21241 Final Project

Haocheng Yang(hryang), Maxwell Chien (mychien)

December 3rd 2021

Introduction

PageRank is a website ranking algorithm created by Larry Page and Sergey Brin back in 1998, it was one of the first search algorithms used by Google. Due to its supremacy and efficiency, it was one of the main reasons behind Google's initial success. The PageRank algorithm ranks the webpages based on how popular a site is, namely how many other sites has links to it and note that the popularity of each site that has a link to it is taken into consideration as well by the algorithm. Indeed, his search algorithm used by Google proved to be a lot more accurate than many of its counterparts at the time.

Linear Algebra Background

Eigenvector and Eigenvalues

An $n \times n$ matrix A has eigenvector x if the linear transformation on x via A only stretches x without changing the direction. The scale of the stretch is called the eigenvalue. An eigenvalue v and its corresponding eigenvector λ has the following property:

$$Ax = \lambda x$$

Markov Matrix

A Markov Matrix M is a $n \times n$ real matrix such that it has all non-negative entries, namely $a_{ij} \geq 0$ for all $i, j \leq n$. In addition, all entries in a column of M must sum to 1:

$$M\mathbf{1} = \mathbf{1}$$

where $\mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$ and is $n \times 1$

Its is obvious from the above equation that a Markov matrix must have eigenvalue of 1 and all other eigenvalues are less than or equal to 1.

Markov Chain/Process[9]

The Markov Process is the action of iterating through some finite power of a Markov Matrix via matrix multiplication. is usually used to predict the probability after some finite steps of transition where each transition probability is independent of the probability any previous transitions. The transition probabilities can be modeled by a Markov Matrix where:

$$a_{ij} = P(j, i)$$

The entry at (i, j) is the transition probability of state j to state i
A Markov chain can be described as:

$$u = M^k * w$$

for some initial state vector w and final state vector u after k transitions and the $n \times n$ Markov matrix that encodes the transition probabilities.
Another property of Markov Matrices it that:

$$\Sigma_{i=1}^{n} u_i = \Sigma_{i=1} w_i,$$

Namely the sum of components of u is equal to the sum of components of w

Perron-Frobenius Theorem[4]

The Perron-Frobenius Theorem states that for a positive $n \times n$ real matrix then:

1.

The matrix has a unique positive largest eigenvalue, an eigenvalue with largest positive value and has algebraic multiplicity = 1

2.

The corresponding eigenvector can be chosen to have strictly positive values and greater than 0

Convergence

A diagonalizable positive $n \times n$ Markov Matrix M with an initial input state w has a steady or equilibrium state vector v, namely:

$$\lim(M^k * w)_{k\to\infty} = cv$$

where v is the eigenvector with corresponding eigenvalue of 1 and c is the coefficient of v in the unique representation of w using the eigenbasis of M.

Consider the eigenbasis: $\{v_1, v_2, v_3, \dots, v_n\}$ since M is diagonalizable and let v_1 be a the eigenvector with eigenvalue of $\lambda_1 = 1$

$$\begin{gathered} w = c_1v_1 + c_2v_2 \cdots + c_nv_n \\ M^kw = M^kc_1v_1 + M^kc_2v_2 \cdots + M^kc_nv_n \\ M^kw = c_1\lambda_1^kv_1 + c_2\lambda_2^kv_2 \cdots + c_n\lambda_nv_n \\ |\lambda_2|, |\lambda_3| \dots |\lambda_n| < 1 \text{ by Perron-Frobenius} \\ \lim_{k\to\infty} M^kw = c_1\lambda_1v_1 = c_1v_1 \end{gathered}$$

General Definitions

Directed Graph

A directed graph G is a collection of vertices V and edges E, consider an edge $e = (v_1, v_2)$, the edge points out from v_1 to v_2, therefore v_2 is reachable from v_1 but not necessarily the other way around.

