CS 111 Assignment 4 Additional Problem: PageRank by the Power Method

Assigned Friday, February 11, 2016

Due by class time Wednesday, February 24, 2016

In class, we computed the PageRank ordering from the adjacency matrix of a directed graph by applying Matlab's built-in `eig()` function. The method we used doesn't work for very large graphs/matrices, because it forms a completely dense n-by-n matrix M, which requires $O(n^2)$ memory to store and $O(n^3)$ time to run `eig()`. In this assignment you will write a Matlab code that works for much larger graphs, using the power method to find the eigenvector, without ever forming a dense matrix. You will start with a Matlab program that I wrote, which uses the power method but still forms the dense matrix M, and you will modify it so that it doesn't need the dense matrix.

1 Code and data

The Matlab directory linked to the GauchoSpace site contains three files that you'll use for this assignment:

- `pagerank1.m`, which is the Matlab code you'll start from.
- `pagerankmats.mat`, which is a data file containing three sample matrices: `EG1` and `EG2` are the tiny examples I did in class, and `EG3` is the 500-vertex web crawl of `harvard.edu`. The cell array `HarvardCrawl` is the URL's of those 500 web pages.
- `bigwebgraph.mat`, which is a data file containing just one matrix, from a crawl of just under a million web pages.

Here is the result of running `pagerank1()` on matrix `EG2`:

```
>> [r,v] = pagerank1(EG2)
Dominant eigenvalue is 1.000000 after 2 iterations.
r =
     3
     4
     1
     2
     5
v =
    0.4054
    0.4054
    0.5777
```

```
    0.5777
    0.0608
```

And here are the ten top-ranked pages in the Harvard web crawl:

```
>> [r,v] = pagerank1(EG3);
Dominant eigenvalue is 1.000000 after 56 iterations.
>> HarvardCrawl(r(1:10))
ans =
    'http://www.harvard.edu'
    'http://www.hbs.edu'
    'http://search.harvard.edu:8765/custom/query.html'
    'http://www.med.harvard.edu'
    'http://www.gse.harvard.edu'
    'http://www.hms.harvard.edu'
    'http://www.ksg.harvard.edu'
    'http://www.hsph.harvard.edu'
    'http://www.gocrimson.com'
    'http://www.hsdm.med.harvard.edu'
```

The routine you write, `pagerank2()`, should be able to duplicate these results, and should also run correctly on the big web graph. If you run `pagerank1()` on the big web graph, Matlab will either just hang or complain that it's out of memory. On my 5-year-old Macbook Air, my own version of `pagerank2()` takes about 24 seconds to run on the big web graph.

2 The power method with a dense matrix

Let's look carefully at `pagerank1()`, which is listed on the last page of this handout. You can watch the code work by setting a debugger breakpoint in it and then running it on the 5-by-5 example matrix `EG2`, using "dbstep" to single-step the code and using the Matlab command line to examine the values of the variables. Say "help debug" to Matlab for more instructions.

Lines 20-26 check the input for validity. Notice the sneaky test on line 24, which verifies that every entry of `E` is either 0 or 1 without using a loop or creating any new big matrices.

Lines 28-39 fill in all 1's in every column that corresponds to a dangling vertex (a vertex with no links out of it). The vector `e` is a column of all 1's, and the vector `d` is a column that has 1's in positions corresponding to dangling vertices and 0's elsewhere. Then `d * e'` is the product of an n-by-1 vector and a 1-by-n vector, so it's an n-by-n matrix (of rank one) containing all the products of an element of `d` and an element of `e`. The resulting matrix `F` may have a lot more nonzeros than `E`; in your `pagerank2()`, you don't want to compute `F` explicitly.

Lines 41-47 scale the matrix to make it column stochastic, and then create the matrix `M` that represents choosing a page uniformly at random 15% of the time. Matrix `M` is completely dense, so you certainly don't want to compute it explicitly in `pagerank2()`.

Lines 49-60 use the power method to find the largest eigenvalue of `M` (which we know is equal to 1 by the Perron-Frobenius theorem) and its associated eigenvector (which gives the PageRank ratings). The loop just multiplies the vector `v` on the left by `M` repeatedly, rescaling it after each multiplication to have norm 1. The loop stops when the vector changes by less than 10^{-6}.

The final line computes the ranking permutation by sorting the eigenvector from largest to smallest element.

