THE UNIVERSITY OF MANCHESTER

SCHOOL OF MATHEMATICS

MATH 30011 PROJECT (SEMESTER ONE)

# Convergence properties and sensitivity of the PageRank problem

*Author:*
Yifan YU (9959198)

*Supervisor:*
Prof. Françoise TISSEUR

January 11, 2019

## Contents

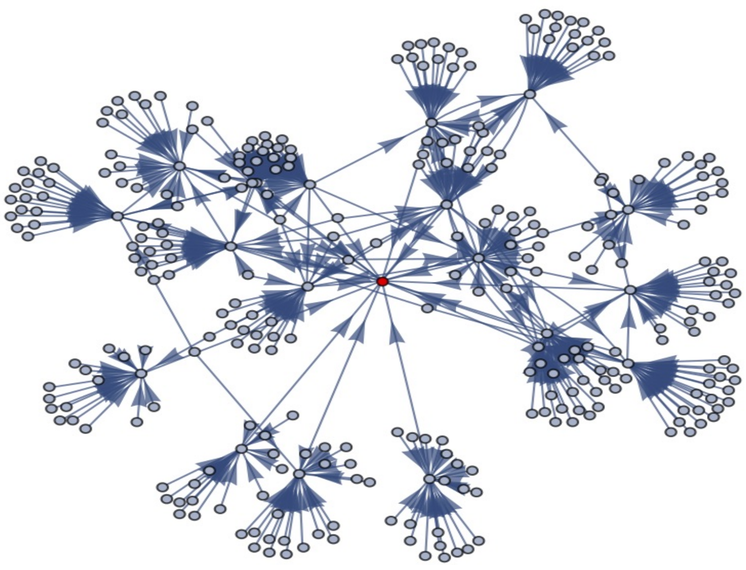


Figure 1: Web structure of Wikipedia Pages starting from ”Pagerank”

Figure 2: Wordcloud–popularity of phases related to ”PageRank ”in web search.

There are 310 vertices (Wikipedia pages) and 409 edges (hyperlinks) with directions indicated if we explore two levels with maximum of 20 webpages per level, starting from the node “PageRank” (see figure 1). And the wordcloud (figure 2) indicates the phases of top popularity related to the keyword “PageRank”.

# 1 Introduction

Information retrieval within the World Wide Web of over 8.1 billion websites has became a new challange, since the web is huge, dynamic and hyperlinked. The precision is the primary concern, since the goal is to maximize the ratio of the number of relevant documents among the total number of documents retrieved. While the performance measure and comparison of web search engines also matter, which involve speed, storage and user satisifaction [18, p.11].

The PageRank Algorithm is designed to measure the importance of every webpage, based on webgraph formed by all pages as nodes and hyperlinks as edges.

## 1.1 Basic components of a web search engine

**• Crawling**

Crawler is a program widely used by search engine to retrieve Web pages. The Web crawler begins with a root set of URLs, and follows all the hyperlinks found on these webpages. Crawling process would never stop, since previously crawlered webpages probably have been updated and the monthly Web changes were estimated to be over 600GB in 1998 [6]. But the updating frequency for each webpage could be completely different, consequently, it is the search engine's role to decide the range of webpages to revisit and the frequencies as well [18, p.15]. Extensive researches focused on efficient crawling, for example, the crawlers based on BackLink and Similarity analysis have obtained approximate 80% hot pages after having visited only 40% of the pages [6].

**• Indexing**

The information returned by web crawlers is parsed, then compressed and stored in the indexing module as inverted files. Descriptors other than page identifier are also involved in relevant document retrieval, such as the location and format of the search terms [18, p.19].

**• Query**

The user queries are linked to corrsponding inverted file in real-time, with results returned under half a second, for example Google processes more than 40,000 search queries on average every second [**?**]. The content relevance sore is query-dependent since it is computed from its inverted file in the content index, while the popularity score is query-independent, and measures the popularity of webpages within the indexing system. Ultimately, the content score and popularity score are combined to rank relevant pages for each query[18, p.23].

## 1.2 Link analysis models: HITS and PageRank

Two link analysis models, HITS and PageRank were developed independently in 1998. *HITS*, the abbreviation for *Hypertext Induced Topic Search*, was proposed by Jon Kleinberg and firstly presented at the Ninth Annual ACM-SIAM Symposium. Sergey Brin and Larry Page collaborated on PageRank as doctoral candidates in computer science at Standford University [18, p.25]. The web search engine Google was originally built to examine the utility of PageRank algorithm [22].

**• HITS**

A page containing many outlinks is a **hub**, and a page with many inlinks is an **authority**. One popularity score for each webpage is calculated by the PageRank method, while both **authority weight** $a(p)$ and **hub weight** $h(p)$ are produced for each webpage $p$ by HITS:

$$a(p) := \sum_{q \to p} h(q),$$

$$h(p) := \sum_{p \to q} a(q).$$

where $p \to q$ denotes that there is a hyperlink from page $p$ to $q$.

Each webpage is assigned with an initial authority score $a^{(0)}(p)$ and hub score $h^{(0)}(p)$. The process is continually repeated with the weights normalized for each iteration, and it finally converges to stable authority and hub weights [8].

**• PageRank**

Imagine a random surfer browsing the web, he either follows an arbitrary hyperlink on the page with equivalent probability or gets tired and moves to a random webpage to restart this process. The teleportation factor $\alpha$ is generally chosen as 0.85 comprimising between efficiency and effectiveness, which indicates that the probability of following the hyperlinks is 85%, and the probability of restarting at a random webpage is 15%. Actually, the constant $\alpha$ controls the convergence of PageRank method and influences the sensitivity of PageRank vector. [18, p. 48]

Note that PageRank is believed to be more efficient than HITS, since PageRank scores of webpages are precomputed. And PageRank is less sensitive to localized link spam since it is based on the entire web structure [14]. Furthermore, the PageRank algorithm appears to be relatively more stable to small perturbations of the link structure in the experiment on the *Cora* database [21].

Generally, the importance of each webpage is measured by the corresponding PageRank value, which is independent of search query. However, in the topic-sensitive PageRank Scheme, multiple important scores are estimated for each webpage, and combined to generate a composite PageRank score based on query topics. The results indicate that the rankings are more accurate comparing to the original PageRank method [14].

## 1.3 Other applications of PageRank

**• Citation among articles**

In the context of citation networks, individual article is regarded as node and each directional citation is regarded as an edge. It is dynamic since articles obtain citations continually, and the popuarity of an article is measured by the number of previous citations. *TimedPageRank* is introduced to assign more weights to more recent edges, and it is recommended to chose $\alpha = 0.5$ in order to maximize the accuracy [9]. While *FutureRank* is believed to be more accurate and useful, which is based on future PageRank score predicted to obtain in the future, involving authorship network and publication time [26].

**•GeneRank**

GeneRank is a modification of PageRank for the interpretation of microarry experiments, which combines prior knowledge about their biological functions and improves the evaluation of experimental results by denoising. Based on the graph of known relationships between genes, *GeneRank* detects genes highly related to those repressed in experiments. And the experimentally best teleportation parameter $\alpha$ is between $0.75$ and $0.85$ [20].

**•PageRank in Sports**

Winner network is another application of PageRank in Sports, where each team is a node, and edge from node $i$ to node $j$ represents that $j$ beats $i$ in the match. The underlying logic is that a sports fan would follow a team until it loses a match between another team, then follow the winner, while it also happens that the sports fan would restart by following an arbitrary team periodically [9].

# 2 The PageRank Algorithm

## 2.1 The Structure of the Web

The web can be abstracted as a graph with nodes and edges, where nodes are the webpages and edges are the hyperlinks between webpages. The webpages having hyperlinks with more webpages are generally considered to be more important [22]. Apart from the number of inlinks to the webpage, the importance of those webpages that link to it also matter, for example, the webpage with a link off the Wikipedia page would probably be ranked higher than those with more obscure links.

## 2.2 Definition of PageRank

**Definition 2.2.1.** *For an arbitrary webpage $P_i$ on the Web, its PageRank value, $r(P_i)$, is defined as the sum of the PageRank values of all webpages that link into $P_i$.*

$$r(P_i) = \sum_{P_j \in H_i} \frac{r(P_j)}{|P_j|}, \tag{2.1}$$

*Where $H_i$ denotes the set of webpages that link to $P_i$, and $|P_j|$ is the number of outlinks from each webpage $P_j$ in the set $H_i$ [18, p. 32].*

It is not necessary that $|P_j| \neq 0$ in general and $|P_j| = 0$ if there is no outlink from page $P_j$, which is called a dangling node(see 2.4).

Therefore, the PageRank value of page $P_i$ is calculated iteratively from the equation (2.1). We denote the PageRank value of $P_i$ as $r_k(P_i)$ in the $k^{th}$ iteration and $r_{k+1}(P_i)$ in the $(k+1)st$ iteration:

$$r_{k+1}(P_i) = \sum_{P_j \in H_i} \frac{r_k(P_j)}{|P_j|}. \tag{2.2}$$

given that the initial value for each webpage is the same: $r_0(P_i) = \frac{1}{n}, \forall i$, if there are $n$ pages on the Web in total. Now consider the tiny web structure in Figure 3 for PageRank values.

Figure 3: Web structure of six pages.

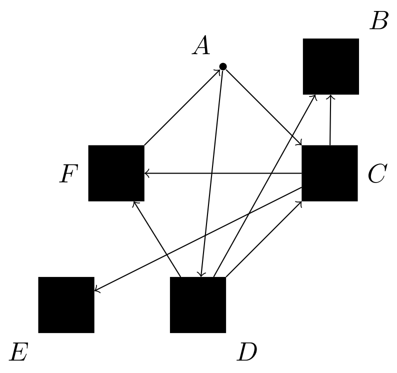


Table 1: List of First two iteration results of Figure3

| Webpages | iteration#0 | iteration#1 | iteration#2 | rank |
|---|---|---|---|---|
| A | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{1}{9}$ | 3 |
| B | $\frac{1}{6}$ | $\frac{5}{18}$ | $\frac{19}{54}$ | 1 |
| C | $\frac{1}{6}$ | $\frac{5}{36}$ | $\frac{1}{9}$ | 3 |
| D | $\frac{1}{6}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | 5 |
| E | $\frac{1}{6}$ | $\frac{2}{9}$ | $\frac{29}{108}$ | 2 |
| F | $\frac{1}{6}$ | $\frac{1}{9}$ | $\frac{4}{54}$ | 6 |

## 2.3 Matrix representation of the PageRank Algorithm

**Definition 2.3.1.** *The Hyperlink matrix represents the web structure, which is defined to be:*

$$H'_{ij} = \begin{cases} 1 & \textit{if there is a hyperlink from page } i \textit{ to } j \\ 0 & \textit{otherwise} \end{cases} \tag{2.3}$$

*Where $H'_{ij}$ is the entry of the $i^{th}$ row and $j^{th}$ column.*

The *hyperlink matrix* is a square matrix of size $n$, where $n$ is the number of nodes(pages) in the web structure. Generally, it is a highly sparse matrix, since the size of the hyperlink matrix of the World Wide Web was estimated to be $8.1$ billion in 2006, but each webpage only links to an average of 10 pages approximately [18, p.20].

The *Hyperlink matrix* corrsponding to Figure3 can be represented as:

$$H' = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}.$$

Define $m_i = \sum\limits_{j=1}^{n} H'_{ij}$ as the row sum of the *hyperlink matrix* $H'$, it indicates the number of outlinks on the page $i$.

**Definition 2.3.2.** *Define a diagonal matrix $D = diag(d_1, \dots, d_n)$, where*

$$d_i = \begin{cases} \frac{1}{m_i} & \text{if } m_i \neq 0 \\ 0 & \text{otherwise} \end{cases}.$$

The *row-normalized hyperlink matrix*, denoted as $H$, is obtained by $DH'$. Hence, take Figure 3 as an example,

$$D = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1, \end{bmatrix},$$

$$H = DH' = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now we introduce the *PageRank vector*

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \end{bmatrix},$$

where $\pi_i \geq 0$ and $\sum\limits_{i=1}^{n} \pi_i = 1 \, \forall i = 1, 2, \dots, n$ ($\pi$ is therefore normalized).

The *PageRank Vector* $\pi$ completely relies on the web structure, not involving a measure of the content relevance of query. The $i^{th}$ entry of *PageRank vector* $\pi$ is the corresponding *PageRank value* of page $i$, where more important pages are attached with larger values.

Now we introduce the notation $\pi^{(k)T}$, a row vector which is the transpose of the PageRank vector at the $k^{th}$ iteration. The matrix representation of equation $(2.1.2)$ can be written as:

$$\pi^{(k+1)T} = \pi^{(k)T} H$$

Suppose the PageRank vector finally converges to $\pi^T$, then:

$$\pi^T = \pi^T H$$

Ultimately, it is converted into an **eigenvalue problem**: The PageRank vector $\pi^T$ is actually an eigenvector of the *row-normalized hyperlink matrix* $H$ with corresponding eigenvalue $1$.

## 2.4 Adjusted Stochastic Model

**Definition 2.4.1.** *A* ***nonnegative matrix*** *is a matrix in which all the elements are non-negative, denoted as* $X \geq 0$

$$x_{ij} \geq 0, \forall i, j. \tag{2.4}$$

*A* ***positive matrix*** *is a matrix in which all the elements are greater than zero, denoted as* $X > 0$.

**Definition 2.4.2.** *An* $n$*-by-*$n$ *matrix is row stochastic if it is a non-negative matrix and for each row, the sum of the entries is* $1$ *[17].*

It is notable that a *row-normalized hyperlink matrix* $H$ is not always *row stochastic*. The webpage $i$ with no outlinks is called a dangling node, in which case, the corresponding row $i$ in $H^{'}$ is a zero row, and $d_i = 0$ in the diagonal matrix $D$. However, if the zero rows in $H$ are replaced by $\frac{1}{n}\mathbf{e^T}$, where $\mathbf{e^T} = (1, 1 \ldots 1)$, then it is adjusted to be stochastic.

**Definition 2.4.3.** *The dangling node vector* $\mathbf{a}$ *is a binary vector defined as:*

$$a_i = \begin{cases} 1 & \textit{if the page } i \textit{ is a dangling node} \\ 0 & \textit{otherwise} \end{cases}.$$

**Definition 2.4.4.** *The stochastic matrix* $S$ *describes that a random surfer would transfer to an arbitrary page randomly, at a dangling node.*

$$S = H + a(\frac{1}{n}e^T). \tag{2.5}$$

$S$ is actually the transition probability matrix for a Markov chain. However, $S$ could still be a reducible matrix with a zero column $j$ (if there is no inlink to page $j$), which does not guarantee it would converge to a unique PageRank vector $\pi^T$ [18]. Consequently, further adjustment is needed.

## 2.5 Google Matrix

Although the random surfer would normally follow the hyperlinks on each webpage, it will periodically get bored at some stage and repeat the process by starting with a random page. The teleportation parameter $\alpha$ represents the proportion of time that the random surfer follows the hyperlinks instead of teleporting to a new page, therefore, $\alpha \in (0, 1)$ [22].

