Computational Physics (P346) Project Report

Google Page Rank Algorithm

Submitted by

Ashlin V Thomas

3rd year Int. MSc Student

School of Physical Sciences

National Institute of Science Education and Research,
Tehsildar Office, Khurda
Pipli, Near, Jatni, Odisha 752050

Submitted to

Dr. Colin Benjamin

Associate Professor

School of Physical Sciences

National Institute of Science Education and Research,
Tehsildar Office, Khurda
Pipli, Near, Jatni, Odisha 752050

Acknowledgements

I wish to express my profound gratitude to our course instructor Dr. Colin Benjamin and the School of Physical Sciences for providing me with an opportunity to undertake this project. Support and guidance provided by the instructor during the coursework and project was critical towards the successful completion of this project. Also, I would like to extend gratitude to my fellow classmates for the support they offered in the process of carrying out this project.

Abstract

The Google PageRank algorithm is a cornerstone of modern web search engines, providing a mechanism to rank web pages based on their link structure. This project explores the mathematical foundation of the algorithm, modeling the internet as a directed graph and employing concepts such as probability vectors and stochastic matrices. An iterative scheme is utilized to compute the PageRank vector, representing the steady-state distribution of web page ranks. The computational challenges of convergence and the scalability of the algorithm are addressed. A modification, the Random Surfer Algorithm, is also discussed, which simulates a user randomly clicking links and resolves issues like dead ends in the hyperlink matrix. Implementation results using Python demonstrate the evolution and convergence of page ranks. The theoretical underpinnings, including the Perron-Frobenius theorem and eigenvector analysis, are validated through numerical experiments, showcasing the robustness and adaptability of the algorithm in handling complex web networks.

Contents

1 Overview **1**

2 Introduction **1**

3 Mathematical Framework **1**
3.1 Internet as a Directed Graph . . . 2
3.2 Probability Vectors and Stochastic Matrices . . . 2
3.3 Hyperlink Matrix . . . 2
3.4 The Role of Graph Theory in PageRank . . . 3
3.5 Foundational lemmas . . . 3

4 The PageRank Algorithm **5**
4.1 PageRank Vector . . . 5
4.2 Iterative Scheme for PageRank . . . 5
4.3 Alternate Formulation: Direct Eigenvector Computation . . . 6
4.4 Python Implementation of the PageRank Algorithm . . . 7
4.4.1 Iterative Scheme . . . 7
4.4.2 Direct Eigenvector Computation . . . 7
4.5 Operation count analysis . . . 8
4.5.1 Iterative scheme . . . 8
4.5.2 Direct eigenvector computation . . . 9
4.6 Random Surfer Algorithm . . . 9
4.6.1 Algorithm Description . . . 9
4.6.2 Python Implementation . . . 9

5 Solutions to Exercise Problems **10**

6 Conclusion **15**

1 Overview

The PageRank algorithm, developed by Google's co-founders Larry Page and Sergey Brin, revolutionized web search by ranking pages based on their importance. This report provides a comprehensive analysis of the PageRank algorithm, its mathematical foundation, and its implementation challenges. We begin by discussing the problem of ranking web pages and the need for a link-based algorithm. We then delve into the mathematical framework, representing the internet as a directed graph and defining key concepts such as probability vectors and stochastic matrices. The PageRank algorithm's iterative scheme and convergence properties are explored, highlighting the algorithm's efficiency in ranking web pages. We also present an alternate formulation using direct eigenvector computation for faster convergence. The Random Surfer Algorithm, a variant of PageRank, is discussed to address dead-end pages and dangling nodes. Python implementations of the PageRank algorithm and the Random Surfer Algorithm are provided, along with solutions to exercise problems. The report concludes with an analysis of the operation count for both algorithms and their impact on search engine efficiency. By elucidating the theoretical underpinnings and practical applications of the PageRank algorithm, this report offers valuable insights into the algorithm's role in organizing the vast information available on the internet.

2 Introduction

The internet comprises billions of interconnected web pages, making it challenging to determine the most relevant results for a given search query. Traditional keyword-matching methods often fail to capture the importance and context of web pages. To address this, Google developed the PageRank algorithm, which ranks web pages based on their link structure. Below, we outline the key aspects of the problem and the algorithm's approach:

- **The Problem:**
  - The internet's vast size and complexity make it difficult to identify relevant search results.
  - Keyword-matching methods can lead to suboptimal rankings by ignoring web page interconnectivity.
- **How Google Search Works:**
  1. A user submits a search query in natural language.
  2. The *Query Module* translates the input into machine-readable language.
  3. The *Ranking Module* determines the relevance and order of web pages.
- **The PageRank Algorithm:**
  - Developed by Sergey Brin and Larry Page, it evaluates:
    - *Quantity of links*: The number of hyperlinks pointing to a page.
    - *Quality of links*: Links from important pages carry more weight.
  - Models the internet as a *directed graph*, where:
    - **Vertices** represent web pages.
    - **Edges** represent hyperlinks between pages.
- **Challenges in Implementation:**
  - **Dead-end pages:** Pages with no outgoing links disrupt the stochastic nature of the hyperlink matrix.
  - **Convergence:** Iterative computation of page ranks is computationally intensive for large networks.

