Final Project

The PageRank Algorithm

1 Introduction

In 1998, two PhD students at Stanford proposed an algorithm that would transform the internet and shape the modern world. Sergey Brin and Larry Page have since used PageRank to create one of the most powerful and influential companies of the 21^{st} century. In this report we describe the underlying theory of PageRank and various methods of implementation.

2 Preliminaries

In order to examine the PageRank algorithm, it is imperative to examine the theory of eigenvectors and eigenvalues, and how they emerge in the study of Markov chains.

2.1 Eigenvalues and Eigenvectors

Definition 2.1: Given a matrix $A \in \mathbb{R}^{n\times n}$, a non-zero vector $v \in \mathbb{R}^n$ is an *eigenvector* of A with associated *eigenvalue* $\lambda \in \mathbb{R}$ if

$$Av = \lambda v. \tag{1}$$

There are various methods of finding the eigenvalues and eigenvectors of a matrix. It immediately follows from equation 1 that $(A - \lambda I)v = 0$ for $v \neq 0$ and so $A - \lambda I$ must not be invertible. So every eigenvalue λ must solve $\det(A - I\lambda) = 0$.

Definition 2.2: An eigenvalue λ^* of A is said to be the *dominant eigenvalue* if $|\lambda^*| > |\tilde{\lambda}|$ for all other eigenvalues $\tilde{\lambda}$ of A. The eigenvectors v^* associated with λ^* are said to be the *dominant eigenvectors*.

We will later study iterative methods for finding the dominant eigenvectors of A.

Theorem 2.1: A square matrix A and its transpose A^T have the same eigenvalues.

proof. Consider an arbitrary eigenvector λ of A. As discussed, λ must solve the equation $\det(A - \lambda I) = 0$. Since the determinant of a matrix is invariant under the transpose operation we know that $\det((A-\lambda I)^T) = 0$. However, since I is symmetric, $(A - \lambda I)^T = A^T - \lambda I$ and so λ solves $\det(A^T - \lambda I) = 0$. We may now conclude that λ is also an eigenvalue of A^T as needed.

2.2 Markov Chains

Definition 2.3: A discrete time *Markov chain* over a finite state space $\mathcal{X} = \{1, ..., n\}$ is a stochastic process $\{X_t\}_{t\in\mathbb{N}}$ where the future state is dependent only on the present state and not the past. Formally we say that

$$\mathbb{P}(X_{t+1} = x_{t+1}|X_1 = x_1, ..., X_t = x_t) = \mathbb{P}(X_{t+1} = x_{t+1}|X_t = x_t) \tag{2}$$

$\mathbb{P}(X_{t+1} = x_{t+1}|X_t = x_t)$ is said to be the transition kernel for time $t + 1$.

In modeling many different systems it is often natural to make the assumption that the future depends only on the present. Consider, for example, a college student aimlessly surfing the web. The website that the student travels to next is likely dependent on hyperlinks found on their current page which they may click on, and not by links on past webpages.

Definition 2.4: Given a matrix $P \in \mathbb{R}^{n\times n}$, a Markov chain is said to be *homogenous* if $\mathbb{P}(X_{t+1} = i|X_t = j) = P_{ji}$ for all $t \in \mathbb{N}$ and $i, j \in \mathcal{X}$. In other words, a *homogenous* Markov chain has a transition kernel that does not depend on time. P is called the *stochastic matrix* for the Markov chain $\{X_t\}_{t\in\mathbb{N}}$.

There are conditions which any stochastic matrix P must satisfy. Each element P_{ji} takes values only in $[0, 1]$ and each row in P must sum to 1.

Theorem 2.2: Every stochastic matrix P has a dominant eigenvalue of 1.

proof. Let $\tilde{v} = [1, 1, ..., 1]^T$. Since the rows of P sum to 1, it follows that $P\tilde{v} = \tilde{v}$ and so P has an eigenvalue of 1. We will now show that any other eigenvalue λ has an absolute value less than 1. Assume for sake of contradiction that P has an eigenvalue λ with $|\lambda| > 1$ and eigenvector v. Then $P^k v = \lambda^k v$ and so $\lim_{k\to\infty} P^k = \infty$. Since $v \neq 0$ this implies that $\exists k \in \mathbb{N},\ i, j \in \mathcal{X}$ such that $P^k_{ij} > 1$. This is a contradiction as we know from **Theorem 6.1** that P^k is a stochastic matrix and therefore cannot have entries which exceed 1. □

3 Formulation

Many search engines existed before Google. However, PageRank's novel approach weights search results based on the *value* of the page and not solely the page's content. The idea is that a user prefers a given site if many other pages contain links to the site. A highly linked page must be popular and therefore reliable. We are now ready to provide an informal description of the algorithm:

1. Identify all pages that contain a given term.

2. Rank them in terms of their *value*.

The remainder of this section will be dedicated to formalizing the *value* metric, with a series of improvements.

3.1 The Equation

Let $\mathcal{X} = \{1, ..., n\}$ be the set of all web pages and v_i be the *value* of page i for all $i \in \mathcal{X}$. If page j has a link to page i we say that $j \to i$.