**Definition 2.5.1.** *The Google matrix is defined as:*

$$G = \alpha S + (1 - \alpha)E \tag{2.6}$$

*where the teleportation matrix $E = \frac{1}{n}ee^T$ is uniform, with $\frac{1}{n}$ at every entry, since the random surfer is equally likely to start with an arbitrary page.*

**Theorem 2.5.1.** *The matrix $G$ is stochastic as a convex combination of $S$ and $E$ that are all stochastic[18, p.37].*

*Proof.* Consider each entry of Google matrix $G$

$$G_{ij} = \alpha S_{ij} + (1-\alpha)E_{ij}$$

$\Rightarrow G_{ij} \geq 0$ since $S_{ij} \geq 0$, $E_{ij} = \frac{1}{n} > 0$ and $\alpha \in (0,1)$

The $i^{th}$ row sum of $G$ is

$$\sum_{j=1}^{n} G_{ij} = \alpha \sum_{j=1}^{n} S_{ij} + (1-\alpha) \sum_{j=1}^{n} E_{ij} = \alpha \cdot 1 + (1-\alpha) \cdot n \cdot \frac{1}{n} = 1. \quad \forall i$$

Hence, the matrix $G$ is stochastic by Definition 2.4.2. □

The Google matrix would focus more on the web structure with larger $\alpha$. Then the PageRank vector is calculated iteratively by $\pi^{(k+1)T} = \pi^{(k)T}G$, hopefully it would finally converge to $\pi^T$ with $\pi^T = \pi^T G$.

**Definition 2.5.2.** *A $n \times n$ square matrix $A$ is said to be primitive, if there exists a $k \in N$ such that $A^k$ is positive. Then every positive matrix is also primitive ($k = 1$) [24].*

If we consider every entry of $G$: $G_{ij} = \alpha S_{ij} + (1-\alpha)E_{ij} = \alpha S_{ij} + (1-\alpha)\frac{1}{n} > 0$ since $\alpha \in (0,1)$ and $0 \leq S_{ij} \leq 1$. Therefore, the positive matrix $G$ is *primitive*.

**Theorem 2.5.2.** *If $A > 0$, $x$ is a positive eigenvector of $A$ corresponding to $\rho(A)$, and $y$ is any positive eigenvector of $A^T$ corresponding to $\rho(A) = \rho(A^T)$, then:*

$$\lim_{k\to\infty} (\frac{A}{\rho(A)})^k = \frac{xy^T}{y^T x} > 0 \tag{2.7}$$

*[28]*

**Theorem 2.5.3.** *Assume that a Markov chain has a positive stochastic matrix $P$. Then $p^{(k)}$ is independent of the initial probability distribution vector $p^{(0)}$,*

$$\lim_{k\to\infty} p^{(k)} = p \tag{2.8}$$

*Where $p^{(k)} = (P^T)^k p^{(0)}$, and $p$ is the stationary probability distribution vector. [28]*

*Proof.* The iteration process is $\pi^{(k)T} = \pi^{(k-1)T}G = \pi^{(k-2)T}G^2 = \cdots = \pi^{(0)T}G^k$, its transpose form is $\pi^{(k)} = (G^T)^k \pi^{(0)}$, and $G$ has been verified to be a positive stochastic matrix.

$G$ is stochastic, $\rho(G) = 1$. The stationary probability distribution vector $\pi$ satisfies $\pi = G^T\pi$ and $e^T\pi = 1$ since it is normalized. By Theorem 2.5.2, take $x = \pi$ and $y = e$ :

$$\lim_{k\to\infty} \pi^{(k)} = \lim_{k\to\infty} (G^T)^k\pi^{(0)} = \lim_{k\to\infty} (\frac{G^T}{\rho(G^T)})^k\pi^{(0)} = \frac{\pi e^T}{e^T\pi}\pi^{(0)} = \frac{\pi(e^T\pi^{(0)})}{1} = \pi \tag{2.9}$$

The stationary probability $\pi^T$ is the *steady-state* probability vector, if sufficient transitions occur, the influence of the starting state would be erased [27]. □

In summary, *the Pagerank Problem* is actually an **eigenvector problem**: finding the normalized eigenvector of $G$ corresponding to the dominant eigenvalue $\lambda = 1$ [18].

$$\begin{cases} \pi^T = \pi^T G \Rightarrow \pi^T(I - G) = 0^T \\ \pi^T e = 1 \end{cases} \tag{2.10}$$

## 2.6 Properties of Google Matrix $G$

**Theorem 2.6.1** (Perron-Frobenius Therorem). *If $A \geq 0$ is irreducible, then:*
*(i) $\rho(A) > 0$.*
*(ii) $\rho(A)$ is an eigenvalue of $A$.*
*(iii) There is an eigenvector $x$ with $x > 0$ and $Ax = \rho(A)x$.*
*(iv) $\rho(A)$ is an eigenvalue of algebraic multiplicity $1$ [28].*

**Lemma 1.** *The second largest eigenvalue of a stochastic matix S satisfis $|\lambda_2| < 1$.*

*Proof.* For any consistent matrix norm $\|\cdot\|$, we have $\rho(S) \leq \|S\|$, since:
$Sx_i = \lambda x_i$ for each eigenvector $x_i \neq 0$. If we write $X = [x_i, x_i, \dots, x_i]$, then $SX = \lambda X$. By the properties of matrix norm: $|\lambda|\|X\| = \|\lambda X\| = \|SX\| \leq \|S\|\|X\|$ By the Theorem 2.6.1, $\rho(S)$ is a eigenvalue of **S**, with algebraic multiplicity $1$. Therefore:

$$\rho(S) \leq \|S\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |S_{ij}| = 1 \tag{2.11}$$

since every row sum of $S$ is 1.
Hence, the eigenvalues of the stochastic matrix $S$ satisfy $1 = |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$ [15]. □

**Theorem 2.6.2.** *Let $S \in \mathbb{R}^{n\times n}$ be row stochastic with eigenvalues $\{1, \lambda_2, \lambda_3, \dots, \lambda_n\}$, then the eigenvalues of $G(\alpha) = \alpha S + (1-\alpha)ev^T$, where $0 < \alpha < 1$ and $v^T$ is a nonnegative row vector with $v^T e = 1$, are $\{1, \alpha\lambda_2, \alpha\lambda_3, \dots, \alpha\lambda_n\}$.*

*Proof.* $Se = e$ since $S$ is row stochastic, indicating that $(1, e)$ is an eigenpair. Let $U = [e \quad X]$ denote an arbitrary invertible matrix with $e$ as the first column. Write $U^{-1}$ as $\begin{bmatrix} u^T \\ V^T \end{bmatrix}$. Then

$$U^{-1}U = \begin{bmatrix} u^T e & u^T X \\ V^T e & V^T X \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

Therefore, $u^T e = 1$ and $V^T e = 0$. Now consider the similarity transformation:

$$U^{-1}SU = \begin{bmatrix} u^T Se & u^T SX \\ V^T Se & V^T SX \end{bmatrix} = \begin{bmatrix} u^T e & u^T SX \\ V^T e & V^T SX \end{bmatrix} = \begin{bmatrix} 1 & u^T SX \\ 0 & V^T SX \end{bmatrix}.$$

It can be deduced that $V^T SX$ has the remaining eigenvalues $\lambda_2, \lambda_3, \dots, \lambda_n$, since $U^{-1}SU$ is similar to $S$ and has same eigenvalues.
Apply the similarity transformation to $G = \alpha S + (1-\alpha)ev^T$

$$\begin{aligned} U^{-1}GU &= U^{-1}(\alpha S + (1-\alpha)ev^T)U \\ &= \alpha U^{-1}SU + (1-\alpha)U^{-1}ev^T U \\ &= \begin{bmatrix} \alpha & \alpha u^T SX \\ 0 & \alpha V^T SX \end{bmatrix} + (1-\alpha)\begin{bmatrix} u^T e \\ V^T e \end{bmatrix} \begin{bmatrix} v^T e & v^T X \end{bmatrix} \\ &= \begin{bmatrix} \alpha & \alpha u^T SX \\ 0 & \alpha V^T SX \end{bmatrix} + (1-\alpha)\begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} 1 & v^T X \end{bmatrix} \qquad (2.12) \\ &= \begin{bmatrix} \alpha & \alpha u^T SX \\ 0 & \alpha V^T SX \end{bmatrix} + (1-\alpha)\begin{bmatrix} 1 & v^T X \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} 1 & \alpha u^T SX + (1-\alpha)v^T X \\ \mathbf{0} & \alpha V^T SX \end{bmatrix}. \end{aligned}$$

Hence, the eigenvalues of the block upper triangular matrix $U^{-1}GU$ are $1, \alpha\lambda_2, \alpha\lambda_3, \dots, \alpha\lambda_n$, which are exactly the eigenvalues of $G$ because of similarity [18]. $\square$

## 2.7 Topic-sensitive PageRank

In the PageRank algorithm mentioned above, a single PageRank vector is computed to measure the relative importance of webpages, based on the web structure. However, it is believed that using a set of precomputed and topic-biased PageRank vectors to generate query-specific PageRank scores is more accurate than a single PageRank vector [14].

The basic idea is to generate 16 topic-sensitive PageRank vectors which are independent of query, then compute the similarity of user's query with each topic, and the composite PageRank score is a linear combination of these topic-sensitive vectors. Note that the corresponding 16 topic-sensitive PageRank vectors for each webpage are computed offline before query, which would not siginificantly increase the query-time, comparing to the original PageRank algorithm [14].

# 3 Computation of stationary vector $\pi^T$ of a Markov chain with transition matrix $G$

$$\begin{aligned} G &= \alpha S + (1-\alpha)E = \alpha S + (1-\alpha)\frac{1}{n}ee^T \\ &= \alpha(H + \frac{1}{n}ae^T) + (1-\alpha)\frac{1}{n}ee^T = \alpha H + (\alpha a + (1-\alpha)e)\frac{1}{n}e^T. \end{aligned} \qquad (3.13)$$

$G$ is a dense matrix, therefore, optimizing the storage and accelerating the calculation are needed.

## 3.1 The Power Method

The power method is simple in execution: by the result (3.13)

$$\begin{aligned}\pi^{(k+1)T} &= \pi^{(k)T}G = \alpha\pi^{(k)T}H + (\alpha\pi^{(k)T}a + (1-\alpha)\pi^{(k)T}e)\frac{1}{n}e^T \\ &= \alpha\pi^{(k)T}H + (\alpha\pi^{(k)T}a + (1-\alpha))\frac{1}{n}e^T.\end{aligned} \tag{3.14}$$

The only vector-matrix multiplication involved in each iteration is $\pi^{(k)T}H$, with computational complexity $O(n)$ since sparse matrix $\mathbf{H}$ has approximate 10 nonzero elements in each row [18].

Moreover, the power method is storage-friendly, comparing to other faster iterative methods, which are infeasible in the context of Google matrix (with size of more than 8.1 billion). And the number of iteration that the power method requires is only $50-100$, which indicates the total computational effort would be approximate $50\,O(n)$ [18].

**• Rate of convergence**

Assume $\pi^{(0)T}$ could be written as a linear combination of the eigenvectors of $G$:

$$\pi^{(0)T} = \sum_{i=1}^{n}\beta_i x_i^T \quad \text{with}\, \beta_1 \neq 0$$

Suppose the eigenvalues of $S$ are $\lambda_i$, $i = 1, 2, \ldots, n$. Suppose that the eigenvalues of $G$ are $\mu_i$, $i = 1, 2, \ldots, n$, where $\lambda_1 = \mu_1 = 1$ and $\mu_i = \alpha\lambda_i$, for $i = 2, 3, \ldots, n$ by Theorem 2.6.2

$$\begin{aligned}\Rightarrow \pi^{(k)T} &= \pi^{(0)T}G^k = (\beta_1 x_1^T + \sum_{i=2}^{n}\beta_i x_i^T)G^k = \beta_1\mu_1^k x_1^T + \sum_{i=2}^{n}\beta_i\mu_i^k x_i^T \\ &= \beta_1\lambda_1^k x_1^T + \sum_{i=2}^{n}\beta_i(\alpha\lambda_i)^k x_i^T = \lambda_1^k\{\beta_1 x_1^T + \sum_{i=2}^{n}\beta_i(\frac{\alpha\lambda_i}{\lambda_1})^k x_i^T\} \\ &= \beta_1 x_1^T + \sum_{i=2}^{n}\beta_i(\frac{\alpha\lambda_i}{\lambda_1})^k x_i^T.\end{aligned} \tag{3.15}$$

where $\lambda_i$ are the eigenvalues of the adjusted stochastic matrix $S$. Then $\pi^{(k)T}$ finally converges to the dominant eigenvector $x_1^T$, and the rate of convergence depends on $|\frac{\alpha\lambda_i}{\lambda_1}|$, for $i = 1, 2, 3, \ldots, n$, among which the subdominant eigenvalue $\alpha\lambda_2$ determines the convergence rate, since $1 = |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$ [27].

It is notable that $|\frac{\alpha\lambda_i}{\lambda_1}| < |\frac{\alpha\lambda_2}{\lambda_1}| < \alpha < 1$, which indicates equation (3.15) will eventually converge. For the commonly chosen value $\alpha = 0.85$, $\alpha^{50} = 0.000295765$, $\alpha^{100} = 8.74767 \times 10^{-8}$, therefore, the accuary is adequate for the PageRank vector, especially when it is combined with content relevance score in search queries.

However, if $\alpha$ is close to 1, and it is probably that $|\alpha\lambda_2| \approx 1$ beacuse of the link structure, it may take a large number of iterations for convergence such that it is unworkable in practice.

## 3.2 Convergence for iterative methods

The iterative methods like Jacobi, Gauss-Seidel and SOR method converge if and only if the spectral radius $\rho < 1$, regardless of the starting vector.

