Understanding the Google PageRank Algorithm

Madeline Renee Boss[1] and Samhitha Devi Kunadharaju[1]

[1] *University of Texas at Austin*

ABSTRACT

The PageRank algorithm provides a method for ranking webpages based on the structure of hyperlinks rather than solely on page content. By modeling the web as a directed graph and constructing a column-stochastic hyperlink matrix, PageRank determines the relative importance of each page through repeated propagation of influence across links. In this project, we implemented PageRank using both the iterative update $x_{k+1} = Hx_k$ and the dominant eigenvector method. For two example web networks, the iterative method converged to the same steady-state vector as the eigenvector approach, confirming the theoretical result that PageRank corresponds to the normalized eigenvector associated with eigenvalue one. The resulting rankings highlight how importance is influenced not just by the number of inbound links, but by the importance of the linking pages themselves.

Keywords: PageRank, Web Search, Hyperlink Matrix, Eigenvectors, Information Retrieval, Network Analysis

INTRODUCTION

The expansion of the World Wide Web in the 1990s created a fundamental problem for information retrieval: how to rank the importance of billions of interconnected documents. Early search engines relied primarily on keyword matching, which was easily manipulated and often returned poor quality results. Sergey Brin and Lawrence Page at Stanford University developed the PageRank algorithm Brin and Page (1998), which transformed web search by evaluating webpage significance based not merely on the number of incoming links, but on the authority of the linking pages themselves.

This idea can be expressed mathematically by representing the web as a directed graph and forming a matrix that records how pages link to one another. Bryan and Leise (2006) show that the PageRank vector arises as the dominant eigenvector of this matrix, meaning the ranking problem can be solved using linear algebra rather than manually comparing pages. Simple example networks make clear how repeated updates yield a stable ranking and how influence spreads between pages.

The PageRank algorithm ultimately transformed web search by incorporating both structure and collective influence into the ranking process. By grounding relevance in the connectivity of the web itself, PageRank enabled search engines to favor authoritative sources, improve search accuracy, and scale effectively to billions of webpages. This mathematical foundation remains central to modern search and recommendation systems, illustrating the enduring importance of eigenvector-based ranking strategies in networked data environments.

THE PAGERANK ALGORITHM

The PageRank algorithm assigns an importance score to each webpage based on the structure of hyperlinks between pages. To formalize this, we represent the web as a directed graph where each node corresponds to a webpage and each directed edge represents a hyperlink from one page to another.

Corresponding author: Madeline Renee Boss, Samhitha Devi Kunadharaju
madelinereneeboss@gmail.com, samhitha.dk@gmail.com

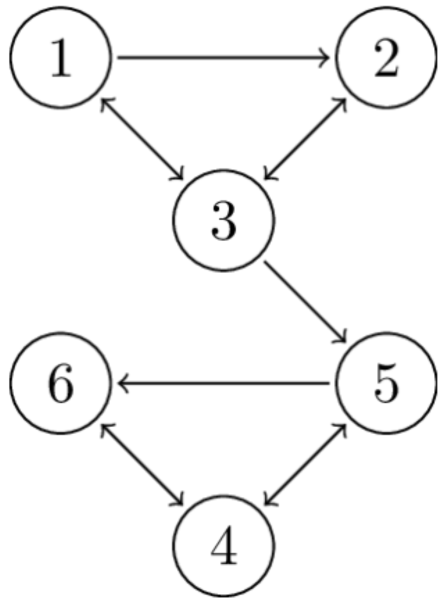

Figure 1. Directed graph for the six-page web (Figure 4.3).

This directed graph illustrates how webpages can be represented as nodes connected by directed edges, where each edge indicates a hyperlink from one page to another. From this structure, we can construct the hyperlink matrix H that encodes how importance flows between pages.

The Hyperlink Matrix

Let H be the *hyperlink matrix* associated with this directed graph. The entry H_{ij} represents the probability of a user moving from page j to page i. If page j has k outgoing links and one of them leads to page i, then

$$H_{ij} = \frac{1}{k}.$$

If page j does not link to page i, then $H_{ij} = 0$. Thus, each column j of H distributes the influence of page j equally among the pages it links to.