This report delves into the PageRank algorithm's underlying mathematics, its implementation challenges, and its impact on search engine efficiency. By analyzing the algorithm's structure and performance, we demonstrate its effectiveness in ranking web pages and organizing the vast information available on the internet.

3 Mathematical Framework

To understand and implement the PageRank algorithm, it is essential to mathematically represent the structure of the internet. This section introduces the conceptual tools and challenges involved in modeling the internet for PageRank calculations.

3.1 Internet as a Directed Graph

The internet can be represented as a directed graph $G = (V, E)$, where:

- V is the set of vertices, representing web pages.
- E is the set of directed edges, representing hyperlinks between the pages.

In this representation, a directed edge $(P_j \to P_i)$ indicates that page P_j contains a hyperlink to page P_i. This structure not only captures the interconnected nature of web pages but also provides a mathematical framework for analyzing their relationships.

Graphs can be classified as directed or undirected. For the internet:

- The graph is **directed**, as hyperlinks have directionality (from one page to another).
- The presence of directed edges allows the application of algorithms that exploit these asymmetrical connections, such as PageRank.

3.2 Probability Vectors and Stochastic Matrices

The ranking process of web pages requires tools from linear algebra, particularly probability vectors and stochastic matrices:

- A **Probability Vector** $p = (p_1\ p_2 \cdots p_n)^T \in \mathbb{R}^n$ describes the probability distribution over n pages. Here:

$$p_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} p_i = 1.$$

  This ensures that probabilities are non-negative and normalized.

- A **Stochastic Matrix** S is a square matrix where each column sums to 1. It is defined as:

$$S = [s_{ij}], \quad \text{where} \quad s_{ij} \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} s_{ij} = 1.$$

  Stochastic matrices describe transitions in systems with probabilities, making them ideal for modeling random surfing behavior.

3.3 Hyperlink Matrix

The connectivity of web pages is encoded in a **Hyperlink Matrix** H. For n pages, H is an $n \times n$ matrix, where each entry H_{ij} is defined as:

$$H_{ij} = \begin{cases} \frac{1}{|P_j|}, & \text{if there is a link from } P_j \text{ to } P_i, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $|P_j|$ is the number of outgoing links from page P_j. This formulation ensures that the rows represent the probability of transitioning between pages based on hyperlinks.

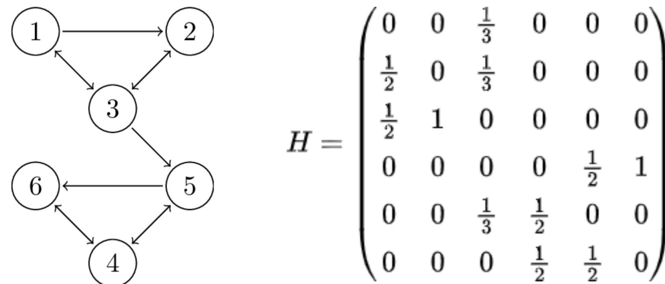

Figure 1: A network of webpages and the corresponding hyperlink matrix.

3.4 The Role of Graph Theory in PageRank

Graph theory plays a pivotal role in PageRank:

- The **in-degree** of a page represents the number of links pointing to it, contributing to its rank.
- The **out-degree** affects the distribution of a page's importance to others.
- Graph traversal techniques are leveraged to simulate user behavior (random surfing).

3.5 Foundational lemmas

The PageRank algorithm is based on the following foundational lemmas:

Lemma 1

The hyperlink matrix corresponding to a a network of webpages, where each page has at least one outlink, is stochastic.

Proof: Let H be an arbitarty hyperlink matrix. Then, for each column j, the sum of the entries is given by:

$$\sum_{i=1}^{n} H_{ij} = \sum_{i=1}^{n} \frac{1}{|P_j|}\gamma_{ij}$$

where

$$\gamma_{ij} = \begin{cases} 1 & \text{if there is a link from } P_j \text{ to } P_i, \\ 0 & \text{otherwise.} \end{cases}$$

Since each page has at least one outlink, $\gamma_{ij} = 1$ for at least one i. Therefore,

$$\sum_{i=1}^{n} H_{ij} = |P_j| \cdot \frac{1}{|P_j|} = 1$$

Since γ_{ij} is non-zero for exactly $|P_j|$ values of i, by the definition of $|P_j|$. Therefore, H is a stochastic matrix. *Hence, the proof.* ■

Lemma 2 (Perron Frobenius Theorem)

If A is a stochastic $n \times n$ matrix, then:

- $\lambda_1 = 1$ is an eigenvalue of A.
- Any other eigenvalue λ_n satisfies : $0 \leq |\lambda_n| < 1$.
- A will have n linearly independent eigenvectors.

Proof

Claim: Let A be a square matrix, then λ is an eigenvalue of A if and only if λ is an eigenvalue of A^T.
Proof of Claim: Let v be an eigenvector of A corresponding to λ. Then,

$$Av = \lambda v \implies v^T A^T = \lambda v^T \implies v^T A^T v = \lambda v^T v \implies (A^T v)^T v = \lambda v^T v$$

Hence, $A^T v = \lambda v$, which implies that λ is an eigenvalue of A^T corresponding to eigenvector v.
The reverse implication follows from the fact that $A = (A^T)^T$. *Hence, the claim.*

Let M be an arbitary $n \times n$ stochastic matrix and let $\mathbf{1} = (1\ 1\ \cdots\ 1)^T \in \mathbb{R}^n$. Then, we can evaluate the following product using the fact that M is stochastic:

$$M^T\mathbf{1} = \begin{pmatrix} \sum_{j=1}^n M_{1j} \\ \sum_{j=1}^n M_{2j} \\ \vdots \\ \sum_{j=1}^n M_{nj} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \mathbf{1} = 1 \cdot \mathbf{1}$$

Hence, 1 is an eigenvalue of M^T, then, it follows from the claim that 1 is an eigenvalue of M.

