Modeling Web Page Importance with PageRank and Markov Chains

MATH UN2015 Applied Linear Algebra Project

Dr. Tommaso Maria Botta

Rohit Biswas (rb3908), Ahana Iyer (ati2110), Sofian Laouzai (sl5680)

December 2025

Abstract

This project investigates the PageRank algorithm as a concrete application of linear algebra and probability for ranking nodes in a directed network. We model a small web as a finite Markov chain whose states are web pages and whose transition probabilities are derived from the hyperlink structure. After correcting for dangling nodes and adding a damping factor, we obtain the so–called Google matrix, a column-stochastic matrix whose dominant eigenvector is the PageRank vector, interpreted as the stationary distribution of a "random surfer." Using Python (NumPy and NetworkX), we implement the power iteration method, verify convergence on toy examples where we can compute the result by hand, and apply the algorithm to small illustrative directed graphs. We then compare PageRank to simpler centrality measures such as in-degree and eigenvector centrality and study how the damping factor and graph structure affect convergence and ranking. Throughout, we connect our findings to the theoretical framework of Markov chains and matrix analysis as presented in standard texts by Grinstead and Snell [4], Meyer [3], and Langville and Meyer [2], and to the original formulation by Brin and Page [1].

1 Introduction

Web search engines must decide which pages are most "important" or relevant to a user query. The PageRank algorithm, introduced by Brin and Page at Google in the late 1990s [1], addressed this problem by exploiting the global hyperlink structure of the web rather than only local content features. In PageRank, a page is considered important if it is linked to by other important pages; the hyperlink network induces a recursive notion of importance.

Mathematically, PageRank provides a canonical example of how ideas from linear algebra and probability (stochastic matrices, eigenvalues and eigenvectors, Markov chains, and stationary distributions) combine in an applied setting. Following our project prospectus, we focus on three guiding questions:

a) How is the PageRank vector defined as the stationary distribution of a Markov chain on a directed graph?

b) How do graph structure and the damping factor influence convergence and the final ranking of nodes?

c) For small networks, how does PageRank compare to simpler centrality measures such as in-degree and eigenvector centrality?

Our goal is not to scale to web-sized graphs, but to understand these questions on small, interpretable networks where we can inspect the matrices and distributions explicitly. This connects the abstract course material (Markov chains, eigenvalues, power iteration) to an influential real-world algorithm [2, 3].

2 Background: Directed Graphs and Markov Chains

2.1 Directed graphs and adjacency matrices

We represent a collection of web pages and hyperlinks as a directed graph $G = (V, E)$, where each vertex $i \in V = \{1, \ldots, n\}$ is a page and a directed edge $j \to i$ indicates that page j contains a hyperlink to page i. The structure of G can be encoded in an *adjacency matrix*

$$A = (a_{ij})_{i,j=1}^{n}, \qquad a_{ij} = \begin{cases} 1, & \text{if there is a link } j \to i, \\ 0, & \text{otherwise.} \end{cases}$$

For PageRank, we are interested not only in the presence of links but in probabilities of following links. Let L_j denote the out-degree of node j, i.e. the number of outgoing links from j. The hyperlink matrix $H = (h_{ij})$ is defined by

$$h_{ij} = \begin{cases} \dfrac{1}{L_j}, & \text{if } j \to i, \\ 0, & \text{otherwise.} \end{cases}$$

Each column of H describes the distribution over next pages when a user at page j chooses one of its links uniformly at random. Ideally, H is column-stochastic (each column sums to 1).

2.2 Markov chains and stationary distributions

A discrete-time Markov chain on a finite state space $\{1, \ldots, n\}$ is specified by a transition matrix $P = (p_{ij})$ where p_{ij} is the probability of moving from state j to state i in one step and each column sums to 1 [4]. A probability vector $x^{(k)} \in \mathbb{R}^n$ evolves according to

$$x^{(k+1)} = P x^{(k)}.$$

A probability vector π is called *stationary* if

$$\pi = P\pi, \quad \pi_i \geq 0, \quad \sum_{i=1}^{n} \pi_i = 1.$$

Under suitable conditions (irreducibility and aperiodicity), the Markov chain has a unique stationary distribution π, and for any initial distribution $x^{(0)}$ we have $x^{(k)} \to \pi$ as $k \to \infty$ [4, 3]. In matrix terms, π is the dominant eigenvector of P corresponding to the eigenvalue $\lambda = 1$.

