Kashif Hussain Student ID: 8444507 MATH36032

PSBC: Assignment 1: Page Rank

24/02/2015

1: Constructing a Stochastic Matrix in MATLAB

I have designed a MATLAB function, `form_s(H)` which does the following:

1. Constructs a stochastic matrix **S** associated with the hyperlink matrix **H** which is the input argument.
2. If the function is fed a sparse hyperlink matrix **H**, then the function returns a sparse stochastic matrix **S**.

The matrix **S** is built from the following equation: $\mathbf{S} = \mathbf{HD} + \frac{1}{n}\boldsymbol{e}\boldsymbol{a}^T$.

Let $\mathbf{D} = \operatorname{diag}(d_j)$ with $d_j = 0$ if page j is a dangling node and $d_j = \left(\sum_{i=1}^{n} h_{ij}\right)^{-1}$ otherwise. Let n be the number of pages of the Web (in this case **H**) and e be the vector $\boldsymbol{e} = [1\,1\ldots1\,]^T \in \mathbb{R}^n$. Also, we define $\boldsymbol{a} \in \mathbb{R}^n$ such that $a_j = 1$ if page j is a dangling node and $a_j = 0$ otherwise.

The function below produces the stochastic matrix S:

```
function S = form_s(H)

%Initialising the vectors as described.
n = size(H,1);
a = zeros(n,1);
e = ones(1,n)';

% This ensures that D is sparse if H is sparse by constructing a sparse %diagonal
matrix D. The function sum(H,1) sums up all the values in a column
D = spdiags(1./sum(H, 1)', 0, n, n);

%If a column has inf value as a result of dividing by 0, change it to 0 %instead
if ~issparse(H)
      D(isinf(D)) = 0;
end

%Make a column matrix where a subscript i is = 1 if it is a dangling node,
%and 0 if it is not as described above.
for i = 1:n
    if sum(H(:,i)) == 0
      a(i,1) = 1;
    end
end

%Using the idea that a sparse matrix is formed if all the components are
%sparse, we check if H is sparse and ensure that a and e are both sparse.
if issparse(H)
    a = sparse(a);
    e = sparse(e);
end

%If sparse, prints a sparse matrix otherwise full matrix.
S = H*D + 1/n*e*a';
End
```

1.1: Testing the Function

A matrix $\mathbf{S} \in \mathbb{R}^{nxn}$ is column stochastic if all its elements are nonnegative and its columns sum to 1. In order to test the function, I will use the following theorem of stochastic matrices:

Theorem 1. *Every column stochastic matrix **S** has 1 as an eigenvalue and there exists an associated eigenvector p with nonnegative entries such that **Sp** = **p**. Moreover, the modulus of every eigenvalue of **S** is less than or equal to 1.*

Constructing the hyperlink matrix as in Figure 1 of the hand-out gives the following matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\mathbf{H_2}$ =

```
(2,1)        1      (6,5)        1
(3,1)        1      (1,7)        1
(5,1)        1
(6,1)        1
(3,2)        1
(4,3)        1
```

The *j*th column of **H** shows the links of the *j*th page and the *i*th row shows the links which point to that *i*th page. $\mathbf{H_2}$ is the sparse representation of **H** in MATLAB.

Running the function `S = form_s(H)`, we get:

```
S =

0              0        0      0.1429         0     0.1429    1.0000

0.2500         0        0      0.1429         0     0.1429         0

0.2500    1.0000        0      0.1429         0     0.1429         0

0              0   1.0000      0.1429         0     0.1429         0

0.2500         0        0      0.1429         0     0.1429         0

0.2500         0        0      0.1429    1.0000     0.1429         0

0              0        0      0.1429         0     0.1429         0
```

A similar result is found by running the function`S2 = form_s(H2)`where **H2** is the matrix $\mathbf{H_2}$

In order to check if they are column stochastic matrices, we can run `sum(S)` to sum their columns to check if they add up to 1, and as we can see, they do.

```
ans =  1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

1.2: The Google Matrix in MATLAB

Let α be the probability that a surfer chooses a link at random.

I have built a function which generates the Google matrix **G** for differing values of α in the range 0.3 to 0.9 increasing in steps of 0.1. The google matrix is represented by the following equation:

$$\mathbf{G}(\alpha) = \alpha\mathbf{S} + (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T \quad 0 < \alpha < 1,$$

where **S** is a column stochastic matrix, $\boldsymbol{e} = [1\,1\ldots1\,]^T \in \mathbb{R}^n$ and $\boldsymbol{v} \in \mathbb{R}^n$ with nonnegative entries v_i such that $\boldsymbol{e}^T\boldsymbol{v} = \sum_{i=1}^{n} v_i = 1$ is the so-called "personalization vector". In my function, $\boldsymbol{v} = \frac{\boldsymbol{e}}{n}$