Let λ_n be an eigenvalue of M. Then, λ_n is an eigenvalue of M^T and let the corresponding eigenvector be $v = (v_1\ v_2\ \cdots v_n)^T$.

$$[M^Tv]_i = M_{1i}v_1 + M_{2i}v_2 + \cdots + M_{ni}v_n = \lambda_n v_i$$
$$\implies |\lambda_n v_i| = |M_{1i}v_1 + M_{2i}v_2 + \cdots + M_{ni}v_n| \leq |M_{1i}v_1| + |M_{2i}v_2| + \cdots + |M_{ni}v_n|$$

Choose $1 \leq k \leq n$ such that $|v_k| = \max_{1 \leq i \leq n} |v_i|$. Then, imposing $i = k$ in the above inequality, we get:

$$|\lambda_n||v_k| \leq M_{1k}|v_1| + M_{2k}|v_2| + \cdots + M_{nk}|v_n|$$
$$\implies |\lambda_n||v_k| \leq M_{1k}|v_k| + M_{2k}|v_k| + \cdots + M_{nk}|v_k| = (M_{1k} + M_{2k} + \cdots + M_{nk})|v_k| = |v_k|$$
$$\implies |\lambda_n| \leq 1$$

Hence, $0 \leq |\lambda_n| \leq 1$.

From the above result, it follows that all non-dominant eigenvalues of M lie within the unit circle in the argand plane. If M is irreducible, which follows from its column stochastic nature, then it follows that all the non-dominant eigenvalues are distinct. Since the eigenspaces corresponding to distinct eigenvalues have a trivial intersection, it follows that we can find n linearly independent eigenvectors, from the basis of each of these eigenspaces. Since, $\mathbb{R}^n$ is n-dimensional, it follows that the eigenvectors of M form a basis for $\mathbb{R}^n$. Therefore, M has n linearly independent eigenvectors.
Hence, the proof. ■

Lemma 3

Let A be an $n \times n$ matrix with n linearly independent eigenvectors $v_1, v_2, \ldots, v_n$ and associated eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Then for any initial vector $x \in \mathbb{R}^n$, we can express A^kx as:

$$A^kx = c_1\lambda_1^k v_1 + c_2\lambda_2^k v_2 + c_3\lambda_3^k v_3 + \cdots + c_n\lambda_n^k v_n$$

where $c_1, c_2, \ldots, c_n$ are constants found by expressing x as a linear combination of the eigenvectors.
Note: We can assume the eigenvalues are ordered such that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$.

Proof: Let x be an arbitrary vector in $\mathbb{R}^n$. Since the eigenvectors $v_1, v_2, \ldots, v_n$ are linearly independent, they form an ordered basis of $\mathbb{R}^n$. Therefore, we can express x as a linear combination of the eigenvectors:

$$x = c_1v_1 + c_2v_2 + c_3v_3 + \cdots + c_nv_n$$

where $c_1, c_2, \ldots, c_n$ are constants.
Now, we shall use the principle of mathematical induction to prove the result.
Base Case: For $k = 1$, we have:

$$Ax = c_1Av_1 + c_2Av_2 + c_3Av_3 + \cdots + c_nAv_n = c_1\lambda_1v_1 + c_2\lambda_2v_2 + c_3\lambda_3v_3 + \cdots + c_n\lambda_nv_n$$

Hence, the result holds for $k = 1$.
Inductive Hypothesis: Assume that the result holds for $k = m$. That is, we have:

$$A^mx = c_1\lambda_1^m v_1 + c_2\lambda_2^m v_2 + c_3\lambda_3^m v_3 + \cdots + c_n\lambda_n^m v_n$$

Inductive Step: We shall prove that the result holds for $k = m + 1$. We have:

$$\begin{aligned} A^{m+1}x &= A(A^m x) \\ &= A(c_1\lambda_1^m v_1 + c_2\lambda_2^m v_2 + c_3\lambda_3^m v_3 + \cdots + c_n\lambda_n^m v_n) \\ &= c_1\lambda_1^{m+1} v_1 + c_2\lambda_2^{m+1} v_2 + c_3\lambda_3^{m+1} v_3 + \cdots + c_n\lambda_n^{m+1} v_n \end{aligned}$$

Hence, the result holds for $k = m + 1$.
Therefore, by the principle of mathematical induction, the result holds for all $k \in \mathbb{N}$.
Hence, the proof. ■

4 The PageRank Algorithm

The Google PageRank algorithm is designed to rank web pages by evaluating their importance through their link structure. This section describes the algorithm in detail, including its iterative scheme, theoretical foundation, and computational considerations.

4.1 PageRank Vector

The PageRank vector x assigns a numerical importance to each web page. Mathematically, the rank of a page P_i can be recursively defined as:

$$r(P_i) = \sum_{Q \in B_i} \frac{r(Q)}{|Q|},$$

where:

- B_i is the set of pages linking to P_i.
- $|Q|$ is the number of outgoing links from page Q.