**Lemma 2.** *Suppose $A \in \mathbb{C}^{n\times n}$, then*

$$\lim_{k\to\infty} A^k = 0 \Leftrightarrow \rho(A) < 1.[28] \tag{3.16}$$

**Lemma 3.** *if the spectral redius $\rho(T) < 1$, then $(I-T)^{-1}$ exists and*

$$(I-T)^{-1} = I + T + T^2 + \cdots = \sum_{k=0}^{\infty} T^k. \tag{3.17}$$

*Proof.* Suppose $(\lambda, x)$ is an eigenpair of $T$ (where $x \neq 0$), then $Tx = \lambda x \Rightarrow (I - T)x = (1-\lambda)x$, therefore, $(1-\lambda, x)$ is an eigenpair of $I - T$. Since $|\lambda| \leq \rho(T) < 1$, $\lambda \neq 1 \Rightarrow 1 - \lambda \neq 0$, then $(I-T)^{-1}$ exists since all the eigenvalues of $(I-T)$ are nonzero.

$$(I-T)(I+T+\cdots+T^n) = (I+T+\cdots+T^n) - (T+T^2+\cdots+T^{n+1}) = I - T^{n+1}$$

$$\Rightarrow \lim_{n\to\infty}(I-T)(I+T+\cdots+T^n) = \lim_{n\to\infty}(I-T^{n+1}) = I - \lim_{n\to\infty} T^{n+1} = I,$$

Since $\lim\limits_{n\to\infty} T^{n+1} = 0$ by Lemma 2 because $\rho(T) < 1$. Thus,

$$(I-T)^{-1} = \lim_{n\to\infty}(I+T+\cdots+T^n) = \sum_{k=0}^{\infty} T^k.[5]$$

□

**Theorem 3.2.1** (convergence)**.** *For any starting vector $x^{(0)} \in \mathbb{R}^n$, the sequence $\{x^{(k)}\}_{k=0}^{\infty}$ given by*

$$x^{(k+1)} = Tx^{(k)} + c, \quad \textit{for each } k \geq 0$$

*converges to the unique solution $x = Tx + c$ iff $\rho(T) < 1$.*

*Proof.* $(\Leftarrow)$

$$\begin{aligned} x^{(k)} &= Tx^{(k-1)} + c \\ &= T(Tx^{(k-2)} + c) + c \\ &= T^2x^{(k-2)} + (T+I)c \\ &\vdots \\ &= T^k x^{(0)} + (T^{k-1} + T^{k-2} + \cdots + T + I)c, \end{aligned}$$

By Lemma 2, when $\rho(T) < 1$, $\lim_{k\to\infty} T^k = 0 \Rightarrow \lim_{k\to\infty} T^k x^{(0)} = 0$. Therefore, by Lemma 3,

$$\begin{aligned}\lim_{k\to\infty} x^{(k)} &= \lim_{k\to\infty} T^k x^{(0)} + \lim_{k\to\infty} (T^{k-1} + T^{k-2} + \cdots + T + I)c \\ &= 0 + \lim_{k\to\infty} (\sum_{k=0}^{n} T^k)c \\ &= (I-T)^{-1}c.\end{aligned}$$

Hence, when $\rho(T) < 1$, the sequence $\{x^{(k)}\}_{k=0}^{\infty}$ converges to $x = (I-T)^{-1}c$, satisfying $x = Tx + c$.
$(\Rightarrow)$ Suppose $x$ is the unique solution to $x = Tx + c$, and $x^{(k)} = Tx^{(k-1)} + c$ for each $k \geq 1$, then

$$\begin{aligned}x - x^{(k)} &= (Tx + c) - (Tx^{(k-1)} + c) = T(x - x^{(k)}), \\ \Rightarrow x - x^{(k)} &= T(x - x^{(k-1)}) = \cdots = T^k(x - x^{(0)}), \\ \Rightarrow \lim_{k\to\infty} T^k(x - x^{(0)}) &= \lim_{k\to\infty} (x - x^{(k)}) = 0.\end{aligned} \tag{3.18}$$

Since the sequence $\{x^{(k)}\}$ is assumed to converge to $x = Tx + c$.
Therefore, $T$ is a convergent matrix with $\lim_{k\to\infty} T^k = 0$ by equation (3.18) since $x - x^{(0)}$ is arbitrary. By Lemma 2, $\rho(T) < 1$ [5]. $\square$

**Theorem 3.2.2.** *Given a system $x = Tx + c$, with $I - T$ invertible, the following statements are equivalent:*
*(1) The iterative method converges to a unique solution.*
*(2) $\rho(T) < 1$.*
*(3) $\|B\| < 1$ for some subordinate matrix norm $\|\cdot\|$.*

## 3.3 Jacobi Method

When solving a system of linear equations

$$Ax = b \tag{3.19}$$

$$\text{where} A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix},$$

decompose the matrix $A$ as:

$$A = L + D + U,$$

where $L$ and $U$ are the lower and upper triangular parts, and

$$D = diag(a_{11}, \dots, a_{nn}).$$

If a splitting of $A \in \mathbb{C}^{n\times n}$ is of the form:

$$A = M - N,$$

then $Ax = b$ can be rewritten as $Mx = Nx + b$, and we can construct a sequence $x^{(k)}$, starting from $x^{(0)}$

$$Mx^{(k+1)} = Nx^{(k)} + b.$$

**Definition 3.3.1** (Jacobi Method). *The Jacobi method for $Ax = b$ is described as:*

$$\begin{aligned} Dx^{(k+1)} &= -(L+U)x^{(k)} + b \\ x^{(k+1)} &= -D^{-1}(L+U)x^{(k)} + D^{-1}b, \end{aligned}$$

*Therefore, the splitting of A is*

$$M = D, \quad N = -(L+U),$$

*The $(k+1)^{th}$ iteration of $x_i$ is*

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}) \quad \text{for } 1 \le i \le n$$

**Theorem 3.3.1** (sufficient convergence condition of Jacobi's method). *if A is strictly diagonally dominant by rows, then the Jacobi's method converges for any starting vector $x^{(0)}$.*

*Proof.* since A is strictly diagonally dominant by rows,

$$|a_{ii}| > \sum_{j\neq i} |a_{ij}|, \quad i = 1, 2, \ldots, n$$

$$\Rightarrow \sum_{j\neq i} |\frac{a_{ij}}{a_{ii}}| < 1, \quad i = 1, 2, \ldots, n$$

Hence,

$$\rho(M^{-1}N) = \rho(-D^{-1}(L+U)) \le \| - D^{-1}(L+U)\|_\infty = \max_i \sum_{j\neq i} |\frac{a_{ij}}{a_{ii}}| < 1$$

□

Rewrite the PageRank problem into the form of linear equations system $Ax = b$:

$$\begin{aligned} &\pi^T = \pi^T G = \pi^T(\alpha S + (1-\alpha)\frac{1}{n}ee^T) = \alpha\pi^T S + (1-\alpha)\frac{1}{n}e^T, \\ &\Rightarrow \pi^T(I - \alpha S) = (1-\alpha)\frac{1}{n}e^T, \\ &\Rightarrow (I - \alpha S^T)\pi = (1-\alpha)\frac{1}{n}e. \end{aligned}$$

Consider the $i^{th}$ row sum of $A = I - \alpha S$,

$$\sum_{j=1}^{n}(I - \alpha S)_{ij} = \sum_{j=1}^{n} I_{ij} - \alpha\sum_{j=1}^{n} S_{ij} = 1 - \alpha \in (0, 1)$$

$$\Rightarrow (I - \alpha S)_{ii} > -\sum_{j\neq i}(I - \alpha S)_{ij} = \sum_{j\neq i} |(I - \alpha S)_{ij}| \tag{3.20}$$

since $-(I - \alpha S)_{ij} = \alpha S_{ij} > 0$ for $j \neq i$. Therefore, Jacobi's method converges for PageRank problem, regardless of the starting vector $x^{(0)}$.

## 3.4 The Method of Gauss-Seidel

**Definition 3.4.1** (Gauss-Seidel Method). *For a given initial vector $x^{(0)}$, the Gauss-Seidel method for $Ax = b$ is described as :*

$$(D + L)x^{(k+1)} = -Ux^{(k)} + b,$$

*Therefore, the splitting of $A$ is*

$$M = D + L, \quad N = -U,$$

*The $(k + 1)^{th}$ iteration of $x_i$ is*

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}). \quad \textit{for } 1 \leq i \leq n$$

**Theorem 3.4.1** (Stein-Rosenberg). *In the linear system $Ax = b$, if $A_{ik} \leq 0$, for each $i \neq k$, and $A_{ii} > 0$ for each $i = 1, 2, \ldots, n$, when the iterative process is written as $x = Tx + c$, only one of the following conditions holds:*
*(a) $0 \leq \rho(T_g) < \rho(T_j) < 1$;*
*(b) $1 < \rho(T_j) < \rho(T_g)$;*
*(c) $\rho(T_j) = \rho(T_g) = 0$;*
*(d) $\rho(T_j) = \rho(T_g) = 1$. [5, p.448]*

Although Gauss-Seidel method is not a parallel implementation, it requires less storage, which is computationally more convenient. In the context of solving the linear system of PageRank problem, the iterative process is $(I - \alpha S^T)\pi = (1 - \alpha)\frac{1}{n}e$(see equation (3.20)), with $(I-\alpha S^T)_{ii} = 1-\alpha S^T_{ii} > 0$ for each $i$ and $(I-\alpha S^T)_{ij} = -\alpha S_{ji} < 0$ for each $i \neq j$. Moreover, the convergence of Jacobi's method in PageRank process (see Theorem 3.3.1) guarantees that $\rho(T_j) < 1$, by Theorem 3.4.1, $0 \leq \rho(T_g) < \rho(T_j) < 1$, then it can be deduced that Gauss-Seidel method convergences faster than Jacobi's method. In practice, the convergence rate of Gauss-Seidel is approximately twice of Jacobi method in PageRank problem [2], and it is practical for solving moderate-size PageRank problem [23].

## 3.5 Comparison for Convergence

In this section, numerical experiments are represented to confirm previous analytical observations. We focus on the web structure obtained by crawling the hyperlinks between wikipedia pages (starting from "PageRank" and $\alpha = 0.85$), with the help of **WikipediaData** in Mathematica which utilizes MediaWiki's API to retrieve articles and hyperlinks. Due to the limit of computing capability, we only explore $4$ levels with maximum of $10$ wikipedia hyperlinks per level, then the web structure is composed of $2657$ vertices and $5920$ edges. The following are the error lists (Table 2)and plot of PageRank vectors in consecutive iterations(Figure 4), measured in $2-$norm.

It is notable that the power method shows overwhelming advantages in computing complexity, which requires approximate $396ms$ for the calculation of PageRank vector in the first 80 iterations, while Jacobi's method and Gauss-Seidel method both require around $10$ minutes. As for the rate of convergence indicated in the plot 4 and table

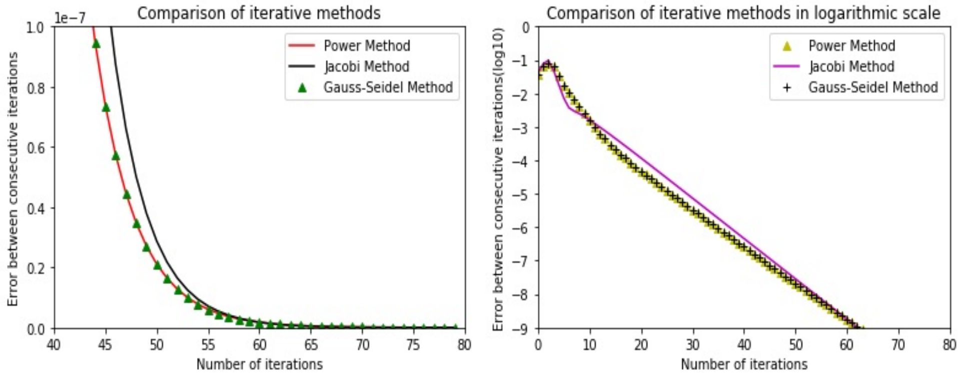


Figure 4: Convergence rate of the power method and Jacobi and Gauss-Seidel method

Table 2: List of error between consecutive iterations of iterative methods

| Iterative methods | Power method | Jacobi's method | Gauss-Seidel method |
|---|---|---|---|
| iteration # 1 | $3.79437876795 \times 10^{-2}$ | $4.2333138748 \times 10^{-2}$ | $3.79437909812 \times 10^{-2}$ |
| iteration # 11 | $1.54434258024 \times 10^{-3}$ | $1.59325199820 \times 10^{-3}$ | $1.54425926347 \times 10^{-3}$ |
| iteration # 21 | $4.7785023356 \times 10^{-5}$ | $1.17073993595 \times 10^{-4}$ | $4.7777424630 \times 10^{-5}$ |
| iteration # 31 | $3.3449461950 \times 10^{-6}$ | $7.418300776278 \times 10^{-6}$ | $3.34331438941 \times 10^{-6}$ |
| iteration # 41 | $2.5980019488 \times 10^{-7}$ | $4.62493987393 \times 10^{-7}$ | $2.59485236784 \times 10^{-7}$ |
| iteration # 51 | $2.11123271400 \times 10^{-8}$ | $2.87556226132 \times 10^{-8}$ | $2.10534621175 \times 10^{-8}$ |
| iteration # 61 | $1.77932832256 \times 10^{-9}$ | $2.87556226132 \times 10^{-9}$ | $2.10534621175 \times 10^{-9}$ |
| iteration # 71 | $1.5941166313 \times 10^{-10}$ | $1.10999172578 \times 10^{-10}$ | $1.5776747689 \times 10^{-10}$ |
| iteration # 80 | $1.99327101094 \times 10^{-11}$ | $9.1039331780 \times 10^{-12}$ | $1.96850057974 \times 10^{-11}$ |

2, the disparity between power method and Gauss-Seidel method is almost not distinguishable, probably limited by the size of hyperlink matrix. While Jacobi's method converges sightly slower.

## 3.6 Successive Overrelaxation(SOR)

The successive Overrelaxation method (SOR) was developed independently and almost simultaneously by Frankel [7] and Young [31]. It was described as "Extraplated Liebmann Method" in [7], but it is believed that the difficulty of determining the optimum $\omega-$value would limit its applications to more complex problems. While the optimal choices of relaxation factor to some partial differential equations problems were presented [31].