```
function G = GoogleMatrix(S)

%Initialising the vectors as described in "Constructing a Stochastic Matrix %in
MATLAB
n = size(S,1);
e = ones(n,1);
v = e/n;

%G is an n by n 3d matrix with 7 layers, 1 for each value of alpha i.e. %Layer 1 will
contain the Google matrix with alpha value 0.3
G = zeros(n,n,7);
alpha = 0.3:0.1:0.9;

%This for loop calculates the Google matrix for all 7 alpha values
for i = 1:7
    k = alpha(i)*S + (1-alpha(i))*v*e';
    G(:,:,i) = k;

end

%This section plots the eigenvalues of the Google matrix using an x symbol %and
eigenvalues of S using a + symbol
plot(eig(G(:,:,1)), 'bx'); hold on;
plot(eig(G(:,:,2)), 'mx'); hold on;
plot(eig(G(:,:,3)), 'yx'); hold on;
plot(eig(G(:,:,4)), 'cx'); hold on;
plot(eig(G(:,:,5)), 'gx'); hold on;
plot(eig(G(:,:,6)), 'kx'); hold on;
plot(eig(G(:,:,7)), 'rx'); hold on;

%if sparse use eigs function as the function eig requires the matrix to be %real and
symmetric..
if issparse(S)
    plot(eigs(S), 'r+');
else
    plot(eig(S), 'r+');
end
title(['Graph of Google Matrix eigenvalues at alpha between 0.3 to 0.9 ' ...
      ' at steps of 0.1 & eigenvalues of S'])
xlabel('x value of complex eigenvalues')
ylabel('y values of complex eigenvalues')
legend('x = eigenvalues of Google Matrix','+ = eigenvalues of S')
```

See the last page for a graph which demonstrates the effect of differing the probability that a surfer randomly clicks on a link. It displays the eigenvalues of $\mathbf{G}(\alpha)$ and $\mathbf{S}$.

We can explain our observations with the use of the following theory:

Theorem 2. *Let* $\boldsymbol{S} \in \mathbb{R}^n$ *be column stochastic with eigenvalues* $\{1, l_2, l_3, \dots, l_n\}$. *Then the eigenvalues* $\boldsymbol{G}(\alpha) = \alpha \boldsymbol{S} + (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T$ *where* $0 < \alpha < 1$*and* $v \in \mathbb{R}^n$ *is a vector with nonnegative elements satisfying* $\boldsymbol{e}^T\boldsymbol{v} = 1$, *are* $\{1, \alpha l_2, \alpha l_3, \dots, \alpha l_n\}$.

As α increases in value closer and closer to 1, the eigenvalues of the Google matrix tends to the eigenvalues of the stochastic matrix **S.** We can see through the use of theorem that the eigenvalues of the google matrix **G** are simply the scaled values of **S** by the value of α.

2: Page Rank and the Power Method

Let $\boldsymbol{p}$ be the PageRank vector where $\boldsymbol{p} \in \mathbb{R}^n$, $p_i > 0$ and satisfies: $\mathbf{G}\boldsymbol{p} = \boldsymbol{p}$ and $\boldsymbol{e}^T\boldsymbol{p}^{(0)} = 1$. The standard approach to solving the PageRank problem is to implement the iteration:

$$\mathbf{p}^{(k)} = \mathbf{G}(\alpha)\mathbf{p}^{(k-1)}, \quad k \geq 1,$$

Using the fact that $\boldsymbol{G}(\alpha)$ is column stochastic and the theory in the handout, we can prove that $\lim_{k\to\infty} p^{(k)} = p$. Following from the equation of $\mathbf{p}^{(k)}$ we can deduce the following:

