Google uses an algorithm to rank pages by importance; a more important page will be higher on the search results list than a less important page. A page’s rank is determined by both how many sites link to it and the respective rank of each site that links to it as follows:

*1*

where is the set of pages that link to and is the number of sites that page links to. The element of the page rank vector contains the rank of page . We use vector as the page rank row vector whose elements are page ’s rank:

*2*

where is the number of pages being ranked. To calculate the page rank vector , we begin by assuming the initial rank for each page is . We then iterate the page rank vector according to:

*3*

This initial endogeneity within the system allows it to be iterated multiple times until it reaches a “steady state” where another iteration will not change the rankings of the pages.

For this project, we present an example mini-web of six different websites:

*4*

A surfer can move from one page to another following the arrows of the mini-web, which represent links directing the surfer to that page. Using the probabilities of moving between the six pages, hyperlink matrix **H** is determined:

*5*

A problem that arises from the way the web is set up is a “rank sink” is created when a surfer gets to page 2 (), for example. Page 2 does not link to any other site, so if a surfer can only travel via links they have nowhere to go in this case. This problem is solved by replacing each entry with 1/6, as there are six pages in this web. Also, an exogenous probability is introduced representing random movement between the six pages with direct linking. Adding the idea of the random movement and fixing the rank sink, the update matrix becomes matrix **G**:

*6*

With **G** being a 6x6 matrix, it can be decomposed into six distinct eigenvalues and six corresponding eigenvectors. The pages’ rankings at each iteration, denoted by vector , can therefore be decomposed into:

*7*

which is the summation of the eigenvectors multiplied by their eigenvalues of **G**. This is equivalent to continuously multiplying , by the update matrix **G** because of the eigenvalue eigenvector relationship:

*8*

Since matrix **G** is a substochastic matrix, one of its six eigenvalues is equal to one, by definition. This is also its largest eigenvalue, in magnitude. Since all other eigenvalues are less than one, as increases with each iteration, the terms with the other eigenvalues approach zero, and approaches a steady state.

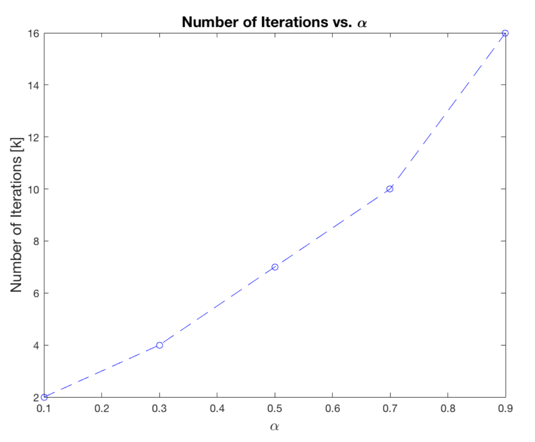
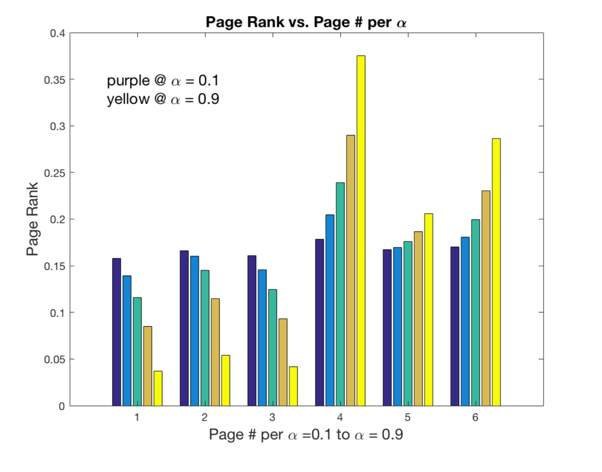
*9*

Using an of 0.9, the steady state is:

*10*

which took sixteen iterations to achieve, and the resulting ranking of the pages would be:

Repeating the process for at 0.7, 0.5, 0.3, and 0.1 are graphed in MATLAB:



where the figure on the left shows the value of on the x-axis with respect to the number of iterations to achieve steady state, which is represented on the y-axis. The figure on the right represents each page’s ranking in the mini-web. The x-axis represents each page in the mini-web. Further, the ranking of each page from left to right, or purple to yellow, corresponds to an of 0.1 to 0.9, respectively. The y-axis represents the magnitude of each page’s ranking as it relates to each .

Based on the result of using various values, 0.1seems to be the optimal value of alpha if one is only focused on computing power, since it takes only two iterations to achieve the steady state. In fact, the value and the number of iterations necessary have a strictly negative relationship, which is makes sense that an increasing adds more movement to the system. For this reason, the smallest possible should be chosen to save computing power. It would not be accurate, however, to simply set to be zero or very small since this is not an accurate representation of the behavior of web surfers. It is optimal, therefore, to choose an alpha that requires few iterations but also predicts a realistic probability of random surfer behavior.

The following pages describe the exact method used in MATLAB to calculate the steady state vector, and how it relates to decomposing the **G** matrix:

../Desktop/LA_Project_1.pdf