If page i is linked to frequently, then we want to assign a higher value to page i. Therefore we could let

$$v_i = \sum_{j=1}^{n} \mathbb{1}_{j \to i} \tag{3}$$

where $\mathbb{1}_{j \to i}$ is 1 if there exists a link from page j to page i and 0 otherwise. We can also formulate equation 3 as a matrix vector multiplication. Let P be an $n \times n$ matrix where $P_{ij} = 1$ if page j contains a link to page i and $P_{ij} = 0$ otherwise. We will call P the connection matrix. Then $v = [v_1, ..., v_n]^T$ can be computed by

$$v = P * \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{4}$$

Improvement 1: Let o_j be the number of links on page j. If page j has many outgoing links then presumably each individual link is less important. Therefore we may modify the matrix P above by letting $P_{ij} = \frac{1}{o_j}$ if page j contains a link to i and 0 otherwise. Equation 3 then becomes:

$$v_i = \sum_{j=1}^{n} \frac{\mathbb{1}_{j \to i}}{o_j} \tag{5}$$

However, in order to avoid dividing by 0 in equation 3 we must make an important adjustment which deals with "dead end" pages. If a page is a "dead end" (contains no link to any other pages) we treat it as if it has a link to every other single page. Therefore we redefine $j \to i$ as *page j has a link to page i or page j has no outgoing links.*

Improvement 2: If page j is a very valued page and contains a link to page i then page i should be assigned a high value as well. Therefore we may modify equation 5

as follows:

$$v_i = \sum_{j=1}^{n} \frac{\mathbb{1}_{j \to i}}{o_j} * v_j \tag{6}$$

Referring back to our matrix vector expression for v it is immediate that we may rewrite the equation above as

$$v = Pv \tag{7}$$

Where $P_{ij} = \frac{\mathbb{1}_{j \to i}}{o_j}$ as before. In other words, v is an eigenvector of P with associated eigenvalue 1.

Moreover, notice that each column in P sums to 1 as

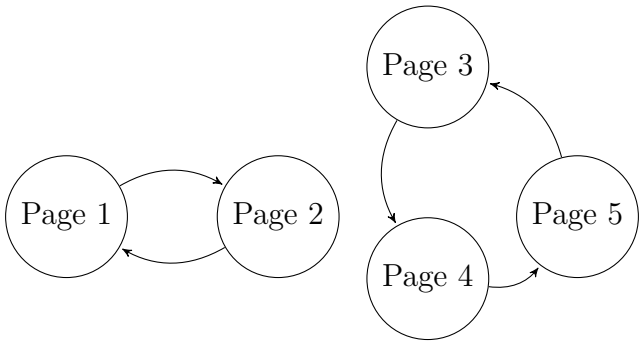
$$\sum_{i=1}^{n} P_{ij} = \sum_{i=1}^{n} \frac{\mathbb{1}_{j \to i}}{o_j} = \frac{1}{o_j} \underbrace{\sum_{i=1}^{n} \mathbb{1}_{j \to i}}_{=o_j} = 1.$$

And so P^T is a stochastic matrix. We may now conclude from theorem **Therorem 2.1** and **Therorem 2.2** that P is guaranteed to have an eigenvalue of 1 and that this eigenvalue will be dominant.

Improvement 3: We know that P is guaranteed to have dominant eigenvalue 1 but we do not know that there is a unique eigenvector which corresponds to this eigenvalue.

However, due to the vast nature of the web, the connection matrix P is *sparse*: P likely contains $O(n)$ non-zero entries as each page only links to a constant number of other pages. The connection matrix from the root `http://www.brown.edu` is an example of this phenomenon, as seen in figure 1. Moreover, we are likely to have groups of web pages connected to each other and not connected to any other groups. These groups are called *clusters*. For example, for the search term `Brown` we may have a cluster corresponding to the color and another corresponding to the school.

Sparse and clustered connection matrices P will often not have a unique eigenvector corresponding to the eigenvalue of 1. Consider, for example, the case of the search term `Brown`. We may have pages 1 and 2 corresponding to the school and pages 3-5 corresponding to the color. The link structure may then resemble the figure below:



In this case we would have connection matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

which we notice has eigenvectors v_1 and v_2, each corresponding to an eigenvalue of 1.

$$v_1 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix}^T \qquad v_2 = \begin{bmatrix} 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}^T$$

A ranking which assigns all the 'color' pages with high scores and 'school' pages with low scores is just as valid as a ranking which assigns the 'color' pages with high scores and 'school' pages with low scores. We would like to only have one valid ranking, and therefore need one final improvement. For this improvement we may turn to **Theorom 3.1** and **Theorom 3.2**.

Theorom 3.1: If $A \in \mathbb{R}^{n \times n}$ is a matrix with $A_{ij} > 0$ for all i, j and has dominant eigenvalue 1, then there is a strictly positive eigenvector associated with the eigenvalue 1.

proof. Let v be an eigenvector such that $Av = v$. Now define v^+ as $v_i^+ = |v_i|$. Notice that $Av^+ \geq v^+$ by the triangle inequality as

$$(Av^+)_i = \sum_{j=1}^{n} A_{ij}|v_j| \geq \left| \sum_{j=1}^{n} A_{ij} v_j \right| = |(Av)_i| = |v_i| = v_i^+.$$

Assume for sake of contradiction that $Av^+ > v^+$. Then $Av > (1+\epsilon)v^+$ for some $\epsilon > 0$. Additionally, $A^n v^+ > (1+\epsilon)^n v^+$ for all $n \in \mathbb{N}$. By Gelfand's formula (1941) we know that that $\rho(A) = \lim_{n\to\infty} \|A^n\|^{1/n}$ for any matrix norm, where $\rho(A)$ is the spectral radius of A, which in our case is 1. However, if $A^n v^+ > (1+\epsilon)^n v^+$ then $\lim_{n\to\infty} \|A^n\|^{1/n} \geq (1+\epsilon)$. This implies that $1 \geq 1 + \epsilon$ which is a contradiction.