$\mathbf{p}^{(k)} = \mathbf{G}\mathbf{p}^{(k-1)} = \mathbf{G}^2\mathbf{p}^{(k-2)} = \cdots = \mathbf{G}^k\mathbf{p}^{(0)}$ and $\boldsymbol{e}^T\boldsymbol{p}^{(k)} = \boldsymbol{e}^T\mathbf{G}^k\boldsymbol{p}^{(0)} = 1$ as $\boldsymbol{e}^T\mathbf{G} = 1$

Let the eigenvalues of $\mathbf{G}$ be $l_1, l_2, \dots, l_n$ ordered so that $1 = |l_1| > |l_2| \geq \dots \geq |l_n|$.

We are assuming that $l_1 = 1$ is a dominant eigenvalue (has the largest magnitude. Further, we also assume that there is a corresponding linearly independent set of eigenvectors $r, v_2, \dots, v_n$. As a result, we can write the initial vector $\mathbf{p}^{(0)}$ as $\mathbf{p}^{(0)} = \alpha_1 r + \sum_{i=2}^{n} \alpha_i v_i$. Assume that $\alpha_1 \neq 0$.

Then $\mathbf{p}^{(k)} = \mathbf{G}^{(k)}\mathbf{p}^{(0)} = \alpha_1\mathbf{G}^{(k)}r + \sum_{i=2}^{n} \alpha_i\mathbf{G}^{(k)}v_i = \alpha_1 r + \sum_{i=2}^{n} \alpha_i l_i^k v_i$.

Therefore, due to the ordering of the eigenvalues of G, $l_i^k \to 0$ as $k \to \infty$ for $i \geq 2$, so $\lim_{k\to\infty} p^{(k)} = \alpha_1 r$ and as $\boldsymbol{e}^T\mathbf{p}^{(k)} = \mathbf{1} = \boldsymbol{e}^T r$ we must have that $\alpha_1 = 1$ which leads us to the conclusion that $\lim_{k\to\infty} p^{(k)} = r$. As a result, the limiting probability that an infinitely dedicated surfer visits any particular Web page is its PageRank.

2.1: Another equation for the Power Method

Using the formula $\mathbf{p}^{(k)} = \mathbf{G}(\alpha)\mathbf{p}^{(k-1)}$ where $k \geq 1$, we can show that it can be written in the form:

$$\mathbf{p}^{(k)} = \alpha\mathbf{S}\mathbf{p}^{(k-1)} + (1-\alpha)\boldsymbol{v}, \quad k \geq 1$$

We have that $\mathbf{G}(\alpha) = \alpha\mathbf{S} + (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T$, therefore:

$\mathbf{p}^{(k)} = \mathbf{G}(\alpha)\mathbf{p}^{(k-1)} = (\alpha\mathbf{S} + (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T)\mathbf{p}^{(k-1)} = \alpha\mathbf{S}\mathbf{p}^{(k-1)} + (1-\alpha)\boldsymbol{v}\boldsymbol{e}^T\mathbf{p}^{(k-1)}$.

As $\mathbf{p}^{(k-1)}$ is a discrete distribution (a Markov chain) which sums to 1 and using the fact that $\boldsymbol{e}^T\mathbf{p}^{(k-1)} = \mathbf{1}$**,** we get our required equation:

$$\mathbf{p}^{(k)} = \alpha\mathbf{S}\mathbf{p}^{(k-1)} + (1-\alpha)\boldsymbol{v}.$$

Using this equation results in less memory being used and processing power if $\mathbf{H}$ is sparse and for large values of n pages as it does not compute a matrix decomposition. This equation saves on calculations done as you do not need to calculate the Google matrix $\mathbf{G}$ in its entirety as well as saving on the calculation of multiplying $(1-\alpha)\boldsymbol{v}\boldsymbol{e}^T$ by $\mathbf{p}^{(k-1)}$ as $\boldsymbol{e}^T\mathbf{p}^{(k-1)} = \mathbf{1}$.

2.2: Designing the Power Method function in MATLAB

This function calculates the page rank for a given hyperlink matrix **H**, personality vector **v** and the probability that a surfer randomly clicks on a webpage *alpha.* Once the absolute value of the function is below 0.00005, the function terminates and returns the number of iterations and the PageRank.