Naive Random Walk Model[3]

This is a naive model that starts a person at some location, then person user will randomly walk on to another location that has an edge coming from his current location with equal probability. Namely, if the current location is connected to c other locations, then for each of the connected locations, the person walking there has a probability of $\frac{1}{c}$. However this model may not end up as a Markov Matrix since if a location has no outgoing edges, then the probability transition would all be zero, resulting in the corresponding column of the matrix having all zeros, which is by definition not a Markov Matrix

Random Surfer Model[5]

This is a model that is commonly used by the PageRank algorithm. Assuming that there is a surfer that starts at some website and will keep on surfing the network for some finite amount of time. At any given site with some links to other sites, the surfer has two options, each happening with some probability.

He may either visit one of the hyperlinks on the current site with uniform probability or the surfer may just visit some random webpage that is on the graph but does not need to be connected with the current site. Let us call the latter event a “teleportation”. This can be viewed as the surfer getting tired after following hyperlinks for some finite amount of time and decide to go to a random site on the network. This model now guarantees a Markov Matrix since it addresses the problem of a site having no outgoing edges as it now has the probability to make random jumps to some other site on the graph and thus resulting the transition probability for that column to sum to 1.

The PageRank Algorithm[3]

The naive random walk PageRank Algorithm is described as follows:

Input: A network of n sites, indexed $1-n$ represented as an adjacency list for a graph G

Output: A list that corresponds to the rank value of each site and can be sorted to display the most relevant site

For each vertex $v \in G$, let us call $L(v)$ the number of outgoing edges it has and let $N(v)$ be the set of neighbors reachable from v, namely $w \in N(v)$ if there is a link at site v to site w

Let $P : (v, w) \to [0, 1]$ be the probability function via:

$$P(v,w) = \tfrac{1}{L(v)} \text{ if } w \in N(v)$$
$$P(v,w) = 0 \text{ otherwise}$$

Namely, the surfer can only visit sites with a hyperlink on the current site, and the probability of visiting each site is uniform, which is just $\frac{1}{\#ofhyperlinks}$

Consider some site v, the probability that the user ends up on v on some iteration k is thus:

$$P_k(v) = \Sigma_{w \in N(v)} \tfrac{P_{k-1}(w)}{N(w)}$$

Namely, it is the sum for all w that has an edge to v, the probability that the user lands on w $\times$ the transition probability from w to v

Let us construct a $n \times n$ matrix M that models the transition probabilities

$$a_{ij} = 0 \text{ if } j \notin N(i)$$
$$a_{ij} = P(i,j) = \frac{1}{L(i)} \text{ if } j \in N(i)$$

Let the initial input vector be $v = \begin{pmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{pmatrix}$ where $v_i = \frac{1}{n}$, meaning that there is a equal probability that the surfer will start on each of the sites.

After iterating some finite times, the matrix will hopefully converge into a steady state, the steady state then holds information about the ranks of each of the website, it can be interpreted as percentage of time that the user spends on that site and the websites will be sorted based on rank and displayed to the user who performed the search.

The steady state can be calculated by taking the calculating the coefficient c of the eigenvector v that corresponds to eigenvalue 1 in the representation of the the initial state using the eigenvectors. Thus, by Convergence, the Markov chain will reach steady state of cv

However, the this naive version of the PageRank algorithm does not model the random surfer model since there was no modeling of the teleportation event, and the transition matrix M might not be guaranteed to be a Markov matrix at all. In the event that M is a Markov matrix, it is not guaranteed to converge to a final state since it is not guaranteed to be positive as some transition probabilities will be zero.

Dampening Factor and Dangling Node

A dangling node is a node on the graph that has no outgoing edges, therefore in a naive random walk algorithm, the transition probability would be zero. We have chosen to handle dangling nodes via the method of "teleportation" in the Random Surfer Model. We assume that at this node, the user will just jump to some site at random in the network and the probability that he jumps to any site in the graph is $\frac{1}{n}$ where n is the number of vertices, which means a user has equal probability of typing the URL of any address into his browser.

The dampening factor for the PageRank algorithm separates it from a naive random walk algorithm. The purpose of the dampening factor is to protect against the edge case that a website either points only to itself or nothing at all. Without it, all walks would converge and stay at that self/non pointing website. Additionally, it makes sure that the matrix is a positive markov matrix.