**Definition 3.6.1.** *The successive overrelaxation method (SOR) is a combination of Jacobi method and Gauss-Seidel method:*

$$(D + \omega L)x^{(k+1)} = ((1 - \omega)D - \omega U)x^{(k)} + \omega b, \tag{3.21}$$

*where $\omega \in \mathbb{R}$ is the relaxation parameter. The corresponding splitting is*

$$M = \frac{1}{\omega}D + L, \quad N = \frac{1-\omega}{\omega}D - U.[16]$$

If $\omega = 1$, the successive overrelaxation method is reduced to Gauss-Seidel method. The optimal $\omega$ is chosen to maximize the rate of convergence, therefore, minimize the spectral radius $\rho(M^{-1}N|_\omega)$ where

$$M^{-1}N = (\frac{1}{\omega}D + L)^{-1} \cdot (\frac{1-\omega}{\omega}D - U). \tag{3.22}$$

**Lemma 4.** *If the inverse $U^{-1}$ of an upper triangular matrix U exists, then it is upper triangular. If the inverse $L^{-1}$ of an lower triangular matrix L exists, then it is lower triangular. [19, p.122]*

**Lemma 5.** *Let $T_n \in \mathbb{R}^{n\times n}$ be an upper or lower triangular matrix, then its determinant $\det(T_n)$ is equal to the product of all the elements on the diagonal.*

$$\det(T_n) = \prod_{k=1}^{n} T_{kk}$$

*[3, p.39]*

**Theorem 3.6.1** (Kahan)**.** *A necessary condition for the SOR method to converge is $|\omega - 1| < 1$ (For $\omega \in \mathbb{R}$ this condition becomes $\omega \in (0, 2)$) [13].*

*Proof.* By equation (3.21)

$$\begin{aligned}
\det(M^{-1}N) &= \det((D + \omega L)^{-1}((1-\omega)D - \omega U)) \\
&= \det((D + \omega L)^{-1}) \det((1-\omega)D - \omega U) \\
&= \det(D^{-1}) \det((1-\omega)D) \\
&= (1-\omega)^n \det(D^{-1}D) \\
&= (1-\omega)^n.
\end{aligned} \tag{3.23}$$

$(D + \omega L)^{-1}$ is a lower triangular matrix by Lemma 4, since $(D + \omega L)$ is a lower matrix. Therefore, $\det(D + \omega L)^{-1} = \prod_{k=1}^{n} ((D + \omega L)^{-1})_{kk} = \prod_{k=1}^{n} D_{kk}^{-1} = \det D^{-1}$. Similarly, $\det((1-\omega)D - \omega U) = \det((1-\omega)D)$.
Suppose $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ are the eigenvalues of $M^{-1}N$, then $\det(M^{-1}N) = (1-\omega)^n = \prod_{i=1}^{n} \lambda_i \leq \lambda_1^n \Rightarrow \lambda_1 \geq (1-\omega)$, therefore, $\rho(M^{-1}N) = \lambda_1 \geq 1 - \omega$.
Hence, the SOR method can converge on if and only if $\rho(M^{-1}N) < 1$ by Theorem 3.2.1, its necessary condition is $|1 - \omega| < 1 \Rightarrow 0 < \omega < 2$ [5]. □

**• Convergence of SOR in PageRank problem**

**Lemma 6.** *Given the adjusted stochastic matrix $S$ and teleportation parameter $\alpha$, if $\lambda$ is an eigenvalue of $T = M^{-1}N$, then $|\frac{\lambda+\omega-1}{\omega}| \leq \max(\alpha, \alpha|\lambda|)$ [11].*

**Theorem 3.6.2** (Sufficient condition). *If $\omega \in (0, \frac{2}{1+\alpha})$, then the spectral radius $\rho(T) < 1$, therefore SOR method is convergent for such $\omega$. [11]*

*Proof.* case 1: $\omega \in [1, \frac{2}{1+\alpha})$. Then the following property holds:

$$1 - \alpha\omega > 1 - \alpha \cdot \frac{2}{1+\alpha} = \frac{1-\alpha}{1+\alpha} > 0 \tag{3.24}$$

The idea is to prove by contradiction. We assume there is an eigenvalue satisfying$|\lambda| \geq 1$, then by Lemma 6:

$$\begin{aligned} |\frac{\lambda + \omega - 1}{\omega}| &\leq \max(\alpha, \alpha|\lambda|) = \alpha|\lambda| \\ \Rightarrow |\lambda + \omega - 1| &\leq \alpha\omega|\lambda| \\ \Rightarrow |\lambda| \leq |\lambda + \omega - 1| + |\omega - 1| &\leq \alpha\omega|\lambda| + |\omega - 1| \quad \text{by triangle inequality} \\ (1 - \alpha\omega)|\lambda| &\leq |\omega - 1| = \omega - 1 \\ |\lambda| &\leq \frac{\omega - 1}{1 - \alpha\omega}. \quad \text{by equation (3.24)} \end{aligned}$$

Moreover, $\omega < \frac{2}{1+\alpha} \Rightarrow \omega + \alpha\omega < 2 \Rightarrow \omega - 1 < 1 - \alpha\omega$, hence $|\lambda| \leq \frac{\omega-1}{1-\alpha\omega} < 1$, which is a contridiction.

case 2: $\omega \in (0, 1]$.

Suppose there exists an eigenvalue $\lambda$ with $|\lambda| \geq 1$, then $|\omega - 1| = 1 - \omega$ and $1 - \omega < 1 - \alpha\omega$.By Lemma 6:

$$|\frac{\lambda + \omega - 1}{\omega}| \leq \max(\alpha, \alpha|\lambda|) = \alpha|\lambda|,$$

Similarly,

$$|\lambda| \leq \frac{|\omega - 1|}{1 - \alpha\omega} = \frac{1 - \omega}{1 - \alpha\omega} < 1,$$

which leads to a contridiction.

In general, if $\omega \in (0, \frac{2}{1+\omega})$, then SOR method converges in the context of PageRank problem [11]. □

**Lemma 7.** *The eigenvalues of the SOR iteration matrix $T = M^{-1}N = -(\frac{1}{\omega}D + L)^{-1}(\frac{\omega-1}{\omega}D + U)$ are the roots of $(\lambda + \omega - 1)^n - \alpha^n\omega^n\lambda = 0$ [11].*

**Theorem 3.6.3** (Necessary condition). *if $\omega \geq \frac{2}{1+\alpha}$, then $\rho(T) \geq 1$ (where $T$ is SOR iteration matrix), with $\rho(T) = 1$ only if $\omega = \frac{2}{1+\alpha}$ [11].*

*Proof.* We define $p_n(\lambda) = (\lambda + \omega - 1)^n - \alpha^n\omega^n\lambda$, then by Lemma 7, any eigenvalue of $T$ satisfies $p_n(\lambda) = 0$. Suppose $\omega > \frac{2}{1+\alpha}$, then $\omega - 2 > \frac{2}{1+\alpha} - 2 = -\frac{2\alpha}{1+\alpha}$ and

$\alpha\omega > \frac{2\alpha}{1+\alpha}$, therefore,

$$p_n(-1) = (\omega-2)^n + \alpha^n\omega^n$$
$$\begin{cases} > -(\frac{2\alpha}{1+\alpha})^n + (\frac{2\alpha}{1+\alpha})^n = 0 \text{ if } n \text{ is odd} \\ > (\frac{2\alpha}{1+\alpha})^n + (\frac{2\alpha}{1+\alpha})^n = 2(\frac{2\alpha}{1+\alpha})^n > 0 \text{ if } n \text{ is even} \end{cases},$$

and

$$\lim_{x\to-\infty} p_n(x) \to -\infty.$$

Since $p_n(x)$ is an continuous function, by the **intermediate value theorem**, there exists a root $\lambda < -1$, therefore $\rho(T) > 1$.

When $\omega = \frac{2}{1+\alpha}$,

$$\begin{aligned} p_n(-1) &= (-1 + \frac{2}{1+\alpha} - 1)^n - \alpha^n(\frac{2}{1+\alpha})^n(-1) \\ &= (-\frac{2\alpha}{1+\alpha})^n + (\frac{2\alpha}{1+\alpha})^n \\ &= 0. \end{aligned} \tag{3.25}$$

Hence, the necessary condition for the convergence of SOR method is $\omega \in (0, \frac{2}{1+\alpha})$ [11]. □

Since $\lim_{\alpha\to 0} \frac{2}{1+\alpha} = 2$, as $\alpha$ approaching $0$, the convergence interval for $\omega$ of the SOR method is approxiamtely $(0, 2)$, while as $\alpha$ approaching $1$, the convergence interval is approximately $\omega \in (0, 1)$ since $\lim_{\alpha\to 1} \frac{2}{1+\alpha} = 1$.

• **Optimal relaxation parameter** $\omega$

**Theorem 3.6.4.** $\omega = 1$ *is optimal in some situations for the PageRank problem, which is exactly Gauss-Seidel method.*

*Proof.* By the results in Theorem 3.6.2,

$$\begin{cases} |\frac{\lambda+\omega-1}{\omega}| & < \alpha|\lambda| \Rightarrow |\lambda| < \frac{|\omega-1|}{1-\alpha\omega} \\ & \text{or} \\ |\frac{\lambda+\omega-1}{\omega}| & < \alpha \Rightarrow |\lambda| < \alpha\omega + |\omega-1| \end{cases} \tag{3.26}$$

Therefore,

$$\lambda < \min_{\omega}(\max(\frac{|\omega-1|}{1-\alpha\omega}, \alpha\omega + |\omega-1|))$$

Since we only need to satisify one of the equations in (3.26).

It is notable that $\min(\alpha\omega + |\omega-1|) = \alpha$, which is obtained when $\omega = 1$, and $\frac{|\omega-1|}{1-\alpha\omega} = 0 < \alpha = \min(\alpha\omega + |\omega-1|)$. Therefore, when $\omega = 1$, $\rho(T) < \alpha$ which is the most strict convergence bound [11]. □

Table 3: List of error between consecutive iterations of iterative methods

| relaxation parameter $\omega$ | $\omega = 0.1$ | $\omega = 0.3$ | $\omega = 0.5$ | $\omega = 0.7$ | $\omega = 0.9$ |
|---|---|---|---|---|---|
| iteration # 1 | $3.83 \times 10^{-3}$ | $1.17 \times 10^{-2}$ | $1.99 \times 10^{-2}$ | $2.85 \times 10^{-2}$ | $3.76 \times 10^{-2}$ |
| iteration # 11 | $2.67 \times 10^{-3}$ | $5.15 \times 10^{-3}$ | $3.35 \times 10^{-3}$ | $1.60 \times 10^{-3}$ | $1.53 \times 10^{-3}$ |
| iteration # 21 | $2.19 \times 10^{-3}$ | $1.43 \times 10^{-3}$ | $3.94 \times 10^{-4}$ | $3.37 \times 10^{-4}$ | $2.02 \times 10^{-4}$ |
| iteration # 31 | $1.60 \times 10^{-3}$ | $4.22 \times 10^{-4}$ | $1.40 \times 10^{-4}$ | $8.56 \times 10^{-5}$ | $2.14 \times 10^{-5}$ |
| iteration # 41 | $1.12 \times 10^{-3}$ | $1.42 \times 10^{-4}$ | $6.28 \times 10^{-5}$ | $1.95 \times 10^{-5}$ | $2.19 \times 10^{-6}$ |
| iteration # 51 | $7.83 \times 10^{-4}$ | $5.75 \times 10^{-5}$ | $2.63 \times 10^{-5}$ | $4.31 \times 10^{-6}$ | $2.23 \times 10^{-7}$ |
| iteration # 61 | $5.41 \times 10^{-4}$ | $3.07 \times 10^{-5}$ | $1.06 \times 10^{-5}$ | $9.38 \times 10^{-7}$ | $2.25 \times 10^{-8}$ |
| iteration # 71 | $3.75 \times 10^{-4}$ | $1.94 \times 10^{-5}$ | $4.14 \times 10^{-6}$ | $2.03 \times 10^{-7}$ | $2.28 \times 10^{-9}$ |
| iteration # 80 | $2.71 \times 10^{-4}$ | $1.32 \times 10^{-5}$ | $1.76 \times 10^{-6}$ | $5.12 \times 10^{-8}$ | $2.91 \times 10^{-10}$ |

Using the default teleportation parameter $\alpha = 0.85$, then $\omega \in (0, \frac{2}{1+\alpha}) = (0, 1.08108)$ guarantees the convergence of SOR method by Theorem 3.6.2. The table 3 confirms this property since all the chosen 5 values of $\omega$ show decreasing errors between consecutive iterations(the web structure remains the same as table 2). Moreover, with increasement of $\omega$ from 0.1 to 0.9, the errors (after few iterations) decay if compared in the same iteration. However, to verify this is not a coincidence, it is reasonable to compute the number of iterations required for a given error ($10^{-3}$) as $\omega$ varying from 0.05 to 1.33 (with step 0.02).

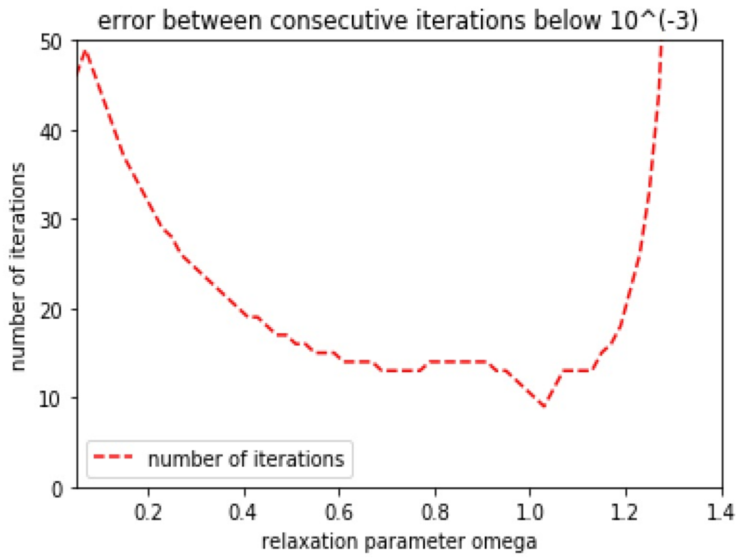


Figure 5: relationship between $\omega$ and the number of iterations required for a given error

```
iteration_number_lst = [46, 49, 46, 43, 40, 37, 35, 33, 31, 29, 28, 26,
                        25, 24, 23, 22, 21, 20, 19, 19, 18, 17, 17, 16,
                        16, 15, 15, 15, 14, 14, 14, 14, 13, 13, 13, 13,
                        13, 14, 14, 14, 14, 14, 14, 14, 13, 13, 12, 11,
                        10, 9, 11, 13, 13, 13, 13, 15, 16, 18, 22, 26,
                        33, 44, 66, 132]

plt.axis([0.05, 1.4, 0, 50])
t = np.arange(0.05,1.33,0.02)
plt.xlabel('relaxation parameter omega')
plt.ylabel('number of iterations')
plt.title('error between consecutive iterations below 10^(-3)')
convergence_times = plt.plot(t,iteration_number_lst,'r--',label= '
                                    number of iterations')
plt.legend()
plt.show()
```

The number of iterations required for error below $10^{-3}$ with the mentioned $64$ values of $\omega$ (from $0.05$ to $1.33$ with step $0.02$) are shown in figure 5, it supports the theoretical result 3.6.4 that the optimal relaxation parameter $\omega$ is approximate $1$ (only requires around $9$ iterations), namely, the Gauss-Seidel method. Note that $0 < \omega < 1$ leads to the spectral radius $\rho(T) < 1$, which guarantees the convergence, while the number of iterations required begin to grow as $\omega$ increasing away from $1$.

## 3.7 Krylov Subspace methods

According to the equation (3.13), $\pi^{(k+1)T} = \alpha\pi^{(k)T}H + (\alpha\pi^{(k)T}a + (1-\alpha))\frac{1}{n}e^T$, apart from the sparse hyperlink matrix $H$ and dangling node vector $a$, we only need to store and update $\pi^{(k)T}$ in the current iteration, which indicates that the power method is storage-friendly. While the other iterative methods for solving linear systems, for example, **GMRES** and **BICGSTAB**, although they are faster algorithms, but require the storage of multiple vectors, therefore, infeasible in the context of PageRank problem [18, p.41].

