MA50177: Scientific Computing
Case Study

Google's PageRank Algorithm

This assignment is about practical aspects of the PageRank algorithm, the heart of the Google search engine. This is an important application of Markov Chain models and eigenvalue computations in discrete mathematics. In particular, we will use the power method to find the dominant eigenvalue and the corresponding eigenvector of a square matrix and analyse its convergence theoretically.

In Part A of the assignment you will be asked to implement the power method, to test it on the matrix arising from the finite difference discretisation of Poisson's equation (discussed in the lectures), and to analyse its convergence. In Part B you will be asked to assemble the connectivity matrix of a subset of the worldwide web from a connectivity list, to implement the PageRank algorithm using the power method to find the PageRank vector for the given subset of the web, and to analyse the convergence again. In Part C of the assignment you will be asked to use compressed row storage to exploit the sparsity of the matrices and to improve the efficiency of your code in Part B.

Google's PageRank Model

In the late 1990's some computer scientists working in link analysis realised that the hyperlink structure of the internet could be used to improve the performance of web search engines. They used ideas from Markov chain models and eigenvalue solvers to compute a ranking of webpages according to their importance purely based on the connectivity of the web. The most successful model both commercially and mathematically was the PageRank algorithm developed by two Computer Science PhD students at Stanford University, Sergey Brin and Larry Page, in 1998 (see [1] for their original paper). Its success was so huge that it created the business giant Google and made Brin and Page multi-millionaires.

The basic idea of PageRank is simple. Let us consider the hyperlink structure of the web as a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of directed edges. The nodes of the graph are the webpages (or a subset of them). We will represent these webpages by integer numbers from 1 to n, i.e. $\mathcal{N} = \{1, \ldots, n\}$. The directed edges of the graph are the hyperlinks on these webpages, i.e. if page i has a link to page j then $(i, j) \in \mathcal{E}$. Let us represent this graph by its connectivity matrix G, i.e. $G_{j,i} = 1$ if there is a link from page i to page j. (Note the order of the indices in $G_{j,i}$. This is important!)

The basic philosophy of PageRank is that the importance of a webpage is governed by the importance of the pages linking to it, i.e. if π_i denotes the pagerank of page i then

$$\pi_j \;=\; \sum_{\{i\,:\,(i,j)\,\in\,\mathcal{E}\}} \frac{1}{c_i}\,\pi_i \tag{1}$$

where $c_i = \sum_{j=1}^{n} G_{j,i}$ is the *out-degree* of page i, i.e. the number of links on page i. Thus, the higher the pagerank of a webpage, the more important it is. A possible algorithm to find the PageRank vector $\boldsymbol{\pi} = (\pi_i)_{i=1,\ldots,n}$ in (1) is to start with an initial guess for the PageRank vector $\boldsymbol{\pi}$ and then loop through the webpages to update this vector until it does not change anymore. Obviously this process is iterative and the question arises if this process will converge to a solution and if this solution is unique and stable.

To do this let us rewrite the problem in matrix form and look at it as an example of a Markov chain. In a Markov chain model the nodes of the graph $\mathcal{G}$ are the *states* of the Markov chain and the connectivity of the graph is represented by a $n \times n$ transition probability matrix P where $P_{j,i}$ is the probability of moving from state i to state j. A matrix P is called *(column-) stochastic* if $\sum_{j=1}^{n} P_{j,i} = 1$ for all $i \in 1, \ldots, n$. If all the entries in P are strictly between 0 and 1 then P is also called *irreducible*. It can be shown quite easily that for a stochastic matrix P the spectral radius $\rho(P) = 1$ and that 1 is an eigenvalue of P. If P is also irreducible, then an important result in matrix theory known as the *Perron-Frobenius Theorem* states that 1 is in fact a simple eigenvalue strictly larger in modulus than all other eigenvalues. Furthermore there exists a unique stationary distribution vector $\boldsymbol{\pi}$ with strictly positive entries $\pi_i \in \mathbb{R}$ such that[1]

$$P\,\boldsymbol{\pi} \;=\; \boldsymbol{\pi} \qquad \text{and} \qquad \boldsymbol{e}^T\boldsymbol{\pi} = 1. \tag{2}$$

where $\boldsymbol{e} = (1, \ldots, 1)^T$, i.e. $\boldsymbol{\pi}$ is the (right) eigenvector corresponding to the eigenvalue 1.