2.3 From the hyperlink matrix to the Google matrix

A naive attempt to define PageRank is to set $P = H$ and take the stationary distribution of the chain defined by H. However, two structural problems arise [2]:

- **Dangling nodes:** If page j has no outgoing links, then $L_j = 0$ and the j-th column of H is all zeros. This column does not define a probability distribution and causes probability mass to "leak".

- **Reducibility and rank sinks:** Even if H is column-stochastic, the underlying graph may decompose into components or contain sinks (subsets of pages that only link internally). In such a case, the Markov chain may not have a unique stationary distribution or may trap probability in certain components.

To address dangling nodes, we replace each zero column of H by a uniform column vector

$$u = \frac{1}{n}\mathbf{1} = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)^T,$$

obtaining a corrected transition matrix H'. This ensures that H' is column-stochastic.

To address reducibility and periodicity, PageRank introduces a *damping factor* $\alpha \in (0, 1)$ and defines the *Google matrix*

$$G = \alpha H' + (1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T.$$

Here $\frac{1}{n}\mathbf{1}\mathbf{1}^T$ is the matrix whose every column equals u. The corresponding random-surfer interpretation is:

- with probability α, follow a hyperlink chosen uniformly from the current page;
- with probability $1 - \alpha$, "teleport" to a uniformly random page.

The matrix G is column-stochastic, irreducible, and aperiodic (primitive), so by the Perron–Frobenius theorem it has a unique positive stationary distribution [5, 2, 3]. This stationary distribution is the PageRank vector.

3 Mathematical Formulation of PageRank

3.1 The PageRank vector as an eigenvector

Let G be the $n \times n$ Google matrix. The PageRank vector $r \in \mathbb{R}^n$ is defined as the unique probability vector satisfying

$$Gr = r, \qquad r_i \geq 0, \quad \sum_{i=1}^{n} r_i = 1. \tag{1}$$

Thus r is a right eigenvector of G with eigenvalue 1. All other eigenvalues λ of G satisfy $|\lambda| < 1$ when $0 < \alpha < 1$ [2]. Intuitively, r_i represents the long-run fraction of time a random surfer spends on page i.

Written componentwise,

$$r_i = \alpha \sum_{j=1}^{n} h'_{ij} r_j + (1 - \alpha)\frac{1}{n}\sum_{j=1}^{n} r_j = \alpha \sum_{j=1}^{n} h'_{ij} r_j + \frac{1 - \alpha}{n},$$

using the fact that $\sum_j r_j = 1$. This makes clear that a page's rank is a convex combination of rank flowing in through hyperlinks and a uniform "teleportation" contribution.

3.2 Existence, uniqueness, and dependence on α

Because G is primitive, the Perron–Frobenius theorem guarantees [3]:

- G has a simple eigenvalue $\lambda_1 = 1$ with a strictly positive eigenvector;
- All other eigenvalues satisfy $|\lambda_k| < 1$.

These properties imply that the Markov chain defined by G has a unique stationary distribution and that power iteration (Section 4) converges to it from any starting distribution.

The choice of α balances two effects [2]:

- Larger α (e.g. $\alpha = 0.95$) makes G closer to H', so PageRank reflects the link structure more strongly but convergence slows because the second-largest eigenvalue $|\lambda_2|$ moves closer to 1.
- Smaller α (e.g. $\alpha = 0.5$) speeds up convergence and moves r closer to the uniform distribution, making rankings less sensitive to fine structural differences.

In practice, $\alpha = 0.85$ has been widely used [1, 2].

4 Computational Implementation: Power Iteration

For small n, the PageRank vector could be found by solving the linear system $(G - I)r = 0$ with the normalization $\sum_i r_i = 1$. However, the original application involved billions of pages, and even in moderate dimensions direct eigenvector computation is costly. Instead, we use the *power iteration* method [3].

4.1 Algorithm

Given a column-stochastic matrix G and a starting probability vector $r^{(0)}$, power iteration generates a sequence

$$r^{(k+1)} = G\, r^{(k)}, \qquad k = 0, 1, 2, \ldots.$$

Under the assumptions above, $r^{(k)} \to r$ as $k \to \infty$, where r is the PageRank vector. Our implementation uses the following steps:

1. **Initialize:** Set $r^{(0)} = \frac{1}{n}\mathbf{1}$, the uniform distribution.
2. **Iteration:** For $k = 0, 1, 2, \ldots$ compute $r^{(k+1)} = Gr^{(k)}$.
3. **Convergence test:** Stop when $\|r^{(k+1)} - r^{(k)}\|_1 < \varepsilon$ for a tolerance ε (we used $\varepsilon = 10^{-8}$).

