Chapter 7

Google PageRank

The world's largest matrix computation. (This chapter is out of date and needs a major overhaul.)

One of the reasons why Google™ is such an effective search engine is the PageRank™ 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 June 2004, n was over 4 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 $g_{ij} = 0$ otherwise. The matrix G can be huge, but it is very sparse. Its jth column shows the links on the jth page. The number of nonzeros in G is the total number of hyperlinks in W.

October 2, 2011

Let r_i and c_j be the row and column sums of G:

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

The quantities r_j and c_j are the *in-degree* and *out-degree* of the jth 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

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

Notice that A comes from scaling the connectivity matrix by its column sums. The jth column is the probability of jumping from the jth page to the other pages on the Web. If the jth 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

$$x = Ax$$

exists and is unique to within a scaling factor. If this scaling factor is chosen so that

$$\sum_i x_i = 1,$$

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

$$(I - A)x = 0.$$

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

$$A = pGD + ez^T$$

where D is the diagonal matrix formed from the reciprocals of the outdegrees,

$$d_{jj} = \begin{cases} 1/c_j & : \quad c_j \neq 0 \\ 0 & : \quad c_j = 0, \end{cases}$$

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

$$z_j = \begin{cases} \delta & : \quad c_j \neq 0 \\ 1/n & : \quad c_j = 0. \end{cases}$$

The rank-one matrix ez^T accounts for the random choices of Web pages that do not follow links. The equation

$$x = Ax$$

can be written

$$(I - pGD)x = \gamma e$$

where

$$\gamma = z^T x.$$

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

$$(I - pGD)x = e$$

can be solved for x. Then the resulting x can be rescaled so that

$$\sum_i x_i = 1.$$

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 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*.

```
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 roundoff error, the computed matrix `I - A` is probably not exactly singular. Even if it is singular, roundoff 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)\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*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 7.1 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 *URL*s. 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 `k`th 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.

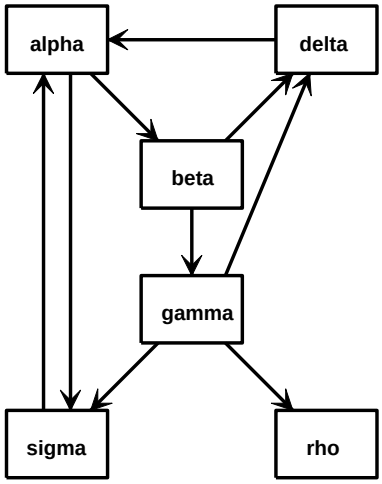

Figure 7.1. *A tiny Web.*

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

```
c =
   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
```

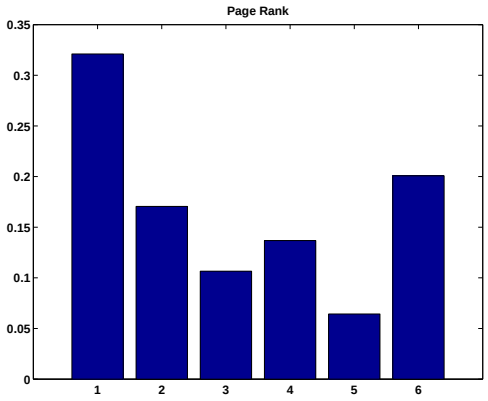

Figure 7.2. *Page Rank for the tiny Web*

The `bar` graph of `x` is shown in figure 7.2. If the URLs are sorted in PageRank order and listed along with their in- and out-degrees, the result is

```
   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.

The `exm` toolbox includes the program `surfer`. 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.harvard.edu',500)
```

accesses the home page of Harvard University and generates a 500-by-500 test case. The graph generated in August 2003 is available in the `exm` toolbox. The statements

```
load harvard500
spy(G)
```

produce a `spy` plot (Figure 7.3) that shows the nonzero structure of the connectivity matrix. The statement

