Homework 7: Page Rank

Michael Overton, Numerical Computing, Spring 2017

April 11, 2017

Google uses a famous scheme, PageRank, to rank web pages. We start with an $n \times n$ *adjacency matrix* M, corresponding to a web of n pages, with $M_{ij} = 1$ if page j has a link to page i and $M_{ij} = 0$ otherwise. Let N_j be the number of pages that page j points to. Then, as long as $N_j > 0$, column j of a matrix A is set according to the following rule (see A&G p. 224)

$$A_{ij} = \begin{cases} \frac{1}{N_j} \text{ if } M_{ij} = 1 \\ 0 \text{ otherwise.} \end{cases}$$

By construction, the jth column of A sums up to one. However, if $N_j = 0$, which happens when page j does not point anywhere (the "dangling node" case), this does not work, so instead we set (see A&G p. 225)

$$A_{ij} = \frac{1}{n} \text{ for all } i = 1, \ldots, n.$$

Then the jth column of A sums to one in this case too.

The "Google matrix" is then defined to be (see A&G, p. 226)

$$G = \alpha A + (1 - \alpha)\frac{1}{n}ee^T$$

where α is a "damping parameter" satisfying $0 < \alpha < 1$ and $e = [1, \ldots, 1]^T$. Let us denote the eigenvalues of G by $\lambda_1, \ldots, \lambda_n$, where

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

that is, ordered by "complex modulus" or "complex magnitude" (see A&G, p. 71).

1. Show that each column of G also sums to one, and therefore $\|G\|_1 = 1$.

2. On p. 226 of A&G it says we could use any vector u instead of $(1/n)e$ in the equation for G, but if we did that, the columns of G might not sum to one. What condition on u would we need to make sure the columns of G still sum to one? (We will stick with $(1/n)e$ in the definition of G.)

3. Show that e, the vector of all ones, is an eigenvector of the *transpose* of the Google matrix corresponding to the eigenvalue one. (Equivalently, e^T is a "left eigenvector" of G corresponding to the eigenvalue one, since $e^T G = e^T$.) Since the eigenvalues of G and G^T are the *same*, this means that G also has an eigenvalue one, but *we do not know the corresponding eigenvector x*. This is what PageRank computes, using the power method.

4. Look at the bottom of p. 77 in A&G. This defines the *spectral radius* of a matrix B, denoted $\rho(B)$, which is the maximum of the eigenvalues of B measured in complex modulus. Thus, using the ordering of λ_j given above, $\rho(G) = |\lambda_1|$. The last line on p. 77 explains that, by definition of an induced norm, the spectral radius of B is always less than or equal to $\|B\|$ for *any* induced norm. What does this tell you about *all* the eigenvalues of G?

The matrix G is a "nonnegative" matrix, in the sense that all its entries $G_{ij} \geq 0$ (this is not the same as being positive semidefinite; in fact, G is not a symmetric matrix). The remarkable theory of nonnegative matrices says that only one eigenvalue of G, namely λ_1, equals one, and that the eigenvector x corresponding to the eigenvalue one is *also* nonnegative (or nonpositive, since we can always scale it by -1, but it's more convenient to take it as nonnegative). It also says that all the *other* eigenvalues of G, λ_j, $j = 2, \ldots, n$, satisfy $|\lambda_j| < 1$.

5. Show that this last conclusion is not true if we allow $\alpha = 1$. In particular, could *all* the eigenvalues of G be *equal to one*? What would this say about the page links in the web?

6. What are the eigenvalues of G when $\alpha = 0$?

7. Write a function whose input is the adjacency matrix M as a sparse matrix, and the damping parameter α, that computes the matrices A and G and the eigenvalues of G using `eig`. This routine requires its input matrix to be full, not sparse, and anyway although A is sparse, G is completely full. Load the adjacency matrix M for the Google 500 graph (see the course web page) which is small enough that the full format for G is not a problem and `eig` is fast. Setting $\alpha = 0.85$, compute the eigenvectors and eigenvalues of G with `[X,Lambda] = eig(G)`, and plot all the eigenvalues as complex numbers in the complex plane: `lambda=diag(Lambda); plot(real(lambda),imag(lambda),'rx'), axis equal` (this plots red crosses). Repeat for $\alpha = 0.5$ and $\alpha = 0.25$, using different colors and symbols. Show them all on the same plot and also plot circles of radius α in compatible colors. Use `legend` to show what's what. What do you notice about the eigenvalues?