3 Getting rid of the dense matrix

Your `pagerank2()` should not compute any of the matrices `F`, `A`, `S`, or `M`. The question is, then, how do you get the effect of the line "`v = M*v`"? You can use the fact that, mathematically, $M = (1-m)A + mS$, so you can get the effect of multiplying a vector by M if you can multiply it both by A and by S. Given a vector v, what vector is Sv? How can you compute that vector without forming S?

Similarly, you can use the fact that $A = FD$, where $D = \text{diag}(1./\text{sum}(F))$, to figure out how to compute Av from v by multiplying a suitable vector x only by the matrix F. Then, finally, you can use $F = E + ed^T$ to figure out how to compute Fx from x: You need to compute ed^Tx from x without forming the matrix ed^T, and you need to compute Ex. In the end, the only matrix you actually need to multiply by is E.

4 What experiments to do

Write and debug a Matlab function `pagerank2()` that has exactly the same input and outputs as `pagerank1()`, but forms no large matrices besides its input matrix `E`. Verify that your code gets the same results as `pagerank1()` on the small examples and the Harvard crawl.

Run your code on the big web graph, timing it with `tic` and `toc`. What is the largest element in the PageRank vector? What is the smallest? Make a histogram of the logarithm of the ratings. Here's how I did that:

```
>> load bigwebgraph
>> tic;[r,v] = pagerank2(E);toc
Dominant eigenvalue is 1.000000 after 71 iterations.
Elapsed time is 23.956778 seconds.
>> max(v)
ans = ...
>> min(v)
ans = ...
>> hist(log10(v),100)
>> title('Histogram of PageRank ratings for BigWebGraph')
>> xlabel('base-10 log of rating')
```

5 What to turn in

Include all of the following in your report:

- Your Matlab source code `pagerank2.m`. If this calls any other Matlab code you wrote, include that too.
- Diary output from your code duplicating the results in Section 1 above.
- Diary output from your code running on the big web graph, with the `tic`/`toc` timer as above, and the max and min values. (Don't print out the values of `r` and `v` for this one!)
- Your histogram, formatted as nicely as you can.

```
function [ranking, vector] = pagerank1(E)
% PAGERANK1 : compute page rank from adjancency matrix
%
% [ranking, vector] = pagerank1(E)
%
% E is a matrix of 0s and 1s,
% where E(i,j) = 1 means that web page (vertex) j has a link to web page i.
%
% This computes page rank by the following steps:
%  1. Add links from any dangling vertices to all vertices.
%  2. Scale the columns to sum to 1.
%  3. Add a constant matrix to represent jumping at random 15% of the time.
%  4. Find the dominant eigenvector with the power method.
%  5. Sort the eigenvector to get the rankings.
%
% The homework problem due February 24 asks you to rewrite this code
% so it never creates a full matrix, or any large matrix other than E.


[nrows,n] = size(E);
if nrows ~= n
    error('E must be square');
end;
if (max(max(E)) ~= 1) || (sum(sum(E)) ~= nnz(E))
    error('E must contain only zeros and ones');
end;

%  1. Add links from any dangling vertices (danglers) to all vertices.
%     Note: This is a different way of doing this than we used in class.
e = ones(n,1);          % vector of all 1's
d = zeros(n,1);         % vector of all 0's
outdegree = sum(E);
for j = 1:n
    if outdegree(j) == 0
        d(j) = 1;
    end;
end;                    % d(j) is now 1 if j is a dangler

F = E + e*d';  % e*d' is an n-by-n matrix with all-1 columns for danglers.

%  2. Scale the columns to sum to 1.
A = F * diag(1 ./ sum(F));

%  3. Add a constant matrix to represent jumping at random 15% of the time.
S = ones(n)/n;
m = 0.15;
M = (1-m) * A + m*S;

%  4. Find the dominant eigenvector.
v = e / n;    % Start with a vector all of whose entries are 1/n.
oldv = zeros(n,1);
iters = 0;

while norm(v-oldv) > 1e-6
    oldv = v;
    v = M*v;
    lambda = norm(v);
    v = v / lambda;
    iters = iters + 1;
end;

fprintf('Dominant eigenvalue is %f after %d iterations.\n', lambda, iters);
vector = v;

%  5. Sort the eigenvector to get the rankings.
[ignore, ranking] = sort(vector,'descend');
```