And so it must be the case that $Av^+ = v^+$. In other words, v^+ is a non-negative eigenvector with eigenvalue 1. To complete our proof we must show that v^+ is strictly positive. We know that $v_i^+ = \sum_{j=1}^n A_{ij} v_j^+$. Since $A_{ij} > 0$ and at least one $v_j^+ > 0$ it follows that $\sum_{j=1}^n A_{ij} v_j^+ > 0$ and so $v_i^+ > 0$ as needed. □

Theorom 3.2: If $A \in \mathbb{R}^{n \times n}$ is a matrix with $A_{ij} > 0$ for all i, j has a dominant eigenvalue of 1, then this eigenvalue has a geometric multiplicity of 1.

proof. From **Theorom 3.1** we know that their is some strictly positive eigenvector $v > 0$ associated with the eigenvalue 1. Assume for sake of contradiction that there is another eigenvector u such that $Au = u$ where u and v are linearly independent. For sufficiently small $\epsilon > 0$ we can clearly see that $v - \epsilon u \geq 0$. Now let

$$\epsilon^* = \sup\{\epsilon > 0 : v - \epsilon u \geq 0\}$$

Clearly $v - \epsilon^* u \geq 0$ but also notice that at least one entry in $v - \epsilon^* u$ is equal to 0, call this entry i. Notice that $A(v - \epsilon^* u) = v - \epsilon^* u$ and since v and u are linearly independent by assumption it must be the case that at least one entry in $v - \epsilon^* u$ is greater than 0. And so

$$v_i - \epsilon u_i = 0 = \sum_{j=1}^{n} \underbrace{A_{ij}(v_j - \epsilon^* u_j)}_{>0 \text{ for at least one } j} > 0$$

which is a contradiction. □

So as our final improvement, we would like to make P_{ij} strictly greater than 0 as this would guarantee that the dominant eigenvalue is positive and unique. To do so we may simply add a positive constant to the connectivity matrix we have previously been working with. Define $C_{ij} = \frac{\mathbb{1}_{j \to i}}{o_j}$ and let $D_{ij} = 1/N$ for all i, j. Now let $P = (1 - r)D + rC$ for $r \in (0, 1)$ so that the columns of P still sum to 1. As before, our value vector v will be the dominant eigenvector of P, which we are now guaranteed is positive and unique.

In summary, the value vector v we wish to find is the solution to following equation:

$$\left(\underbrace{(1 - r)D + rC}_{P} \right) v = v \text{ with } r \in (0, 1),\ D_{ij} = \frac{1}{n},\ C_{ij} = \frac{\mathbb{1}_{j \to i}}{o_j} \tag{8}$$

Since $v > 0$ and eigenvalues are unaffected by scaling, we will take our solution to be the scaled version of v such $\sum_{i=1}^n v_i = 1$.

3.2 The Surfer

Let us return to the example of a college student aimlessly browsing the web. We call this student *the surfer*. At each time t, *the surfer* clicks on a link listed on the page

they are currently browsing with probability $r \in (0,1)$. However, with probability $1-r$ the student travels to a random website. If we let $X_t \in \mathcal{X}$ denote the web page the student is on at time t, then $\{X_t\}_{t\in\mathbb{N}}$ is a Markov chain. Moreover, X_t has the following transition kernel:

$$\mathbb{P}(X_{t+1} = i | X_t = j) = \frac{1-r}{n} + \frac{r\mathbb{1}_{j\to i}}{o_j} \tag{9}$$

Notice that equation 9 corresponds exactly to P_{ij} from equation 8. Therefore, P^T is a stochastic matrix for $\{X_t\}_{t\in\mathbb{N}}$.

Definition 3.1: A vector π is said to be the stationary vector of a Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with transition matrix T if $\pi T = \pi$.

Under the right conditions (see **Theorem 6.2** for more information)

$$\lim_{t\to\infty} \mathbb{P}(X_t = i) = \pi_i$$

Letting $\pi = v^T$ and $T = P^T$ we now we arrive at an even more intuitive notion of the value v_i of each page. The value of page i is the fraction of time that *the surfer* will spend on page i.

4 Eigenvalue/Eigenvector Problem Approach

We now begin to examine methods which find the value vector v that we defined. In this section we use the information that v is the unique, dominant eigenvector of equation 8 and simply look at algorithms for finding the dominant eigenvector.

4.1 The Power Method

We present now a simplified version of the Power Iteration Algorithm for the matrix P we have defined and provide its proof. For any vector $u \in \mathbb{R}^n$, write u as a linear combination of orthonormal eigenvectors of P. Say that $q_1, ..., q_m$ are the orthonormal eigenvectors of P with associated eigenvalues $\lambda_1, ...\lambda_m$ then

$$u = a_1 q_1 + ... + a_m q_m. \tag{10}$$

From equation 10 it follows that

$$P^k u = a_1 P^k q_1 + ... + a_m P^k q_m = a_1 \lambda_1^k q_1 + ... + a_m \lambda_m^k q_m \tag{11}$$

However, we already know that P has dominant eigenvalue 1 and so $\lambda_1 = 1$ and $\lambda_i^k \to 0$ as $k \to \infty$ for all other eigenvalues λ_i since $|\lambda_i| < 1$. Therefore equation 11

implies that $P^k u \to a_1 q_1$ as $k \to \infty$. Recall that we are looking for the dominant eigenvalue v and since v sums to 1 we may simply set $v = \frac{aq_1}{\|aq_1\|_1}$ and we will be done.

In practice, instead of taking $k \to \infty$ we may just proceed until $\|Pv - v\|_2 < \epsilon$. This leaves us with our first implementation of the PageRank algorithm:

```
function [v, res] = simple_power_iteration(G, r, eps, max_iter)
    n = length(G);
    D = (1/n)*ones(n,n);

    % If a page has no outgoing links, treat it as if it has outgoing links
    % to every other page (see Improvement 1).
    e = ones(n,1);
    for col = 1:n
        if sum(G(:,col)) == 0; G(:,col) = e; end
    end

    % We may generate matrix C by multiplying G by O where O is diagonal
    % with O(i,i) = 1/{# outgoing links from page i}
    O = diag(1./sum(G));
    C = G*O;

    % Now form P from equation 8.
    P = (1-r)*D + r*C;

    % res will store the norm between v and v_old
    res = zeros(max_iter,1);

    v = (1/n)*ones(n,1);
    for i = 1:max_iter
        v_old = v;
        v = P*v;
        res(i) = norm(v - v_old);
        if  res(i) < eps; break; end
    end
    v = v/sum(v);
end
```