### • GMRES(Generalized minimal residual method)

GMRES was developed by Saad and Schultz in 1986, as a generalization of Paige and Saunders' MINRES algorithm. When solving the large linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is invertible and nonsymmetric, the basic idea is to start with an initial guess $x_0$ and $r_0 = b - Ax_0$, a correlation $z_m$ in the $m^{th}$ iteration is in the Krylov Subspace

$$\mathcal{K}_m = span\{r_0, Ar_0, \cdots, A^{m-1}r_0\},$$

which satisfies

$$\min_{z\in\mathcal{K}_m} \|b - A(x_0 + z)\|_2. \tag{3.27}$$

At the $m^{th}$ iteration, $x_m = x_0 + z_m$, and the residual norm (3.27) is nonincreasing in this process [25][29].

Consider the application of GMRES method to the linear system (3.20):

$$(I - \alpha S^T)\pi = (1-\alpha)\frac{1}{n}e$$

Note that $A = I - \alpha S^T \in \mathbb{R}^{n\times n}$ is sparse with only $11n$ nonzero elements approximately, therefore, computing complexity caused by the matrix-vector product is $O(n)$ in each iteration, with other $O(nm)$ floating point operations at the $m^{th}$ iteration. And for GMRES(m), we need $(m+2)n$ storage locations, therefore, infeasible in the PageRank problem as $n \sim O(10^{10})$[25], although it shows significant advantage in dealing with small-size web structure (only requires $376ms$ to find the PageRank vector corresponding to the $2657$ Wikipedia pages within the error of $10^{-6}$).

However, it is pointed out that ”near singularity” and ”stagnation” (slow rate of convergence) are two significant disadvantages of original GMRES method, but a modified and shifted GMRES(m) algorithm was proposed to solve the problem of near singularity and a polynomial preconditioner would help to improve convergence [30].

**• BICG and BICGSTAB**

BICG (Biconjugate gradient method) is another Krylov space method for solving non-Hermitian linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ (preferrably after applying a preconditioner to the matrix in the original linear system). This iterative method starts with an initial guess $x_0$ and generates $r_i$ in each iteration with $r_i = b - Ax_i$ satisfying:

(1) $r_n$ lies in the Krylov space:

$$K_m(A, r_0) = span\{r_0, Ar_0, \cdots, A^{m-1}r_0\},$$

(2) $r_n$ is orthogonal to another Krylov space with a distinct initial guess $y_0$ and Hermitian $A^*$:

$$r_n \perp \mathcal{L}_m(A^*, y_0) = span\{y_0, A^*y_0, \cdots, (A^*)^{m-1}y_0\}.[12]$$

However, BICGSTAB( biconjugate gradient stabilized method) is numerically more stable comparing to BICG, with faster convergence and less storage issues [1]. The rate of convergence of BICGSTAB is similar to GMRES, it requires $361ms$ for the implementation of the mentioned Wikipedia pages structure.

# 4 The Sensitivity of teleportation parameter $\alpha$

A markov chain is obtained by perturbing the transition matrix $G$ with a uniform teleportation parameter $\alpha$, PageRank vector is defined as its stationary state [4]. $\alpha$ is originally chosen to be $0.85$ by Page and Brin, which is a compromise between the reflection of web structure and calculation efficiency [22]. When $\alpha = 0$, the influence of web structure vanishes, only the random surfer process remains. As $\alpha$ approaching to 1, more importance is attached to the web structure.

The emphasis of this chapter is the sensitivity of PageRank value with respect to the teleportation parameter $\alpha$ ($0 < \alpha < 1$). We will examine the derivative and apply perturbation theory to measure how PageRank responds to a small change in $\alpha$.

## 4.1 PageRank's derivative with respect to $\alpha$

We denote the derivative of $\pi^T$ with respect to $\alpha$ as $\frac{d\pi^T}{d\alpha}$, which is commonly used as an approximate measure of how PageRank vector responds as $\alpha$ varies sightly, but the actual changes cannot be deduced from it. For the $j^{th}$ entry of $\pi$, the positive sign of $\frac{d\pi_j^T}{d\alpha}$ indicates that the corrsponding PageRank value $P_j$ increases with the small increase of $\alpha$. If $\frac{d\pi^T}{d\alpha}$ is large in magnitude, then the $j^{th}$ PageRank value $P_j$ is sensitive to the small change in $\alpha$ [18, p.57].

**Theorem 4.1.1.** *The $j^{th}$ component of PageRank vector is*

$$\pi_j^T(\alpha) = \frac{D_j(\alpha)}{\sum\limits_{i=1}^{n} D_i}, \tag{4.28}$$

*Where $D_j$ is the $j^{th}$ principal minor determinant of order $n-1$ in $I - G(\alpha)$*

$$D_j = \det \begin{bmatrix} 1-G_{11} & \cdots & -G_{1(j-1)} & -G_{1(j+1)} & \cdots & -G_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -G_{(j-1)1} & \cdots & 1-G_{(j-1)(j-1)} & -G_{(j-1)(j+1)} & \cdots & -G_{(j-1)n} \\ -G_{(j+1)1} & \cdots & -G_{(j+1)(j-1)} & 1-G_{(j+1)(j+1)} & \cdots & -G_{(j+1)n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & \cdots & -G_{n(j-1)} & -G_{n(j+1)} & \cdots & 1-G_{nn} \end{bmatrix}.$$

*Proof.* Let $A = I - G$, where $I, G \in \mathbb{R}^{n\times n}$. Let $e = (1, 1, \dots 1)^T$ be a column vector of dimension $n$. Then

$$\begin{cases} Ie = e \\ Ge = e \text{ since G is row stochastic} \end{cases} \Rightarrow Ae = (I-G)e = Ie - Ge = 0. \tag{4.29}$$

Therefore, A is singular since $e \neq 0$ but $e$ lies in the null space of $A$, and $\det(A) = 0$. Moreover, $dim(null(A)) = 1 \Rightarrow rank(A) = n - dim(null(A)) = n - 1$

$$A \cdot [adj(A)] = [adj(A)] \cdot A = \det(A)I = 0. \tag{4.30}$$

Since $adj(A)$ lies in the null space of $A$, and $adj(A) \neq 0$, $dim(adj(A)) = 1$. By equation (4.29), we can deduce that each column of $adj(A)$ is a multiple of $e$.
$D_i = adj(A)_{ii}$, therefore, $adj(A) = e(D_1, D_2, \dots, D_n)$
Similarly, $A^T\pi = (I-G)^T\pi = I\pi - G^T\pi = 0$. By second half of (4.30): $A^T \cdot [adj(A)^T] = ([adj(A)] \cdot A)^T = 0$. For the same reason, every column of $adj(A)^T$ is a multiple of $\pi$, therefore, every row of $adj(A)$ is a multiple of $\pi^T$.

$$\Rightarrow \mu\pi^T = (D_1, D_2, \dots, D_n) \text{ for some } \mu \neq 0 \tag{4.31}$$

Multiply $e$ on the both side:

$$\mu = \mu\pi^T e = (D_1, D_2, \dots, D_n)e = \sum_{j=1}^{n} D_j.$$

Hence, by (4.31), $\pi^T = \frac{1}{\mu}(D_1, D_2, \ldots, D_n) = \frac{1}{\sum j{=}1^n D_j}(D_1, D_2, \ldots, D_n)$,
Then the PageRank value of the $j^{th}$ component is:

$$\Rightarrow \pi_j^T = \frac{D_j}{\sum\limits_{i=1}^{n} D_i}.$$

[18, p.66] $\square$

**Definition 4.1.1** (p-norm). *For $p \geq 1$, the p-norm is defined as: [28]*

$$\| x \|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}} = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}.$$

**Lemma 8** (Hölder's inequality). *[28]*

$$|x^* y| \leq ||x||_p ||y||_q \text{ with } \frac{1}{p} + \frac{1}{q} = 1. \tag{4.32}$$

**Theorem 4.1.2.** *Suppose the PageRank vector is $\pi^T = (\pi_1, \pi_2, \ldots, \pi_n)$, then its derivative satisifies*

$$|\pi_j'(\alpha)| \leq \frac{1}{1-\alpha} \textit{ for each j =1,2, \ldots, n}$$

$$\|\frac{d\pi^T}{d\alpha}\|_1 \leq \frac{2}{1-\alpha}.$$

*Proof.*

$$\pi^T = \pi^T G = \pi^T[\alpha S + (1-\alpha)\frac{1}{n}ee^T],$$

$$\Rightarrow \pi^T(I - \alpha S) = (1-\alpha)\frac{1}{n}(\pi^T e)e^T = (1-\alpha)\frac{1}{n}e^T. \tag{4.33}$$

Differentiate both sides with respect to $\alpha$:

$$\frac{d\pi^T}{d\alpha}(I - \alpha S) = \pi^T S - \frac{1}{n}e^T,$$

$$\Rightarrow \frac{d\pi^T}{d\alpha} = [\pi^T S - \frac{1}{n}e^T](I - \alpha S)^{-1}. \tag{4.34}$$

The $i^{th}$ component of (4.34) is:

$$\frac{d\pi_i^T}{d\alpha} = [\pi^T S - \frac{1}{n}e^T](I - \alpha S)^{-1}e_i, \tag{4.35}$$

By Lemma 8, if we choose $p = 1$ and $q \to \infty$, which satisifies $\frac{1}{p} + \frac{1}{q} \to 1$. When $x, y \in \mathbb{R}^n$:

$$|x^* y| = |x^T y| \leq ||x||_1 ||y||_\infty.$$

Suppose that $x$ is orthogonal to $e \Rightarrow x^T e = 0$, then:

$$|x^T y| = |x^T(y - \alpha e)| \leq ||x||_1 ||y - \alpha e||_\infty, \tag{4.36}$$

$$||y-\alpha e||_\infty = \max|y_i-\alpha| = \begin{cases} \max y_i - \alpha \text{ if } \alpha < \frac{\max y_i + \min y_i}{2} (1) \\ \frac{\max y_i - \min y_i}{2} \text{ if } \alpha = \frac{\max y_i + \min y_i}{2} (2) \\ \alpha - \min y_i \text{ if } \alpha > \frac{\max y_i + \min y_i}{2} (3) \end{cases}. \quad (4.37)$$

In the case (1) and (3), the result in equation (4.37) $> \frac{\max y_i - \min y_i}{2}$.
Therefore, $\min ||y-\alpha e||_\infty = \frac{\max y_i - \min y_i}{2}$, if and only if $\alpha = \frac{\max y_i + \min y_i}{2}$.
Let $x = \pi^T S - \frac{1}{n}e^T = \pi^T(S - \frac{1}{n}ee^T)$, and the $i^{th}$ component of $(S-\frac{1}{n}ee^T)e$ is $\sum_{k=1}^{n}(S-\frac{1}{n}ee^T)_{ik}\cdot e_k = (\sum_{k=1}^{n} S_{ik} - n\cdot\frac{1}{n})\cdot 1 = 1-1 = 0 \Rightarrow (S-\frac{1}{n}ee^T)e$ is a zero vector of dimension $\mathbb{R}^n$. Therefore, $x^Te = (S-\frac{1}{n}ee^T)^Te = 0$.
Let $y = (I-\alpha S)^{-1}e_i$. Using the result in equation (4.35) and (4.36)

$$|\frac{d\pi_i^T}{d\alpha}| = [\pi^T S - \frac{1}{n}e^T](I-\alpha S)^{-1}e_i \le ||\pi^T S - \frac{1}{n}e^T||_1||(I-\alpha S)^{-1}e_i||_\infty \le ||\pi^T S - \frac{1}{n}e^T||_1 \cdot \frac{\max y_i - \min y_i}{2}. \quad (4.38)$$

By triangle inequality and since $S_i \le 1, \forall i = 1,2,\ldots,n$

$$||\pi^T S - \frac{1}{n}e^T||_1 = \sum_{i=1}^{n}|(\pi^T S - \frac{1}{n}e^T)_i| \le \sum_{i=1}^{n}|\pi^T S_i| + \frac{1}{n}\sum_{i=1}^{n}|e_i^T| \le 1+1 = 2,$$

Then the equation (4.38) becomes:

$$|\frac{d\pi_i^T}{d\alpha}| \le \max y_i - \min y_i. \quad (4.39)$$

where $y = (I-\alpha S)^{-1}e_i$, it is the $i^{th}$ column of $(I-\alpha S)^{-1}$.
Since

$$(I-\alpha S)e = e - \alpha S e = (1-\alpha)e \Rightarrow (I-\alpha S)^{-1}e = \frac{1}{1-\alpha}e,$$

$$y = (I-\alpha S)^{-1}e_i = \frac{1}{1-\alpha}e_i \Rightarrow y_{\min} \ge 0, \quad (4.40)$$

$$\begin{aligned} y_{\max} &= \max_{1\le k\le n}(I-\alpha S)^{-1}_{ik} \le \max_{1\le i,k\le n}(I-\alpha S)^{-1}_{ik} \\ &\le \|(I-\alpha S)^{-1}\|_\infty = \|(I-\alpha S)^{-1}e\|_\infty \\ &= \|\frac{1}{1-\alpha}e\|_\infty = \frac{1}{1-\alpha}. \end{aligned} \quad (4.41)$$

Since $\|(I-\alpha S)^{-1}\|_\infty$ is the max row sum of $(I-\alpha S)^{-1}$, and each entry of $(I-\alpha S)e$ is the corresponding row sum.
Consequently, Substitute equations (4.40) and (4.41) into (4.39)

$$|\frac{d\pi_i^T}{d\alpha}| \le \max y_i - \min y_i \le \frac{1}{1-\alpha}. \quad (4.42)$$

In order to prove $\|\frac{d\pi^T(\alpha)}{d\alpha}\|_1 \le \frac{2}{1-\alpha}$, we consider the transpose of equation (4.34):

$$\frac{d\pi}{d\alpha} = ((I-\alpha S)^{-1})^T(\pi^T S - \frac{1}{n}e^T)^T,$$

Then we take the $1-$norm on the both sides:

$$\begin{aligned}\|\frac{d\pi}{d\alpha}\|_1 &= \|((I-\alpha S)^{-1})^T(\pi^T S - \frac{1}{n}e^T)^T\|_1 \\ &\le \|((I-\alpha S)^{-1})^T\|_1 \|(\pi^T S - \frac{1}{n}e^T)^T\|_1 \\ &\le \|(I-\alpha S)^{-1}\|_\infty \|\pi^T S - \frac{1}{n}e^T\|_\infty.\end{aligned}$$

Since the subordinate 1-norm is consistent ($\|AB\|_1 \le \|A\|_1\|B\|_1$) and $\|A^T\|_1 = \max\limits_{1\le j\le n}\sum\limits_{i=1}^{m}|a_{ij}^T| = \max\limits_{1\le j\le n}\sum\limits_{i=1}^{m}|a_{ji}| = \|A\|_\infty$.

