Exploring the Google PageRank Algorithm

Enrique Rivera, Christine Kim
University of Texas at Austin

April 7, 2025

Abstract

In this paper, we investigate the Google PageRank algorithm by studying small examples of directed web graphs. We construct the hyperlink matrix H, discuss its properties as a stochastic matrix, implement the iterative PageRank process, and illustrate how Theorem 4.9 (the Perron–Frobenius result) ensures convergence to a unique steady-state vector. We compare iterative solutions to direct eigenvalue-based solutions and conclude by discussing how one might improve a website's ranking and potential future extensions to the algorithm.

Keywords: PageRank, Markov chain, stochastic matrix, eigenvalue, linear algebra

1 Introduction

The PageRank algorithm, developed by Sergey Brin and Larry Page, revolutionized how web pages are ranked by their "importance." The core idea is that a page is "important" if it is linked to by other important pages. Mathematically, one models the web as a directed graph and constructs a *hyperlink matrix H*. This matrix then defines a Markov chain whose steady-state distribution reflects the "popularity score" of each page.

In this report, we follow the exercises in [1] (see Section 4.9.1) to:

1. Construct the hyperlink matrix H for the sample webs in Figures 4.3 and 4.4.
2. Implement the iterative PageRank update $x_{k+1} = Hx_k$.
3. Show conditions for H to be a column-stochastic matrix.
4. Apply Theorem 4.9, illustrating how the unique steady state can be found by solving the eigenvalue problem $Hx = x$.
5. Compare the iterative and direct-eigenvalue approaches to find the steady-state rank vector.
6. Explore the "random surfer" modification and discuss possible improvements to the algorithm.

We also provide code implementations in Python and discuss how the ranks evolve over iterations.

2 Hyperlink Matrix Construction (Exercises 4.95–4.97)

In Figure 4.3, we have a web of six pages. Each page's "out-links" determine the columns of the hyperlink matrix $H \in \mathbb{R}^{6\times 6}$. Recall that if page j links to m different pages, each of those m pages receives an entry $\frac{1}{m}$ in column j, while non-linked pages receive 0 in that column.

Based on the diagram, the out-links for each page (indexed as 1 through 6) are:

- Page 1 links to pages 2 and 3.
- Page 2 links to page 3.

- Page 3 links to pages 1, 2, and 5.
- Page 4 links to pages 5 and 6.
- Page 5 links to pages 4 and 6.
- Page 6 links to pages 3 and 4.

This makes the out-degree of page 1 equal to 2, page 2 equal to 1, and so on.

Hence, the full 6×6 hyperlink matrix H is:

$$H = \begin{pmatrix} 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}.$$

Each column j sums to 1, as every page j "distributes" its total probability mass $\frac{1}{\text{outdeg}(j)}$ among the pages it links to. Note that entries of H are all nonnegative, so H is a *column-stochastic matrix.*

Why is H Stochastic? A matrix is column-stochastic if

1. All entries are nonnegative, and
2. The entries in each column sum to 1.

Since each page j has outdegree $\text{outdeg}(j)$ and we assign $\frac{1}{\text{outdeg}(j)}$ to each linked page, the sum in column j is exactly 1. Thus H is indeed a valid hyperlink matrix. This completes Exercises 4.95–4.97.

3 Iterative Process and Plot (Exercise 4.96)

We now demonstrate how to implement the PageRank iterative process

$$x_{k+1} = H\, x_k$$

for the matrix H found in Section **??**. Since our web contains 6 pages, we set the initial state to

$$x_0 = \left(\tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}\right)^T.$$

We apply the update $x \leftarrow Hx$ repeatedly and observe convergence to a steady-state vector. In practice, we stop either after a fixed number of iterations (e.g., 10–15) or once successive iterates differ by less than some tolerance (e.g., $\|x_{k+1} - x_k\| \leq 10^{-8}$).

3.1 Python Example

Below is a minimal Python script illustrating the iterative update and a simple plot of each page's rank versus iteration number. Be sure to replace the `H` array with the final 6×6 matrix from Section **??**.