We may run the algorithm and time it on 500 pages with the root of `http://www.brown.edu`.

```
>> load('brown500.mat')
>> r = 0.85; eps = 0.0001; max_iter = 100;
>> tic; [v, res] = simple_power_iteration(G, r, eps, max_iter); toc;
Elapsed time is 0.014564 seconds.
```

And finally we may view that `http://www.brown.edu` is number 1.

```
-------top 5 websites linking to http://www.brown.edu-------
    pagerank   in out  url
  1 5.210e-02 310  11  http://www.brown.edu
150 3.845e-02  39   1  http://xmlns.com/foaf/0.1
148 2.755e-02  38   1  http://purl.org/rss/1.0/modules/content
118 2.553e-02 225  14  http://www.brown.edu/contact
 15 2.490e-02 214   5  http://directory.brown.edu
```

4.2 Accelerating the Power Method: Sparsity

The most intensive computation in the algorithm presented above is the $O(n^2)$ matrix vector multiplication Pv. In this subsection we present a method to speed up this computation which relies upon the *sparsity* of the matrix G where $G_{ij} = \mathbb{1}_{\{j \text{ has a link to page } i\}}$.

Notice that we may rewrite the matrix P from equation 8 to take advantage of sparsity as follows:

$$P = ez + rGD \tag{12}$$

where

- $e = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}^T \in \mathbb{R}^{n \times 1}$.
- $z \in \mathbb{R}^{1 \times n}$ and $z_i = \frac{1-r}{n}$ if page i has a non-zero number of outgoing links and $z_i = \frac{1}{n}$ otherwise. This gives us that, as before, if a page has no outgoing links we assume that it has an outgoing like to every other page.
- D is diagonal with $D_{ii} = 1/(\text{number of links outgoing from page } i)$ unless there are no outgoing links from page i in which case $D_{ii} = 0$.

Now we observe that $Px = e(zx) + r(GD)x$ where GD is sparse. So the matrix multiplication we are now doing will be sparse and therefore faster. This leaves us with the following algorithm:

```
function [v, res] = sparse_power_iteration(G, r, eps, max_iter)
    n = length(G);

    c = sum(G,1);
    k = find(c~=0);
    D = sparse(k,k,1./c(k),n,n);
    T = sparse(r*G*D);

    z = sparse(((1-r)*(c~=0) + (c==0))/n);
    e = ones(n,1);
```

```
    res = zeros(max_iter,1);

    v = (1/n)*ones(n,1);
    for i = 1:max_iter
        v_old = v;
        v = e*(z*v) + T*v;
        res(i) = norm(v - v_old);
        if  res(i) < eps; break; end
    end
    v = v/sum(v);
end
```

We immediately see a speed up with the same results:

```
>> tic; [v, res] = simple_power_iteration(G, r, eps, max_iter); toc;
Elapsed time is 0.013368 seconds.
>> tic; [v2, res2] = sparse_power_iteration(G, r, eps, max_iter); toc;
Elapsed time is 0.001007 seconds.
>> disp(norm(v - v2));
   8.9962e-16
```

Let us examine speed for different values of n. We hypothesize that using sparsity the algorithm complexity will change from $O(n^2)$ to $O(n)$, as sparse matrix vector multiplication is $O(n)$. To confirm this hypothesis we randomly generate a sparse matrix for different values of n from 10 to 1000 and average the algorithm time for each run, as seen on figure 2.

4.3 Accelerating the Power Method: Adaptive Computation

Some entries in v may converge faster than others. Therefore at each step of the preceding algorithm we need only recompute the value v_i for entries i at which $|v_i^{(k)} - v_i^{(k-1)}| > \epsilon$.

The `for` loop then becomes:

```
    v = (1/n)*ones(n,1);
    not_converged = ones(n,1)==1;
    for i = 1:max_iter
        v_old = v;
        tmp = T(not_converged,:)*v;
        v(not_converged) = e(not_converged)*...
            (z(not_converged)*v_old(not_converged)) + tmp;
        not_converged = abs(v - v_old) > delta;
        res(i) = norm(v - v_old);
```