Using the fact that $\|\pi^T S - \frac{1}{n}e^T\|_\infty \le \|\pi^T S - \frac{1}{n}e^T\|_1 \le 2$ by equation (4.39) and $\|(I-\alpha S)^{-1}\|_\infty = \|(I-\alpha S)^{-1}e\|_\infty = \|\frac{1}{1-\alpha}e\|_\infty = \frac{1}{1-\alpha}$, we can deduce that

$$\|\frac{d\pi^T}{d\alpha}\|_1 = \|\frac{d\pi}{d\alpha}\|_1 \le \frac{1}{1-\alpha}\cdot 2 = \frac{2}{1-\alpha}.$$

Hence, the derivative of the PageRank vector $\pi^T$ satisifies $|\pi_j'(\alpha)| \le \frac{1}{1-\alpha}$ for each j =1,2, . . . , n and $\|\frac{d\pi^T}{d\alpha}\|_1 \le \frac{2}{1-\alpha}$ [18, p.66-67].

Note that both the derivative of PageRank vector measure in $1-$norm and each of its components are bounded above by terms involving $\alpha$, and as $\alpha \to 1$, both the bounds $\to \infty$, which indicates the PageRank vector is more sensitive to small changes in $\alpha$. □

**Lemma 9.** *Suppose $A(t)x(t) = b(t)$ and the entries in $A(t)$, $x(t)$ and $b(t)$ are differentiable. If $A(t)^{-1}$, then:*

$$\frac{dA(t)^{-1}}{dt} = -A(t)^{-1}A'(t)A(t)^{-1}.[19]$$

**Theorem 4.1.3.** *The derivative of PageRank vector $\pi^T(\alpha)$ with respect to $\alpha$ is given by*

$$\frac{d\pi^T(\alpha)}{d\alpha} = -\frac{1}{n}e^T(I-S)(I-\alpha S)^{-2}.$$

*Proof.*

$$\pi^T = \pi^T\cdot G = \pi^T\cdot(\alpha S + (1-\alpha)\frac{1}{n}ee^T),$$

$$\Rightarrow \pi^T\cdot(I-\alpha S-(1-\alpha)\frac{1}{n}ee^T) = 0^T.$$

Multiply both sides on the right by $(I-\alpha S)^{-1}$,

$$\pi^T(I-(1-\alpha)\frac{1}{n}ee^T(I-\alpha S)^{-1}) = 0^T,$$

$$\Rightarrow \pi^T = \pi^T(1-\alpha)\frac{1}{n}ee^T(I-\alpha S)^{-1} = (1-\alpha)\frac{1}{n}e^T(I-\alpha S)^{-1}.$$

By Lemma 9, $\frac{d}{d\alpha}(I-\alpha S)^{-1} = -(I-\alpha S)^{-1}(-S)(I-\alpha S)^{-1} = (I-\alpha S)^{-1}S(I-\alpha S)^{-1}$, then

$$\begin{aligned}
\frac{d\pi^T}{d\alpha} &= \frac{d}{d\alpha}[(1-\alpha)\frac{1}{n}e^T(I-\alpha S)^{-1}] \\
&= (1-\alpha)\frac{1}{n}e^T\frac{d}{d\alpha}(I-\alpha S)^{-1} - \frac{1}{n}e^T(I-\alpha S)^{-1} \\
&= (1-\alpha)\frac{1}{n}e^T(I-\alpha S)^{-1}S(I-\alpha S)^{-1} - \frac{1}{n}e^T(I-\alpha S)^{-1} \\
&= -\frac{1}{n}e^T(I-\alpha S)^{-1}[I-(1-\alpha)S(I-\alpha S)^{-1}] \qquad (4.43)\\
&= -\frac{1}{n}e^T(I-\alpha S)^{-1}[(I-\alpha S)-(1-\alpha)S](I-\alpha S)^{-1} \\
&= -\frac{1}{n}e^T(I-\alpha S)^{-1}(I-S)(I-\alpha S)^{-1} \\
&= -\frac{1}{n}e^T(I-S)(I-\alpha S)^{-2}.
\end{aligned}$$

$(I-S)$ and $(I-\alpha S)^{-1}$ commute since by Lemma 3 and $\rho(\alpha S)<1$, $(I-\alpha S)^{-1} = \sum_{k=0}^{n}\alpha^k S^k$, each term $S^k$ commutes with both $I$ and $S$.

In particular,

$$\lim_{\alpha\to 0}\frac{d\pi^T}{d\alpha} = -\frac{1}{n}e^T(I-S)I^{-2} = -\frac{1}{n}e^T(I-S),$$

and

$$\lim_{\alpha\to 1}\frac{d\pi^T}{d\alpha} = -\frac{1}{n}e^T(I-S)(I-S)^{-2} = -\frac{1}{n}e^T(I-S)^{-1}.$$

[18, p.68] □

## 4.2 Higher order derivatives

**Theorem 4.2.1** (Exact formula for derivatives)**.** *If we denote $(I-\alpha S)$ as $M(\alpha)$, then the following properties hold:*
*(1). $\pi'(\alpha) = (\pi^T S - \frac{1}{n}e^T)M(\alpha)^{-1}$;*
*(2). $\frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}} = (k+1)\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}SM(\alpha)^{-1}\ \forall k\geq 1$.*

*Proof.* (1) $M(\alpha) = I-\alpha S \Rightarrow M'(\alpha) = -S$. From equation (3.20), $(I-\alpha S^T)\pi = (1-\alpha)\frac{1}{n}e \Rightarrow \pi^T(I-\alpha S) = (1-\alpha)\frac{1}{n}e^T \Rightarrow \pi^T M(\alpha) = (1-\alpha)\frac{1}{n}e^T$.

Differentiate it on the both sides:

$$\begin{aligned}
\frac{d\pi^T}{d\alpha}M(\alpha) + \pi^T M'(\alpha) &= -\frac{1}{n}e^T \\
\frac{d\pi^T}{d\alpha}M(\alpha) &= -\frac{1}{n}e^T - \pi^T M'(\alpha) \qquad (4.44)\\
\frac{d\pi^T}{d\alpha}M(\alpha) &= -\frac{1}{n}e^T + \pi^T S,
\end{aligned}$$

Since $M(\alpha) = I - \alpha S$ is invertible,

$$\frac{d\pi^T}{d\alpha} = (\pi^T S - \frac{1}{n}e^T)M(\alpha)^{-1}.$$

(2) Now we prove by induction to deduce the $(k+1)^{st}$ derivative of $\pi^T$ (for $k \geq 1$).

base case ($k = 1$): differentiate equation (4.44) again to obtain:

$$\begin{aligned}
\frac{d\pi^T}{d\alpha}M'(\alpha) + \frac{d^2\pi^T}{d\alpha^2}M(\alpha) &= \frac{d\pi^T}{d\alpha}S \\
\frac{d^2\pi^T}{d\alpha^2}M(\alpha) &= \frac{d\pi^T}{d\alpha}S - \frac{d\pi^T}{d\alpha}M'(\alpha) \\
\frac{d^2\pi^T}{d\alpha^2}M(\alpha) &= \frac{d\pi^T}{d\alpha}S + \frac{d\pi^T}{d\alpha}S = 2\frac{d\pi^T}{d\alpha}S.
\end{aligned}$$

Therefore,

$$\frac{d^2\pi^T}{d\alpha^2} = 2\frac{d\pi^T}{d\alpha}SM(\alpha)^{-1},$$

which satisifies the formula for $k = 1$.

Inductive steps: assume $\frac{d^{(k)}\pi^T}{d\alpha^{(k)}} = k\frac{d^{(k-1)}\pi^T}{d\alpha^{(k-1)}}SM(\alpha)^{-1}$ hold, then

$$\begin{aligned}
\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}M(\alpha) &= k\frac{d^{(k-1)}\pi^T}{d\alpha^{(k-1)}}S \\
\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}M'(\alpha) + \frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}}M(\alpha) &= k\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S \\
\frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}}M(\alpha) &= k\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S - \frac{d^{(k)}\pi^T}{d\alpha^{(k)}}M'(\alpha) \\
\frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}}M(\alpha) &= k\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S + \frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S = (k+1)\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S \\
\frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}} &= k\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S + \frac{d^{(k)}\pi^T}{d\alpha^{(k)}}S = (k+1)\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}SM(\alpha)^{-1}.
\end{aligned}$$

Hence, we have proved that the $(k+1)^{st}$ derivative of PageRank vector $\pi^T$ is $\frac{d^{(k+1)}\pi^T}{d\alpha^{(k+1)}} = (k+1)\frac{d^{(k)}\pi^T}{d\alpha^{(k)}}SM(\alpha)^{-1}$ by induction [4]. □

**Corollary 1.** *The $k^{th}$ derivative of PageRank vector $\pi^T$ is*

$$\frac{d^{(k)}\pi^T}{d\alpha^{(k)}} = k!\pi^T(SM(\alpha)^{-1} - \frac{1}{1-\alpha}I)(SM(\alpha)^{-1})^{k-1}. \tag{4.45}$$

*Proof.* By Theorem 4.2.1

$$\begin{aligned}
\pi'(\alpha) &= (\pi^T S - \frac{1}{n}e^T)M(\alpha)^{-1} \\
&= \pi^T SM(\alpha)^{-1} - \frac{1}{n}e^T M(\alpha)^{-1} \\
&= \pi^T SM(\alpha)^{-1} - \frac{1}{1-\alpha}\pi^T \\
&= \pi^T(SM(\alpha)^{-1} - \frac{1}{1-\alpha}I),
\end{aligned}$$

Since $\pi^T M(\alpha) = (1-\alpha)\frac{1}{n}e^T \Rightarrow \frac{1}{n}e^T M(\alpha)^{-1} = \frac{1}{1-\alpha}\pi^T$.

Therefore, using the property that $\frac{d^{(k)}\pi^T}{d\alpha^{(k)}} = k\frac{d^{(k-1)}\pi^T}{d\alpha^{(k-1)}}SM(\alpha)^{-1}$,

$$\begin{aligned}
\frac{d^{(k)}\pi^T}{d\alpha^{(k)}} &= k[(k-1)\frac{d^{(k-2)}\pi^T}{d\alpha^{(k-2)}}SM(\alpha)^{-1}]SM(\alpha)^{-1} \\
&= k(k-1)\frac{d^{(k-2)}\pi^T}{d\alpha^{(k-2)}}(SM(\alpha)^{-1})^2 \\
&= k(k-1)(k-2)\frac{d^{(k-3)}\pi^T}{d\alpha^{(k-3)}}(SM(\alpha)^{-1})^3 \\
&\vdots \\
&= k(k-1)\cdots 2\frac{d\pi^T}{d\alpha}(SM(\alpha)^{-1})^{k-1} \\
&= k!\pi^T(SM(\alpha)^{-1} - \frac{1}{1-\alpha}I)(SM(\alpha)^{-1})^{k-1}.
\end{aligned} \tag{4.46}$$

[4] □

Although we have derived the exact formula for the $k^{th}$ derivative of $\pi^T$ based on $\pi^T$ or $\frac{d^{(k-1)}\pi^T}{d\alpha^{k-1}}$, but it is computationally infeasible to invert $M(\alpha) = I - \alpha S$. Instead, we would try to approximate $\pi^T$ and its derivatives for given $\alpha$ after ensuring the convergence.

**Definition 4.2.1.** *The sequence $\pi_0^{T(0)}(\alpha), \pi_1^{T(0)}(\alpha), \ldots, \pi_n^{T(0)}(\alpha), \cdots$ is defined as the PageRank vector in each iteration, with $\lim_{k\to\infty}\pi_k^{T(0)}(\alpha) = \pi^T$.*
*Similarly, the sequence $\pi_0^{T(1)}(\alpha), \pi_1^{T(1)}(\alpha), \ldots, \pi_n^{T(1)}(\alpha), \cdots$ is defined as the first order derivative of PageRank vector in each iteration, with $\lim_{k\to\infty}\pi_k^{T(1)}(\alpha) = \frac{d\pi^T}{d\alpha}$.*
*However, the sequence $\pi_0^{T(t)}(\alpha), \pi_1^{T(t)}(\alpha), \ldots, \pi_n^{T(t)}(\alpha), \cdots$ will not converge to the $t^{th}$ derivative in general. Moreover, define an associated sequence $l_0^{(k)}(\alpha), l_1^{(k)}(\alpha), \cdots, l_t^{(k)}(\alpha)$ as:*

$$\begin{aligned}
l_t^{(0)}(\alpha) &= \pi_t^{T(0)}(\alpha), \\
l_t^{(1)}(\alpha) &= \pi_t^{T(1)}(\alpha) - \frac{1}{1-\alpha}\pi_t^{T(0)}(\alpha), \\
l_t^{(k)}(\alpha) &= k\pi_t^{T(k)}(\alpha) \quad \forall k \geq 2.
\end{aligned} \tag{4.47}$$

*[4]*

**Theorem 4.2.2** (approximation for derivatives of $\pi^T$). *With $l_t^{(1)}(\alpha)$ defined in equation (4.47),*

$$\lim_{t\to\infty} l_t^{(1)}(\alpha) = \frac{d\pi^T}{d\alpha},$$

*as t $\to \infty$, the difference is*

$$\|l_t^{(1)}(\alpha) - \frac{d\pi^T}{d\alpha}\| = O(t\alpha^t).$$

*And*

$$\lim_{t\to\infty} l_t^{(k)}(\alpha) = \pi^{T(k)}(\alpha),$$

*the difference is $O(t^k\alpha^t)$ in norm. [4]*

It is pointed out that PageRank and its derivatives are nearly proportional in a numerical experiment (although whether this property still holds for the world wide web structure is not addressed), a reasonable explanation is that pages with larger PageRank are more sensitive to changes in $\alpha$ as they are more dependent on all inlinks [10].

## 4.3 Perturbation analysis of PageRank vector

Apart from examining the derivative of PageRank vector, we will also explore how the PageRank vector responses to a small perturbation in Google matrix.

