MACM 316 – Computing Assignment 3

Due Date: June 9th, at 11pm.

You must upload both your code (to Computing Assignment 3 - scripts) and your report (to Computing Assignment 3). The assignment is due at 11:00pm. I have set the due time in Canvas to 11:05pm and if Canvas indicates that you submitted late, you will be given 0 on the assignment. Your computing report must be exactly 1 page.

- Please read the **Guidelines for Assignments** first.
- Keep in mind that Canvas discussions are open forums.
- Acknowledge any collaborations and assistance from colleagues/TAs/instructor.

A. Computing Assignment – Google's page rank

Remark: to complete this assignment, you should download the files *harvard500.m* and *setup_page_rank.m*. You may also find it useful to look at the in-class demo *TicToc.m* (posted on lecture notes page).

In this assignment we will implement a toy version of Google's world renowned PageRank algorithm. One of the reasons why Google is such an effective search engine is the PageRank algorithm developed by Googles 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. Instead, 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 an internet user surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next. Then if certain web pages have more incoming links then our user will visit those pages more often. Furthermore, our user can end up in a dead end at pages with no outgoing links, or cycle around cliques of interconnected pages. To avoid such scenarios we assume that a certain fraction of the time, he/she will simply choose a random page from the Web. Such an internet surfer can be modelled using a theoretical random walk 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 it has a lot of incoming links and also other pages with high rank link to it. Our goal is to compute this page rank for a simple collection of webpages from the Harvard University website.

Start your script by calling the script *setup_page_rank.m* to read the variables U, G, z, p and N that are used throughout the assignment. The cell vector U contains a list of 500 web URLs from the Harvard website. We will rank these URLs based on their importance. p is a fixed number and N is the length of the vector z. The matrix G is the *connectivity matrix* of the URLs. $G_{ij} = 1$ if there is a hyperlink to page i from page j and $G_{ij} = 0$ otherwise. The matrix G can be very large but it has very few nonzero entries. The number of nonzero entries in G is the total number of hyperlinks between our URLs. Now define the vector c using

$$c_i = \sum_j G_{ij}.$$

In otherwords c_i is the total number of hyperlinks coming into page i. Next, define D as the diagonal matrix with entries

$$D_{ij} = \begin{cases} 1/c_i & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j. \end{cases}$$

Finally, construct the weight matrix A using the following expression:

```
A = p*G*D + ones(N, 1)*z';
```

Note that p, N and z are provided by the *setup_page_rank.m* script and you only need to construct the D matrix. The rank of each URL is contained in a normalized vector x (a vector of length one) which is the nontrivial solution to

$$x = Ax.$$

In order to solve for x you will implement a very elegant method called the *power method*. You should Google it.

a) Start by defining the initial vector x_0

```
x_0 = ones(N,1)./N;
```

Then the solution x can be approximated by implementing the following iteration:

```
x = x_0;
for j = 1:k

    x = A*x;
    x = x./norm(x);

end
```

Take $k = 15$ and provide a table that includes the five largest entries of x along with the corresponding URLs. **Hint 1: The top URL should be 'www.harvard.edu'. Hint 2: Look up MATLAB's 'sort' command.**

b) Now wrap the above for loop in another one and repeat the calculations for $k = 2^4, 2^5, \cdots, 2^{11}$. Then use the *tic* and *toc* commands to record the time required to estimate x for each value of k. Provide a loglog plot of the computation time versus k. Does this plot agree with your expectation of the operation counts?

c) Finally, repeat the calculations of part (b) by taking advantage of the sparsity of G and D. Replace your inner for loop with the following:

```
x = x_0;
G = sparse(G);
D = sparse(D);
GG = p*G*D;

for j = 1:k

    x = GG*x + ones(N,1)*(z'*x);
    x = x./norm(x);
end
```

Use the tic toc commands again and provide a plot of the computation time versus k. I recommend plotting this data on the loglog plot of part (b) for better comparison (look up the 'hold' command in MATLAB). Describe your observations.