Homework 7
MATH 504

1. Reading from Sauer textbook

   - Section 12.1.1 and 12.1.2 discuss power iteration. See Reality Check 12 for the Page Rank algorithm.

2. Attached you will find two files, `nodes.txt` and `edges.txt` which give the edges and nodes for the web pages at Hollins University (a small university in Virginia). Each edge entry is two numbers, reflecting a link in one web page, given by the first number, to another web page, given by the second number. `nodes.txt` is in fact not needed, but it provides the web page names which makes things more interesting.

   Recall that the PageRank algorithm involves constructing a matrix A which models the random movements of a web surfer. The matrix A models a web surfer clicking randomly on links in a page. In class, we discussed that the rate of power iteration depends on the ratio $|\lambda_2/\lambda_1|$. The smaller this ratio, the quicker the convergence. For large matrices, λ_2/λ_1 will be very close to 1 and convergence will be slow. As discussed in Sauer, one way that Google dealt with this slow convergence is by slightly changing the web surfer model. We select a number $p \in [0, 1]$. Then, we assume that with probability p the web surfer randomly, with uniform probability, jumps to any page in the network given in A and with probability $(1-p)$ the web surfer randomly, with uniform probability, jumps to a page with a link given in the current page. This means that the original matrix A is replaced by a matrix M where

   $$M = (1-p)A + pB, \tag{1}$$

   and where I leave it to you to determine the matrix B. Apparently, Google originally used roughly $p = .15$ for their ranking algorithm.

   In this problem you'll apply power iteration to compute the dominant eigenvector v which has eigenvalue 1 and satisfies $M^T v = v$. (REMEMBER, you need to consider M^T not M!) Remember, the v you compute will have to be normalized to sum to 1, rather than the usual normalization of length 1. If you normalize v in this way then v_i will represent the probability of being at page i after a long time surfing the web. If we sort the pages in descending order of their v_i we get the PageRank ordering.

   A nice package for working with graphs is **igraph**. Two commands you will find useful are

   - `graph_from_data_frame` - Given a data.frame with two columns, in which each row represents an edge, creates an igraph object.
   - `as_adjacency_matrix` - Given an igraph object, returns an adjacency matrix. The adjacency matrix has a 1 if an edge connects two nodes. TO GET A, NORMALIZE THE ROWS OF THE ADJACENCY MATRIX

SO THAT EACH ROW SUMS TO 1. Use these two commands and **read.table** to extract the A matrix from the dataset edge file.

ONE FINAL ISSUE IS THAT SOME WEB PAGES HAVE NO LINKS IN THEM. THIS MEANS THAT OUR A MATRIX WILL HAVE SOME ROWS WITH ALL 0's. TO AVOID THIS SITUATION, ADD EDGES SO THAT EACH PAGE HAS AN EDGE (LINK) CONNECTING IT TO ITSELF.

(a) To get your code running and for orientation, build a toy network of 5 nodes (pages). Make sure that there is a path from each node in the network to every other node. Using different values of p, compute the dominant eigenvector v using power iteration and use **eigen** to determine the first two eigenvalues. (Remember to tranpose your matrix!) Discuss how p affects the distance between λ_1 and λ_2 and how this relates to the number of iterations needed for power iteration. If $\lambda = a + bi$, that is λ is complex, then it's size is $\sqrt{a^2 + b^2}$ and you can compare how far this is from 1.

(b) Now consider the Hollins Universtiy network and set $p = .15$. Compute the dominant eigenvector of M^T using power iteration. Try to use **eigen** to compute v; you'll see that eigen doesn't work well here. Determine the highest rated web page.

3. The Stanford Large Network Database is an excellent resource for large networks.

`http://snap.stanford.edu/data/`

A network of all web pages at Notre Dame university is available here

`http://snap.stanford.edu/data/web-NotreDame.html`

The network has about 300,000 nodes and about 1,000,000 edges. Look at the Notre Dame file to see the data format. Names have been stripped and the nodes in the network are simply numbered. Each row in the datafile contains two numbers, corresponding to a start node and end node connected by a directed edge. You can load the data into R using **read.table** as usual.

As we discussed in class, doing a single iteration of power method is $O(n^2)$. For the Hollins University dataset this is not too bad, but for the Notre Dame network this would be very slow and, besides, we can't store the full A matrix.. However, the adjacency matrix (i.e. A) of most networks will be sparse, meaning most entries will be zero. Both in terms of storage and matrix multiplication, great savings can be achieved by not considering 0 entries. This comes down to representing the matrix A in a data structure different than the usual $n \times n$ grid and exploiting this data structure to reduce the $O(n^2)$ to $O(n)$. We will not consider the specific data structures used to store sparse matrices, but instead simply use sparse matrices through R packages. The

Matrix package can handle sparse matrices. A nice, short writeup about R packages dealing with sparse matrices is given here

`www.johnmyleswhite.com/notebook/2011/10/31/using-sparse-matrices-in-r`

NOTICE that M is never sparse even though A is typically sparse because B has all non-zero entries. However, the matrix B is simple enough that we don't need to actually store it. More precisely, when you calculate Mv, you need to calculate $(1-p)Av$ through matrix multiplication, but pBv has a simple formula in terms of the sum of entries of v. BE VERY CAREFUL IN YOUR IMPLEMENTATION. If you have two matrices, say U and W, one of which is stored as a sparse matrix and one as a dense matrix, then $U+V$ will implicitly be converted to a dense matrix. Use the R function **class** to determine how your matrices are being stored as you apply power iteration in order to make sure that you are storing and manipulating your matrices to take advantage of sparseness.

To get a sparse matrix, use the `as_adjacency_matrix` function from igraph with the **sparse** flag set to TRUE.

(a) Try to create an A matrix with and without sparsity for this graph. You will see that without sparsity, R will run out of memory.

(b) Find the dominant eigenvector of M^T and determine the most popular pages. Use $p = .15$.