**Theorem 4.3.1.** *Suppose the original Google matrix is $G = \alpha S + (1-\alpha)E$ with PageRank vector $\pi^T$, and the updated Google matrix is $\widetilde{G} = \alpha\widetilde{S} + (1-\alpha)E$ with corresponding PageRank vector $\widetilde{\pi}^T$. Let $V$ be the set of updated webpages, then*

$$\|\pi^T - \widetilde{\pi}^T\|_1 \le \frac{2\alpha}{1-\alpha}\sum_{i\in V}\pi_i.$$

*[18, p.69]*

*Proof.* Define $R = S - \widetilde{S}$ as the perturbation between the two adjusted stochastic matrices. Since $\pi^T$ and $\widetilde{\pi}^T$ are the dominant eigenvectors of $G$ and $\widetilde{G}$, $\pi^T = \pi^T G$ and $\widetilde{\pi}^T = \widetilde{\pi}^T G$. Moreover, $(1-\alpha)\pi^T E = (1-\alpha)\widetilde{\pi}^T E = \frac{1-\alpha}{n}(1,1,\cdots,1)^T$ as $E_{ij} = \frac{1}{n}\,\forall i,j$ and $\pi^T$ is normalized, then:

$$\begin{aligned}
\pi^T - \widetilde{\pi}^T &= \pi^T G - \widetilde{\pi}^T\widetilde{G}\\
&= \pi^T(\alpha S + (1-\alpha)E) - \widetilde{\pi}^T(\alpha\widetilde{S} + (1-\alpha)E)\\
&= \alpha\pi^T S - \alpha\widetilde{\pi}^T\widetilde{S}\\
&= \alpha\pi^T(S - \widetilde{S}) + \alpha(\pi^T - \widetilde{\pi}^T)\widetilde{S}\\
&= \alpha\pi^T R + \alpha(\pi^T - \widetilde{\pi}^T)\widetilde{S},
\end{aligned}$$

Note that $I - \alpha\widetilde{S}$ is invertible, therefore,

$$\begin{aligned}
(\pi^T - \widetilde{\pi}^T)(I - \alpha\widetilde{S}) &= \alpha\pi^T R\\
\Rightarrow \pi^T - \widetilde{\pi}^T &= \alpha\pi^T R(I - \alpha\widetilde{S})^{-1},
\end{aligned} \tag{4.48}$$

Since $\widetilde{S}e = e$, $(I - \alpha\widetilde{S})e = (1-\alpha)e \Rightarrow (I - \alpha\widetilde{S})^{-1}e = \frac{1}{1-\alpha}e$, therefore,

$$\|(I - \alpha\widetilde{S})^{-1}\|_\infty = \|(I - \alpha\widetilde{S})^{-1}e\|_\infty = \|\frac{1}{1-\alpha}e\|_\infty = \frac{1}{1-\alpha}.$$

Using $1-$norm and $\infty-$ norm to measure equation (4.48)

$$\begin{aligned}
\|\pi^T - \widetilde{\pi}^T\|_1 &\le \alpha\|\pi^T R\|_1\|(I - \alpha\widetilde{S})^{-1}\|_\infty\\
&\le \frac{\alpha}{1-\alpha}\|\pi^T R\|_1.
\end{aligned} \tag{4.49}$$

Without loss of generality, we suppose that the $n$ updated webpages correspond to the top $n$ rows of $R$ ($S$ and $\widetilde{S}$ differ in the first $n$ rows) and $\pi_1^T$ corresponds to the PageRank value of updated webpages, then $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$,

$$\begin{aligned}\|\pi^T R\|_1 &= \|(\pi_1^T \ \pi_2^T) \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \|_1 = \|\pi_1^T R_1\|_1 \\ &\leq \|\pi_1^T\|_1 \|R_1\|_\infty = \|\pi_1^T\|_1 \|S_1 - \widetilde{S}_1\|_\infty \\ &\leq \|\pi_1^T\|_1 (\|S_1\|_\infty + \|\widetilde{S}_1\|_\infty) \leq 2\|\pi_1^T\|_1,\end{aligned}$$

Substitute it into equation (4.49),

$$\begin{aligned}\|\pi^T - \widetilde{\pi}^T\|_1 &\leq \frac{2\alpha}{1-\alpha}\|\pi_1^T\|_1 \\ &= \frac{2\alpha}{1-\alpha}\sum_{i \in V} \pi_i,\end{aligned}$$

where $V$ is the set of webpages that have been updated. □

In general, the parameter $\frac{2\alpha}{1-\alpha}$ in the bound tends to $\infty$ as $\alpha \to 1$, which indicates that as $\alpha$ approaching 1, the PageRank vector behaves more sensitive to small perturbation in web structure [18, p.69].

## 4.4 Experimental results

Table 4: 10 Wikipedia pages with highest PageRank scores

| $\alpha = 0.1$ | $\alpha = 0.5$ | $\alpha = 0.85$ | $\alpha = 0.99$ |
|---|---|---|---|
| Acacia sensu lato | Google Search | Spamdexing | Spamdexing |
| FAQ | Carl Linnaeus | Web indexing | Meta element |
| Aloe | Spamdexing | Web crawler | Web indexing |
| Jennifer Lopez | Web crawler | Meta element | Web crawler |
| Joke | Acacia sensu lato | Search engine optimization | Search engine optimization |
| Agapanthus africanus | Web indexing | PageRank | Link farm |
| List of artificial intelligence projects | Search engine optimization | Carl Linnaeus | PageRank |
| Meta element | Meta element | List of algorithms | Search engine (computing) |
| Anna Kournikova | PageRank | Link farm | Earley parser |
| Google Search | List of algorithms | Search engine (computing) | DNSBL |

Now we focus on the small-size web structure composed of Wikipedia pages if we start from ”PageRank” and explore 4 levels with the maximum of 10 pages per level, table 4 shows 10 wikipedia pages with top PageRank values, given different value of $\alpha$. A few things change with the $\alpha$ parameter, it is notable that the web structure is attached with more importance with larger $\alpha$ (since the top 10 wikipedia pages are more relevant to ”PageRank”). When $\alpha = 0.1$, random surfer behavior matters more and the gaps in PageRank score are small (the Wikipedia page with the largest PageRank score is 0.000959088, while the $10^{th}$ is 0.000728698), therefore, the majority of the top 10 Wikipedia pages are irrelavant to ”PageRank”. However, when $\alpha = 0.99$, nearly all the top 10 Wikipedia pages are closely related to the topic ”PageRank”.

# 5 Conclusion

PageRank vector is defined as the stationary state of the Markov chain by perturbating the transition matrix $G$ with a uniform teleportation parameter $\alpha$, where $\alpha \in (0, 1)$ represents the proportion of time that a random surfer follows the hyperlinks rather than teleports to an arbitrary new webpage and $\alpha$ is commonly chosen to be $0.85$. The Google matrix $G$ is primitive, irreducible and stochastic, therefore, the unique stationary probability distribution vector $\pi^T$ exists.

$$\pi^T = \pi^T G,$$

It can be viewed as an eigenvalue problem: PageRank vector $\pi^T$ is the dominant eigenvector of $G$ with eigenvalue $1$ (algebraic multiplicity $1$).

The first emphasis of this report is the comparison of numerical methods in computing the PageRank vector $\pi^T$. Although the size of Google matrix $G$ is extremely large ($8.1$ billion in $2006$), the web structure invloved in all the numerical experiments in this report is composed of $2657$ Wikipedia pages and $5920$ hyperlinks (starting from the Wikipedia page "PageRank" and the hyperlinks within $4$ levels are crawled). The power method is the most efficient method if compared in computing time, while the power method and Gauss-Seidel method have similar rates of convergence, sightly better than Jacobi's method. The SOR method is convergent for the PageRank problem if the relaxation parameter $\omega \in (0, \frac{2}{1+\alpha})$, and it is deduced that the optimal $\omega$ is approximate $1$ and it is also confirmed in the experiment. Furthermore, two Krylov methods (GMRES and BICGSTAB) are equivalently efficient in computing $\pi^T$, however, they are infeasible in the real web structure because of storage issues.

The sensitivity of teleportation parameter $\alpha$ is also researched with analysis of derivative of $\pi^T$ and perturbation method. The explicit formulae of the derivative of $\pi^T$ are given, and the bounds of both derivative of $\pi^T$ in $1-$norm and each component are derived. Furthermore, the exact formula and approximation of higher derivatives of $\pi^T$ are also presented. It is obvious that as $\alpha$ approaching $1$, the PageRank vector tends to be more sensitive to small perturbation in the web structure. When comparing the top $10$ Wikipedia pages with highest PageRank scores as $\alpha$ varies, larger $\alpha$ leads to Wikipedia pages more relevant to the topic PageRank, and the differences in PageRank scores are more significant.

Future work could focus on involving the web structure of larger size to better simulate the world wide web and improving the computing efficiency in programming. Moreover, the relationship between PageRank and its derivative could be further researched, and also their roles in detecting link spams.

## References

[1] K. Ahuja, P. Benner, E. De Sturler, and L. Feng, *Recycling bicgstab with an application to parametric model order reduction*, SIAM Journal on Scientific Computing, 37 (2015), pp. S429–S446.

[2] P. Berkhin, *A survey on pagerank computing*, Internet Mathematics, 2 (2005), pp. 73–120.

[3] D. S. Bernstein, *Matrix mathematics: Theory, facts, and formulas with application to linear systems theory*, vol. 41, Princeton university press Princeton, 2005.

[4] P. Boldi, M. Santini, and S. Vigna, *Pagerank as a function of the damping factor*, in Proceedings of the 14th international conference on World Wide Web, ACM, 2005, pp. 557–566.

[5] R. L. Burden and J. D. Faires, *Numerical analysis*, Cengage Learning, 9 (2010).

[6] J. Cho, H. Garcia-Molina, and L. Page, *Efficient crawling through url ordering*, Computer Networks and ISDN Systems, 30 (1998), pp. 161–172.

[7] S. P. Frankel, *Convergence rates of iterative treatments of partial differential equations*, Mathematical Tables and Other Aids to Computation, 4 (1950), pp. 65–75.

[8] D. Gibson, J. Kleinberg, and P. Raghavan, *Inferring web communities from link topology*, in Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems: links, objects, time and space—structure in hypermedia systems, ACM, 1998, pp. 225–234.

[9] D. F. Gleich, *Pagerank beyond the web*, SIAM Review, 57 (2015), pp. 321–363.

[10] D. F. Gleich and M. Saunders, *Models and algorithms for pagerank sensitivity*, Stanford University Stanford, CA, 2009.

[11] C. Greif and D. Kurokawa, *A note on the convergence of sor for the pagerank problem*, SIAM Journal on Scientific Computing, 33 (2011), pp. 3201–3209.

[12] M. H. Gutknecht, *Variants of bicgstab for matrices with complex spectrum*, SIAM journal on scientific computing, 14 (1993), pp. 1020–1033.

[13] A. Hadjidimos, *Successive overrelaxation (sor) and related methods*, Journal of Computational and Applied Mathematics, 123 (2000), pp. 177–199.

[14] T. Haveliwala, *Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search*, IEEE transactions on knowledge and data engineering, 15 (2003), pp. 784–796.

[15] T. Haveliwala and S. Kamvar, *The second eigenvalue of the google matrix*, tech. rep., Stanford, 2003.

[16] N. J. HIGHAM AND F. TISSEUR, *Lecture note for math46101/math66101 numerical linear algebra: Iterative method*, 2018.

[17] C. R. JOHNSON, *Row stochastic matrices similar to doubly stochastic matrices*, Linear and Multilinear Algebra, 10 (1981), pp. 113–130.

[18] A. N. LANGVILLE AND C. D. MEYER, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006.

[19] C. D. MEYER, *Matrix analysis and applied linear algebra*, vol. 71, Siam, 2000.

[20] J. L. MORRISON, R. BREITLING, D. J. HIGHAM, AND D. R. GILBERT, *Generank: using search engine technology for the analysis of microarray experiments*, BMC bioinformatics, 6 (2005), p. 233.

[21] A. Y. NG, A. X. ZHENG, AND M. I. JORDAN, *Link analysis, eigenvectors and stability*, in International Joint Conference on Artificial Intelligence, vol. 17, LAWRENCE ERLBAUM ASSOCIATES LTD, 2001, pp. 903–910.

[22] L. PAGE, S. BRIN, R. MOTWANI, AND T. WINOGRAD, *The pagerank citation ranking: Bringing order to the web.*, tech. rep., Stanford InfoLab, 1999.

[23] M. PETKOVIĆ AND N. KJURKCHIEV, *A note on the convergence of the weierstrass sor method for polynomial roots*, Journal of computational and applied mathematics, 80 (1997), pp. 163–168.

[24] G. RUDOLPH, *Convergence analysis of canonical genetic algorithms*, IEEE transactions on neural networks, 5 (1994), pp. 96–101.

[25] Y. SAAD AND M. H. SCHULTZ, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing, 7 (1986), pp. 856–869.

[26] H. SAYYADI AND L. GETOOR, *Futurerank: Ranking scientific articles by predicting their future pagerank*, in Proceedings of the 2009 SIAM International Conference on Data Mining, SIAM, 2009, pp. 533–544.

[27] W. J. STEWART, *Introduction to the numerical solution of Markov chains*, Princeton University Press, 1994.

[28] F. TISSEUR, *Lecture note for math36001: Norm; perron-frobenius theory*, 2018.

[29] H. F. WALKER, *Implementation of the gmres method using householder transformations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 152–163.

[30] G. WU, Y.-C. WANG, AND X.-Q. JIN, *A preconditioned and shifted gmres algorithm for the pagerank problem with multiple damping factors*, SIAM Journal on Scientific Computing, 34 (2012), pp. A2558–A2575.

[31] D. YOUNG, *Iterative methods for solving partial difference equations of elliptic type*, Transactions of the American Mathematical Society, 76 (1954), pp. 92–111.

## 6 Appendix

The following is the basic simulation of PageRank Algorithm in Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 21 16:33:06 2018

@author: yifan yu
"""

import numpy as np
from numpy import linalg as LA
import scipy as sci
from scipy import sparse
import sympy as sp
import random
import scipy.sparse.linalg as spla
import matplotlib.pyplot as plt
import math

class NotSquareMatrixError(Exception):
    pass
class Index_Error(Exception):
    pass

def random_hyperlink_matrix(m):
    """
    input integer 'm', return row-normalized random square matrix of
                                    size 'm'
    (since the intelligent surfer may be more likely to jump to content
                                    -filled pages)
    """
    # create a square matrix of size 'm'
    random_matrix = np.random.rand(m, m)
    # normalizing rows of 'random_matrix'
    row_sums = random_matrix.sum(axis=1)
    new_matrix = random_matrix / row_sums[:, np.newaxis]
    return new_matrix

def random_lst(num):
    """
    input integer'num'
    return a list of length 'num' with random integer element between 0
                                     and 'num'-1
    """
    end = num - 1
    lst = []
    for j in range(num):
        lst.append(random.randint(0,end))
    return lst
```

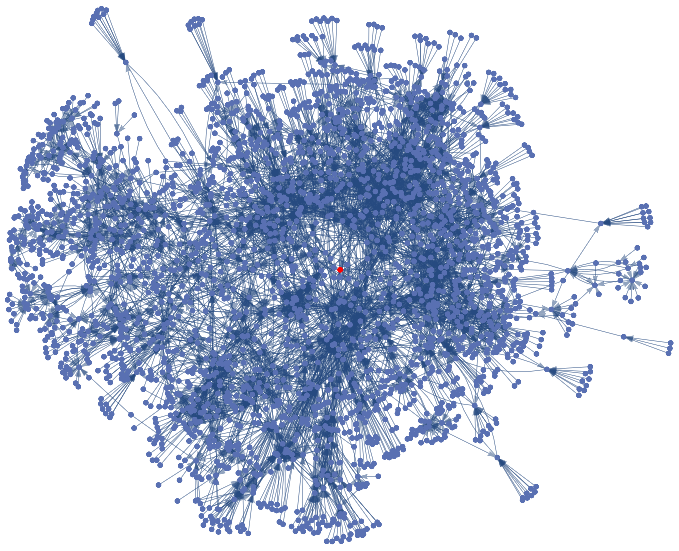
```
def same_index(I,J):
    """
    This function is used to exam if 'I' and 'J' indicate same element
                                    more than once
    input two lists 'I' and 'J'
    return a boolian value
    """
    I_index = list()
    for i in range(len(I)):
        I_index.append((I[i-1],J[i-1]))
    if len(I_index) == len(set(I_index)):
        return True
    else:
        return False