```
function [p,cnt] = pm_pagerank(H,v,alpha)
%PM_PAGERANK PageRank by power method.
%   [P,CNT] = PM_PAGERANK(H,V,ALPHA) computes the PageRank vector p
%   associated with the connectivity matrix H, scaling parameter ALPHA and
%   personalization vector V.
%   CNT counts the number of iterations

n = length(H); %Number of Pages
S = form_s(H); %Constructs a stochastic matrix
u = (1-alpha)*v;
p = ones(n,1)/n;
p_prev = zeros(n,1);
cnt = 0; %Counting the number of iterations
while max(abs(p-p_prev)) > 1e-5
    p_prev = p;
    p = alpha*S*p_prev + u;
    cnt = cnt +1;
end
```

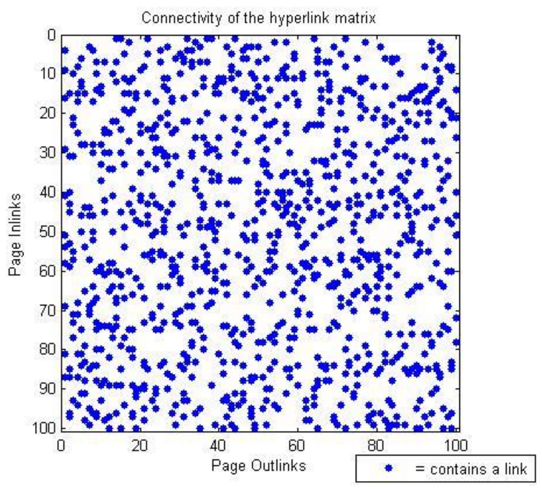
2.3: Constructing a random hyperlink matrix

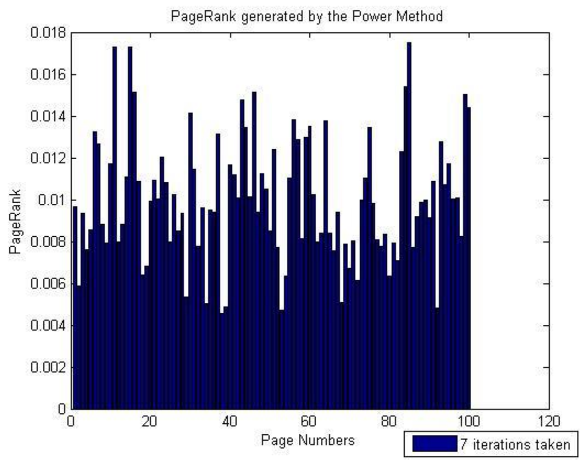
The continuing page shows my randweb function which produces a (psedo)random hyperlink matrix $\mathbf{H}$ with a specified amount of pages $n \in \mathbb{Z}$. The additional parameters are used for the following:

Outlinks: Defines the number of outlinks each page should have "on average"
Deviation: Defines how much the number of outlinks per page can deviate from the mean by.
Dangling: Defines how many dangling nodes are specified to be part of the hyperlink matrix.
Reaverage: Provides the option to re-randomize a certain amount of nodes such that the calculated mean is the one you specified as the number of outlinks "on average".
Returnsp: Provides the option to return a sparse matrix instead.

Initially the function normally distributes random pages with the number of outlinks specified as the mean and how much is can deviate by is given by the deviation input argument. If a certain amount of dangling nodes are specified, the function randomly chooses which nodes are to become dangling nodes. Reaverage, if selected, then re-averages the matrix by choosing a random page and assigning it a random hyperlink to another page until the average reaches that as specified by the outlinks argument.

The main for loop then generates a hyperlink matrix by first checking how many hyperlinks $h \in \mathbb{Z}$ can be assigned to a page $l \in \mathbb{Z}$ by examining the normally distributed matrix of integer values. Then the page is assigned h hyperlinks randomly, taking care to check if a page l has already been assigned or if the random page it has been assigned is itself. The function returns a (pseudo)randomly generated hyperlink matrix. Below is a spy plot showing the connectedness of the matrix and a bar chart showing the PageRank of 100 pages with 100 outlinks.





For the PageRank I chose an alpha value of 0.9. To confirm that it is indeed the correct Pagerank, I computed: **Gp** which returned the same **p** value satisfying the following equation: **Gp** = **p.**

In computer code, I used the following: `G(:,:,7) * p.`

```
function Z = randweb(pages,outlinks,deviation,dangling,reaverage,returnsp)
%pages: The number of pages the random web must have
%outlinks: The number of outlinks each page should have on average.
%normally distributes values with mean as the value of outlinks and deviates by the
value of deviation.
M = round(deviation.*randn(pages,1) + outlinks);

