Politecnico di Torino

Master's degree in Data Science and Engineering

Computational Linear Algebra

Homework 1
PageRank

Lisa Vandi 360247
Riccardo Polazzi 361352

Academic Year 2025/2026

Contents

Chapter 1

The Mathematics of PageRank

1.1 Mathematical Formulation

The PageRank algorithm models the World Wide Web as a directed graph $G = (V, E)$, where V represents the set of n web pages and E represents the set of hyperlinks connecting them. The core intuition, derived from the *democracy of the web* concept, is that a page's importance is determined by the importance of the pages linking to it.
Let x_k denote the quantitative importance score of page k. A basic formulation suggests that a page distributes its importance evenly among its outgoing links. If page j has n_j outgoing links, the score transfer to page k is x_j/n_j. The importance score of any page k is defined recursively as the sum of the importance scores received from all pages linking to it (its *backlinks*). Let $L_k \subset \{1, \ldots, n\}$ denote the set of pages with a link to page k. The score x_k is given by:

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j} \tag{1.1}$$

This equation implies that the importance of a page is derived exclusively from the network structure and the importance of its neighbors.

To solve the ranking system efficiently for the entire web, we translate the recursive relationship between page scores into matrix notation. We define the *link matrix* A of size $n \times n$ as follows:

$$A_{ij} = \begin{cases} 1/n_j & \text{if page } j \text{ links to page } i \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

where n_j represents the *out-degree* (the number of outgoing links) of page j. Using this notation, the system of linear equations that defines the importance of each page can be expressed compactly as:

$$A\mathbf{x} = \mathbf{x} \tag{1.3}$$

where $\mathbf{x} = [x_1, \ldots, x_n]^T$ is the **importance vector** containing the scores of all pages.

From a linear algebra perspective, the equation $A\mathbf{x} = \mathbf{x}$ is a specific instance of the general **eigenvalue problem**:

$$A\mathbf{x} = \lambda\mathbf{x} \tag{1.4}$$

In our case, the PageRank solution corresponds to the scenario where the eigenvalue λ is exactly 1. Therefore, calculating the ranking of the web pages is equivalent to finding the **eigenvector x** associated with the eigenvalue $\lambda = 1$.

Assuming the web graph contains no *dangling nodes* (pages with no outgoing links, i.e., $n_j > 0$ for all j), every column of A sums to exactly 1. This characteristic defines A as a **column-stochastic matrix**. A fundamental property of such matrices (as stated in **Proposition 1** in the paper) ensures that $\lambda = 1$ is always an eigenvalue and that at least one corresponding eigenvector exists, thereby guaranteeing a solution to the ranking problem.

1.2 Problems with matrix A

The raw link matrix A poses two challenges that necessitate modifications to guarantee a valid ranking.

1.2.1 Disconnected components

The first critical challenge arises from the topological structure of the Web. The raw link matrix A fails to provide a valid ranking if the graph is not strongly connected.
Indeed, the fundamental requirement for a robust ranking algorithm is **uniqueness**.
Mathematically, the ranking vector $\mathbf{x}$ is unique (subject to the normalization $\sum x_i = 1$) if and only if the dimension of the eigenspace corresponding to the eigenvalue $\lambda = 1$ is exactly 1:

$$\dim(V_1(A)) = 1$$

However, the World Wide Web is not a single connected component. If the Web W consists of r disconnected sub-webs $W_1, \ldots, W_r$, then $\dim(V_1(A)) \geq r$. Hence, there is no unique importance score vector $\mathbf{x} \in V_1(A)$ with $\sum x_i = 1$.
With $\dim(V_1(A)) > 1$, there are infinite valid ranking vectors, making it impossible to determine which one represents the "true" importance.

Solution to uniqueness: the Google Matrix M

To force uniqueness, we must modify the system such that the graph becomes effectively connected. We introduce the **Google Matrix** M, defined as:

$$M = (1 - m)A + mS \tag{1.5}$$

where S is an $n \times n$ matrix with all entries equal to $1/n$, and m is the damping factor (typically 0.15). The term mS acts as a perturbation that equally distributes probability mass to every page in the network, regardless of the link structure.

Physically, this implements the "Random Surfer" model:

- With probability $(1 - m)$, the user follows the existing links (matrix A).
- With probability m, the user "teleports" to a random page chosen uniformly from the entire Web (matrix S).

This artificial teleportation creates a link between every pair of pages. Consequently, disconnected sub-webs cease to exist in the mathematical model; the graph becomes a single, strongly connected component where every node is reachable from every other node.

Mathematical proof of Uniqueness

From a linear algebra perspective, the introduction of the term mS (where $m > 0$) ensures that M is a **positive matrix** ($M_{ij} > 0$ for all i, j). The strict positivity of M, combined with the fact that M is column-stochastic (as it is a sum of column-stochastic matrices), guarantees that $\dim(V_1(M)) = 1$, solving the uniqueness problem.