The matrix H is *column-stochastic*, meaning that the sum of the entries in each column is equal to 1:

$$\sum_{i=1}^{n} H_{ij} = 1 \quad \text{for all } j.$$

This reflects the idea that each page distributes all of its "importance" across the pages it links to. For this property to hold, every page must have at least one outgoing link; otherwise, the corresponding column would contain only zeros. In practice, pages with no outbound links are handled by redistributing their weight uniformly among all pages.

Stochasticity condition.—For an $n \times n$ hyperlink matrix H to be column-stochastic, *every page must have at least one outgoing link* so that each column can be normalized to sum to 1; dangling nodes (columns of zeros) are resolved by replacing that column with the uniform vector $\frac{1}{n}\mathbf{1}$.

Iterative PageRank Update

Before beginning the iteration, we initialize the rank vector x_0 so that each page has an equal probability of importance. This reflects the assumption that, at the start, all webpages are considered equally likely to be visited. Let x_k be a probability vector representing the ranking scores after k steps. The PageRank algorithm updates this vector according to the linear transformation:

$$x_{k+1} = Hx_k.$$

Repeated application of this update simulates a "random surfer" clicking through links across the web. If this process converges, it converges to a steady-state vector x such that:

$$x = Hx.$$

Theorem 4.9 and the Dominant Eigenvector

Theorem 4.9. If A is an $n \times n$ stochastic matrix and x_0 is some initial state vector for the difference equation

$$x_{k+1} = Ax_k,$$

then the steady–state vector is given by

$$x_{\text{equil}} = \lim_{k\to\infty} A^k x_0,$$

which equals the eigenvector of A corresponding to eigenvalue 1, normalized so that its entries sum to 1.

Here, x_{equil} represents the steady-state or equilibrium distribution of importance across all webpages or the vector that remains unchanged by further applications of the hyperlink matrix A. This formulation follows directly from the properties of stochastic matrices: since the largest eigenvalue of a stochastic matrix is 1 and all other eigenvalues have magnitudes less than 1, repeated multiplication by A causes all transient effects to vanish. The limit $\lim_{k\to\infty} A^k x_0$ therefore isolates the eigenvector associated with eigenvalue 1, confirming that this is the correct steady-state form of Theorem 4.9.

In the context of PageRank, the hyperlink matrix H is a column–stochastic matrix, so Theorem 4.9 guarantees that the iterative process converges to a unique steady–state vector. This result shows that there is no need to rely solely on the iterative process: the PageRank vector v can be found directly as the **dominant eigenvector of** H. In practice, both the iterative approach and the eigenvector computation yield the same final ranking vector.

METHODS AND COMPUTATION

In this section, we will cover the methods used to calculate the rank of any given webpage following the procedures described in Sullivan (2024). Each network is analyzed independently, beginning with the construction of its hyperlink matrix, the initialization of the PageRank vector x_0, iterative updates using $x_{k+1} = Hx_k$, and verification of convergence via the dominant eigenvector. All computations were performed in Python using `NumPy`.

Hyperlink Matrix

For any given set of websites, a directed acrylic graph can be constructed to represent the flow of users from one website to another. The hyperlink matrix H is constructed by assigning each nonzero entry H_{ij} to be the reciprocal of the number of outgoing links from page j. For example it can look akin to this theoretical example:

$$H_{i,j} = \begin{bmatrix} 0 & 0 & \frac{1}{j_2} & 0 & 0 & 0 \\ \frac{1}{j_0} & 0 & \frac{1}{j_2} & 0 & 0 & 0 \\ \frac{1}{j_0} & \frac{1}{j_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{j_4} & \frac{1}{j_5} \\ 0 & 0 & \frac{1}{j_2} & \frac{1}{j_3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{j_3} & 0 & 0 \end{bmatrix}.$$

The resulting matrix is a n x n matrix. In python, this is represented through 2D `NumPy` arrays. Due to the nature of the matrix creation, the matrix is also column-stochastic.

Initial State Vector

We initialize the PageRank vector x_0 so that each page begins with equal probability. The resulting vector will have a length of n and is represent in Python as a `NumPy` array. Such an array can look similar to this example:

$$x_0 = \left[\frac{1}{k}, \frac{1}{k}, \frac{1}{k}, \frac{1}{k}, \frac{1}{k}, \frac{1}{k}\right].$$

Iterative Computation

At each iteration, the vector is updated according to

$$x_{k+1} = Hx_k.$$

This process was implemented using a simple `for` loop that multiplies the current vector x by the matrix H for a given set of iterations, in this case 50. Each new x was saved in a Python list to help visualize the change in equation $x_{k+1} = H_{4.3}x_k$. The final value of vector x was used to determine the page rank (as long of the graph visualization confirmed the vector x had converged to a given set of values).