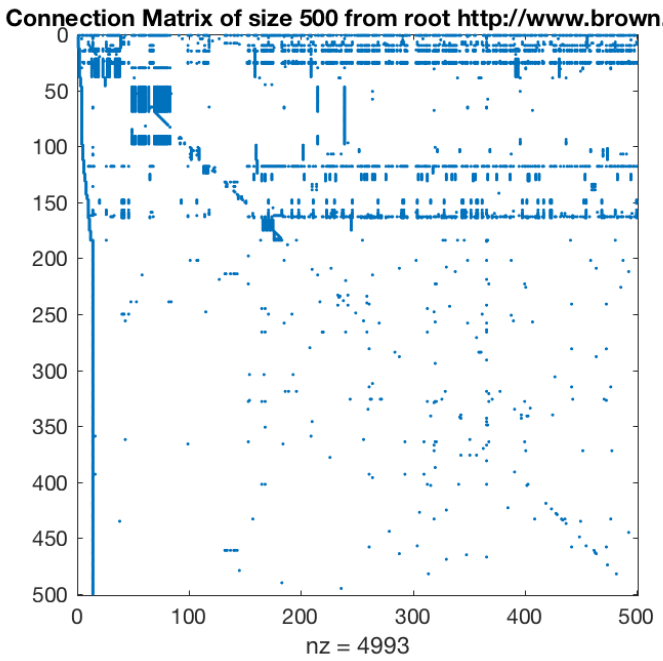


Figure 1: Exemplar Connection Matrix

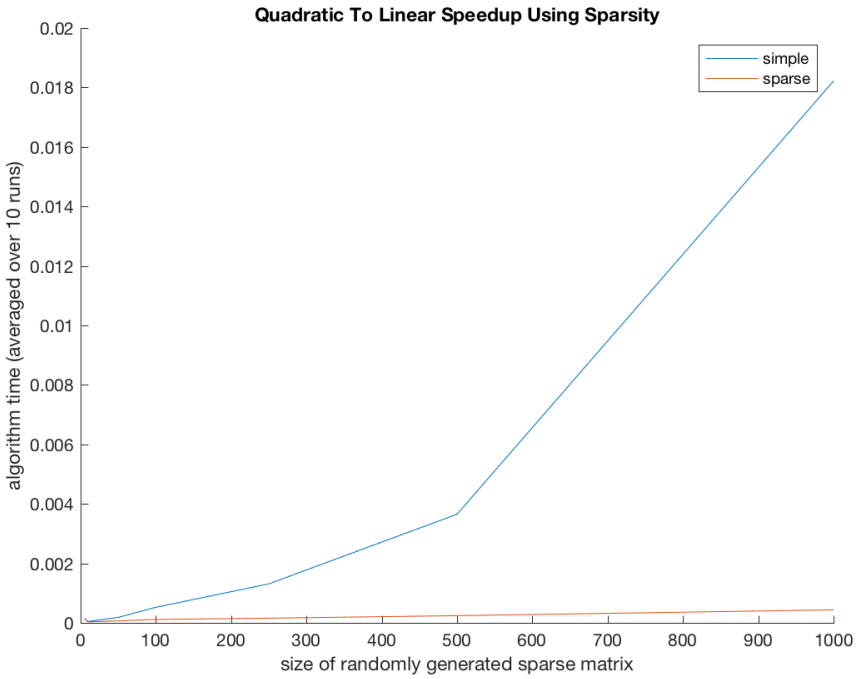


Figure 2: Difference in Algorithm Speed, $\epsilon = 10^{-5},\ r = 0.85$

```
        if  res(i) < eps; break; end
    end
    v = v/sum(v);
```

We find that for $\delta = \epsilon/2n$, roughly half the entries will have already converged by the iteration before we break out of the loop. Unsurprisingly, the results from this algorithm are slightly different than the results from the two previous algorithms. However, the algorithms always match on the entries with the highest value (which are the most important).

4.4 Accelerating the Power Method: Aitken Extrapolation

The Aitken Extrapolation method may allow us to find the dominant eigenvector faster. The idea is to that three consecutive vectors from the algorithm $v^{(k-2)}, v^{(k-1)}$ and $v^{(k)}$ can be used to obtain an estimate of the dominant eigenvector, which will speed up convergence. Assume that u_1, u_2 are the largest eigenvalues and that $v^{(k-2)}$ can be approximately expressed as a linear combination of u_1 and u_2. Then

$$v^{(k-2)} = u_1 + \alpha_2 u_2 \tag{13}$$
$$v^{(k-1)} = u_1 + \alpha_2 \lambda_2 u_2 \tag{14}$$
$$v^{(k)} = u_1 + \alpha_2 \lambda_2^2 u_2 \tag{15}$$

as $\lambda_1 = 1$. Now let g and h be defined as

$$g_i = \left(x_i^{(k-1)} - x_i^{(k-2)}\right)^2 \tag{16}$$
$$h_i = x_i^{(k)} - 2x_i^{(k-1)} + x_i^{(k-2)} \tag{17}$$

From some algebra we obtain that $g_i = \alpha_2^2(\lambda_2 - 1)^2(u_2)_i^2$ and $h_i = \alpha_2(\lambda_2 - 1)^2(u_2)_i$ and so $f = g/h = \alpha_2 u_2$. Consequently $u_1 = v^{(k-2)} - f$. Of course this does not give us the actual u_1 as then we would be done. This method relied on the assumption that $v^{(k-2)}$ can be written as a linear combination of the first two eigenvectors, which is an approximation. However, we hope this approximation is close.

The code for this algorithm is as follows.

```
function [v, res] = aitken_power_iteration(G, r, eps, max_iter)
    n = length(G);

    c = sum(G,1);
    k = find(c~=0);
    D = sparse(k,k,1./c(k),n,n);
    T = sparse(r*G*D);
```

```
    z = sparse(((1-r)*(c~=0) + (c==0))/n);
    e = ones(n,1);

    res = zeros(max_iter,1);

    v = zeros(n, max_iter);
    v(:,1) = (1/n)*ones(n,1);
    for i = 2:max_iter
        v(:,i) = e*(z*v(:,i-1)) + T*v(:,i-1);
        res(i) = norm(v(:,i) - v(:,i-1));
        if  res(i) < eps; break; end
        if rand < 0.2 && i >= 3
            v(:,i) = aitken(v(:,i-2), v(:,i-1), v(:,i));
        end
    end
    v = v(:,i)/sum(v(:,i));
end

function u = aitken(v1, v2, v3)
    g = (v2 - v1).^2;
    h = v3 - 2*v2 + v1;
    u = v1 - g./h;
end
```

And as you can see from figure 3, where the residual norms are plotted on a log scale, the Aitken algorithm converges slightly faster. As a drawback this algorithm is slower and does not appear to work all of the time. In 3, the value of r was set to 0.99 to slow down convergence. Figure 4 shows the results for $r = 0.85$.

5 Linear Solver Approach

Recall that in formula 12 we state that the value vector v is the dominant eigenvector of the matrix $P = ez + rGD$. And so $(ez + rGD)v = v$. Moving all terms to the same side we arrive at

$$(I - ez - rGD)v = 0 \tag{18}$$

which in turn implies that

$$(I - rGD)v = e\gamma \tag{19}$$

where $\gamma = zv$. Recall that z is a row vector and v is a column vector and so zv is a constant. As a result we may just take $\gamma = 1$, then solve the system for v and rescale v so that $\sum_{i=1}^{n} v_i = 1$.