This conclusion is derived from the interaction of the following properties established in the paper:

1. **Proposition 2**: For a positive, column-stochastic matrix like M, any eigenvector in the subspace $V_1(M)$ must have components of the **same sign** (either all strictly positive or all strictly negative). Vectors with mixed signs are forbidden in this eigenspace.
2. **Proposition 3**: Given two linearly independent vectors, it is always possible to form a linear combination that results in a vector with mixed signs.

Based on these propositions, **Lemma 3.2** demonstrates that $V_1(M)$ cannot contain two linearly independent vectors. If it did, by Proposition 3, we could combine them to form a vector with mixed signs. Since $V_1(M)$ is a vector subspace, this new vector would necessarily belong to $V_1(M)$. However, this is impossible, as **Proposition 2** explicitly forbids vectors with mixed signs in that eigenspace.

Consequently, the dimension of $V_1(M)$ must be exactly 1. This guarantees the existence of a unique eigenvector with positive components summing to 1: the unique PageRank vector.

1.2.2 Dangling Nodes

The second major issue regarding matrix A concerns dangling nodes.
In the context of the PageRank algorithm, **dangling nodes** are defined as web pages that possess no outgoing hyperlinks. In a directed graph representing the Web, these nodes act as dead ends for navigation.

The presence of dangling nodes profoundly alters the structure of the link matrix A: it makes the matrix **column-substochastic**.
Indeed, each dangling node produces a column containing exclusively zeros within the matrix A. This implies that the sum of the elements in certain columns is equal to zero, and therefore less than or equal to one.

The substochastic nature of the raw matrix A introduces several difficulties in the ranking calculation:

1. **Violation of Proposition 1 (loss of eigenvalue 1)**: while for a column-stochastic matrix it is guaranteed that 1 is an eigenvalue, for a substochastic matrix this is no longer guaranteed. Without the eigenvalue 1, it is not possible to identify an eigenvector x such that $Ax = x$ in a standard manner to assign importance scores.

2. **Persistence of substochasticity with M**: even if we attempt to apply the damping factor directly to the raw matrix A using the formula $M = (1-m)A + mS$, the problem persists. For a dangling node (where the column of A is zero), the corresponding column in M sums to:

   $$(1-m) \cdot 0 + m \cdot 1 = m$$

   Since $m < 1$ (typically 0.15), the resulting matrix is not column-stochastic. Consequently, the standard theorems cannot be fully applied to guarantee the existence of a unique, valid ranking vector $\mathbf{x}$ normalized such that $\sum x_i = 1$.

Handling Dangling Nodes

To ensure the matrix A is perfectly column-stochastic, a mathematical transformation is applied to the columns corresponding to dangling nodes. Specifically, for every node j belonging to the set of dangling nodes, the null column is replaced with a uniform probability distribution:

$$\forall j \in \text{Dangling Set}, \quad A_{ij} = \frac{1}{N} \quad \forall i = 1 \ldots N \tag{1.6}$$

This operation ensures that the total probability mass is preserved during iterations, effectively modeling the behavior of a random surfer who, upon reaching a node with no outgoing links, performs a "random jump" to any node in the network with equal probability.

Crucially, the addition of these virtual links to every node in the system **does not alter the ranking hierarchy**. Since the probability mass from a dangling node is distributed perfectly equitably among all N nodes, no individual node receives a competitive advantage or a relative increase in importance compared to others. This approach stabilizes the stochastic matrix and guarantees the existence of a unique stationary distribution while preserving the structural integrity and the relative importance defined by the original set of links.

1.3 The Power Method and Optimization

Given the immense size of the Web, finding the eigenvector with a direct method is computationally infeasible. We employ the **Power Method**, an iterative algorithm that converges to the dominant eigenvector, to solve the eigenvector problem $M\mathbf{x} = \mathbf{x}$.

The iterative step is defined as:

$$\mathbf{x}^{(k)} = M\mathbf{x}^{(k-1)} \tag{1.7}$$

Recursively applying this formula leads to the relation with the initial distribution vector $\mathbf{x}^{(0)}$:

$$\mathbf{x}^{(k)} = M^k\mathbf{x}^{(0)} \tag{1.8}$$

However, constructing the dense matrix M explicitly is computationally prohibitive ($O(n^2)$ memory). To solve this, we exploit the structure of M to compute the product implicitly using the sparsity of A.
The iteration is mathematically rewritten as:

$$\mathbf{x}^{(k)} = (1-m)A\mathbf{x}^{(k-1)} + m\mathbf{s} \tag{1.9}$$

where $\mathbf{s}$ is the vector with all entries $1/n$. This formulation reduces the complexity significantly, as A is sparse.

1.3.1 Convergence Analysis

The convergence of the iterative process $x_k = Mx_{k-1}$ to the steady-state eigenvector q is established through two key propositions.

Proposition 4: Contraction in the Zero-Sum Subspace

Proposition 4 focuses on the subspace $V \subset \mathbb{R}^n$ consisting of vectors v such that $\sum v_j = 0$. It states that for any $v \in V$:

$$||Mv||_1 \leq c||v||_1 \tag{1.10}$$

where $c = \max_{1\leq j\leq n} |1 - 2\min_{1\leq i\leq n} M_{ij}| < 1$. This proves that M acts as a **contraction mapping** on the subspace of vectors whose components sum to zero.