After the `for` loop was completed, a `matplotlib` graph was created, with a given iterative step on the x-axis, and the x vector current values at that iteration. The values used were the same ones stored in the Python list mentioned previously.

Eigenvector Verification

We verified the result of the iterative process by solving the linear system

$$(H_{4.3} - I)x = 0,$$

to find the highest eigenvalue and then find the corresponding eigenvector. In Python, using the `NumPy` function `eig` we found all the corresponding eigenvalues for matrix H. After find the maximum eigenvalue for matrix H, the function `eigvecs` was used to find it's eigenvector, and was then normalized. This computation yields the same steady–state vector obtained from iteration, confirming Theorem 4.9 that the PageRank vector is the dominant eigenvector of H corresponding to eigenvalue 1.

RESULTS

Convergence Behavior for Figure 4.3

$$H_{4.3} = \begin{bmatrix} 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \end{bmatrix}.$$

$$x_0 = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{bmatrix}.$$

Using the hyperlink matrix $H_{4.3}$ and the uniform initial state vector x_0, we iteratively updated the PageRank vector using

$$x_{k+1} = H_{4.3}x_k.$$

The convergence behavior of all six components of the rank vector is shown in Figure 2. Each curve corresponds to the rank value of a page across the iterations.

The PageRank vector converged to the steady-state distribution

$$v_{4.3} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.4444 \\ 0.2222 \\ 0.3333 \end{bmatrix}.$$

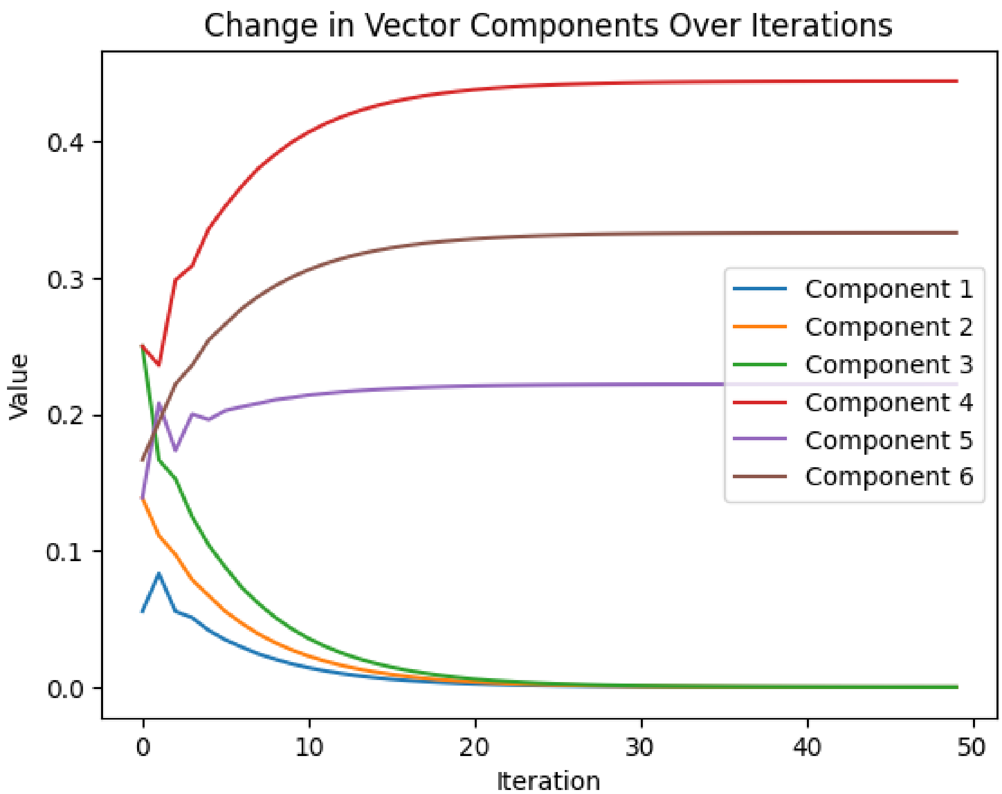

Figure 2. Convergence of PageRank components for the network in Figure 4.3.