This formula reflects the idea that a page's importance is derived from the ranks of the pages linking to it, distributed proportionally to their out-degrees.

4.2 Iterative Scheme for PageRank

The core of the PageRank algorithm lies in an iterative computation process to determine the rank of each web page. The steps are as follows:

1. **Initialization:** All page ranks are initialized to an equal value:

   $$x_0 = \frac{1}{N}\mathbf{1},$$

   where N is the total number of web pages, and $\mathbf{1}$ is a vector of ones. This ensures that the initial rank distribution is uniform.

2. **Iteration:** At each step, the page rank vector is updated using the equation:

   $$x_{n+1} = Hx_n,$$

   where H is the hyperlink matrix. Each iteration represents a "random surfer" transitioning between pages based on their hyperlinks.

3. **Convergence:** The iterations continue until the rank vector x reaches a steady state, where:

   $$\|x_{n+1} - x_n\| < \epsilon,$$

   with ϵ being a small tolerance value.

This description of the page rank algorithm arises several questions:

- What is the theoretical foundation for the convergence of the PageRank algorithm?
- Can we optimize the iterative scheme for faster convergence?
- What are the computational challenges in implementing PageRank for large networks?
- How does the algorithm handle dead-end pages?

We will address these questions in the subsequent sections.

4.3 Alternate Formulation: Direct Eigenvector Computation

We begin the discussion of convergence by stating a very important theorem that provides the theoretical foundation for the convergence of the PageRank algorithm.

Theorem 1

If the hyperlink matrix H of certain network of pages is stochastic with $v_1 = (v_{1,1}, v_{1,2}, \cdots, v_{1,n})^T$ being its dominant right eigenvector. Then the iterative scheme, previously defined, converges to -

$$\lim_{k\to\infty} H^k x_0 = \left(\frac{1}{\sum_{i=1}^n v_{1,i}}\right) v_1 \tag{1}$$

Proof: Let H be a stochastic matrix with eigenvalues $\lambda_1 = 1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$ and the corresponding eigenvectors $v_1, v_2, \cdots, v_n$. Since H is stochastic, it follows from Lemma 2 that all the eigenvectors are linearly independent. Therefore, we can express the inital pagerank vector x_0 as a linear combination of the eigenvectors of H:

$$x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

After k iterations, we can express x_k in the following form using Lemma 3:

$$\begin{aligned} x_k = H^k x_0 &= c_1 \lambda_1^k v_1 + c_2 \lambda_2^k v_2 + \cdots + c_n \lambda_n^k v_n \\ \implies x_k &= c_1 v_1 + c_2 \lambda_2^k v_2 + \cdots + c_n \lambda_n^k v_n \end{aligned}$$

Since $|\lambda_j| < 1$ for $j \geq 2$, it follows that $\lim_{k\to\infty} \lambda_j^k = 0$ for $j \geq 2$. Therefore, we have:

$$\lim_{k\to\infty} x_k = c_1 v_1$$

Claim: Let x_k be the pagerank vector after k iterations. Then, the sum of the entries of x_k is 1.
Proof of Claim: We will use the principle of mathematical induction to prove the claim.
Base Case: For $k = 0$, we have the inital pagerank vector $x_0 = \frac{1}{N}(1\ 1\ \cdots\ 1)^T$ and the sum of the entries is $N\frac{1}{N} = 1$.
Inductive Hypothesis: Assume that the sum of the entries of x_m is 1. That is,

$$\sum_{i=1}^n x_{m,i} = 1$$

Inductive Step: We shall prove that the sum of the entries of x_{m+1} is 1. We have:

$$\sum_{i=1}^n x_{m+1,i} = \sum_{i=1}^n (Hx_m)_i = \sum_{i=1}^n \left(\sum_{j=1}^n H_{ij} x_{m,j}\right) = \sum_{j=1}^n \left(\sum_{i=1}^n H_{ij}\right) x_{m,j}$$

Since H is stochastic, it follows that $\sum_{i=1}^n H_{ij} = 1$ for all j. Therefore, we have:

$$\sum_{i=1}^n x_{m+1,i} = \sum_{j=1}^n x_{m,j} = 1$$

Hence, the sum of the entries of x_{m+1} is 1.
Therefore, by the principle of mathematical induction, the sum of the entries of x_k is 1 for all $k \in \mathbb{N}$.

Since the sum of the entries of x_k is 1 for all $k \in \mathbb{N}$ and $\lim_{k\to\infty} x_k = c_1 v_1$, it follows that sum of the entries of $c_1 v_1$ is 1. Therefore, $c_1 = \left(\sum_{i=1}^{n} v_{1,i}\right)^{-1}$. Hence, we have:

$$\lim_{k\to\infty} x_k = \left(\frac{1}{\sum_{i=1}^{n} v_{1,i}}\right) v_1$$

Hence, the proof. ■

The above theorem provides a theoretical foundation for the convergence of the PageRank algorithm. Apart from this, the theorem also provides a direct method to compute the steady-state rank distribution of the network of webpages. This can be acheived by performing Gaussian elimination on the matrix $H - I$ to obtain the dominant eigenvector. This approach is computationally efficient and guarantees convergence to the correct solution.