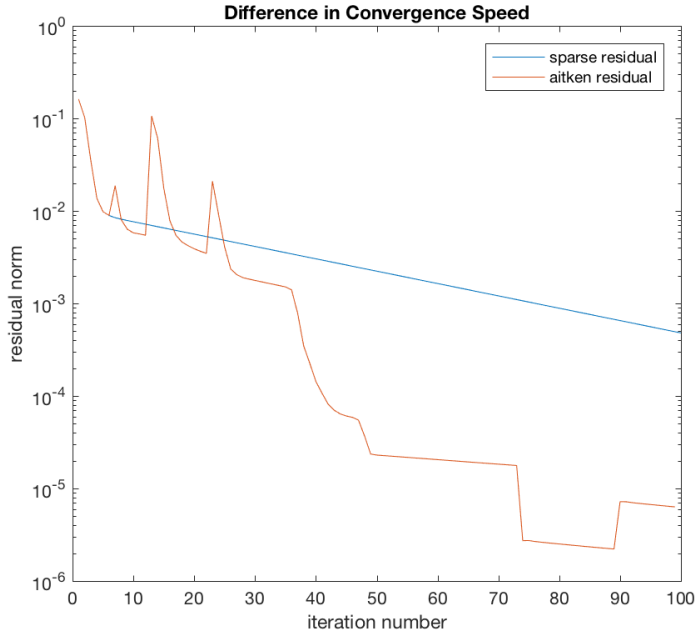


Figure 3: Difference in Convergence Speed, $r = 0.99$

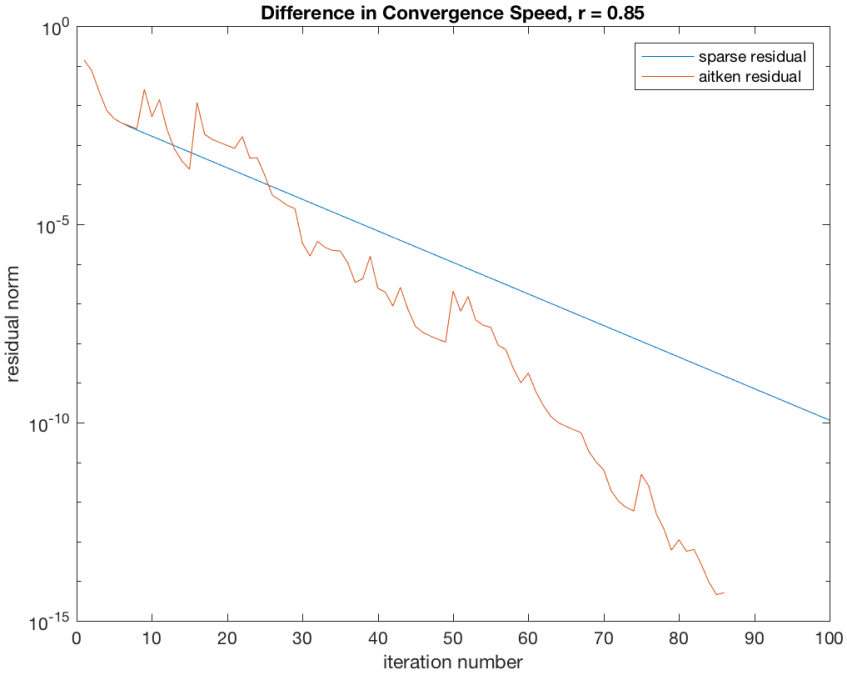


Figure 4: Difference in Convergence Speed, $r = 0.85$

5.1 Direct

We now have a linear system

$$(I - rGD)v = e \tag{20}$$

which we need to solve. First we will do so with MATLABs built in linear solver.

```
load('brown500.mat')
% Get a reference solution from the Power Method
r = 0.85; eps = 1e-6; max_iter = 100;
[v_power, ~] = sparse_power_iteration(G, r, eps, max_iter);

% Use MATLABs linear system solver to solve directly.
c = sum(G,1);
k = find(c~=0);
n = length(G);
D = sparse(k,k,1./c(k),n,n);
e = ones(n,1);
I = speye(n,n);
v_direct = (I - r*G*D)\e;
v_direct = v_direct/sum(v_direct);

% Show the error
disp(norm(v_power - v_direct));
```

Running this code we get and the result `4.5158e-06` which is reassuring as we had set $\epsilon = 10^{-6}$.

However, we can also write the direct solver ourselves. Though there are many ways we could accomplish this, we proceed with Gaussian Elimination. This will allow us to find LU such that $P(I - rGD) = LU$ where L is lower triangular, U is upper triangular, and P is our pivot matrix. Next we may use simple forward substitution to solve for w in $Lw = Pe$ then backwards substitution to solve for v in $Uv = w$. Finally we rescale v so that $\sum_{i=1}^{n} v_i = 1$. The code for doing so is as follows:

```
>> [L,U,P] = gepp(I-r*G*D);
>> Pe = P*e;
>> w = forwardsub(L,Pe);
>> v_gepp = backsub(U,w);
>> v_gepp = v_gepp/sum(v_gepp);
>> disp(norm(v_gepp - v_direct))
   9.2805e-17
```

Which returns the same solution as MATLABs solver as expected. All the code for `gepp`, `backsub` and `forwardsub` can be found in the appendix.

5.2 Iterative (Preconditioned) GMRES method: A Krylov Subspace Method for Non-Symmetric Linear Systems

Iterative methods are also extremely effective for solving linear systems. Krylov Subspace algorithms create a Krylov Subspace $\mathcal{K}_n = \text{span}\{b, Ab, ..., A^{n-1}b\}$ then find an approximate solution x_n in $\mathcal{K}_n$. The Conjugate Gradient method is the most popular algorithm though it only works for symmetric, positive definite systems.