Because G is column-stochastic, each $r^{(k)}$ remains a probability vector. The convergence rate is controlled by $|\lambda_2|$, the magnitude of the second-largest eigenvalue of G, so the number of iterations is $O(\log(1/\varepsilon)/\log(1/|\lambda_2|))$ [2].

4.2 Implementation details

We implemented PageRank in Python using NumPy to store H, G, and $r^{(k)}$ as arrays. Directed graphs were created using NetworkX, from which we extracted adjacency information and out-degrees. For the small networks considered here, we used dense matrices for simplicity; in large-scale settings, sparse matrix structures would be essential [2].

To validate the implementation, we:

- Checked that G is column-stochastic numerically (column sums equal 1 up to rounding).
- Verified on tiny graphs (three to five nodes) that power iteration converges to the same distribution obtained by solving $(G - I)r = 0$ directly with a linear solver.
- Monitored $\|r^{(k+1)} - r^{(k)}\|_1$ as a function of k to visualize convergence.

4.3 Python implementation of PageRank

To operationalize the PageRank framework developed in Sections 2.3 and 4, we implemented the algorithm directly in Python using NumPy and NetworkX. This subsection serves as a computational counterpart to the theoretical construction, translating the definitions of H, H', and G and the power iteration method into executable code. The implementation proceeds in three steps:

i) setup and helper functions,

ii) matrix construction (H, H', and G),

iii) power iteration.

4.3.1 Setup and example graph

We first construct the directed graph used in Section 5.3 and establish a consistent mapping between node labels and matrix indices.

Listing 1: Setup and example directed graph

```
import numpy as np
import networkx as nx
from typing import Dict, List, Tuple

ALPHA = 0.85          # Damping factor
TOLERANCE = 1e-8      # Convergence tolerance

def create_example_graph() -> nx.DiGraph:
    """
    Creates the 4-node directed graph from Section 5.3.
    Links: 1->2, 1->3, 2->3, 3->1, 4->3
    """
    G = nx.DiGraph()
    G.add_edges_from([
        (1, 2),
```

```
        (1, 3),
        (2, 3),
        (3, 1),
        (4, 3)
    ])
    return G

def get_node_indices(G: nx.DiGraph) -> Dict[int, int]:
    """Maps node labels to matrix indices."""
    return {node: i for i, node in enumerate(G.nodes)}
```

4.3.2 Matrix construction: H and dangling-node correction

The following code constructs the hyperlink matrix H defined in Section 2.1 and identifies dangling nodes. Columns corresponding to dangling nodes are later replaced with the uniform vector, yielding H' as described in Section 2.3.

Listing 2: Construction of the hyperlink matrix H

```
def build_hyperlink_matrix(
    G: nx.DiGraph,
    n: int,
    node_indices: Dict[int, int]
) -> Tuple[np.ndarray, List[int]]:

    H = np.zeros((n, n))
    dangling_nodes = []

    for j_node, j in node_indices.items():
        out_degree = G.out_degree(j_node)

        if out_degree == 0:
            # Dangling node
            dangling_nodes.append(j)
        else:
            for i_node in G.successors(j_node):
                i = node_indices[i_node]
                H[i, j] = 1.0 / out_degree

    return H, dangling_nodes
```

Dangling nodes are corrected by replacing zero columns with the uniform probability vector $u = (1/n)\mathbf{1}$.

Listing 3: Dangling-node correction to obtain H'

```
def correct_dangling_nodes(
    H: np.ndarray,
    dangling_nodes: List[int],
    n: int
) -> np.ndarray:

    H_prime = H.copy()
```

```
    u = np.ones(n) / n

    for j in dangling_nodes:
        H_prime[:, j] = u

    return H_prime
```

4.3.3 Google matrix construction

Using the corrected hyperlink matrix H', we form the Google matrix

$$G = \alpha H' + (1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T,$$

as defined in Section 2.3.

Listing 4: Construction of the Google matrix G

```
def build_google_matrix(
    H_prime: np.ndarray,
    alpha: float,
    n: int
) -> np.ndarray:

    teleport_matrix = np.ones((n, n)) / n
    G = alpha * H_prime + (1 - alpha) * teleport_matrix

    return G
```

4.3.4 Power iteration method

Finally, following the power iteration method developed in Section 4, we compute the PageRank vector. Starting from the uniform distribution, the iteration proceeds until convergence in the ℓ^1 norm.