4.4 Python Implementation of the PageRank Algorithm

4.4.1 Iterative Scheme

The PageRank algorithm can be implemented in Python using the NumPy library. The following code snippet demonstrates the implementation of the iterative scheme:

Program 1: Python Implementation of the iterative scheme for PageRank

```
import numpy as np

def pagerank_iterative(H, max_iter=1000, tol=1e-9):
    N = H.shape[0]
    # Initialize PageRank vector
    x = np.ones(N) / N
    ranks_history = [x.copy()]

    for k in range(max_iter):
        x_new = H @ x  # Update using the hyperlink matrix
        ranks_history.append(x_new.copy())

        # Check for convergence
        if np.linalg.norm(x_new - x, 1) < tol:
            break
        x = x_new

    return x, ranks_history
```

In this code snippet, the function `pagerank_iterative` accepts the hyperlink matrix `H` and two optional arguments `max_iter` and `tol`. The function initializes the PageRank vector `x` as a vector of ones divided by the number of nodes `N`. It then iteratively updates the PageRank vector using the iterative sceheme. The function also keeps track of the history of PageRank vectors at each iteration in the `ranks_history` list. This helps us to study the convergence behavior of the algorithm.

4.4.2 Direct Eigenvector Computation

We had seen that instead of iteratively computing the PageRank vector, we can directly calculate the dominant eigenvector of the hyperlink matrix. This is acheived by performing Gaussian elimination on the matrix $H - I$, followed by back substitution to obtain the solution of the system of equations $(H - I)x = 0$. Since 1 is an

eigenvalue of H, it follows that $H - I$ is singular and the final row in the row-echelon form of $H - I$ will be all zeros. Hence, we impose an additional constraint that the sum of the entries of x is 1 to obtain the dominant eigenvector. This is achieved by fixing the final entry of solution vector to be 1, followed by solving the system of equations and finally dividing the solution vector by the sum of its entries to obtain the dominant eigenvector. The following code snippet demonstrates the implementation of this approach:

Program 2: Python Implementation of the direct eigenvector computation for PageRank

```
import numpy as np

def row_echelon_form(A): #Gaussian Elimination with partial pivoting
    A = A.astype(float)
    rows, cols = A.shape

    for i in range(min(rows, cols)):
        # Find the pivot row using partial pivoting
        max_row = np.argmax(np.abs(A[i:, i])) + i
        A[[i, max_row]] = A[[max_row, i]]

        # Make the pivot element 1
        if A[i, i] == 0:
            continue
        A[i] = A[i] / A[i, i]

        # Eliminate the column entries below the pivot
        for j in range(i + 1, rows):
            A[j] = A[j] - A[j, i] * A[i]

    return A

def pagerank_gauelim(H):
    N = H.shape[0]
    A = H - np.eye(N)
    A = row_echelon_form(A)
    bs = np.zeros(N)
    xs = np.zeros(N)
    xs[-1] = 1
    for i in reversed(range(N-1)):
        xs[i] = (bs[i] - A[i,i+1:]@xs[i+1:])/A[i,i]
        # Backward substitution
    return xs/np.sum(xs)
```

4.5 Operation count analysis

4.5.1 Iterative scheme

The operation count for the iterative scheme can be analyzed as follows:

- Each iteration involves a matrix-vector multiplication, requiring $O(n^2)$ operations.
- Suppose that the algorithm converges in m iterations. Then, the total number of operations is $O(mn^2)$.

For the iterative scheme, we need to consider the number of iterations required for convergence, which can vary based on the network structure and the convergence criteria. So, we cannot provide a precise operation count without knowing a reasonably good upper bound for m.

4.5.2 Direct eigenvector computation

The operation count for the direct eigenvector computation can be analyzed as follows:

- Gaussian elimination requires $O(2n^3/3)$ operations.
- Back substitution requires $O(n^2)$ operations.
- The final division operation requires $O(n)$ operations.
- Hence, the total operation count is $O(n^3)$.

4.6 Random Surfer Algorithm

4.6.1 Algorithm Description

The Random Surfer Algorithm provides a simplified yet effective model for simulating the behavior of a user navigating the web. This approach is designed to address challenges such as dead ends (pages with no outgoing links) and dangling nodes (pages with no incoming links) by introducing randomness into the user's browsing behavior. The key features of the algorithm are as follows:

- **Random Clicks:** It assumes that a user randomly clicks on links available on a web page, navigating to the next page based on the hyperlink structure.
- **Hypothetical Links:** To account for the possibility of dead ends or disconnected pages, hypothetical links are added between all web pages. This modification ensures that the user can jump to any page from any other page, maintaining the continuity of the browsing process.
- **Adjusted Probabilities:** The probability of following a hypothetical link is set to half the probability of following a real link. This adjustment ensures that real hyperlinks remain the dominant factor in determining a page's importance while introducing enough randomness to handle structural anomalies in the hyperlink matrix.

This algorithm enhances the robustness of the PageRank computation by combining deterministic transitions via hyperlinks with probabilistic jumps between pages, providing a realistic and mathematically sound framework for web navigation modeling.