Proposition 5: Uniqueness and Global Convergence

Proposition 5 asserts that every positive column-stochastic matrix M possesses a **unique** vector q with positive components such that $Mq = q$ and $||q||_1 = 1$. The primary significance of this result is that the PageRank vector q can be computed as the limit of the power sequence for any initial probability vector x_0:

$$\lim_{k\to\infty} M^k x_0 = q \tag{1.11}$$

The connection to Proposition 4 is found in how the algorithm handles the **initial error vector** v. Any starting guess x_0 is essentially the sum of the true ranking and this error ($x_0 = q + v$). Since both x_0 and q are normalized to 1, the sum of the components of v must be zero, which identifies it as a member of the **subspace** V defined in Proposition 4.

Because v falls within this specific subspace, the contraction property established in Proposition 4 ensures that the error is diminished with each iteration until it vanishes. This explains why the choice of the initial vector x_0 does not affect the final result: the iterative process naturally "filters out" the error component to reveal the unique steady-state vector q.

Rate of Convergence

Proposition 4 establishes a constant $c = \max_{1\leq j\leq n} |1 - 2\min_{1\leq i\leq n} M_{ij}|$ that represents the **maximum theoretical limit** for error reduction in each iteration. This constant serves as a worst-case bound, guaranteeing that the error norm decreases by at least a factor of c at each step. However, in practice, this bound is often considered "pessimistic".

The actual asymptotic behavior of the algorithm is governed by the spectral properties of the matrix M:

- **The Spectral Gap**: The power method converges asymptotically according to the second largest eigenvalue of M, denoted as λ_2, such that the error at each step follows the relation $||Mx_k - q||_1 \approx |\lambda_2| \cdot ||x_{k-1} - q||_1$.

- **Influence of the Damping Factor**: For Google's modified matrix $M = (1-m)A + mS$, it can be shown that $|\lambda_2| \leq 1 - m$. In a typical web model, while the theoretical bound c might be as high as 0.94, the actual convergence rate dictated by $1 - m$ (using $m = 0.15$) is a significantly more efficient 0.85.

- **The m Trade-off**: The choice of the damping factor m represents a fundamental trade-off: a smaller m (closer to 0) preserves more of the original link structure of the web but increases $|\lambda_2|$, thereby slowing down the computation time.

- **Computational Efficiency**: With the standard implementation of $m = 0.15$, the error is reduced by approximately 15% in every iteration. This rapid decay is what enables Google to process the world's largest matrix computation within a reasonable timeframe.

- **Convergence Criterion**: Iterations continue until the difference between successive rank vectors (the residual) falls below a specified tolerance τ:

$$||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||_1 < \tau \tag{1.12}$$

Chapter 2

Computational Implementation

The implementation of the PageRank algorithm was developed in Python using the `NumPy` and `SciPy` libraries. The code is designed to process the *Hollins* dataset ($N = 6012$ pages) and demonstrates the practical application of the Power Method.
The implementation strategy balances mathematical clarity with computational efficiency, addressing the construction of the link matrix and the resolution of dangling nodes.

2.1 Matrix construction strategy

The structure of the link matrix A is strictly dictated by the topology of the analyzed dataset. The *Hollins* dataset consists of $N = 6012$ nodes and 23875 edges. A defining characteristic of this network is the high prevalence of dangling nodes (3189 pages).

In line with the mathematical framework established in the previous sections, the presence of such dangling nodes creates a dual effect on the matrix A:

- **Theoretical effect (sub-stochasticity)**: since dangling nodes have no outgoing links, they generate columns containing exclusively zeros, making the matrix column-substochastic.
- **Computational effect (sparsity)**: the abundance of zero-columns significantly contributes to the sparsity of the matrix, reducing the number of non-zero entries to orders of magnitude less than N^2.

Our implementation strategy prioritizes the preservation of this sparsity to guarantee execution speed and memory efficiency.
While the theoretical remedy for dangling nodes would require physically filling

these zero-columns with dense uniform vectors ($1/N$), doing so would catastrophically destroy the sparsity benefit. For this dataset, it would transform a lightweight structure into a dense matrix, making the computation inefficient.

Therefore, we chose not to store the $1/N$ probabilities in memory. Instead, we apply the dangling node patch virtually via algebraic decomposition during the calculation step.
To support this efficient approach, we chose to use the **Compressed Sparse Row (CSR)** format: it allows us to store the matrix in its raw, sub-stochastic form, occupying memory only for the existing links ($O(E)$), while mathematically simulating a fully stochastic matrix through code logic.

2.2 Implementation of the Compressed Sparse Row (CSR) Format

To make PageRank calculations computationally feasible for large-scale graphs, we adopted the **Compressed Sparse Row (CSR)** format. This choice is necessitated by the extreme sparsity of web graphs: the link matrix consists almost entirely of zeros, as each page typically links to only a tiny fraction of the total nodes in the network.