def hyperlink_matrix(I,J):
    """
    input lists 'I' and 'J' to specify the nonzero element
    (the xth elements i in 'I'and j in 'J' means there is a hyperlink
                                    from i to j)
    return the row normalized hyperlink matrix
    (all outlinks from a page are given equal weight in terms of the
                                    random surfer's hyperlinking
                                    probabilities)
    """
    if len(I) != len(J):
        raise Index_Error("The nonzero elements' indices are not one-to
                                    -one correspond")
    else:
        l = len(I)
        # elements equal to 1 if there is a hyperlink
        V = [1] * l
        B = sparse.coo_matrix((V,(I,J)),shape=(l,l)).todense()
        # delete the first row and first column(all zeros)of the
                                    hyperlink matrix
        B = np.delete(B,0,0)
        B = np.delete(B,0,1)
        # delete the last few zero rows and zero columns of the
                                    hyperlink matrix
        B = np.delete(B,np.s_[max(I+J):l],0)
        B = np.delete(B,np.s_[max(I+J):l],1)
        print(B)
        row_sum = np.sum(B,axis=1).tolist()
        # remove the list symbol in every element of 'row_sum'
        row_sum_lst = [x[0] for x in row_sum]
        # create a diagonal matrix with entries 1/cj(the links on jth
                                    page) if cj!=0
        # and 0 otherwise
        row_sum_inverse = [1/x if x!=0 else 0 for x in row_sum_lst]
        row_sum_inverse_array = np.array(row_sum_inverse)
        Diag_matrix = np.diag(row_sum_inverse_array)

    return np.matmul(Diag_matrix, B)


def matrix_size(x):
    """
    input a square matrix 'x'
    return its size
    """
    n = np.shape(x)
    # raise an error if not a square matrix
    if n[0] != n[1]:
        raise NotSquareMatrixError("The matrix is not a square atrix")
    else:
        return n[0]



def zero_rows(x):
    """
    input a matrix 'x'
    return the indices of its zero-rows
    """
    return list(np.where(~x.any(axis=1))[0])


def normalized_hyperlink(H):
    """
    input the original hyperlink matrix of wikipedia webpages
    return the row normalized form
    """
    row_sums = H.sum(axis=1)
    for i in range(len(row_sums)):
        if row_sums[i] != 0:
            H[i:] = H[i:]/row_sums[i]
    return H

def adjust_matrix(x):
    """
    add dangling node vector to original matrix
    The random surfer can hyperlink to any page at random, after
                                    entering a dangling node)
    """
    n = matrix_size(x)
    # entries in 'dangling_node' is 1 if the page is a dangling node,
                                    and 0 otherwise
    dangling_node = np.zeros((n,1))
    for i in zero_rows(x):
        dangling_node[i][0] = 1
    # generate 1xn row vector with all entries as 1/n
    row_vector = 1/n * np.ones((1,n),dtype=int)
    # adjust the matrix to be H+a(1/n*e)
    S = x + np.matmul(dangling_node,row_vector)
    return S

def Google_matrix(x,alpha):
    """
    input a matrix 'x', and parameter 'alpha' between 0 and 1
    return the corresponding Google Matrix
    """
    n = matrix_size(x)
    # teleportation matrix 'E' is uniform (the surfer is equally likely
                                     to jump
    # to any page when teleporting
    E = 1/n * np.ones((n,n),dtype=int)
    # G = alpha*S+(1-alpha)*1/n*e*eT
    G = alpha * adjust_matrix(x) + (1-alpha) * E
    return G

def PageRank_vector_iteration(x,alpha,k):
    """
    input a row normalized hyperlink matrix 'x' and number of
                                    itearation 'k'
    retun the corresponding PageRank vector
    """
    n = matrix_size(x)
    PR_vector = 1/n * np.ones((n,1),dtype=int)
    # transpose the PageRank vector
    PR_transpose = PR_vector.transpose()
    for i in range(k):
        New_PR_transpose = np.matmul(PR_transpose,Google_matrix(x,alpha
                                    ))
        PR_transpose = New_PR_transpose
    return PR_transpose



def PageRank_vector(x,alpha,e):
    """
    input a row normalized hyperlink matrix 'x' and tolerance of
                                    convergence 'e'
    retun the corresponding PageRank vector
    """
    n = matrix_size(x)
    PR_vector = 1/n * np.ones((n,1),dtype=int)
    # transpose the PageRank vector
    PR_transpose = PR_vector.transpose()
    # difference is originally between 'PR_transpose' and zero-vector
    difference = PR_transpose
    counter = 0
    # stop the iteration until the power method has converged to a
                                    tolerance of 10^(-10)
```

```python
    while LA.norm(difference) >= e:
        New_PR_transpose = np.matmul(PR_transpose,Google_matrix(x,alpha
                                            ))
        difference = New_PR_transpose - PR_transpose
        PR_transpose = New_PR_transpose
        counter += 1
    print(counter)
    return PR_transpose



def PageRank_vector_alpha(x,k):
    """
    input the a row normalized hyperlink matrix 'x' and the number of
                                        iterations 'k'
    return the corresponding PageRank vector with symbolic parameter
                                        alpha
    """
    n = matrix_size(x)
    # create the google matrix with symbolic parameter alpha reserved
    E = 1/n * np.ones((n,n),dtype=int)
    # create symbolic parameter alpha as 'a'
    a = sp.symbols('a')
    G = a * adjust_matrix(x) + (1-a) * E
    PR_transpose = 1/n * np.ones((1,n),dtype=int)
    # calculate the matrix multiplication with object arrays for 'k'
                                        times
    for i in range(k):
        PR_transpose = np.dot(np.array(PR_transpose,object),np.array(G,
                                            object))

    # expand the multiplication of polynomials
    return np.array([sp.expand(x) for x in PR_transpose[0]])


def PageRank_vector_derivative(x,k,i):
    """
    input a PageRank vector with symbolic parameter 'a'(after 'k'th
                                        iteration)
    return its 'i'th order derivative of every component
    """
    y = PageRank_vector_alpha(x,k)
    a = sp.symbols('a')
    return [sp.diff(x,a,i) for x in y]





def rank(x,alpha,e):
    """
    input the PageRank vector, return the rank for each entries
    """
    array = PageRank_vector(x,alpha,e)
    order = array.argsort()
    ranks = order.argsort()
    return ranks




def power_pagerank(H,alpha,iteration):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha' and
                                    iteration times
    return the error list of PageRank vector calculated by the power
                                    method.
    pi^(k+1)T = alpha pi^{(k)T} + (alpha pi^{(k)T}a +(1-alpha))1/n e^T
    """
    n = matrix_size(H)
    PR_vector = 1/n * np.ones((n,1),dtype=int)
    # transpose the PageRank vector
    PR_transpose = PR_vector.transpose()
    dangling_node = np.zeros((n,1))
    for i in zero_rows(H):
        dangling_node[i][0] = 1
    v = 1/n * np.ones((1,n),dtype=int)
    error_lst = list()
    for i in range(iteration):
        part = alpha * np.matmul(PR_transpose,dangling_node)+(1-alpha)
        New_PR_transpose = alpha * np.matmul(PR_transpose,H)+ part*v
        difference = New_PR_transpose - PR_transpose
        error = LA.norm(difference)
        error_lst.append(error)
        PR_transpose = New_PR_transpose
    return error_lst

def Jacobi(A,b,iteration):
    """
    input matrix 'A', vector 'b' and iteration times
    return the list of error of jacobi's method in each iteration
    """
    n = matrix_size(A)
    x = 1/n * np.ones((n,1),dtype=int)
    x_new = np.zeros((n,1))
    error_lst = list()
    for k in range(iteration):

        for i in range(n):
            sigma = 0
            for j in range(n):
                if j != i:
                    sigma = sigma + float(A[i][j]) * float(x[j][0])
            x_new[i] = (b[i][0] - sigma)/A[i][i]
        difference = x_new - x
        error = LA.norm(difference)
        error_lst.append(error)
        x = np.copy(x_new)


    return error_lst


def Jacobi_PageRank(H,alpha,iteration):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha' and
                                    iteration times
    return the error list of PageRank vector calculated by Jacobi
                                    Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    return Jacobi(A,b,iteration)


def Gauss_Seidel(A,b,iteration):
    """
    input matrix 'A', vector 'b' and iteration times
    return the list of error of Gauss-Seidel's method in each iteration
    """
    n = matrix_size(A)
    x = 1/n * np.ones((n,1),dtype=int)
    x_new = np.zeros((n,1))
    error_lst = list()
    for k in range(iteration):
        for i in range(n):
            sigma = 0
            for j in range(0,i):
                sigma = sigma + float(A[i][j]) * float(x_new[j][0])
            for j in range(i+1,n):
                sigma = sigma + float(A[i][j]) * float(x[j][0])
            x_new[i] = (b[i][0] - sigma)/A[i][i]


        difference = x_new - x
        error = LA.norm(difference)
        error_lst.append(error)
        x = np.copy(x_new)

    return error_lst
```

```
def Gauss_Seidel_PageRank(H,alpha,iteration):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha' and
                                    iteration times
    return the error list of PageRank vector calculated by Gauss-Seidel
                                     Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    return Gauss_Seidel(A,b,iteration)


def SOR(A,b,omega,iteration):
    """
    input matrix 'A', vector 'b', relaxation parameter 'omega' and
                                    iteration times
    return the list of error of Successive over-relaxation (SOR) method
                                     in each iteration
    """
    n = matrix_size(A)
    x = 1/n * np.ones((n,1),dtype=int)
    x_new = np.zeros((n,1))
    error_lst = list()
    for k in range(iteration):
        for i in range(n):
            sigma = 0
            for j in range(0,i):
                sigma = sigma + float(A[i][j]) * float(x_new[j][0])
            for j in range(i+1,n):
                sigma = sigma + float(A[i][j]) * float(x[j][0])
            x_new[i] = (1-omega)* x[i] + omega * (b[i][0] - sigma)/A[i]
                                          [i]

        difference = x_new - x
        error = LA.norm(difference)
        error_lst.append(error)
        x = np.copy(x_new)

    return error_lst



def SOR_PageRank(H,alpha,omega,iteration):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha'
    relaxation parameter 'omega' and iteration times
    return the error list of PageRank vector calculated by SOR Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    return SOR(A,b,omega,iteration)

def SOR_2(A,b,omega,e):
    """
    input matrix 'A', vector 'b',
    relaxation parameter 'omega' and error 'e'
    return iteration times for Successive over-relaxation (SOR) method
    converges within required error
    """
    n = matrix_size(A)
    x = 1/n * np.ones((n,1),dtype=int)
    x_new = np.zeros((n,1))
    error = 1
    count = 0
    while error >= e:
        for i in range(n):
            sigma = 0
            for j in range(0,i):
                sigma = sigma + float(A[i][j]) * float(x_new[j][0])
            for j in range(i+1,n):
                sigma = sigma + float(A[i][j]) * float(x[j][0])
            x_new[i] = (1-omega)* x[i] + omega * (b[i][0] - sigma)/A[i]
                                                    [i]

        difference = x_new - x
        error = LA.norm(difference)
        count += 1
        x = np.copy(x_new)

    return count

def SOR_2_PageRank(H,alpha,omega,e):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha'
    relaxation parameter 'omega' and error 'e'
    return the iteration times of PageRank vector calculated by SOR
                                            Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    return SOR_2(A,b,omega,e)




def error_plot(H,alpha,iteration):
    """
    input Hyperlink matrix 'H',teleportation parameter 'alpha' and
                                    iteration times
    return the plot of error of Power Method, Jacobi Method and Gauss-
                                    Seidel with respect to
                                    iteration times
    """
    error_lst_1 = power_pagerank(H,alpha,iteration)
    error_lst_2 = Gauss_Seidel_PageRank(H,alpha,iteration)
    error_lst_3 = Jacobi_PageRank(H,alpha,iteration)
    print(error_lst_1, error_lst_2,error_lst_3)
    plt.axis([0, iteration, 0, 10**(-17)])
    t = np.arange(0,iteration,1)
    plt.xlabel('Number of iterations')
    plt.ylabel('Error between consecutive iterations')
    plt.title('Comparison of iterative methods')
    power = plt.plot(t,error_lst_1,'r--',label= 'Power Method')
    Gauss = plt.plot(t,error_lst_2,'k',label = 'Jacobi Method')
    Jacobi = plt.plot(t,error_lst_3,'g^',label = 'Gauss-Seidel Method')
    plt.legend()
    plt.show()
    return


def GMRES_PageRank(H,alpha,tol):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha' and
                                    iteration times
    return the error list of PageRank vector calculated by GMRES Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    x0 = 1/n * np.ones((n,1))
    return spla.gmres(A,b,x0,tol)



def BICGSTAB_PageRank(H,alpha,tol):
    """
    input hyperlink matrix 'H', teleportation parameter 'alpha' and
                                    iteration times
    return the error list of PageRank vector calculated by BICGSTAB
                                    Mathod
    (I-alpha*S)pi^{T} = (1-alpha) * 1/n * e^T
    """
    n = matrix_size(H)
    I = np.identity(n)
    S = adjust_matrix(H)
    A = I - alpha * S.transpose()
    b = (1-alpha)/n * np.ones((n,1),dtype=int)
    x0 = 1/n * np.ones((n,1))
    return spla.bicgstab(A,b,x0,tol)



def main():
    H = mmread("/Users/zifeiyu/desktop/mat.mtx").toarray()
    normalized_H = normalized_hyperlink(H)
    #print(np.shape(normalized_H))
    adjust_H = adjust_matrix(normalized_H)
    G = Google_matrix(normalized_H,alpha = 0.85)
    plt.axis([1, 2700, 1, 2700])
    t = np.arange(1,2658,1)
    for i in range(20):
        pi = rank(normalized_H,alpha=(1+i)/20,e=10**(-10)).tolist()[0]
        convergence_times = plt.plot(t,pi,'r--',label= 'number of
                                    iterations')
    plt.show()
```

```
In[∘]:= links =
          WikipediaData["PageRank", "BacklinksRules", "MaxLevelItems" → 10, "MaxLevel" → 4];
        g = Graph[links, VertexLabels → Placed["Name", Tooltip],
          VertexStyle → {"PageRank" → Red}]
```

Out[∘]=



```
In[∘]:= EdgeCount[g]

Out[∘]= 5920

In[∘]:= VertexCount[g]

Out[∘]= 2657

        Column[Part[VertexList[g], Ordering[PageRankCentrality[g, 0.99], All, Greater]]]


In[∘]:= hyperlinkmat = AdjacencyMatrix[links] // MatrixForm

Out[∘]//MatrixForm=
        ( … 1 … )
        large output   show less   show more   show all   set size limit...

In[∘]:= Export["/Users/zifeiyu/desktop/mat.mtx", hyperlinkmat, "MTX"]

Out[∘]= /Users/zifeiyu/desktop/mat.mtx
```