4.6.2 Python Implementation

The following code snippet demonstrates the implementation of the Random Surfer Algorithm in Python:

Program 3: Python Implementation of the Random Surfer Algorithm

```
import numpy as np
def positive_entry_pos(inlist):
  pos = []
  for i in range(len(inlist)):
    if inlist[i]>0:
      pos.append(i)
  return pos
def random_surfer(H, max_iter=1000, tol=1e-9):
  N = H.shape[0]
  random_surfer_H = []
  for i in H.T:
    pos = positive_entry_pos(i)
    k = N + len(pos) - 1
    random_surfer_H.append([2/k if j in pos else 1/k for j in range(N)])
  random_surfer_H = np.array(random_surfer_H).T
  for i in range(N):
    random_surfer_H[i][i] =0
  return pagerank_gauelim(random_surfer_H,max_iter, tol)
```

The function `random_surfer` accepts the hyperlink matrix `H` as input and computes a new hyperlink matrix `random_surfer_H` by adding hypothetical links and adjusting probabilities. The adjusted hyperlink matrix is then used to compute the PageRank vector using the dominant eigenvector method (One can also use the iterative scheme). This implementation provides a comprehensive solution for handling dead ends and dangling nodes in the web graph.

5 Solutions to Exercise Problems

Exercise 4.94

Problem: Prove lemma 3.
Solution: The proof of lemma 3 is provided in the section on foundational lemmas 3.5.

Exercise 4.95

Problem: Construct the hyperlink matrix corresponding to the network of webpages shown in Figure 1.
Solution: The corresponding hyperlink matrix has been added alongwith the network in the same figure.

Exercise 4.96

Problem: Write code to implement the iterative process defined previously. Make a plot that shows how the rank evolves over the iterations.
Solution: The code for the iterative process has been provided in the section on Python implementation of the PageRank algorithm. We make use of it in the following code snippet to plot the evolution of the rank over the iterations:

Program 4: Python code to plot the evolution of the rank over the iterations

```
import numpy as np
import matplotlib.pyplot as plt

def pagerank_iterative(H, max_iter=1000, tol=1e-9):
    N = H.shape[0]
    x = np.ones(N) / N
    ranks_history = [x.copy()]
    for k in range(max_iter):
        x_new = H @ x
        ranks_history.append(x_new.copy())
        if np.linalg.norm(x_new - x, 1) < tol:
            break
        x = x_new
    return x, ranks_history

H = np.array([[0, 0, 1/3, 0, 0, 0],
    [1/2, 0, 1/3, 0, 0, 0],
    [1/2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1/2, 1],
    [0, 0, 1/3, 1/2, 0, 0],
    [0, 0, 0, 1/2, 1/2, 0]])
final_ranks, ranks_history = pagerank_iterative(H)

for i in range(len(ranks_history[0])):
    plt.plot(range(len(ranks_history)), [rank[i] for rank in ranks_history],
             label=f'Page {i+1}')
plt.title('Evolution of PageRank over Iterations')
plt.xlabel('Iteration')
plt.ylabel('PageRank Value')
plt.legend()
plt.grid()
plt.show()
```

One can run this code snippet to obtain the following plot that shows how the rank evolves over the iterations for the network of pages given in Figure 1.

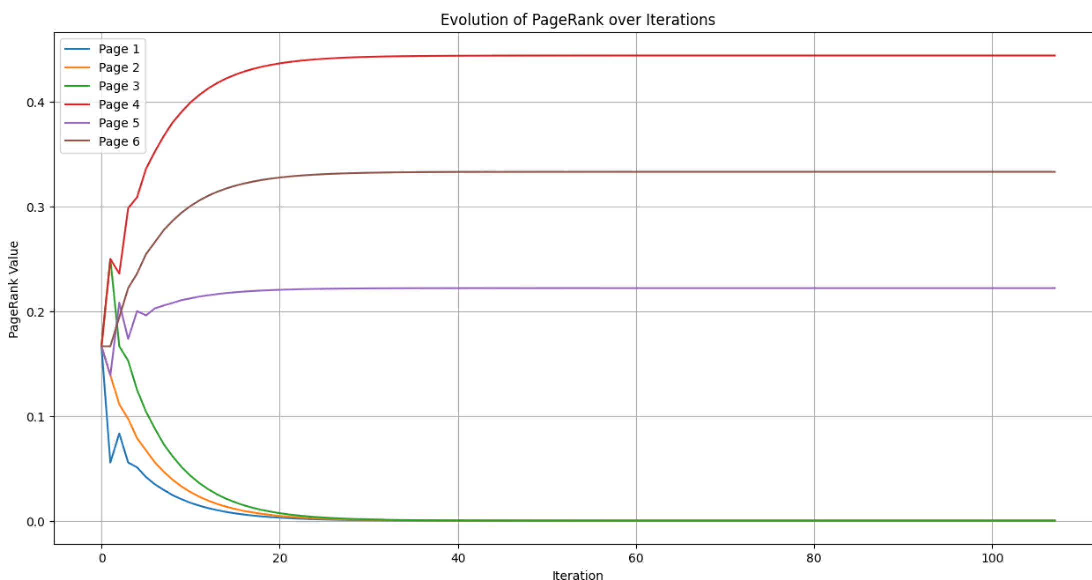

Figure 2: Plot showing the evolution of the rank over the iterations.

Exercise 4.97

Problem: What must be true about a collection of n pages such that an $n \times n$ hyperlink matrix H is a stochastic matrix?
Solution: From lemma 1, we know that the hyperlink matrix corresponding to a network of webpages is stochastic if each page has at least one outlink. A proof of this lemma has been provided in the section on foundational lemmas 3.5.