```
pagerank(U,G)
```

computes page ranks, produces a bar graph (Figure 7.4) of the ranks, and prints the most highly ranked URLs in PageRank order.

For the `harvard500` data, the dozen most highly ranked pages are

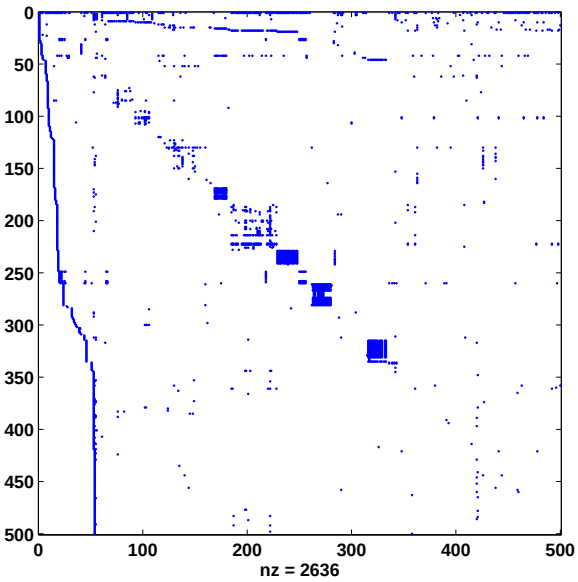

Figure 7.3. *Spy plot of the* `harvard500` *graph.*

	page-rank	in	out	url
1	0.0843	195	26	http://www.harvard.edu
10	0.0167	21	18	http://www.hbs.edu
42	0.0166	42	0	http://search.harvard.edu:8765/custom/query.html
130	0.0163	24	12	http://www.med.harvard.edu
18	0.0139	45	46	http://www.gse.harvard.edu
15	0.0131	16	49	http://www.hms.harvard.edu
9	0.0114	21	27	http://www.ksg.harvard.edu
17	0.0111	13	6	http://www.hsph.harvard.edu
46	0.0100	18	21	http://www.gocrimson.com
13	0.0086	9	1	http://www.hsdm.med.harvard.edu
260	0.0086	26	1	http://search.harvard.edu:8765/query.html
19	0.0084	23	21	http://www.radcliffe.edu

The URL where the search began, `www.harvard.edu`, dominates. Like most universities, Harvard is organized into various colleges Harvard Medical School, the Harvard Business School, and the Radcliffe Institute. You can see that the home pages of these schools have high PageRank. With a different sample, such as the one generated by Google itself, the ranks would be different.

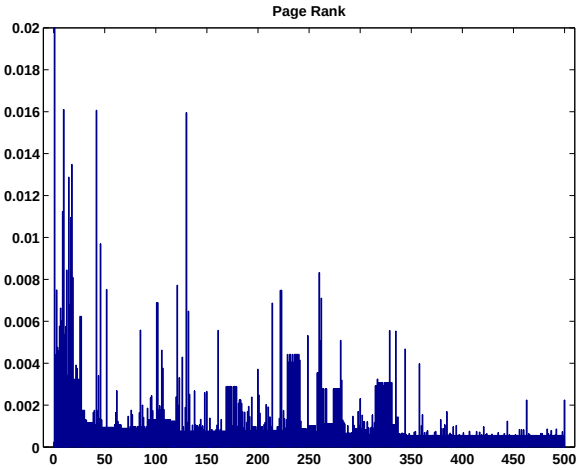

Figure 7.4. *PageRank of the* `harvard500` *graph.*

Further Reading

Further reading on matrix computation includes books by Demmel [**?**], Golub and Van Loan [**?**], Stewart [**?**, **?**], and Trefethen and Bau [**?**]. The definitive references on Fortran matrix computation software are the LAPACK Users' Guide and Web site [**?**]. The MATLAB sparse matrix data structure and operations are described in [**?**]. Information available on Web sites about PageRank includes a brief explanation at Google [**?**], a technical report by Page, Brin, and colleagues [**?**], and a comprehensive survey by Langville and Meyer [**?**].

Recap

```
%% Page Rank Chapter Recap
% This is an executable program that illustrates the statements
% introduced in the Page Rank Chapter of "Experiments in MATLAB".
% You can access it with
%
%    pagerank_recap
%    edit pagerank_recap
%    publish pagerank_recap
%
%  Related EXM programs
%
%  surfer
%  pagerank

%% Sparse matrices
```

```
   n = 6
   i = [2 6 3 4 4 5 6 1 1]
   j = [1 1 2 2 3 3 3 4 6]
   G = sparse(i,j,1,n,n)
   spy(G)

%% Page Rank
   p = 0.85;
   delta = (1-p)/n;
   c = sum(G,1);
   k = find(c~=0);
   D = sparse(k,k,1./c(k),n,n);
   e = ones(n,1);j
   I = speye(n,n);
   x = (I - p*G*D)\e;
   x = x/sum(x)

%% Conventional power method
   z = ((1-p)*(c~=0) + (c==0))/n;
   A = p*G*D + e*z;
   x = e/n;
   oldx = zeros(n,1);
   while norm(x - oldx) > .01
      oldx = x;
      x = A*x;
   end
   x = x/sum(x)

