Università della Svizzera italiana

Facoltà di scienze informatiche

Informatics Bachelor Course Numerical Computing **Academic Year 2017/2018**

Instructor: Prof. Olaf Schenk TA: Fabio Verbosio, Aryan Eftekhari

Assignment 1 - PageRank Algorithm

Due date: Tuesday 10 October 2017, 10:30

The purpose of this assignment[1] is to learn the importance of numerical linear algebra algorithms to solve fundamental linear algebra problems that occur in search engines.

Page-Rank Algorithm -- Linear Systems

One of the reasons why Google™ is such an effective search engine is the Page-Rank™ 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 in the order of their PageRank. Imagine surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next. This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages. So, a certain fraction of the time, simply choose a random page from the Web. 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. A page has high rank if other pages with high rank 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 actually varies with time, but by the end of 2002, n was over 3 billion. 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:

$$r_i = \sum_j g_{ij}, \quad c_j = \sum_i g_{ij}, \tag{1}$$

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

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 the n-by-n matrix whose elements are

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

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, that is 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 δ, the probability of jumping from one page to another without following a link. If $n = 4 \cdot 10^9$ and $p = 0.85$, then $\delta = 3.75 \cdot 10^{-11}$. The 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 in matrix theory known as the *Perron–Frobenius theorem* applies to such matrices. It concludes that a nonzero solution of the equation

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

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 is *Google's PageRank*. The elements of x are all positive and less than one.

The 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 successive vectors agree to 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

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

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^T accounts for the random choices of Web pages that do not follow links. The equation

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

can be written

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

where

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

We do not know the value of γ because it depends upon 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 the vector z is not actually 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/sum(x);
```

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].

```
A = p*G*D + delta
x = (I - A)\e
x = x/sum(x)
```

At first glance, this appears to be a very dangerous idea. Because $(I - A)$ is theoretically singular, with exact computation some diagonal element of the upper triangular factor of $(I - A)$ should be zero and this computation should fail. But with roundo error, the computed matrix $(I - A)$ is probably not exactly singular. Even if it is singular, roundo 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, $(I - A)\backslash e$, 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 $A \times x$ 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.

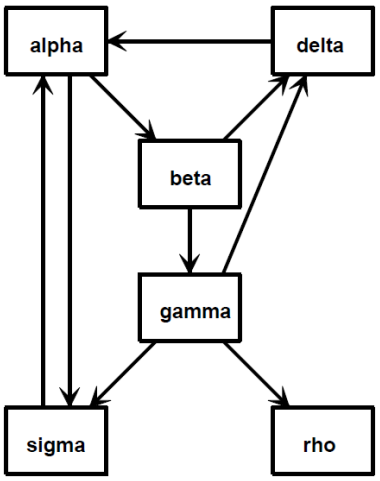

Figure 1. A tiny web graph.

Figure 1 is the graph for a tiny example, with $n = 6$ instead of $n = 4 \cdot 10^9$. Pages on the Web are identified by strings known as uniform resource locators, or URLs. Most URLs begin with `http` because they use the hypertext transfer protocol. In Matlab, we can store the URLs as an array of strings in a cell array. This example involves a 6-by-1 cell array.

```
U = {'http://www.alpha.com'
     'http://www.beta.com'
     'http://www.gamma.com'
     'http://www.delta.com'
     'http://www.rho.com'
     'http://www.sigma.com'}
```

Two different kinds of indexing into cell arrays are possible. Parentheses denote subarrays, including individual cells, and curly braces denote the contents of the cells. If k is a scalar, then $U(k)$ is a 1-by-1 cell array consisting of the kth cell in U, while $U\{k\}$ is the string in that cell. Thus $U(1)$ is a single cell and $U\{1\}$ is the string

'http://www.alpha.com'. Think of mail boxes with addresses on a city street. B(502) is the box at number 502, while B{502} is the mail in that box.

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

```
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/sum(x)
```

solve the sparse linear system to produce

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

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

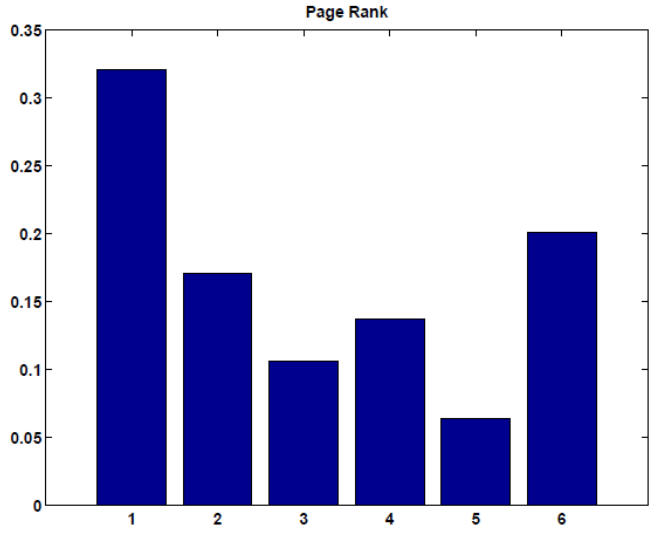

Figure 2. Page Rank for the tiny web graph

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

Notice that the column sums of A are all equal to one. This assignment includes in the directory `files_data` the Matlab file `surfer.m`. A statement like

```
[U,G] = surfer('http://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. The function uses urlread, which was introduced in Matlab 6.5, along with underlying Java utilities to access the Web. 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('http://www.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 graph was generated in September 2014 and will be called the *ETH500 data* set. The statements

```
spy(G)
```