A popular technique is to instead solve $M^{-1}Ax = M^{-1}b$. This is called *Left Preconditioning*. Alternatively, we can *right precondition* by solving $AM^{-1}u = b$ then letting $x = M^{-1}u$.

One common issue is that the vectors $\{b, Ab, ..., A^{n-1}b\}$ can be close to linearly dependent, and so it is helpful to find an orthogonal basis for them. We can use Arnoldi Iteration do so, which works almost exactly as the Modified Gram-Schmidt algorithm. This brings us to the pseudocode of the GMRES algorithm (without preconditioning).

for $n = 1, 2, ...$

1. Get $Q_n = [q_1 \ q_2 \ ... \ q_n]$ where $\{q_1, ..., q_n\}$ is an orthogonal basis for $\mathcal{K}_n$ using the Arnoldi iteration. (With the Arnoldi iteration we also obtain an $n+1 \times n$ upper Hessenberg matrix H_n where $AQ_n = Q_{n+1}H_n$.)
2. Compute the best y_n. In other words, the y_n which minimizes $\|H_n y_n - \beta e_1\|$. Because Q_n is orthogonal, this is identical to $\|Ax_n - b\|$ where $\beta = \|b - Ax_0\|$. (This is a least squares problem. We can use Givens Rotation to find a matrix which we can multiply with H_n to get an upper triangular system, which can then be easily solved.)
3. Let $x_n = Q_n y_n$ and break if residual below threshold.

Code on the internet was a very helpful basis for the implementation of the algorithm, though I could not find any code that used preconditioning and so I made the necessary adjustments independently. Since the MATLAB code for Right Preconditioned GMRES is long I have placed it in the appendix.

We run this code with no preconditioner and the Jacobi preconditioner and plot the residual norms. Of course we also compare the the page-rank value we obtained from solving the direct system to ensure the result is correct. The Jacobi preconditioner speeds up convergence as expected, as showcased by figure 5, generated by the following code.

```
load('brown500.mat')
```

```
% Use MATLABs linear system solver to solve directly.
c = sum(G,1);
k = find(c~=0);
n = length(G);
D = sparse(k,k,1./c(k),n,n);
e = ones(n,1);
I = speye(n,n);
r = 0.9; A = I - r*G*D;
v_direct = (I - r*G*D)\e;
v_direct = v_direct/sum(v_direct);

% GMRES, no preconditioner
[x1,res1] = gmres_impl(A,e, eye(length(A)));
% GMRES, Jacobi preconditioner
[x2,res2] = gmres_impl(A,e, diag(diag(A)));

% Showcase norms, should be machine 0
norm(x1-v_direct)
norm(x2-v_direct)

% Plot
semilogy(res1)
hold on
semilogy(res2)
legend('No Preconditioner', 'Jacobi Preconditioner');
xlabel('Iteration #'); ylabel('Residual');
title('GMRES right preconditoned - cut off 1e-14')

ans =

   3.8380e-16


ans =

   9.1449e-17
```

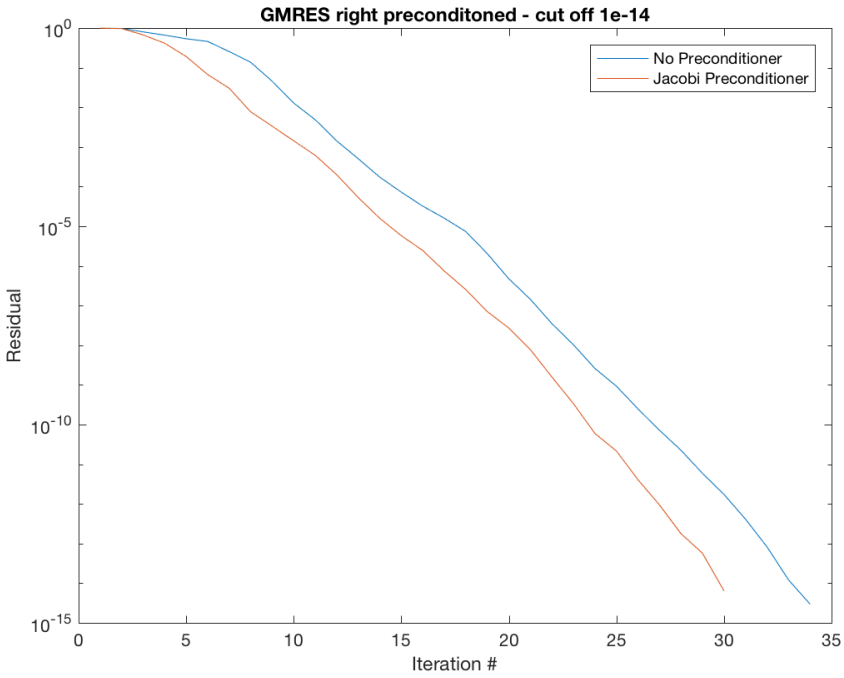


Figure 5: GMRES

6 Appendix

6.1 Omitted Theorems

Theorem 6.1: For any homogenous Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with stochastic matrix P it follows that $\mathbb{P}(X_{k+t} = i|X_k = j) = P^k_{ij}$ for all $t \in \mathbb{N}$, $i, j \in \mathcal{X}$.