2.2.1 Correspondence Between Theory and Code

According to the formal definition, a sparse matrix $A \in \mathbb{R}^{m \times n}$ is encoded using three primary arrays:

- **`data` (or AA)**: Contains the real values $a_{i,j}$ of the non-zero entries, ordered by row. In our script, this array stores the transition probabilities calculated as $1/\text{out\_degree}(src)$.
- **`indices` (or JA)**: Stores the column indices for each corresponding value in the `data` array. In the context of PageRank, these represent the source nodes of the links.
- **`indptr` (or IA)**: An array of length $m + 1$ that recursively defines the starting and ending points for each row's data. Formally, $IA[i + 1] = IA[i] + \text{number of non-zero entries in row } i$.

2.2.2 Matrix Construction: From COO to CSR

Our implementation utilizes the `scipy.sparse` library. The process begins by gathering link data into the **Coordinate (COO)** format. The COO format is an intuitive storage scheme that represents a matrix as a collection of triplets: *(row_index, column_index, value)*.

This format is ideal for the initial data-loading phase because it allows for the incremental building of the matrix as links are read line-by-line from the input file (e.g., *hollins.dat*). However, while COO is efficient for construction, it is not optimized for arithmetic operations. Therefore, once the lists of rows, columns, and values are populated, we convert the structure into a CSR matrix:

```
A_sparse = sparse.csr_matrix((data, (rows, cols)), shape=(N,
N))
```

This conversion transforms the unordered triplets into a compressed, row-aligned structure that is significantly faster for the matrix-vector products required by the Power Method.

2.2.3 Computational and Memory Efficiency

The adoption of the CSR format provides two critical advantages observed during testing on the *Hollins* dataset:

1. **Memory Footprint Reduction**: Storage requirements are reduced from $O(N^2)$ to $O(Nz+m)$, where Nz is the number of non-zero elements. For the *Hollins* dataset ($N = 6012$), a dense matrix would require over 36 million entries. By using CSR, we only store the $Nz = 23,875$ valid links, achieving a memory saving of over 99%.

2. **Matrix-Vector Multiplication Optimization**: The Power Method relies heavily on the repeated calculation of Ax. Since CSR stores non-zero elements in contiguous memory, it allows the algorithm to iterate *only* over existing links while completely ignoring zero entries. This reduces the computational complexity of each iteration from $O(N^2)$ to $O(Nz)$, enabling the algorithm to scale to much larger graphs.

```python
for src, tgt in links:
    src_idx = src - 1
    tgt_idx = tgt - 1

    # If a node appears as a source, it has outgoing links -> Not Dangling
    is_dangling[src_idx] = False

    # Calculate transition probability: 1 / out_degree
    val = 1.0 / out_degree[src]

    rows.append(tgt_idx)
    cols.append(src_idx)
    data.append(val)

# Convert to CSR format for efficient multiplication
A_sparse = sparse.csr_matrix((data, (rows, cols)), shape=(N, N))
```

Listing 2.1: Sparse Matrix Construction Logic

2.3 Handling Dangling Nodes: The Algebraic Decomposition

To reconcile the memory benefits of the sparse matrix ($O(E)$ storage) with the theoretical requirement of a column-stochastic matrix, we employ an **algebraic decomposition strategy** during the iterative process.

In a standard dense implementation, stochasticity is enforced by modifying the matrix structure filling empty columns with $1/N$. In our sparse implementation, we leave the matrix A_{sparse} as column-substochastic (where dangling columns sum to zero).
Consequently, a standard multiplication $A_{sparse}\mathbf{x}$ would result in a leakage of probability mass: the importance associated with dangling nodes would simply disappear from the system.

To correct this loss without destroying sparsity, we compute the missing mass dynamically and reinject it into the system. If D is the set of indices corresponding to dangling nodes, the iterative step (Equation 1.9) is decomposed into three distinct components:

$$x_{new} = (1-m)Ax + (1-m)\left[\frac{\text{dangling\_mass}}{N}\right] + \frac{m}{N} \tag{2.1}$$

$$\mathbf{x}_{new} = \underbrace{(1-m)A_{sparse}\mathbf{x}}_{\text{Flow from Links}} + \underbrace{(1-m)\left(\sum_{j\in D} x_j\right)\frac{\mathbf{1}}{\mathbf{N}}}_{\text{Virtual Patch}} + \underbrace{\frac{m}{N}}_{\text{Teleportation}} \tag{2.2}$$

This decomposition mathematically guarantees that the operator remains effectively column-stochastic, despite the underlying matrix being sub-stochastic. The validity of this approach rests on the conservation of probability mass:

- **Loss detection**: the term $A_{sparse}\mathbf{x}$ propagates probability mass only from nodes with outgoing links. The mass residing in dangling nodes, defined as $\Omega = \sum_{j\in D} x_j$, is effectively lost in this step because the corresponding columns in A_{sparse} are zero.
- **Mass recovery (the Virtual Patch)**: the second term in Equation 2.2 captures this exact lost mass Ω and redistributes it according to the theoretical definition of dangling nodes (a uniform jump to any page).
  By adding the scalar value $(1-m)\Omega/N$ to every node, we mathematically simulate the presence of physical links from every dangling node to every other node, without allocating memory for them.
- **Proof of stochasticity**: the column-stochasticity is not violated because the total probability is conserved.
  Let Σ_{valid} be the total probability mass currently held by non-dangling

nodes. The sum of the elements in the vector resulting from the first term is $(1-m)\Sigma_{valid}$. The sum of the elements from the virtual patch is $(1-m)\Omega$. Since $\Sigma_{valid} + \Omega = 1$ (the total probability of the previous iteration), the total sum of $\mathbf{x}_{new}$ becomes:

$$\sum_N \mathbf{x}_{new} = (1-m)(\Sigma_{valid} + \Omega) + \sum_N \frac{m}{N} = (1-m)(1) + m = 1$$

This proof confirms that our algebraic implementation strictly adheres to the theoretical requirements of the PageRank algorithm (matrix stochasticity) while maintaining the computational efficiency of sparse matrix operations.

2.4 Iterative Power Method

The core calculation is performed by the function `calculate_pagerank`. The algorithm does not construct the dense Google Matrix M explicitly. Instead, it iterates the decomposition formula described above until convergence.

```
# --- POWER METHOD LOOP ---

x = np.full(N, 1.0/N)
teleport_contribution = m / N
iterations = 0

for k in range(max_iter):
    x_prev = x.copy()

    # Step A: Standard Matrix Multiplication
    Ax = A_sparse.dot(x_prev)

    # Step B: Implicit Dangling Node Handling
    # Total mass lost of dangling nodes
    dangling_mass_sum = np.sum(x_prev[dangling_indices])

    # We redistribute this lost mass evenly to ALL nodes (1/N),
    # applied with the damping factor (1-m).
    dangling_correction = (1 - m) * (dangling_mass_sum / N)

    # Step C: Combine Everything
    x = (1 - m) * Ax + dangling_correction +
teleport_contribution

    # Step D: Convergence Check (L1 Norm)
    diff = np.sum(np.abs(x - x_prev))
    iterations = k + 1

    if diff < tol:
        break
```

Listing 2.2: PageRank Calculation using Power Method

Chapter 3

Experimental Results

The implemented algorithm was tested on three distinct topologies. For all tests, we used a damping factor $m = 0.15$ and a tolerance $\tau = 10^{-7}$.

3.1 Test Case 1: Connected Web

The first test involved the 4-page connected graph described in Figure 3.1: it is a strongly connected component where probability mass flows freely between all nodes.

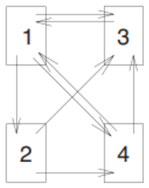


Figure 3.1: Connected graph topology (4 nodes)

```python
A_4pages = np.array([
    [0.0, 0.0, 1.0, 0.5],
    [1/3, 0.0, 0.0, 0.0],
    [1/3, 0.5, 0.0, 0.5],
    [1/3, 0.5, 0.0, 0.0]
])

scores, iters = calculate_pagerank(
    A_4pages,
    N=4,
    dangling_indices=False, # No dangling nodes in this graph
    m=0.15)
```

Listing 3.1: Numpy representation of the 4-node connected graph

The algorithm converged in **21** iterations, computing the following rank:

Page ID	**Score** (x_i)	**Rank**
Page 1	0.3682	1
Page 3	0.2880	2
Page 4	0.2021	3
Page 2	0.1418	4

Table 3.1: PageRank Results for Figure 3.1

Although Page 3 has the highest number of backlinks (receiving votes from all other pages), Page 1 emerges as the most authoritative. This phenomenon is explained by the dynamics of score distribution: Page 1 receives a decisive vote from Page 3 which, having only a single outgoing link directed specifically toward Page 1, transfers its **entire weight** to it without any dilution.

This confirms that the importance of a page depends not only on the number of votes received, but also on the authority of the voters and how they distribute their score.

3.2 Test Case 2: Disconnected Web

The second test used the 5-page graph from Figure 3.2 which consists of two disconnected subwebs ($W_1 = \{1, 2\}$ and $W_2 = \{3, 4, 5\}$).

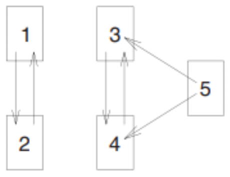


Figure 3.2: Disconnected graph topology (5 nodes)

```python
# Note the zeros indicating no links between Group A and Group B
A_5pages = np.array([
    [0.0, 1.0, 0.0, 0.0, 0.0],
    [1.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 1.0, 0.5],
    [0.0, 0.0, 1.0, 0.0, 0.5],
    [0.0, 0.0, 0.0, 0.0, 0.0]
])

# The damping factor m=0.15 acts as the bridge between components
scores_5, iters_5 = calculate_pagerank(A_5pages, N=5,
    dangling_indices=False, m=0.15)
```

Listing 3.2: Matrix setup for disconnected components

- **Analysis:** with $m = 0$, the dimension of the eigenspace would be 2. The damping factor $m = 0.15$ successfully connects the components via teleportation.
- **Convergence:** the algorithm converged in **2** iterations, providing the following rank.