8. Again for each of $\alpha = 0.85$, 0.5 and 0.25, set x to the *column of X corresponding to the eigenvalue one in* `lambda` (the eigenvalues may not be delivered in sorted order), normalize it so $\|x\|_1 = 1$, and double check that it is a positive vector (multiplying by -1 if it is a negative vector; if it has mixed signs you have a bug). Sort its components with `[xsort,index]=sort(x,'descend')`, and print the largest 10 components along with the corresponding URL names in `url`, namely `url(index(1:10))`. These are the highest 10 ranked web pages in the Google 500 graph (the result may depend on α).

9. Now write a function to implement the power method for computing the desired page rank vector x, which requires *matrix-vector products* using the formula for G: do *not* store the full matrix G. A key point is that you can multiply $(1/n)ee^T$ onto a vector v to give you $(1/n)ee^Tv = (1/n)(e^Tv)e$

without computing or storing ee^T. For the same reason, do not explicitly replace the zero columns of A by the vectors $(1/n)e$; you just need to remember to include these in the matrix vector product.

Set the inital vector to $(1/n)e$. Since G has column sums equal to one, you can omit the normalization step in the power method: the vector should always have 1-norm equal to one. Since we know the eigenvalue is one, instead of the Rayleigh quotient we can use the eigenvector residual norm $\mu = \|Gv - v\|_1$ as a termination criterion (but compute Gv only once, not twice, in the loop). Test your code on the Google 500 matrix with $\alpha = 0.85$, running it until μ is "sufficiently small", and make sure that the final v is reasonably close to the normalized x found by `eig`; if not, you have a bug. Output a vector of all the residuals μ and plot it against the iteration count. What is the "rate of convergence", that is, by approximately what factor is the residual reduced each step in the final stages of the iteration? How does this relate to $|\lambda_2|$, the second largest eigenvalue delivered by `eig`? (You can use `sort(lambda,'descend')` to sort the eigenvalues, since it sorts complex numbers by complex modulus, as desired). Repeat for the other two values of α, namely, 0.5 and 0.25.

10. Once you are sure your power method code is working correctly, load the Purdue 77587 graph which gives A directly, *not* the adjacency matrix M. The column sums of A are already one and there are no zero columns. It is no problem to store the very sparse matrix A, but storing G would result in running out of memory. The power method should still work very efficiently. If it is too slow for $\alpha = 0.85$, focus on $\alpha = 0.5$ and $\alpha = 0.25$. Plot the μ values, and print the top 10 page ranks and corresponding URLs. Estimate the modulus of the second largest eigenvalue, $|\lambda_2|$, *based on the convergence rate of* μ *and your previous observations.*

11. Extend your power method code to implement "subspace iteration", defined on A&G p. 339, which I prefer to call the "block power" method. The orthogonalization step, computed by `[Vnew,R]=qr(V,0)`, is essential; without it, all the columns of V will converge to the eigenvector x. Now the idea is that the columns of the $n \times m$ matrix V should converge to the first m Schur vectors in the Schur decomposition of A, with eigenvalues ordered by complex modulus. To be efficient, m must be small, and to converge, we need to ensure $|\lambda_m| > |\lambda_{m+1}|$. For this reason, $m = 2$ is a bad choice: **explain why.** Try $m = 3$. Then, hopefully, the eigenvalues of the final 3×3 matrix $V^T AV$ will approximate the eigenvalues $\lambda_1 = 1$, λ_2, λ_3 of G. Try this first on the Google 500 matrix, for the three values of α, comparing what you get from subspace iteration with the λ_2, λ_3 obtained from `eig`. Then, if that works, try the Purdue 77587 graph and report what you get from subspace iteration with your *estimated* $|\lambda_2|$ reported in the previous question.

Credits: I collected the Google graph data using Cleve Moler's `surfer.m`. The Purdue graph dates from 2001 and was taken from

`http://www.cs.purdue.edu/homes/dgleich/cs515-2015/homeworks/purdue_web.mat`