Listing 5: Power iteration for PageRank

```
def power_iteration(
    G: np.ndarray,
    n: int,
    tolerance: float
) -> np.ndarray:

    r_current = np.ones(n) / n
    iteration_count = 0

    while True:
        r_next = G @ r_current

        if np.linalg.norm(r_next - r_current, ord=1) < tolerance:
            break

```

```
        r_current = r_next
        iteration_count += 1

        if iteration_count > 1000:
            print("Warning: Maximum iterations reached.")
            break

    print(f"Converged after {iteration_count} iterations.")
    return r_next
```

4.3.5 Connection to experimental results

All PageRank values reported in Section 5 were computed using this implementation. The code directly mirrors the mathematical definitions of H, H', G, and the stationary distribution equation $Gr = r$, ensuring consistency between theory and computation.

5 Experiments on Small Directed Networks

We now address our three research questions using small, hand-crafted graphs that illustrate the role of graph structure, damping, and comparison with other centrality measures.

5.1 A simple 3-page cycle

Consider three pages A, B, C with links

$$A \to B, \quad B \to C, \quad C \to A.$$

The adjacency structure is a directed 3-cycle. There are no dangling nodes, and the hyperlink matrix H is already column-stochastic. For any $\alpha \in (0, 1)$, the resulting Google matrix G is symmetric under cyclic permutation of the nodes, so the PageRank vector is

$$r = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)^T.$$

Numerically, power iteration converged to the uniform distribution from any starting vector. This example illustrates that PageRank reduces to uniform ranking when the graph is highly symmetric.

5.2 Structural problems: dangling nodes and rank sinks

Next, we constructed examples to illustrate how dangling nodes and rank sinks affect the ranking and why the damping factor is essential.

Dangling node. A *dangling node* is a page with no outgoing links. In the hyperlink matrix H, this corresponds to a column of zeros. If we naïvely iterate $r^{(k+1)} = Hr^{(k)}$, the total probability $\sum_i r_i^{(k)}$ decreases whenever probability flows into the dangling node's column. The standard PageRank fix is to replace each zero column by the uniform vector $u = (1/n, \ldots, 1/n)^T$, which we did in our implementation.

Rank sink. A *rank sink* is a set of pages that only link to each other and not to the rest of the graph. Without damping, probability that enters the sink never leaves, and the PageRank mass eventually concentrates entirely within the sink. With $\alpha < 1$, teleportation provides a nonzero probability of jumping into or out of every component at each step. In our experiments, we saw that:

- For $\alpha = 1$, the power iteration on H' did not converge to a unique stationary distribution when the graph contained a sink; the limiting distribution depended on the starting vector.
- For $\alpha = 0.85$, the same graph had a unique PageRank vector in which sink nodes still had relatively high scores but did not absorb the entire probability mass.

This directly illustrates the role of the damping factor in turning the Markov chain primitive and ensuring a unique stationary distribution.

5.3 Comparison with in-degree and eigenvector centrality

To compare PageRank with simpler centrality measures, we considered the following 4-node directed graph:

$$\begin{aligned} &1 \to 2,\ 1 \to 3,\\ &2 \to 3,\\ &3 \to 1,\\ &4 \to 3. \end{aligned}$$

Node 3 receives links from three different pages $(1, 2, 4)$ and is intuitively the most central.

In-degree centrality

The in-degree of node i is the number of incoming links:

$$\deg^-(1) = 1, \quad \deg^-(2) = 1, \quad \deg^-(3) = 3, \quad \deg^-(4) = 0.$$

Thus in-degree centrality ranks node 3 highest, nodes 1 and 2 tied, and node 4 last.

Eigenvector centrality

Eigenvector centrality assigns scores x_i proportional to the sum of scores of neighbors linking to i, i.e. $x = \lambda A^T x$ [3]. For this graph, the dominant eigenvector of A^T (normalized to sum to 1) is approximately

$$x \approx (0.325,\ 0.245,\ 0.430,\ 0.000)^T.$$

Node 3 is again highest, followed by node 1, then node 2, and node 4 effectively gets score 0.

PageRank

With $\alpha = 0.85$ and the standard teleportation term, the PageRank vector for this graph (computed by our implementation) is approximately

$$r \approx (0.373,\ 0.196,\ 0.394,\ 0.038)^T.$$

We observe:

- Node 3 still has the highest rank, reflecting its role as the primary hub.
- Node 1 is second, since it is part of a 3-cycle with node 3.
- Node 2 has moderate rank.
- Node 4 has a small but non-zero rank because teleportation ensures every node maintains some probability mass even if it is structurally peripheral.