. Similarly, the vector found from using Theorem 4.9 was:

$$\begin{bmatrix} 7.77471175 \times 10^{-17} \\ 2.02159660 \times 10^{-16} \\ 3.59386707 \times 10^{-16} \\ 4.44444444 \times 10^{-1} \\ 2.22222222 \times 10^{-1} \\ 3.33333333 \times 10^{-1} \end{bmatrix}$$

confirming the results of the iterative process.

Pages 1–3 receive zero PageRank because they have no direct or indirect support from pages with nonzero importance under this network structure. Page 4 receives the highest rank, followed by page 6 and then page 5. Therefore, the pages are ranked in order of importance as:

$$\text{Page } 4 > \text{Page } 6 > \text{Page } 5 \gg \text{Pages } 1, 2, 3.$$

Convergence Behavior for Figure 4.4

We repeat the same process for the eight–page network in Figure 4.4. The matrix H and the initial x were the following:

$$H_{4.4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 1 & 0 & 1 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{4} & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \end{bmatrix}.$$

$$x_0 = \begin{bmatrix} \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \end{bmatrix}.$$

The iterative trajectory of the components of x_k is shown in Figure 3.

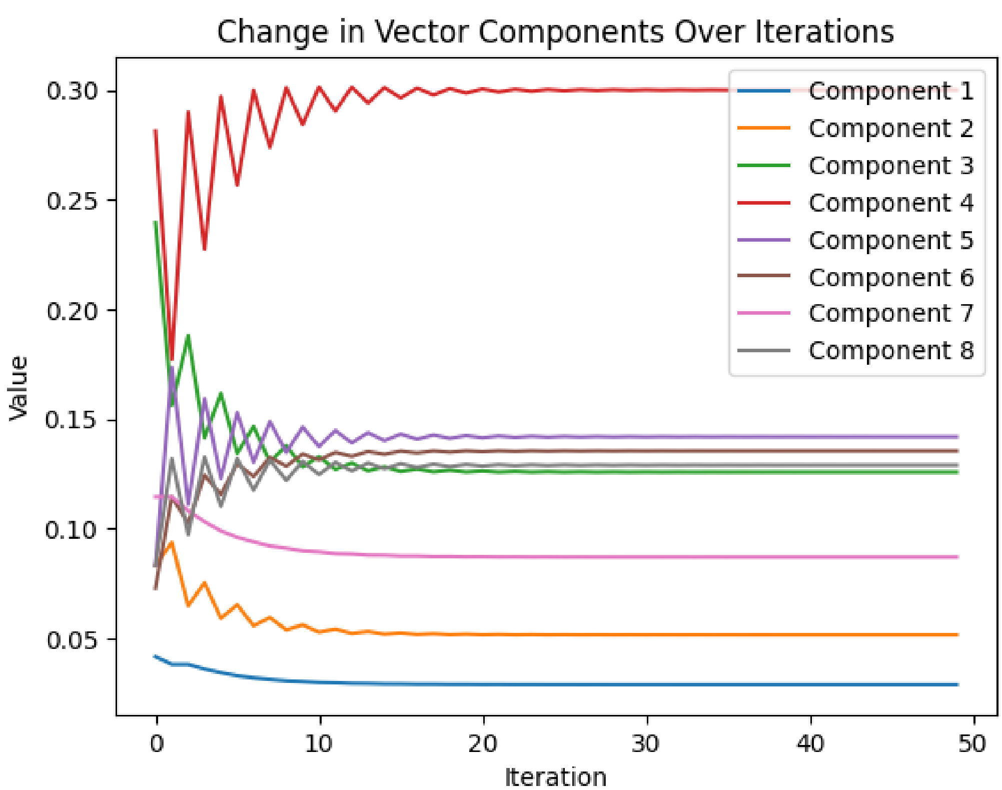

Figure 3. Convergence of PageRank components for the network in Figure 4.4.