Page ID	**Score (x_i)**	**Rank**
Page 3	0.2850	1 (Tie)
Page 4	0.2850	1 (Tie)
Page 1	0.2000	3 (Tie)
Page 2	0.2000	3 (Tie)
Page 5	0.0300	5

Table 3.2: PageRank Results for Figure 3.2

The results highlight two key behaviors of the algorithm.
First, the **perfect symmetry** within the sub-webs is reflected in the scores: Page 1 and Page 2 are topologically identical (mutual links), as are Page 3 and Page 4, resulting in tied rankings.
Second, Page 5 receives the lowest score (0.0300). This occurs because Page 5 acts as a pure ”source” node with outgoing links but no incoming links from other pages. Its score is derived exclusively from the **random teleportation component (m/N)**, confirming that the algorithm correctly identifies and penalizes nodes that do not receive votes from the rest of the network.

3.3 Test Case 3: Hollins Dataset

We applied the algorithm to the `hollins.dat` dataset, representing a real-world subset of the web:

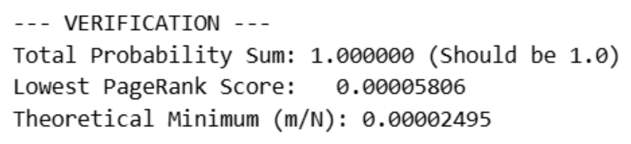
- **Nodes (n):** 6012, of which 3189 are dangling nodes.
- **Edges:** 23875
- **Convergence:** completed in **71** iterations.

```python
filename = 'hollins.dat'
m = 0.15

# 1. Build the Sparse Matrix from file
A_sparse, N, dangling_nodes = build_sparse_link_matrix(filename)

# 2. Calculate PageRank
scores, iters = calculate_pagerank(A_sparse, N, dangling_nodes, m=m
    )

# 3. Mathematical Verification
total_prob = np.sum(scores)
min_score = np.min(scores)
expected_min = m / N
```

Listing 3.3: Processing the Hollins Dataset and Verification

Rank	Page ID	Score
1	2	0.019879
2	37	0.009288
3	38	0.008610
4	61	0.008065
5	52	0.008027
6	43	0.007165
7	425	0.006583
8	27	0.005989
9	28	0.005572
10	4023	0.004452

Table 3.3: Top 10 Pages in Hollins Dataset



```
--- VERIFICATION ---
Total Probability Sum: 1.000000 (Should be 1.0)
Lowest PageRank Score:   0.00005806
Theoretical Minimum (m/N): 0.00002495
```

Figure 3.3: Mathematical verification of PageRank algorithm

The application of the PageRank algorithm to the Hollins dataset reveals meaningful structural properties of the network:

1. **The homepage effect**: the page with ID **2** is identified as the most important node with a score of 0.019879, which is more than double the score of the second-ranked page. In the context of the Hollins University web domain, Page 2 corresponds to the homepage (`http://www.hollins.edu/`).

It is expected to be the authority hub, receiving inbound links from navigation bars across most sub-pages in the domain.

2. **Minimum score verification**: we observed a minimum importance score of 5.8×10^{-5} in the computed vector. The theoretical lower bound for PageRank with $m = 0.15$ is the score a page would receive if it had absolutely no incoming links, receiving importance only through teleportation:

$$x_{min}^{theoretical} = \frac{m}{N} = \frac{0.15}{6012} \approx 2.5 \times 10^{-5}$$

   Since our observed minimum (5.8×10^{-5}) is strictly greater than the theoretical floor (2.5×10^{-5}), this confirms that the damping factor is correctly distributing probability mass to all nodes, and no node is left with zero importance.

3. **Convergence scale**: the algorithm required 71 iterations to converge. While higher than the simple test cases (21 iterations), this is consistent with the spectral gap theory for larger, more complex graphs. It demonstrates that the sparse CSR implementation scales effectively, handling the increased dimensionality without computational bottlenecks.

Chapter 4

Exercises

4.1 Exercise 1

We analyze a scenario of **Ranking Manipulation** where the owners of Page 3 attempt to artificially boost their ranking.
Recall that the importance score x_k is derived from the linear system $Ax = x$. We begin by establishing the baseline scores for the original 4-page topology shown in Figure 3.1.

4.1.1 Mathematical Analysis: Before and After Modification

Case 1: Initial 4-Page Web

In the original configuration, Page 3 links only to Page 1 ($n_3 = 1$), effectively transferring its entire voting weight to that node.

The resulting link matrix A is:

$$A = \begin{bmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \tag{4.1}$$

The importance score eigenvector, derived from the Paper, for this matrix is approximately:

$$x \approx [0.387, 0.129, 0.290, 0.194]^T$$

In this state, $x_1 > x_3$ ($0.387 > 0.290$). Even though Page 3 has more backlinks, Page 1 is more important because it receives the full weight of Page 3's vote.