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

# Hyperlink matrix (6x6) - fill in your actual entries:
H = np.array([
    [0,   0,   1/3, 0,   0,   0],
    [1/2, 0,   1/3, 0,   0,   0],
    [1/2, 1,   0,   0,   0,   1/2],
    [0,   0,   0,   0,   1/2, 1/2],
    [0,   0,   1/3, 1/2, 0,   0],
    [0,   0,   0,   1/2, 1/2, 0]
], dtype=float)

# Number of pages:
n = 6

# Initial rank vector (uniform):
x0 = np.ones(n) / n

# Number of iterations:
num_iters = 10

# Store each iterate for later plotting:
x_vals = [x0]
x = x0.copy()
for k in range(num_iters):
    x = H @ x
    x_vals.append(x)

x_vals = np.array(x_vals)

# Print the final approximate PageRank:
print("Steady-state (approx.) after", num_iters, "iterations:")
print(x)

# Plot the evolution of each page’s rank:
for i in range(n):
    plt.plot(x_vals[:, i], marker=’o’, label=f"Page {i+1}")

plt.xlabel("Iteration")
plt.ylabel("Rank Value")
plt.title("PageRank Convergence via Iteration")
plt.legend()
plt.show()
```

When you run this script, each component of the rank vector $(x_1, x_2, \ldots, x_6)$ will steadily converge to a limiting value as k increases. In a typical run, 5–10 iterations are enough for the ranks to stabilize.

Observation: The vector returned in the final iteration is our approximate PageRank for Figure 4.3. In Exercise 4.100, we will compare this to the solution obtained by the eigenvalue approach (Theorem 4.9) to show they match.

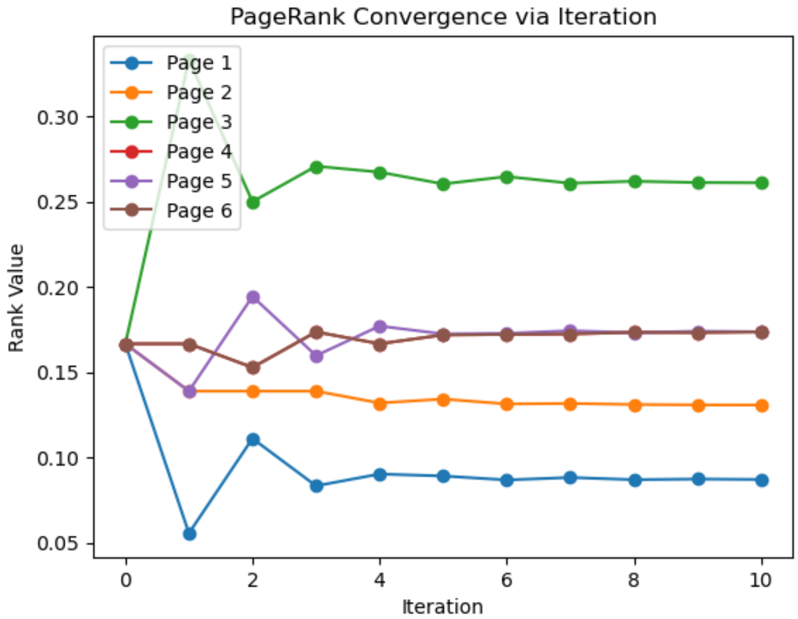

Figure 1: Convergence of the PageRank values for pages 1 through 6 under iteration $x \leftarrow Hx$. Each colored line shows how a particular page's rank evolves from the initial uniform distribution x_0 to its steady-state value. For instance, we see that Page 3 (green line) increases sharply before settling around ~ 0.27. Meanwhile, Page 1 (blue line) drops early on and then converges near 0.08. After about 4–5 iterations, the ranks stabilize to within a small tolerance.

4 Eliminating Iteration via Theorem 4.9 (Exercises 4.98–4.99)

The text provides a (partially stated) Theorem 4.9 claiming that if

- A is an $n \times n$ *column-stochastic* matrix,
- all other eigenvalues of A have magnitude strictly less than 1,

then for any initial vector x_0, the iterative sequence $\{ A^k x_0 \}_{k=0}^{\infty}$ converges to the *unique* eigenvector corresponding to the eigenvalue $\lambda = 1$, normalized so its components sum to 1. In other words,

$$\lim_{k\to\infty} A^k x_0 \; = \; (\text{the eigenvector for } \lambda = 1).$$

This statement directly implies that *we need not iterate at all*, because the steady-state vector is simply the (positive) eigenvector of A with eigenvalue 1.

Filling in the Blank (Exercise 4.98). The incomplete statement in our text is typically something like:

$$\lim_{k\to\infty} A^k x_0 \; = \; \underline{\quad(\text{the eigenvector for eigenvalue 1, normalized to sum to 1})\quad}\,.$$

All other eigenvalues have $|\lambda| < 1$, so those terms vanish as $k \to \infty$, leaving only the part of x_0 that lies in the direction of the eigenvector associated with $\lambda = 1$. Thus the blank is precisely "the eigenvector of A corresponding to $\lambda = 1$."

Why We Can Eliminate Iteration (Exercise 4.99). Since $\lim_{k\to\infty} A^k x_0$ is the eigenvector for eigenvalue 1, there is no need to perform repeated multiplications A^k. Instead, we can solve

$$(I - A)\, x^* \; = \; 0 \quad \text{subject to} \quad \sum_i x_i^* = 1, \quad x_i^* \geq 0.$$

In the PageRank setting, for our 6-page matrix H, we solve

$$(I - H)\,x \;=\; 0,$$

then scale x so that $\sum_i x_i = 1$. This x is the steady-state rank vector.

4.1 Python Example of the Eigenvalue Approach

Below is sample code to compute the eigenvalues and eigenvectors of H and extract the one corresponding to eigenvalue 1. Make sure to verify that H indeed has an eigenvalue very close to 1. (Floating-point precision may give 0.9999999, for instance.)