%Creates a specific number of dangling nodes if requested.
if dangling > 0
    D = round(deviation.*randn(dangling,1) +outlinks);
    %Add the specified amount of dangling nodes
    for i = 1:dangling
        k = randi([1 pages]);
        while M(k,1) == 0 %Randomly selected the same dangling node
            k = randi([1,pages]);
        end
        M(k,1) = 0;
    end
    %Randomly reaverage other nodes such that the mean is equal to outlinks
    if reaverage == 1
        for s = 1:sum(D)
            j = randi([1,pages]);
            while M(j,1) == 0 %Randomly selected the same dangling node
                j = randi([1,pages]);
            end
            M(j,1) = M(j,1) +1;
        end
    end
end
Z = zeros(pages,pages);

%Generates hyperlink matrix by randomly assigning page l the value of %M(l,:) as the
number of outlinks to each page.
%NOTE: Diagonal set to 0 i.e. Discounting webpage's links to itself
%NOTE: Dangling node identified by column of 0's
for l = 1:pages
    for m = 1:M(l,:) %For each outlink belonging to a page
        r = randi([1,pages]); %Chooses a random page
        %check if outlink exists or links to itself
        if Z(r,l) ~= 1 && r ~=l
            Z(r,l) = 1;
        else
      %if it does, re-choose a random page until both conditions are false.
             while r == l || Z(r,l) == 1
                r = randi([1,pages]);
             end
             Z(r,l) = 1;
        end
    end
end

%check if wanted in sparse format
if returnsp == 1
    Z = sparse(Z);
end
```

3: A Random Walk compared to PageRank

3.1: The Problem

In order to construct a weighted directed graph, we can do the following:
Let $n \in \mathbb{Z}$ be the number of vertices and $s \in \mathbb{Z}$ be the number of surfers at each vertex. We can set the weight $w_k \in \mathbb{R}$ for each edge which the page hyperlinks to as $k = 1, \ldots, l \in \mathbb{Z}$ where $\sum_{k=1}^{l} w_k = 1$ for every vertex n and define it as the probability that a surfer s moves along that edge at each step of the stochastic process.

3.2: The Connection

The stochastic process converges to a stable directed graph in which the final states of each vertex relates to the popularity of the page which is equal to the page rank. Ultimately, the distribution of the surfers should be equal to that of the PageRank. As a result, we only need to keep track of the surfers at each vertex and further avoids matrix multiplication which is present in the PageRank power method.

3.3: The Function

The function below constructs the above process. Initially, it constructs a stochastic matrix **S** to hold the weights w_k of each edge which is a hyperlink to another page. It then computes a random walk for a specified number of iterations through the use of a histogram before finally returning the ending states of each vertex along with the vertex popularity.

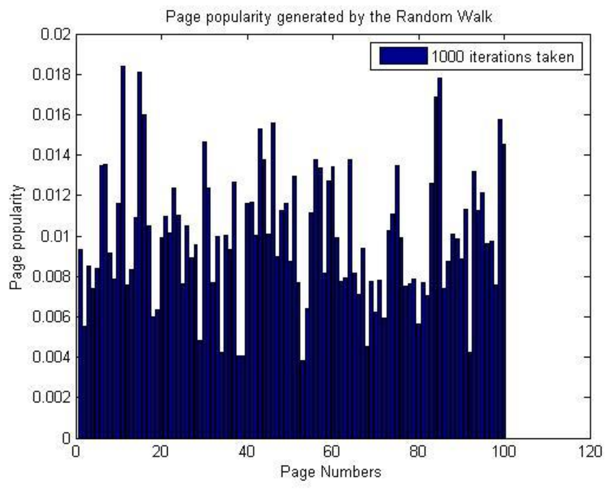
```
function [x, sumx] = randomwalk(A,surfers,iterations)
%A: The hyperlink matrix to be used
%surfers: The starting amount of surfers at each vertex
%iterations: How many times the function must iterate.