%% Sparse power method
   G = p*G*D;
   x = e/n;
   oldx = zeros(n,1);
   while norm(x - oldx) > .01
      oldx = x;
      x = G*x + e*(z*x);
   end
   x = x/sum(x)

%% Inverse iteration
   x = (I - A)\e;
   x = x/sum(x)

%% Bar graph
   bar(x)
   title('Page Rank')
```

Exercises

7.1 Use `surfer` and `pagerank` to compute PageRanks for some subset of the Web that you choose. Do you see any interesting structure in the results?

7.2 Suppose that `U` and `G` are the URL cell array and the connectivity matrix produced by `surfer` and that `k` is an integer. Explain what

```
U{k}, U(k), G(k,:), G(:,k), U(G(k,:)), U(G(:,k))
```

are.

7.3 The connectivity matrix for the `harvard500` data set has four 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 indices around 170 and the other three have indices in the 200s and 300s. Mathematically, a graph with every node connected to every other node is known as a *clique*. Identify the organizations within the Harvard community that are responsible for these near cliques.

7.4 A Web connectivity matrix G has $g_{ij} = 1$ if it is possible to get to page i from page j with one click. If you multiply the matrix by itself, the entries of the matrix G^2 count the number of different paths of length two to page i from page j. The matrix power G^p shows the number of paths of length p.

(a) For the `harvard500` data set, find the power p where the number of nonzeros stops increasing. In other words, for any `q` greater than `p`, `nnz(G^q)` is equal to `nnz(G^p)`.

(b) What fraction of the entries in G^p are nonzero?

(c) Use `subplot` and `spy` to show the nonzeros in the successive powers.

(d) Is there a set of interconnected pages that do not link to the other pages?

7.5 The function `surfer` uses a subfunction, `hashfun`, to speed up the search for a possibly new URL in the list of URLs that have already been processed. Find two different URLs on The MathWorks home page `http://www.mathworks.com` that have the same `hashfun` value.

7.6 Figure 7.5 is the graph of another six-node subset of the Web. In this example, there are two disjoint subgraphs.

(a) What is the connectivity matrix G?

(b) What are the PageRanks if the hyperlink transition probability p is the default value 0.85?

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

7.7 The function `pagerank(U,G)` computes PageRanks by solving a sparse linear

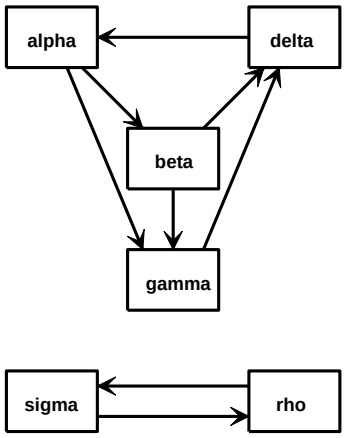

Figure 7.5. *Another tiny Web.*

system. It then plots a bar graph and prints the dominant URLs.

(a) Create `pagerank1(G)` by modifying `pagerank` so that it just computes the PageRanks, but does not do any plotting or printing.

(b) Create `pagerank2(G)` by modifying `pagerank1` 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?

(c) Create `pagerank3(G)` by modifying `pagerank1` 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?

(d) 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.

7.8 Here is yet another function for computing PageRank. This version uses the power method, but does not do any matrix operations. Only the link structure of the connectivity matrix is involved.

```
function [x,cnt] = pagerankpow(G)
```

```
% PAGERANKPOW  PageRank by power method.
% x = pagerankpow(G) is the PageRank of the graph G.
% [x,cnt] = pagerankpow(G)
%    counts the number of iterations.

% Link structure

[n,n] = size(G);
for j = 1:n
   L{j} = find(G(:,j));
   c(j) = length(L{j});
end

% Power method

p = .85;
delta = (1-p)/n;
x = ones(n,1)/n;
z = zeros(n,1);
cnt = 0;
while max(abs(x-z)) > .0001
   z = x;
   x = zeros(n,1);
   for j = 1:n
      if c(j) == 0
         x = x + z(j)/n;
      else
         x(L{j}) = x(L{j}) + z(j)/c(j);
      end
   end
   x = p*x + delta;
   cnt = cnt+1;
end
```

(a) How do the storage requirements and execution time of this function compare with the three `pagerank` functions from the previous exercise?

(b) Use this function as a template to write a function that computes PageRank in some other programming language.