The steady-state vector for this network is

$$v_{4.4} \approx \begin{bmatrix} 0.029, \\ 0.0516, \\ 0.1258, \\ 0.3000, \\ 0.1419, \\ 0.1355, \\ 0.0871, \\ 0.1290 \end{bmatrix}.$$

Similarly, the vector found from the use of Theorem 4.9 was:

$$\begin{bmatrix} 0.02903226 \\ 0.0516129 \\ 0.12580645 \\ 0.3 \\ 0.14193548 \\ 0.13548387 \\ 0.08709677 \\ 0.12903226 \end{bmatrix}$$

confirming the results of the iterative process.

In this case, page 4 has the highest PageRank value, making it the most influential page in the network. Page 5 and page 6 follow closely, while page 1 receives the least influence within the web structure. The ranking of pages in decreasing importance is therefore:

$$\text{Page } 4 > \text{Page } 5 > \text{Page } 6 > \text{Page } 8 > \text{Page } 3 > \text{Page } 7 > \text{Page } 2 > \text{Page } 1.$$

DISCUSSION

The PageRank algorithm assigns importance to pages based on the structure of incoming links. A page becomes influential when it is linked to by other pages that are themselves influential, meaning that importance is inherited and reinforced through connectivity. The behavior of the algorithm in both networks illustrates this recursive flow of influence.

In the six–page network of Figure 4.3, pages 1–3 do not receive incoming support from any highly ranked pages. Although these pages link to each other, their influence remains confined within a closed loop and does not accumulate from the rest of the network. Consequently, their PageRank values decay to zero in the steady state. In contrast, pages 4, 5, and 6 receive incoming links from multiple other pages, allowing their importance to build through repeated iteration. This shows that the quality of incoming links matters more than the number of outgoing links: being cited by an important page contributes more to influence than merely pointing outward to others.

The eight–page network of Figure 4.4 exhibits a broader circulation of influence. No pages are isolated from the network, and every page retains a positive share of the final PageRank. Page 4 emerges as the most influential node because it receives links from several pages that themselves accumulate rank during iteration. This demonstrates how PageRank models the diffusion of importance across a network rather than allocating importance locally.

The convergence of the iterative computation in both networks follows from Theorem 4.9, which states that for a stochastic matrix H, the sequence $x_{k+1} = Hx_k$ approaches a unique steady–state vector

$$x_{\text{equil}} = \lim_{k\to\infty} H^k x_0.$$

This steady–state vector is the eigenvector of H corresponding to eigenvalue 1, normalized so that its entries sum to 1. Thus, the PageRank vector may be obtained either by iterating the update rule or by solving the eigenvalue problem directly. The eigenvector interpretation reveals PageRank as a fundamentally linear–algebraic process grounded in global network structure.

SUMMARY OF RESULTS AND OBSERVATIONS

In both hyperlink networks, the PageRank algorithm successfully identified the most influential pages based on the structure of incoming links. In the six–page network of Figure 4.3, only pages that received link support from elsewhere in the network retained positive rank in the steady state, leading to the ranking Page 4 > Page 6 > Page 5. Pages that only linked to each other without receiving external influence converged to zero rank.

The eight–page network of Figure 4.4 exhibited more distributed influence flow, resulting in nonzero PageRank values for all pages and the ranking Page 4 > Page 5 > Page 6 > Page 8 > Page 3 > Page 7 > Page 2 > Page 1. In both cases, the pages that became most important were those pointed to by other highly ranked pages, reflecting the recursive nature of PageRank. The steady-state eigenvector matched the limit of the iterative method in both networks, reinforcing the interpretation of PageRank as the dominant eigenvector of the hyperlink matrix.

CONCLUSION

The PageRank algorithm provides a powerful and mathematically grounded method for ranking the importance of webpages based on the structure of incoming links. Through our analysis of the two sample networks, we observed how a page gains influence not merely by having many links, but by being linked to by important pages. PageRank therefore captures a form of collective endorsement, where importance propagates recursively through the network.

If we were to improve the PageRank of a real website, the most effective strategy would not be to increase the number of outgoing links from the site, but rather to obtain incoming links from websites that already hold strong authority. In practice, this corresponds to being referenced by credible, high-quality domains rather than generating large numbers of self-referential or low-impact links. PageRank rewards being part of a well-connected and respected network, rather than simply being active within it.