Let us return to PageRank and write (1) as a Markov chain model. Let $P_{j,i} = G_{j,i}/c_i$ if there is a link from page i to page j, i.e. we assume that it is equally likely to follow any of the outgoing links on a webpage. Then the stationary distribution vector $\boldsymbol{\pi}$ in (2) is also a solution to (1). This is supposed to model the behaviour of a "random surfer" with infinite time at hand. However, there is still one problem with this model in that webpages do exist which have no outgoing links, so called *dangling pages*. This means a random surfer could be caught up in one of them and never resurface. From a Markov chain point of view this would be a *closed* state and the transition matrix would not be a stochastic matrix. We overcome this problem by adding a link from this page to every other page. i.e. $P_{j,i} = 1/n$ for all j, if i is a dangling page. For our random surfer this means that when he arrives at a dangling page he types in the address of another random webpage to continue surfing. Now P is a stochastic matrix but it is still not irreducible in general. To render P irreducible we add the possibility for the random surfer to jump to a random webpage at any time ignoring the hyperlink structure of the web. We assume the probability of this is $(1 - \alpha)$ where $\alpha \in (0, 1)$. (Google uses apparently $\alpha = 0.85$.) Now the final form of the transition probability matrix is

$$P \;=\; \alpha(G\hat{C} + \boldsymbol{v}\boldsymbol{a}^T) \;+\; (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T \tag{3}$$

where $\hat{C}$ is the $n \times n$ diagonal matrix with entries $\hat{C}_{i,i} = 1/c_i$ if i is not a dangling page and $\hat{C}_{i,i} = 0$ otherwise,

[1]Note that most books on Markov chain models use the transpose of P defined here in which case the stationary distribution vector is a left eigenvector of P to the eigenvalue 1. In his talk in the Numerical Analysis Seminar Zhivko Stoyanov also used this notation.

$$\boldsymbol{a} = (a_i)_{i=1,\ldots,n} \quad \text{with} \quad a_i = \begin{cases} 1 & \text{if } i \text{ is a dangling page} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and $\boldsymbol{v}$ is the so-called *personalisation vector*. We will only use

$$\boldsymbol{v} \;=\; \tfrac{1}{n}\,\boldsymbol{e}\;. \tag{5}$$

This matrix is stochastic and irreducible and so the problem (2) has a unique solution, the PageRank vector $\boldsymbol{\pi}$. To find $\boldsymbol{\pi}$ we will use the Power Method discussed in lectures.

For more information on PageRank see Section 2.11 of [2], the original paper [1] on PageRank by Page, Brin et al, the slides from Zhivko Stoyanov's Numerical Analysis Seminar talk [3], or one of the other references [4-6] given below.

To produce a set of webpages and their hyperlink structure we will use the `MATLAB` function `surfer.m` provided in `~masrs/ma50177/assignment`. This is a slightly modified version of the `MATLAB` function given in Cleve Moler's book [2] where you can also find a description of how to use it. My modified version avoids opening some local webpages at Bath which might create problems with BUCS for you. Therefore, please **only use** my version of `surfer.m`. In addition to returning a vector U of webpages and the connectivity matrix G on exit, my modified version also writes the connectivity matrix to a file `graph.txt` in the format

```
n nnz

2 1
3 1
1 2
. .
. .
. .
```

i.e. the first row of `graph.txt` contains `n`, the number of nodes, and `nnz`, the number of edges (or links). The subsequent `nnz` rows in `graph.txt` contain the indices i and j of the nonzero entries in G. To use `surfer.m` you need to set the proxy in the `Web Preferences` for `MATLAB`:

- In the `MATLAB` window go to `File` ⟶ `Preferences...` ⟶ `Web`
- Set the `Proxy host` to `wwwcache.bath.ac.uk` and the `Proxy port` to `3128`.

Please use `surfer.m` with moderation, or we might get problems with BUCS.

There is also a `MATLAB` function `pagerank.m` available in [2] which you may use to find the pagerank vector in `MATLAB`. Note, however, that `pagerank.m` implements a slightly different version of the PageRank algorithm, so the solution vector and therefore the pagerank ordering might be slightly different as well.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Technical Report, Stanford University, 1998. (available under `http://dbpubs.stanford.edu:8090/pub/1999-66`)

[2] C.B. Moler, *Numerical Computing with Matlab*, SIAM, 2004. (also available under `http://www.mathworks.com/moler/chapters.html`)

[3] Z. Stoyanov, *How Google works: Eigenvector methods for PageRank*, Numerical Analysis Seminar, University of Bath, 11th March 2005. (slides available under `http://students.bath.ac.uk/mapzvs/Google.pdf`))

[4] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin, *PageRank Computation and the Structure of the Web*, 11th Int. World Wide Web Conference, Honolulu, May 2002. (available under `http://www2002.org/CDROM/poster/173.pdf`)

[5] D. Higham and A. Taylor, *The Sleekest Link Algorithm*, IMA Mathematics Today, Vol. 39, 2003. (also available under `http://www.charlesclee.com/pagerank.pdf`)