%Can only be used with full matrices
if issparse(A)
    A = full(A);
end


S = form_s(A);
[n,~] = size(S);
%Initialize surfer locations
x = surfers*ones(n,1);
edgeWeight = S;
%Allow teleportation from dangling node
for i = 1:n
    if S(i,i) == 1/n
%If at dangling node, teleport to any other node with probability 1/n
       edgeWeight(:,i) = 1/n;
    end
end

%Compute the walk for some iterations. At each step, a surfer moves from their
%current vertex to another vertex
for k = 1:iterations
    y = zeros(n,1); %where people end up
    for ivertx=1:n
        r = rand(x(ivertx),1);
        ws = cumsum(edgeWeight(:,ivertx)); %weights
        u = histc(r,[0, ws']);
       if (size(y) ~= size(u))
            u = u';
        end
        y = y + u(1:end-1);
    end
    x = y;
end
%calculate rank of each page
sumx = x./(surfers*n);
end
```

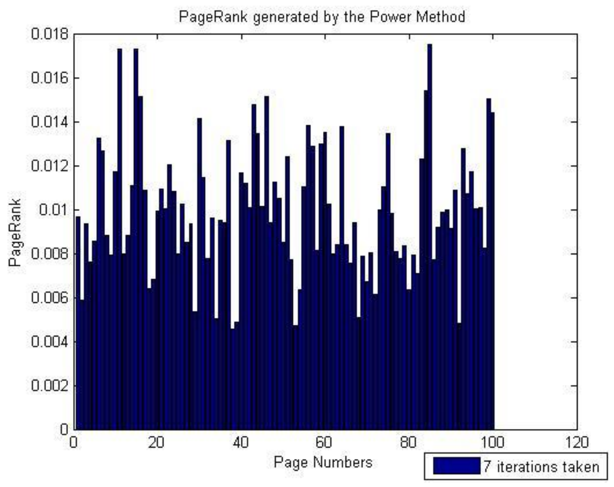
3.4: The Comparison

Below are two bar charts, one showing the PageRank of each page and the other the result of my randomwalk function. As we can seem after 1000 iterations, the random walk shows similar characteristics to that of the page rank with its troughs and peaks differing only by small amounts. It is converging to a steady states similar to that produced by the PageRank graph and it is the one I expected.

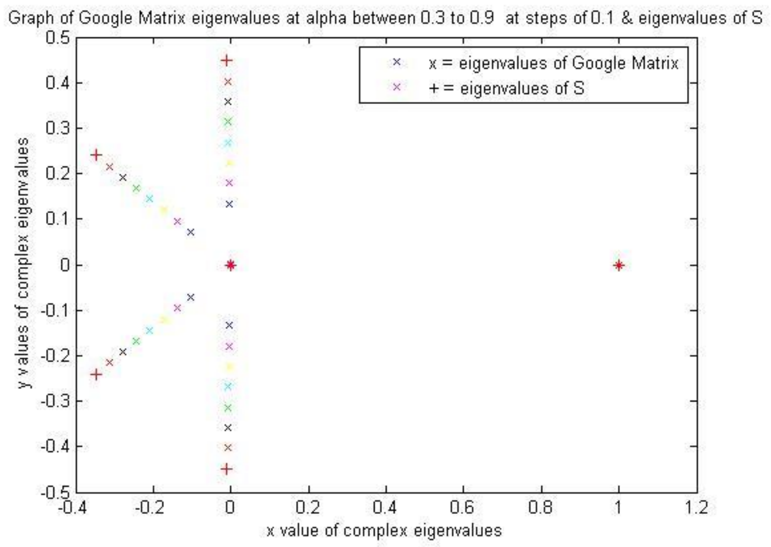
However, in the case of running a random walk with numerous dangling nodes (as is the case on the World Wide Web), the randomwalk converges to those dangling nodes as they have no outlinks which lead to other nodes. As a result, my function implemented a 'teleportation' option when a surfer happens to land on a dangling node in which they are given probability $\frac{1}{n}$ of teleporting to one of the other n vertices and in doing so, the function converges to the values produced by the PageRank.

Another important consequence to notice is the effect of having the number of surfers being an integer. In this case, we introduce rounding as if there is 1 surfer on node 1 with a 50% probability of going to node 2 or node 3, the surfer must choose one or the other as you cannot have ½ a surfer. This leads to a discrete distribution. If the surfers were not defined as integers, we would have non-discrete distribution and would have no rounding and as a result, it would take less iterations to match the distribution to that of the PageRank. However, it is more realistic to model a surfer as an integer as a surfer chooses only 1 page to visit.





1.2: The Google Matrix in MATLAB (Plot)


Graph of Google Matrix eigenvalues at alpha between 0.3 to 0.9 at steps of 0.1 & eigenvalues of S
x = eigenvalues of Google Matrix
+ = eigenvalues of S
y values of complex eigenvalues
x value of complex eigenvalues