Exercise 4.98

Problem: Let A be an $n \times n$ stochastic matrix and x_0 is some initial vector for the difference equation $x_{n+1} = Ax_n$, where x_n is the vector at the nth iteration. Find the steady state solution of the difference equation. State arguments in support of your answer.
Solution: The steady state solution of the difference equation $x_{n+1} = Ax_n$ is given by -

$$\lim_{n\to\infty} x_n = \left(\frac{1}{\sum_{i=1}^{n} v_{1,i}}\right) v_1$$

where, v_1 is the dominant eigenvector of the stochastic matrix A. A proof of this result has been provided in the section 4.3 on the direct eigenvector computation for PageRank.

Exercise 4.99

Problem: Discuss how Theorem 1 greatly simplifies the PageRank iterative process. In other words, there is no reason to iterate at all. Instead, what can be found out?
Solution: Theorem 1 provides a direct method to compute the steady-state rank distribution of the network of webpages, by computing the dominant eigenvector of the hyperlink matrix. This approach can be implemented as follows -

1. Perform Gaussian elimination on the matrix $H - I$ to obtain its row-reduced echelon form.
2. Fix the final entry of the solution vector to be 1 and solve the system of equations $(H - I)x = 0$. This is done beacuse the matrix $H - I$ is singular and the final row in the row-echelon form of $H - I$ will be all zeros. This leaves us a degree of freedom to fix the final entry of the solution vector.

3. Perform backward substitution to obtain the solution vector.
4. Divide the solution vector by the sum of its entries to obtain the steady-state rank distribution.

Exercise 4.100

Problem: Find the steady state rank distribution of the network of webpages shown in Figure 1 using the direct eigenvector computation method. Compare the result with the iterative scheme.
Solution: We use the functions `pagerank_iterative` and `pagerank_gauelim`, defined in the section 4.4, for the hyperlink matrix shown in Figure 1 to compute the steady state rank distribution using the iterative scheme and the direct eigenvector computation method respectively. The following code snippet demonstrates this:

Program 5: Python code to compare the iterative scheme and direct eigenvector computation method

```
import numpy as np

H = np.array([[0, 0, 1/3, 0, 0, 0],
    [1/2, 0, 1/3, 0, 0, 0],
    [1/2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1/2, 1],
    [0, 0, 1/3, 1/2, 0, 0],
    [0, 0, 0, 1/2, 1/2, 0]])

final_ranks_iterative, ranks_history = pagerank_iterative(H)
final_ranks_gauelim = pagerank_gauelim(H)

print("Final ranks obtained using the iterative scheme : ")
print(np.round(final_ranks_iterative, 4))
print("Final ranks obtained using the dominant eigenvector method : ")
print(np.round(final_ranks_gauelim, 4))
```

Running this code snippet provides the following output:

```
Final ranks obtained using the iterative scheme :
[0.     0.     0.     0.4444 0.2222 0.3333]
Final ranks obtained using the dominant eigenvector method :
[0.     0.     0.     0.4444 0.2222 0.3333]
```

Hence, we can rank the pages $P_i, 1 \leq i \leq 6$ in the decreasing order of importance as follows -
$P_4 > P_6 > P_5 > P_1 = P_2 = P_3$.
The output shows that the final ranks obtained using the iterative scheme and the direct eigenvector computation method are the same. This is expected as both methods converge to the same solution.

Exercise 4.101

Problem: For the network of pages shown in figure 3,

(a) Write the hyperlink matrix and the initial state x_0 for the iterative scheme.

(b) Find the steady state PageRank using both the iterative scheme and the direct eigenvector computation method.

(c) Rank the pages in the order of importance.

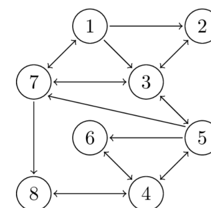

Figure 3: Network of webpages.

Solution:
(a) The hyperlink matrix and the initial state x_0 are as follows:

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 1 & 0 & 1 \\ 0 & 0 & \frac{1}{2} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{4} & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{2} & 0 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \end{pmatrix}, \quad x_0 = \begin{pmatrix} 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \end{pmatrix}$$

(b) The steady state PageRank obtained using both the iterative scheme and the direct eigenvector computation method is as follows:

```
Final ranks obtained using the iterative scheme :
[0.03169 0.01056 0.09155 0.32746 0.15493 0.14789 0.09507 0.14085]
Final ranks obtained using the dominant eigenvector method :
[0.03169 0.01056 0.09155 0.32746 0.15493 0.14789 0.09507 0.14085]
```

One can arrive at the above output by running the code snippet provided in the solution to exercise 4.100 with the hyperlink matrix in this exercise.

(c) The pages can be ranked in the decreasing order of importance as follows -
$P_4 > P_6 > P_5 > P_3 > P_8 > P_7 > P_2 > P_1$.