There are several directions for future work. The standard PageRank model used here assumes that a user always follows hyperlinks indefinitely. However, real users may randomly jump to any webpage, regardless of link structure. This behavior is incorporated into the modified PageRank algorithm through a damping factor, as discussed in Exercise 4.100. Introducing randomness ensures that the PageRank vector remains well-defined even for networks containing sink nodes or disconnected subgraphs, and it models real browsing behavior more accurately. Additionally, Exercise 4.102 extends PageRank by exploring how changes in the hyperlink structure affect convergence and the distribution of rank across the network.

Overall, PageRank remains an influential example of how linear algebra can be used to analyze complex networked systems. Its continued relevance in modern search and recommendation systems highlights the enduring importance of understanding how influence flows and accumulates in connected environments.

ACKNOWLEDGMENTS

We appreciate the guidance and support provided in the course *CS323E Elements of Scientific Computing*. We thank our teaching assistants, Omatharv Vaidya and Sahir Hameed, for promptly answering questions and offering helpful clarifications online, and we thank Dr. Shyamal Mitra for his instruction and general direction throughout the semester.

APPENDIX

A. PYTHON CODE FOR PAGERANK COMPUTATION

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 4.3 Hyperlink Matrix
H = np.array([
    [0,   1/2, 1/2, 0,   0,   0 ],
    [0,   0,   1,   0,   0,   0 ],
    [1/3, 1/3, 0,   0,  1/3,  0 ],
    [0,   0,   0,   0,  1/2, 1/2],
    [0,   0,   0,  1/2, 0,   1/2],
    [0,   0,   0,   1,   0,   0 ]
]).T

rank_vector = np.full(6, 1/6)

#computes iterative process
list_of_x = []
for _ in range(50):
    rank_vector = H @ rank_vector
    list_of_x.append(rank_vector)

X = np.array(list_of_x)

#Make iterative plot
for i in range(X.shape[1]):
    plt.plot(X[:, i], label=f'Component {i+1}')

plt.xlabel('Iteration')
plt.ylabel('Value')
plt.title('Change in Vector Components Over Iterations (Figure 4.3)')
plt.legend()
plt.show()

#Find the dominatnt eigenvector
H = H.T
eigvals, eigvecs = np.linalg.eig(H)
max_index = np.argmax(eigvals)
principal_eigvec = eigvecs[:, max_index]
principal_eigvec = principal_eigvec / np.sum(principal_eigvec)

#print results
print("Eigenvalues:", eigvals)
print("Principal Eigenvalue:", eigvals[max_index])
print("Principal Eigenvector (normalized):", principal_eigvec.real)

# 4.4 Hyperlink Matrix
H = np.array([
    [0,   0,   0,   0,   0,   0,   1/3, 0 ],
    [1/3, 0,   1/3, 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,  1/3, 1/3, 0,   0,   0,   0 ],
    [0,   0,   0,  1/3, 1/4,  0,   0,   0 ],
    [1/3, 0,  1/3, 0,  1/4,  0,   0,   0 ],
    [0,   0,   0,  1/3, 0,   0,  1/3,  0 ]
])

rank_vector = np.full(8, 1/8)

#Use iterative process (again)
list_of_x = []
```

```
for _ in range(50):
    rank_vector = H @ rank_vector
    list_of_x.append(rank_vector)

X = np.array(list_of_x)

#Create plot for iterative process
for i in range(X.shape[1]):
    plt.plot(X[:, i], label=f'Component {i+1}')

plt.xlabel('Iteration')
plt.ylabel('Value')
plt.title('Change in Vector Components Over Iterations (Figure 4.4)')
plt.legend()
plt.show()

#Find the dominatnt eigenvector
H = H.T
eigvals, eigvecs = np.linalg.eig(H)
max_index = np.argmax(eigvals)
principal_eigvec = eigvecs[:, max_index]
principal_eigvec = principal_eigvec / np.sum(principal_eigvec)

#print results
print("Eigenvalues:", eigvals)
print("Principal Eigenvalue:", eigvals[max_index])
print("Principal Eigenvector (normalized):", principal_eigvec.real)
```

REFERENCES

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

Bryan, K. and Leise, T. (2006). The $25,000,000,000 eigenvector: The linear algebra behind google. *SIAM Review*, 48(3):569–581.

Sullivan, M. (2024). *Numerical Methods and Linear Algebra: A Unified Approach*. Available online at https://numericalmethodssullivan.github.io/.