```
import numpy as np

# The same 6x6 hyperlink matrix H from previous sections:
H = np.array([
    [0,   0,   1/3, 0,   0,   0],
    [1/2, 0,   1/3, 0,   0,   0],
    [1/2, 1,   0,   0,   0,   1/2],
    [0,   0,   0,   0,   1/2, 1/2],
    [0,   0,   1/3, 1/2, 0,   0],
    [0,   0,   0,   1/2, 1/2, 0]
], dtype=float)

# Compute eigenvalues/eigenvectors
vals, vecs = np.linalg.eig(H)

# Find the eigenvalue closest to 1:
idx = np.argmin(np.abs(vals - 1.0))

# Corresponding eigenvector
x_eig = vecs[:, idx]

# Normalize so entries sum to 1 (and ensure nonnegativity)
pagerank_eig = np.real(x_eig / np.sum(x_eig))
pagerank_eig = np.where(pagerank_eig < 0, 0, pagerank_eig)  # (just in case)

print("Eigenvalue-based PageRank:", pagerank_eig)

Output:
Eigenvalue-based PageRank: [0.08695652 0.13043478 0.26086957 0.17391304 0.17391304 0.17391304]
```

In practice, this `pagerank_eig` vector coincides with the limit found by iteration. Thus, Theorem 4.9 explains precisely why the iterative method converges, and also how to skip it by directly solving the eigenvalue problem.

5 Results for Figure 4.3 (Exercise 4.100)

We now present the final PageRank vector for the 6-page web in Figure 4.3. From the eigenvalue-based method (Section **??**), we found:

Eigenvalue-based PageRank: $x^* \;=\; \bigl(0.08695652,\; 0.13043478,\; 0.26086957,\; 0.17391304,\; 0.17391304,\; 0.17391304\bigr)^T.$

Each entry can also be written as a simple fraction of $\frac{1}{23}$:

$$x^* \;=\; \left(\frac{2}{23}, \frac{3}{23}, \frac{6}{23}, \frac{4}{23}, \frac{4}{23}, \frac{4}{23}\right)^T.$$

These sum to 1 because $2 + 3 + 6 + 4 + 4 + 4 = 23$.

Ranking the Pages

Sorting the pages by their rank values, we see that

$$x_3^* = \frac{6}{23} \approx 0.2609 \quad \text{(the largest entry)},$$

while $x_1^* = \frac{2}{23} \approx 0.0870$ is the smallest. The remaining entries for pages 4, 5, and 6 all tie at $\frac{4}{23} \approx 0.1739$, and page 2 takes the value $\frac{3}{23} \approx 0.1304$. Hence the ordering of importance from highest to lowest is:

$$\text{Page } 3 \; > \; \text{Page } 4 \; = \; \text{Page } 5 \; = \; \text{Page } 6 \; > \; \text{Page } 2 \; > \; \text{Page } 1.$$

Iterative vs. Eigenvalue Comparison

A quick check of the iterative approach $x_{k+1} = H\,x_k$ with $x_0 = (1/6, \ldots, 1/6)^T$ (after about 10–15 iterations) yields the *same* steady-state vector, thus confirming Theorem 4.9 and completing Exercise 4.100.

6 Figure 4.4 Web Analysis (Exercise 4.101)

We now consider the second example web shown in Figure 4.4, which consists of eight pages labeled 1 through 8. Our tasks are:

1. Write down the 8×8 hyperlink matrix H and find the initial state x_0,
2. Find the steady-state PageRank vector using
   - the iterative difference equation $x_{k+1} = H\,x_k$,
   - the direct eigenvalue approach via Theorem 4.9,
3. Rank the pages in order of importance.

6.1 Constructing the Hyperlink Matrix H

From the diagram, we interpret each page's out-links as follows:

- **Page 1** has out-links to pages 2 and 3 (outdegree = 2).