This example shows that PageRank and eigenvector centrality produce similar relative rankings in a simple graph without dangling nodes, but PageRank's teleportation term prevents nodes from ever having rank exactly zero. For larger or more complex directed networks (especially with sinks or dangling nodes), PageRank can differ much more significantly from in-degree and eigenvector centrality [2].

Explicit matrices for the 4-node example

For completeness, we record the matrices associated with this graph. With nodes ordered as $(1, 2, 3, 4)$, the adjacency matrix is

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Since every node has at least one outgoing link, there are no dangling nodes, and the hyperlink matrix is already column-stochastic:

$$H = H' = \begin{pmatrix} 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

With damping factor $\alpha = 0.85$ and $n = 4$, the Google matrix

$$G = \alpha H' + (1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T$$

is

$$G = \begin{pmatrix} 0.0375 & 0.0375 & 0.8875 & 0.0375 \\ 0.4625 & 0.0375 & 0.0375 & 0.0375 \\ 0.4625 & 0.8875 & 0.0375 & 0.8875 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \end{pmatrix}.$$

5.4 Effect of the damping factor

Finally, we studied how varying the damping factor α affects both convergence and rankings. On the 4-node graph above, we ran power iteration for $\alpha = 0.5$, $\alpha = 0.85$, and $\alpha = 0.95$. We observed the following qualitative patterns, consistent with the theory [2]:

- For $\alpha = 0.5$, the PageRank vector was closer to uniform: differences between node scores were modest, and the number of iterations to reach the tolerance ε was small.
- For $\alpha = 0.85$, the ranking emphasized the hyperlink structure more strongly while still converging in a reasonable number of iterations.
- For $\alpha = 0.95$, node 3's advantage became even more pronounced and low-importance nodes were pushed closer to 0, but the number of iterations roughly doubled, reflecting slower convergence as $|\lambda_2|$ approached 1.

These experiments support the usual choice $\alpha \approx 0.85$ as a compromise between sensitivity to structure and computational efficiency [1, 2].

6 Discussion and Connection to Course Concepts

The PageRank algorithm ties together several themes from applied linear algebra and probability:

- **Stochastic matrices and Markov chains:** The Google matrix G is a column-stochastic matrix whose powers G^k describe the evolution of a random surfer. The stationary distribution r is both a Markov chain steady state and a dominant eigenvector [4, 3].
- **Eigenvalues, eigenvectors, and power iteration:** Power iteration is a simple iterative method that exploits the spectral gap between the largest eigenvalue and the rest. Our experiments make this spectral viewpoint concrete.
- **Structural properties of graphs:** Dangling nodes, reducibility, and sinks illustrate how purely combinatorial features of a directed graph affect the behavior of the associated Markov chain and the need for regularization via teleportation [2].
- **Comparison of centrality notions:** In-degree centrality, eigenvector centrality, and PageRank are all expressible in linear algebraic terms but capture different ideas of importance. PageRank's probabilistic interpretation as a long-run visit probability makes it especially suitable for ranking in information retrieval [1].

Working through explicit small examples allowed us to verify the theoretical claims of the literature and to see concretely how the mathematical assumptions in theorems such as Perron–Frobenius manifest in computations.

7 Conclusion

We modeled web page importance using the PageRank algorithm and analyzed it through the lens of linear algebra and Markov chains. Starting from a directed graph, we built the hyperlink matrix H, corrected dangling nodes to obtain H', and formed the Google matrix G using a damping factor and teleportation. We then used power iteration to compute the PageRank vector and examined its dependence on graph structure and the damping parameter.

On small networks, we saw that PageRank:

- Can be interpreted as the stationary distribution of a random surfer on a directed graph.
- Resolves structural issues such as dangling nodes and rank sinks through teleportation.
- Often agrees qualitatively with eigenvector centrality on simple graphs but differs from in-degree centrality, especially when the *source* of incoming links matters.
- Becomes more sensitive to the hyperlink structure and slower to converge as α increases.

Beyond its historical importance in web search, PageRank serves as a rich case study in applied linear algebra, illustrating how abstract concepts like eigenvalues, stochastic matrices, and Markov chains directly inform the design and analysis of widely used algorithms.

References

[1] S. Brin and L. Page, The anatomy of a large-scale hypertextual Web search engine, *Computer Networks*, 30(1–7):107–117, 1998.

[2] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006.

[3] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000.

[4] C. M. Grinstead and J. L. Snell, *Introduction to Probability*, 2nd edition, American Mathematical Society, 1997.

[5] (Example survey) P. Berkhin, A survey on PageRank computing, *Internet Mathematics*, 2(1):73–120, 2005.