proof. Since the transition kernel is time invariant, showing that $\mathbb{P}(X_t = i|X_0 = j) = P^k_{ij}$ for all $t \in \mathbb{N}$ is sufficient in proving our claim. We now proceed inductively. For $t = 1$ we have that $\mathbb{P}(X_1 = i|X_0 = j) = P^1_{ij}$ by definition. Now assume that for some fixed $t = k$, $\mathbb{P}(X_k = i|X_0 = j) = P^k_{ij}$ for all states $i, j \in \mathcal{X}$. We now observe that

$$\begin{aligned}\mathbb{P}(X_{k+1} = i|X_0 = j) &= \sum_{\ell=1}^{n} \underbrace{\mathbb{P}(X_{k+1} = i|X_k = \ell)}_{=P_{i\ell}} * \underbrace{\mathbb{P}(X_k = \ell|X_0 = j)}_{=P^k_{\ell j}} \\ &= \sum_{\ell=1}^{n} P^k_{i\ell} P_{\ell j} \\ &= P^{k+1}_{ij}\end{aligned}$$

and we have therefore proved our claim by induction. □

Theorem 6.2: If $\{X_t\}$ is an *irreducible* Markov chain with transition matrix T and stationary distribution π then $\lim_{t\to\infty} \frac{1}{t}\sum_{s=1}^{t} f(X_s)$ converges almost surely to $\mathbb{E}[f(X)]$ where $X \sim \pi$.

This theorem is called **The Ergodic Theorem**. The proof is an application of stopping times and the strong law of large numbers. Since the proof relies more on probability theory we do not present it fully. Essentially, a sequence of stopping times $\tau_{x,n}$ are constructed where $\tau_{x,n}$ is n^{th} time that the system visits state $x \in \mathcal{X}$. Then, for each x, an iid sequence of random variables $Y_1, Y_2, ...$ where $Y_n = \sum_{s=\tau_{x,n-1}}^{\tau_{x,n}-1} f(X_s)$. Since the Y_n are iid we can invoke the SLLN.

6.2 Omitted Code

GEPP

```
function [L, U, P] = gepp(A)

U = A; L = eye(size(A)); P = eye(size(A));

m = size(A,1);

for k = 1:m-1
    vec = U(k:m,k);

    [val, i] = max(abs(vec));
    alpha_k = i+k-1;

    if (alpha_k ~= k)
        tmp = U(k,k:m);
        U(k,k:m) = U(alpha_k,k:m);
        U(alpha_k,k:m) = tmp;

        tmp = vec(i);
        vec(i) = vec(1);
        vec(1) = tmp;

        tmp = L(k,1:k-1);
        L(k,1:k-1) = L(alpha_k,1:k-1);
        L(alpha_k,1:k-1) = tmp;

        tmp = P(k,:);
        P(k,:) = P(alpha_k,:);
        P(alpha_k,:) = tmp;

    end
```

```
    mul_k = vec(2:end)/vec(1,1);

    L(k+1:m,k) = mul_k;
    U(k+1:m,k+1:m) = U(k+1:m,k+1:m) - mul_k*U(k,k+1:m);
end

U = triu(U);

return
```

Forwardsub and Backwardsub

```
function [ x ] = forwardsub( L, b )

    n = length(b);
    x = zeros(n,1);

    for k = 1:n
        x(k) = (1/L(k,k))*(b(k) - L(k,1:k-1)*x(1:k-1));
    end

end


function [ x ] = backsub( U, b )

    n = length(b);
    x = zeros(n,1);

    for k = n:-1:1
        x(k) = (1/U(k,k))*(b(k) - U(k,k+1:end)*x(k+1:end));
    end

end
```

Right Preconditioned GMRES

```
function [x, res] = gmres_impl(A, b, M)
  n = length(A);
  % initial guess is 0's of length n
  x = zeros(n,1);
  % set max_iters to 100
  max_iters = 40; res = zeros(max_iters,1);
  % set error threshold
  eps = 1e-14;
  % norm of B will be used
  norm_b = norm(b);
  % e1 will be used
```

```
  e1 = zeros(n,1); e1(1) = 1;
  % initial residual
  r = b - A*(M\x);
  norm_r = norm(r);
  % will be used for GR
  s = zeros(max_iters,1);
  c = zeros(max_iters,1);
  % H is a hessenberg matrix, Q(:,1:k) is basis for
  % k dimensional krylov subspace
  H = zeros(n,n); Q = zeros(n,n);
  Q(:,1) = r/norm_r;
  beta = norm_r*e1;
  for k = 1:max_iters
    % get Hessenberg matrix H and basis Q for Krylov subspace
    [H(1:k+1,k), Q(:,k+1)] = arnoldi_func(A, Q, k, M);
    % apply GR
    [H(1:k+1,k), c(k), s(k)] = apply_GR(H(1:k+1,k), c, s, k);
    % update res
    beta(k+1) = -s(k)*beta(k);
    beta(k) = c(k)*beta(k);
    res(k)=abs(beta(k+1))/norm_b;
    if res(k) < eps; break; end
  end
  % get result
  y = H(1:k,1:k)\beta(1:k);
  x = x+M\(Q(:,1:k)*y);
  % scale because page rank vec normalizes to 1
  x = x/sum(x);

  res = res(res>0);
end
% arnoldi
function [h, q] = arnoldi_func(A, Q, k, M)
  h = zeros(k,1); q = A*(M\Q(:,k));
  for i = 1:k
    h(i)= q'*Q(:,i); q = q - h(i)*Q(:,i);
  end
  h(k+1) = norm(q); q = q/h(k+1);
end
% apply givens rotation
function [h, ck, sk] = apply_GR(h, c, s, k)
  for i = 1:k-1
    tmp = c(i)*h(i) + s(i)*h(i+1);
    h(i+1) = -s(i)*h(i) + c(i)*h(i+1);
    h(i)= tmp;
  end
  [ck, sk] = GR(h(k), h(k+1));
  h(k) = ck*h(k) + sk*h(k+1); h(k+1) = 0;
end
% get givens rotation c and s
function [c, s] = GR(u, v)
```

```
  if u == 0
    c = 0; s = 1;
  else
    t = sqrt(u^2 + v^2);
    c = abs(u)/t; s = c*v/u;
  end
end
```