Case 2: Modified 5-Page Web

To boost Page 3's score, owners create Page 5, which links to Page 3, and they modify Page 3 to link back to Page 5. This changes the number of outgoing links for Page 3 to $n_3 = 2$. The link structure changes as follows:

- **Page 3:** previously linked only to Page 1 ($n_3 = 1$). Now links to Page 1 and Page 5 ($n_3 = 2$). Consequently, the probability mass from Page 3 is split: $A_{1,3} = 0.5$ and $A_{5,3} = 0.5$.
- **Page 5:** links only to Page 3 ($n_5 = 1$). Thus, $A_{3,5} = 1.0$.

The modified link matrix $\tilde{A}$ becomes:

$$\tilde{A} = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 & 1 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 \end{bmatrix} \tag{4.2}$$

The score equations for the modified topology are:

- $x_1 = \frac{x_3}{2} + \frac{x_4}{2}$
- $x_2 = \frac{x_1}{3}$
- $x_3 = \frac{x_1}{3} + \frac{x_2}{2} + \frac{x_4}{2} + x_5 = \frac{x_1}{3} + \frac{x_1}{6} + \frac{x_1}{4} + \frac{x_3}{2}$
- $x_4 = \frac{x_1}{3} + \frac{x_2}{2} = \frac{x_1}{3} + \frac{x_1}{6} = \frac{x_1}{2}$
- $x_5 = \frac{x_3}{2}$

Substituting the values into the equation for x_1:

$$x_1 = \frac{x_3}{2} + \frac{1}{2}\left(\frac{x_1}{2}\right) \implies x_1 = \frac{x_3}{2} + \frac{x_1}{4} \implies \frac{3}{4}x_1 = \frac{x_3}{2} \implies \mathbf{x_3 = 1.5x_1} \tag{4.3}$$

4.1.2 Conclusion

From this exercise, it is clear that the PageRank system can be manipulated through strategic linking. By creating a circle of links between two pages, you create a "trap" that keeps the importance points circulating within a small group rather than letting them flow to the rest of the web.

The theoretical derivation establishes a precise relationship: $x_3 = 1.5x_1$. These findings are corroborated by the computational implementation using the Power Method:

$$x_3 \approx 0.3673 \quad \text{and} \quad x_1 \approx 0.2449$$

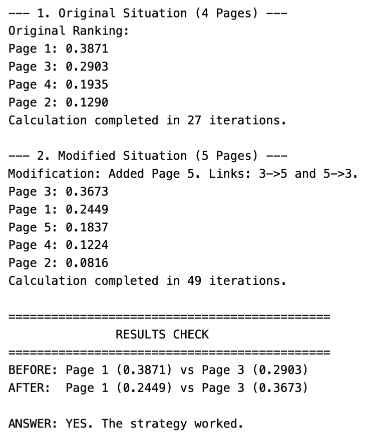
Calculating the ratio from these numerical values confirms the algebraic solution:

$$\frac{x_3}{x_1} = \frac{0.3673}{0.2449} \approx 1.5$$

This demonstrates that the iterative algorithm converges to the exact theoretical distribution derived from the linear system.

```python
# Original A Matrix (4x4)
A_orig_dense = np.array([
    [0.0, 0.0, 1.0, 0.5],
    [1/3, 0.0, 0.0, 0.0],
    [1/3, 0.5, 0.0, 0.5],
    [1/3, 0.5, 0.0, 0.0]
])

A_orig_sparse = sparse.csr_matrix(A_orig_dense)
scores_orig, iters_orig = calculate_pagerank(A_orig_sparse,4,False,
    m=0.0)

# Modified A Matrix (5x5) with trap 3 <-> 5
A_mod_dense = np.array([
    [0.0, 0.0, 0.5, 0.5, 0.0],
    [1/3, 0.0, 0.0, 0.0, 0.0],
    [1/3, 0.5, 0.0, 0.5, 1.0],  # Page 3 receives from 5
    [1/3, 0.5, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.5, 0.0, 0.0]   # Page 5 receives from 3
])

A_mod_sparse = sparse.csr_matrix(A_mod_dense)
scores_mod, iters_mod = calculate_pagerank(A_mod_sparse,5,False,m
    =0.0)
```

Listing 4.1: Exercise 1: Modified Adjacency Matrix Setup



```
--- 1. Original Situation (4 Pages) ---
Original Ranking:
Page 1: 0.3871
Page 3: 0.2903
Page 4: 0.1935
Page 2: 0.1290
Calculation completed in 27 iterations.

--- 2. Modified Situation (5 Pages) ---
Modification: Added Page 5. Links: 3->5 and 5->3.
Page 3: 0.3673
Page 1: 0.2449
Page 5: 0.1837
Page 4: 0.1224
Page 2: 0.0816
Calculation completed in 49 iterations.

============================================
               RESULTS CHECK
============================================
BEFORE: Page 1 (0.3871) vs Page 3 (0.2903)
AFTER:  Page 1 (0.2449) vs Page 3 (0.3673)

ANSWER: YES. The strategy worked.
```

Figure 4.1: Computational results showing the modified PageRank scores. The ranking of Page 3 surpasses Page 1.