Exercise 4.102

Problem: Implement the random surfer algorithm for the network of webpages shown in Figure 3. Compare your ranking to the non-random surfer results from the previous problem.
Solution: The following code snippet demonstrates the implementation of the Random Surfer Algorithm for the network of webpages shown in Figure 3:

Program 6: Python code to implement the Random Surfer Algorithm

```
import numpy as np

H = np.array([
    [0,0,0,0,0,0,1/3,0],
    [1/3,0,0,0,0,0,0,0],
    [1/3,1,0,0,1/4,0,1/3,0],
    [0,0,0,0,1/4,1,0,1],
    [0,0,1/2,1/3,0,0,0,0],
    [0,0,0,1/3,1/4,0,0,0],
    [1/3,0,1/2,0,1/4,0,0,0],
    [0,0,0,1/3,0,0,1/3,0]
])

final_ranks_iter = random_surfer(H,method= 'iterative')

print("Final PageRank values using random surfer algorithm
      implemented using iterative scheme:")
print(np.round(final_ranks_iter, 5))

final_ranks_gauelim = random_surfer(H,method= 'gauelim')

print("Final PageRank values using random surfer algorithm
      implemented using dominant eigenvector method:")
print(np.round(final_ranks_gauelim, 5))
```

Running this code snippet, making use of the previously defined functions `pagerank_iterative`, `pagerank_gauelim` and `random_surfer`, provides the following output:

```
Final PageRank values using random surfer algorithm implemented using iterative scheme:
[0.1114  0.10689 0.14274 0.13704 0.12712 0.11945 0.13425 0.12111]
Final PageRank values using random surfer algorithm implemented using dominant eigenvector method:
[0.1114  0.10689 0.14274 0.13704 0.12712 0.11945 0.13425 0.12111]
```

Based on this output, we can rank the pages in the decreasing order of importance as follows -
$P_3 > P_4 > P_7 > P_5 > P_8 > P_6 > P_2 > P_1$.
We can find that the ranking of pages have changed quite significantly when compared to the non-random surfer results from the previous problem. Some of the observations are -

- We find that the ranks of all pages lie between 0.1 and 0.15 in the random surfer algorithm, whereas the ranks obtained using the non-random surfer algorithm lie between 0.03 and 0.33. This indicates that the random surfer algorithm provides a more uniform distribution of ranks across the pages since it take cares of dead ends and dangling nodes.
- The ranks of pages P_3 and P_7 increase significantly in the random surfer algorithm compared to the non-random surfer algorithm. Although P_3 and P_7 have a substantial number of incoming links, a surfer can jump from P_3 to P_5 but cannot return to P_3 or P_7. Consequently, the surfer becomes confined to pages P_4, P_5, P_6, and P_8, leading to an increase in the ranks of these pages in the non-random surfer algorithm. The random surfer algorithm addresses this issue by introducing hypothetical links between all pages, thereby significantly boosting the ranks of P_3 and P_7.
- Due to the same reason as above, we find a decrease in the ranks of pages P_6 and P_5 in the random surfer algorithm compared to the non-random surfer algorithm.
- The ranks of pages P_1 and P_2 remain relatively unchanged in both the random surfer and non-random surfer algorithms. This is because these pages have a relatively low number of incoming links and are not significantly affected by the random surfer behavior.
- Therefore, the random surfer algorithm provides a more realistic and robust ranking of web pages by incorporating the behavior of a user navigating the web in a probabilistic manner.

Note: Complete Python scripts for each of the exercises, wherever applicable, can be accessed from the GitHub repository: https://github.com/Ashlin-V-Thomas/Google-PageRank-Algorithm. If you wish to run the scripts, you can clone the repository and execute the Python files, ensuring that the required libraries are installed. The code snippets provided in the solutions to the exercises are excerpts from the complete Python scripts. The complete scripts contain additional functionalities to enhance the user experience.

Solutions to the Questions during the Presentation

Question 1

Question: How is the iterative scheme similar or different from the power iteration method?
Answer: The iterative scheme is similar to the power iteration method, as both are used to find the dominant eigenvector of a matrix. But, in the iterative scheme, we neither do normalise the vector after each iteration nor do we need to find the eigenvalue.

Question 2

Question: What if we have a dead end in the network of webpages?
Answer: In the presence of a dead end, the hyperlink matrix will have a column with all zeros. This will make the matrix non-stochastic. In such cases, we can implement the Random Surfer Algorithm, which adds hypothetical links between all pages to ensure that the surfer can navigate to any page from any other page. This approach helps in handling dead ends and dangling nodes in the network.

6 Conclusion

The PageRank algorithm is a powerful tool for ranking web pages based on their importance. By leveraging the principles of linear algebra and graph theory, PageRank provides a robust and scalable solution for evaluating the significance of web content. The theoretical foundation of PageRank, including the convergence properties and computational considerations, offers valuable insights into the algorithm's behavior. By exploring the iterative scheme, direct eigenvector computation, and the Random Surfer Algorithm, we gain a comprehensive understanding of the PageRank methodology and its practical applications. Through the exercises and code implementations, we have demonstrated the effectiveness of PageRank in ranking web pages and handling structural anomalies in web networks. By combining theoretical analysis with practical implementations, we have highlighted the versatility and reliability of the PageRank algorithm in modern web search applications.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[2] Amy N Langville and Carl D Meyer. A survey of eigenvector methods for web information retrieval. *SIAM review*, 47(1):135–161, 2005.

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Stanford InfoLab*, 1999.

[4] Eric Sullivan. The google page rank algorithm. *Numerical Methods : An Inquiry-Based Approach with Python*.