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

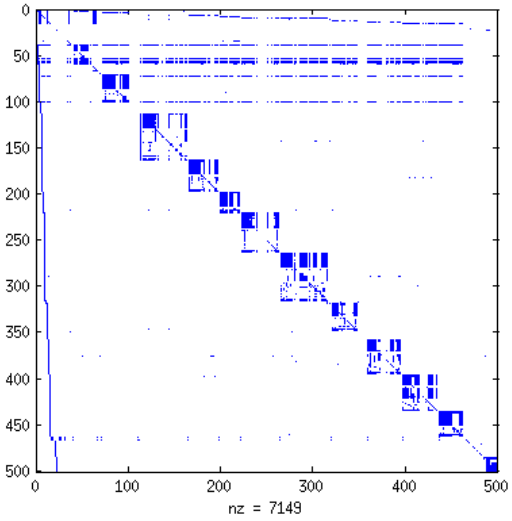

Figure 3. Spy plot of the Department of Computer Science (ETH Zurich) web graph.

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 dozen highly ranked pages are

```
>> pagerank(U,G)
     page-rank  in  out  url
  39   0.0391  311    0  http://www.ethz.ch
  57   0.0297  291    0  http://www.zope.org
  58   0.0297  291    0  http://www.infrae.com
  53   0.0287  288    0  http://www.cd.ethz.ch/services/web
  72   0.0223  235    0  http://www.ethz.ch/index_EN
 101   0.0192  233    5  http://www.hk.ethz.ch/index_EN
```

466	0.0100	18	0	http://purl.org/dc/elements/1.1
376	0.0098	18	0	http://www.inf.ethz.ch
463	0.0092	17	0	http://ns.adobe.com/xap/1.0
464	0.0092	17	0	http://ns.adobe.com/xap/1.0/mm
487	0.0080	13	2	http://www.finanzen.ch
59	0.0071	54	5	http://www.hk.ethz.ch
486	0.0061	13	1	http://www.handelszeitung.ch

The URL where the search began, www.inf.ethz.ch, dominates.

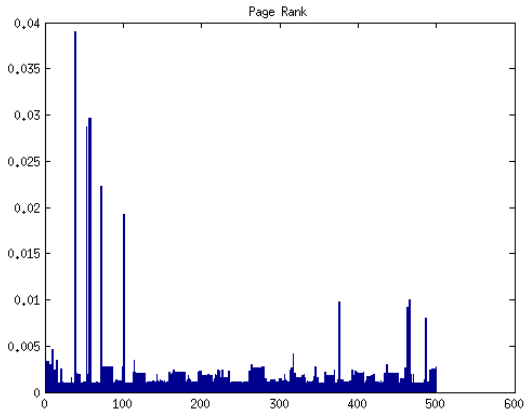

Figure 4. PageRank of the Department of Computer Science (ETH Zurich) web graph.

1. Solve the following problems:

1.1. Other webgraphs [10 points]

Use `surfer.m` and `pagerank.m` to compute PageRanks for some subset of the Web that you choose. Do you see any interesting structure in the results? Report on three PageRanks for different webgraphs by showing the connectivity matrix, the PageRanks, and the ten most important entries in the graph.

1.2. Connectivity matrix and subcliques [10 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 with every node connected to every other node is known as a clique. Identify the organizations within the ETH community that are responsible for these near cliques.

1.3. Connectivity matrix and disjoint subgraphs [10 points]

Figure 5 is the graph of another six-node subset of the Web. In this example, there are two disjoint subgraphs.

1. What is the connectivity matrix G?

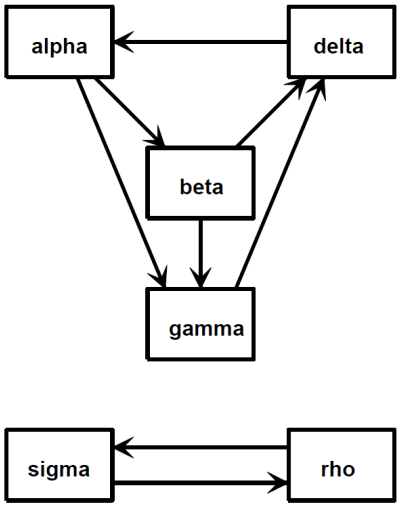

Figure 5. Another tiny Web.

2. What are the PageRanks if the hyperlink transition probability p is the default value 0.85?

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

1.4. PageRanks by solving a sparse linear system [70 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. Familiarize yourself with the **power method** and the related **inverse iteration** for solving Eigenvalue problems [1, 2, 3].

2. Create `pagerank1(G)` by modifying `pagerank.m` to use inverse iteration instead of solving the sparse linear system. The key statements are

```
x = (I - A)\e
x = x/sum(x)
```

   What should be done in the unlikely event that the backslash operation involves a division by zero?

3. Create `pagerank2(G)` 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?

4. Use your functions to compute the PageRanks of the six-node example discussed in the text. Make sure you get the correct result from each of your three functions.

5. Use your functions `pagerank1.m` and `pagerank2.m` to compute the PageRanks of the six-node example discussed in the text. Make sure you get the correct result from each of your two functions.

6. Use your functions `pagerank1.m` and `pagerank2.m` to compute the PageRanks of the three selected graphs from the first assignment. Report on the convergence of e.g. the power iteration for these subgraphs and summarize the advantage of the power method implemented in `pagerank2.m` against the original implementation in `pagerank.m`

In-class assistance

If you experience difficulties in solving the problems above, please come to class either on Tuesday 10:30-12:15, SI-006 or on Thursday 08:30-10:15, SI-006

Additional technical notes

You only need Matlab for this assignment.

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