4.2 Exercise 11

In this exercise, we recalculate the page ranking to the same modified graph from Exercise 1, which includes Page 5 and its reciprocal link cycle with Page

3 creating a loop, but this time using the Google Matrix M:

$$M = (1 - m)A + mS \tag{4.4}$$

4.2.1 Mathematical Resolution

1. Construction of the Modified Link Matrix A

The column-stochastic link matrix A is:

$$A = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 & 1 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 \end{bmatrix} \tag{4.5}$$

2. Calculation of the Google Matrix M

Using $n = 5$ and $m = 0.15$, the matrix S consists of entries $1/5 = 0.2$. The final matrix M used for the ranking is:

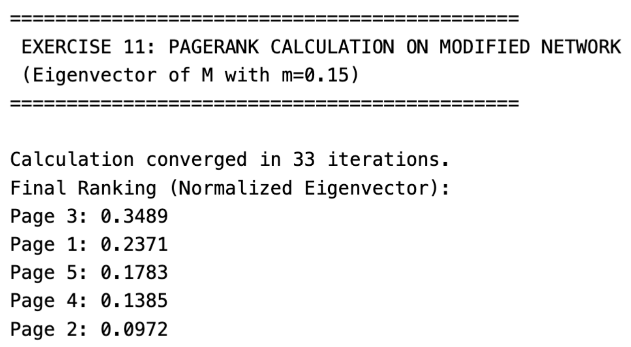
$$M = 0.85 \begin{bmatrix} 0 & 0 & 0.5 & 0.5 & 0 \\ 0.33 & 0 & 0 & 0 & 0 \\ 0.33 & 0.5 & 0 & 0.5 & 1 \\ 0.33 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 \end{bmatrix} + 0.15 \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix} \tag{4.6}$$

3. Implementation and Results

The ranking is computed as the unique steady-state vector q such that $Mq = q$. This corresponds to the Random Surfer Model: the surfer follows a link with 85% probability or "teleports" to a random page with 15% probability[. Mathematically, q is efficiently found using the Power Method ($x_k = Mx_{k-1}$), which is guaranteed to converge regardless of the initial starting vector x_0.

```python
A_ex11_dense = np.array([
    [0.0, 0.0, 0.5, 0.5, 0.0],
    [1/3, 0.0, 0.0, 0.0, 0.0],
    [1/3, 0.5, 0.0, 0.5, 1.0],
    [1/3, 0.5, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.5, 0.0, 0.0]
])

A_ex11_sparse = sparse.csr_matrix(A_ex11_dense)
scores_ex11, iters_ex11 = calculate_pagerank(A_ex11_sparse, 5,
    False, m=0.15)
```

Listing 4.2: Exercise 1: Modified Adjacency



```
============================================
 EXERCISE 11: PAGERANK CALCULATION ON MODIFIED NETWORK
 (Eigenvector of M with m=0.15)
============================================

Calculation converged in 33 iterations.
Final Ranking (Normalized Eigenvector):
Page 3: 0.3489
Page 1: 0.2371
Page 5: 0.1783
Page 4: 0.1385
Page 2: 0.0972
```

Figure 4.2: Results of Exercise 11

4.2.2 Interpretation: the Random Surfer Model

By introducing the jump probability m, the algorithm ensures that the random surfer can eventually "break out" of any artificial trap. This makes the ranking more robust against strategic manipulation and ensures mathematical stability across the global web.

The comparison reveals the stabilizing effect of the Google Matrix:

1. **Mitigation of Manipulation:** In Exercise 1 ($m = 0$), the strategic link loop between Page 3 and Page 5 allowed Page 3 to reach a massive score of 0.3673. In Exercise 11 ($m = 0.15$), the introduction of teleportation introduces a probability "leak" in the loop. The random surfer has a 15% chance at each step to escape the trap and jump to other pages. Consequently, the score of Page 3 drops significantly (from 0.3673 to 0.3489).

2. **Global Redistribution and Mitigation of Strategic Inflation:** Although Page 3 remains the top-ranked node due to its structural prominence, the artificial boost provided by Page 5 is significantly dampened. This is particularly evident in Page 5's score, which sees a substantial drop; since its only source of importance is the reciprocal link from Page 3, it is heavily affected by the 15% "leakage" introduced by the damping factor. Instead of the authority being entirely trapped in the $3 \leftrightarrow 5$ cycle, a portion of it is now constantly redistributed to the rest of the network via the teleportation term mS, preventing artificial link structures from monopolizing the ranking.

3. **Conclusion:** In conclusion, Exercise 11 demonstrates that the Google Matrix M effectively mitigates strategic link manipulation. While the reciprocal structure between Page 3 and Page 5 still influences the ranking, the 15% damping factor prevents artificial score inflation by redistributing authority globally. This ensures a more balanced, robust, and mathematically stable ranking compared to the "pure democracy" model used in Exercise 1.