{ for all } k \geq 0$$

Let us supose that $\sum_i^n x_i^k = 1$ and proof that $\sum_i^n x_i^{k+1} = 1$. Using (11) we have that

$$\sum_{i}^{n} x_i^{k+1} = \sum_{j|c_j=0} x_j + m \sum_{j|c_j\neq 0} x_j + (1-m) \sum_{i}^{n} \sum_{j|c_j\neq 0} A_{ij} x_j =$$

$$= \sum_{j|c_j=0} x_j + m \sum_{j|c_j\neq 0} x_j + (1-m) \sum_{j|c_j\neq 0}^{n} x_j \sum_{i} A_{ij} =$$

$$= \sum_{j|c_j=0} x_j + m \sum_{j|c_j\neq 0} x_j + (1-m) \sum_{j|c_j\neq 0}^{n} x_j = \sum_{i}^{n} x_i^k = 1$$

where the second equality holds because the non-zero columns of A add to one. This implies that

$$\sum_{j|c_j \neq 0} x_j = 1 - \sum_{j|c_j=0} x_j$$

And retaking (11) we have

$$\begin{aligned} x_i^{k+1} &= \sum_{j|c_j=0} \frac{x_j}{n} + \frac{m}{n} \sum_{j|c_j \neq 0} x_j + (1-m) \sum_{c|c_j \neq 0} A_{ij} x_j = \\ &= \sum_{j|c_j=0} \frac{x_j}{n} + \frac{m}{n} \big(1 - \sum_{j|c_j=0} x_j\big) + (1-m) \sum_{i|i \in L_j} \frac{x_j}{cj} = \\ &= (1-m) \sum_{j|c_j=0} \frac{x_j}{n} + \frac{m}{n} + (1-m) \sum_{i|i \in L_j} \frac{x_j}{c_j} \end{aligned}$$

which is the same expression than (9) as we wanted to see.