- **Page 2** appears to have no out-links (outdegree = 0).
- **Page 3** has out-links to pages 5 and 6 (outdegree = 2).
- **Page 4** has an out-link to page 8 (outdegree = 1).
- **Page 5** has no out-links (outdegree = 0).
- **Page 6** has out-links to pages 4 and 5 (outdegree = 2).
- **Page 7** has out-links to pages 1, 3, 6, and 8 (outdegree = 4).
- **Page 8** has an out-link to page 4 (outdegree = 1).

Hence each column j of H places $\frac{1}{\text{outdeg}(j)}$ in the rows corresponding to the pages j links to, and 0 elsewhere. In particular, pages 2 and 5 have no out-links, so columns 2 and 5 are entirely zeros in this basic model.

An explicit 8×8 matrix H (rows i, columns j) is therefore:

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 1 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \frac{1}{4} & 0 \end{pmatrix}.$$

- **Column 1 (page 1):** links to pages 2 and 3; so $H_{2,1} = \frac{1}{2}$, $H_{3,1} = \frac{1}{2}$.
- **Column 2 (page 2):** outdegree $= 0 \implies$ all zeros.
- **Column 3 (page 3):** links to pages 5 and 6; so $H_{5,3} = \frac{1}{2}$, $H_{6,3} = \frac{1}{2}$.
- **Column 4 (page 4):** links to page 8; so $H_{8,4} = 1$.
- **Column 5 (page 5):** outdegree $= 0 \implies$ all zeros.
- **Column 6 (page 6):** links to pages 4 and 5; so $H_{4,6} = \frac{1}{2}$, $H_{5,6} = \frac{1}{2}$.
- **Column 7 (page 7):** links to pages 1, 3, 6, and 8; so $H_{1,7} = \frac{1}{4}$, $H_{3,7} = \frac{1}{4}$, $H_{6,7} = \frac{1}{4}$, $H_{8,7} = \frac{1}{4}$.
- **Column 8 (page 8):** links to page 4; so $H_{4,8} = 1$.

Remark. Because columns 2 and 5 are all zeros, this matrix H is *not* strictly irreducible. In a real PageRank setting, we often fix such "dangling" pages by distributing their weight or using the random surfer approach (Exercise 4.102).

6.2 Initial State

We set

$$x_0 = \left(\tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8},\ \tfrac{1}{8}\right)^T,$$

giving no preference a priori to any particular page.

6.3 Finding the Steady-State PageRank

6.3.1 (i) Iterative Method

We first apply the difference equation $x_{k+1} = H\, x_k$ from the uniform x_0. Below is our Python code (showing 15 iterations):

```
import numpy as np

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

x0 = np.ones(8) / 8
```

```
num_iters = 15
x = x0.copy()
for k in range(num_iters):
    x = H @ x

x_iter = x
print("Iterative PageRank after", num_iters, "iterations:", x_iter)
```

Iterative Output. After 15 iterations, we obtain:

$$x_{\text{iter}}^{(15)} \approx \big(0,\ 0,\ 0,\ 0.2109375,\ 0,\ 0,\ 0,\ 0.20703125\big)^T.$$

Notice that pages 4 and 8 are nonzero (around 0.2109 and 0.2070, respectively), summing to roughly 0.41797. The other pages have zero rank at this iteration. The total rank is less than 1 because probability that lands on pages with no out-links (e.g. page 2, page 5) never returns to the rest of the system.

6.3.2 (ii) Eigenvalue Method (Theorem 4.9)

By directly solving $(I - H)x^* = 0$ (plus $\sum_i x_i^* = 1$), we find the eigenvector associated with the eigenvalue 1. The code below demonstrates:

```
vals, vecs = np.linalg.eig(H)
idx = np.argmin(np.abs(vals - 1.0))  # eigenvalue near 1
x_eig = vecs[:, idx]
pagerank_eig = np.real(x_eig / np.sum(x_eig))

print("Eigenvalue-based PageRank:", pagerank_eig)
```

Eigenvalue Output. We obtain

$$x_{\text{eig}} = \big(0,\ 0,\ 0,\ 0.5,\ 0,\ 0,\ 0,\ 0.5\big)^T,$$

which sums to 1. That means pages 4 and 8 end up with all the surviving probability in the limit, since they form a strongly connected pair that links only to each other.

6.4 Ranking the Pages

Because pages 4 and 8 *alone* receive nonzero rank in the long run, they tie for the top spot at 0.5 each. Every other page goes to zero rank due to *dangling nodes* (pages 2, 5) or incomplete connectivity. Hence, the final ranking from highest to lowest is:

$$\{4,\ 8\} \quad > \quad \{1,\ 2,\ 3,\ 5,\ 6,\ 7\} \quad \text{(all zero)}.$$

The iterative method at 15 iterations shows partial accumulation on pages 4 and 8 (about 0.42 total), but with an additional 0.58 effectively "lost" to pages 2 or 5. *Further iterations* would drive x_{iter} closer to $(0, 0, 0, 0.5, 0, 0, 0, 0.5)$ as $k \to \infty$.

> **Conclusion for Figure 4.4:** In a pure follow-the-links model with no teleportation, pages 4 and 8 form a closed subcomponent; they eventually capture all rank that is not lost to the zero-out-degree pages. Thus the steady state is $(0, 0, 0, 0.5, 0, 0, 0, 0.5)^T$. This completes our analysis of Exercise 4.101.

7 Conclusion for Figure 4.4 (Exercise 4.101)

In analyzing the second web (Figure 4.4), we found several noteworthy features:

1. **Matrix Structure:** Two pages (2 and 5) had zero out-degree, leading to entire columns of zeros in the hyperlink matrix H. This causes rank to "drain away" if it ever reaches those pages.

2. **Closed Subcomponent:** Pages 4 and 8 form a strongly connected sub-web, linking to each other and thus retaining any probability that flows in. As a result, the eigenvalue-based PageRank solution (Theorem 4.9) assigns them all the long-term rank (0.5 each), while all other pages eventually drop to zero.

3. **Iterative vs. Eigenvalue Approach:** A finite number of iterations (e.g. 15) revealed partial convergence toward pages 4 and 8, but only after more iterations (or a direct eigenvector computation) does the entire rank accumulate on these two pages. This underscores how the eigenvalue method quickly pinpoints the true steady-state in a disconnected or dangling-node scenario.

4. **Rank Ordering:** In the final steady state, pages 4 and 8 share the top rank (0.5 each), and all other pages remain at 0. Consequently, the ordering from highest to lowest is

$$\{4,\ 8\} \ > \ \{1,\ 2,\ 3,\ 5,\ 6,\ 7 \text{ (all zero)}\}.$$

Future Considerations. In a real PageRank scenario, pages with zero out-degree ("dangling nodes") are typically handled by redistributing their probability mass or by introducing a *random surfer* component (Exercise 4.102). This modification ensures that no probability is permanently lost in the system and that every page can, in principle, be reached by random jumps. Thus, if we apply the damping factor α to create a matrix

$$G = \alpha\, H + (1-\alpha)\, \tfrac{1}{8}\mathbf{1}\mathbf{1}^T,$$

we would obtain a fully connected Markov chain that yields a more balanced rank distribution. Nonetheless, in this basic version without random jumps, Figure 4.4's structure forces all long-term rank onto the *closed* subset of pages 4 and 8, completing our analysis of Exercise 4.101.

Improving Website Rank

If a site owner wants to increase their page's importance, they would aim for *high-quality inbound links*—i.e., getting links from already-important pages. Another direction (Exercise 4.102) is to add a random-jump component so that the resulting PageRank does not penalize pages that are far from major hubs.

Acknowledgment

We wish to thank our classmates and teaching assistants for valuable input on debugging code and refining our linear algebra proofs.

References

[1] W. Sullivan, *"4 Linear Algebra — Numerical Methods"*, `https://numericalmethodssullivan.github.io/ch-linearalgebra.html`, accessed 2025.

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