PageRank Computation via the Power Method

John Edwards

May 10, 2025

Email:jtedwards@udallas.edu

Abstract

This report explores the implementation of the Power Method to compute the PageRank of web pages. By representing the web as a directed graph and using the transition matrix derived from link structures, we apply iterative numerical techniques to approximate the dominant eigenvector. The method offers computational efficiency for large-scale problems. The results validate the method's practical use in ranking algorithms like Google's PageRank.

1 Introduction

PageRank is an algorithm developed by Google to rank web pages in their search results. It models the internet as a directed graph, where pages link to one another. The importance of each page is determined by computing a steady-state vector, which is the dominant eigenvector of a transition matrix that represents the long-term probabilities of landing on a page. Computing this eigenvector using direct methods can be computationally expensive. Instead, the Power Method, an iterative technique, is used for its simplicity and efficiency on large matrices.

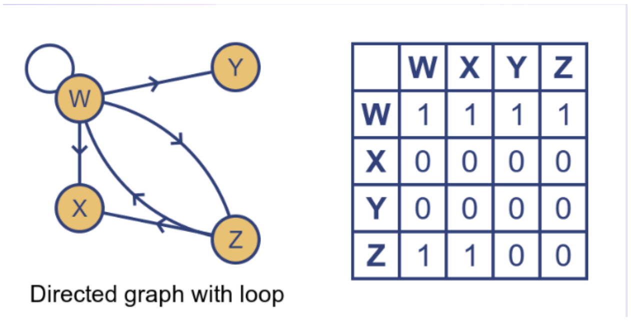
2 Methodology

We begin with an adjacency matrix representing links between web pages. This matrix is then converted into a stochastic matrix (transition matrix), where each column represents a probability distribution.

To compute the PageRank vector, we apply the Power Method:

1. Initialize a rank vector $R^{(0)}$.
2. Multiply $R^{(k+1)} = TR^{(k)}$.
3. Normalize $R^{(k+1)}$.
4. Repeat until convergence: $\|R^{(k+1)} - R^{(k)}\| < \epsilon$.

We implemented this algorithm in Python using NumPy for matrix operations. Damping was not applied in this implementation but can be added for more realistic modeling.



	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

Figure 1: Exammple Adjacentcy Matrix

3 Results

The computation of the PageRank vector relies on two matrices: the Link Matrix and the Transition Matrix.

Link Matrix

The Link Matrix is constructed based on the adjacency relationships between web pages. A value of 1 in a cell indicates the presence of a link from one page to another. Each column sums to 1 (representing a probability distribution of outbound links).

$$\text{Link Matrix} = \begin{bmatrix} 0.5 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.0 & 0.0 \end{bmatrix}$$

Transition Matrix

To handle the issue of dangling nodes (pages with no outbound links), and to ensure stochasticity, the Link Matrix is modified using a damping factor. This results in the Transition Matrix, which alters the original structure with a uniform probability to ensure all states are reachable.

$$\text{Transition Matrix} = \begin{bmatrix} 0.4625 & 0.4625 & 0.8875 & 0.8875 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \\ 0.0375 & 0.0375 & 0.0375 & 0.0375 \\ 0.4625 & 0.4625 & 0.0375 & 0.0375 \end{bmatrix}$$

This matrix ensures the Markov process is irreducible and aperiodic, guaranteeing convergence of the Power Method (Stanford PageRank). After 17 iterations, the method converges to a steady-state vector representing the final PageRank scores.

PageRank Vector

The resulting PageRank vector is as follows:

$$R = \begin{bmatrix} 0.6116 \\ 0.0375 \\ 0.0375 \\ 0.3134 \end{bmatrix}$$

Page	Rank
A	0.6116
B	0.0375
C	0.0375
D	0.3134

This steady-state vector indicates the relative importance of each page in the network.

This vector indicates that Page A has the highest rank, followed by D, B, and C, reflecting their relative importance in the network.

4 Discussion

The Power Method proves effective for computing dominant eigenvectors in sparse and large systems. Its simplicity makes it easy to implement, yet its convergence speed depends on the eigenvalue gap. For more realistic web graphs, incorporating damping and personalization vectors (as done in Google's actual PageRank) could improve stability and relevance. The method is still computationally efficient and simple to implement.

5 Conclusion

We successfully applied the Power Method to approximate the PageRank vector of a small web graph. This method illustrates how iterative techniques in numerical linear algebra can solve real-world problems like web ranking efficiently. The project highlights the connection between numerical methods and practical applications in computer science.

Problem 1: PageRank & Power Method Implementation

```
import numpy as np

def link_matrix(matrix):
#adjacentcy matrix for directed graph cpmverted into link matrix
    n = matrix.shape[0]
    L = np.zeros((n,n))
    #square matrix the same size as the adj matrix
    for j in range(n):
        links = np.where(matrix[:, j])[0]

        # finds the index of page connects to j
        n_link = len(links)
        if n_link > 0:
            L[links,j] = 1/n_link
            # 1/j  where the page j links page i
    return L

def transition_matrix(link_matrix,dampening, e_jumpvec):
    n = link_matrix.shape[0]
    T = np.zeros((n,n))

    for j in range(n):
        if np.sum(link_matrix[:,j]) == 0: # if no links then colum = e
            T[:, j] = e_jumpvec
        else:
            T[:,j] = (1-dampening) * link_matrix[:,j] + dampening*e_jumpvec
            # Tij = (1-d) * Lij + d * ei
    return T

def power_iteration(T, tolerance, iterations):
    n = T.shape[0]
    R = np.ones(n) / n

    for i in range(iterations):
        R_next = np.matmul(T,R)
        # T is already normalized
        if np.linalg.norm(R_next - R, 1) < tolerance:
        # check how much values changes as an absolute value
            print(f"Converge in {i+1} iterations")
            break
        R = R_next
    return R



matrix1 = np.array([
    [0, 0, 1, 0],
    [0, 0, 1, 0],
    [1, 1, 0, 1],
    [0, 1, 0, 0],
], dtype=float)

example_matrix = np.array([
    [1, 1, 1, 1],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [1, 1, 0, 0],
], dtype=float)
```

```python
def print_output(matrix):
    n = matrix.shape[0]
    L = link_matrix(matrix)
    print(f"Link Matrix: \n{L}")
    e = np.ones(n) / n
    # probability that a jump between link lands on another value in the matrix
    d = 0.15
    # probability a user will visit another link
    T = transition_matrix(L, d, e)
    print(f"Transition Matrix: \n{T}")
    tolerance = 1e-6
    iteration = 100
    R = power_iteration(T, tolerance, iteration)

    for i, val in enumerate(R):
        print(f"Page {i}: {val:.4f}")

print("Start of Matrix 1 Demo")
print_output(matrix1)
print("End of Matrix 1")
print("Start of Matrix 2 Demo")
print_output(example_matrix)
print("End of Matrix 2")
```

References

[1] Binod Pant et al. *Google PageRank Explained via Power Iteration.* Stanford University.

[2] GeeksforGeeks. *Power Method – Determine Largest Eigenvalue and Eigenvector in Python.*

[3] Alfio Quarteroni et al. *Scientific Computing with MATLAB and Octave.*