The dampening factor is represented by a scalar, d, which is usually 0.85. In the real world, $1 - d$ is meant to represent the probability that a user goes to a random website instead of continuing to follow the chain of links. To represent this in matrix form, this is applied by multiplying the adjacency matrix by d, and then adding $(1-d)/$ (num of websites) to every single element in the newly scaled matrix. By doing so, it essentially gives every website a chance to be stumbled upon randomly as well as making sure the matrix is a positive markov matrix.

$$\hat{M} = d * M + \frac{1}{n}B \text{ where } B \text{ is a } n \times n \text{ and } B = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

The new matrix $\hat{M}$ now becomes a positive Markov matrix that models the random surfer model, and $\hat{M}$ is guaranteed to have a unique largest eigenvalue by the Perron-Frobenius theorem, and the value is must be 1 since the eigenvalues of Markov Matrices are bounded. Therefore, we can then take a scalar-multiple of the unique eigenvector that corresponds to eigenvalue of 1 as the dominating vector for the end state and as the steady state that $\hat{M}$ converges to since all other eigenvectors eventually approach 0 as the Markov chain increases as their corresponding eigenvalues $|\lambda| < 1$

Code

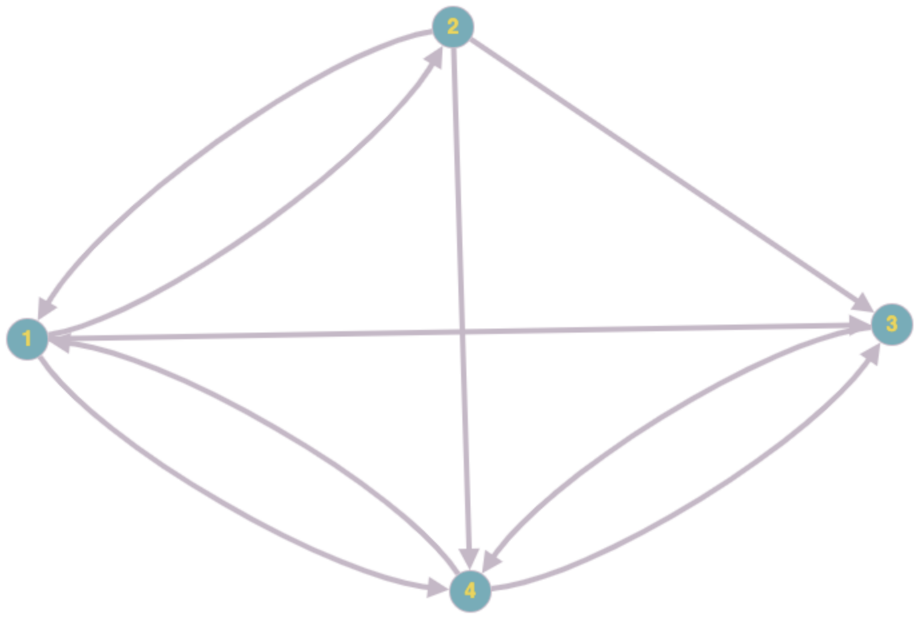
```
using LinearAlgebra

print("Please enter your input, with whitespaces separating different sites and commas separating links for a single site, if there a site has no links, enter NULL for that site, website will be numbered based on position in input \n ")
Array1 = String[] #Create an empty array of strings that will eventually be filled as the adjacency list of each site
for x in split(readline()) #Puts each adjacency list represented as strings in the input in the array
     push!(Array1,x)

end
node = length(Array1)
 #total number of sites
Markov = zeros(Float64,node, node)
#Initialize a n x n matrix with values 0

for i in eachindex(Array1)
    #for each site's adjacency list
    if (cmp("NULL", Array1[i])==0)
```

```julia
        #if the site has no linked list going outwards
            for j in eachindex(Markov[:,i])
                Markov[j,i] = 1/node
                #transition probability of this site to every other site is 1/n in accordance with the random surf model

            end
    else
        for x in split(Array1[i], "," )
            #if the site has some links
            connect = length(split(Array1[i], ","))
            neighbor = parse(Int64, x)
            Markov[neighbor,i ] = 1/connect
            #the transition probability is 1/(number of neighbors) if they are connected and 0 otherwise
        end
    end
end
d = 0.85
#dampening factor
matrix1 = fill(Float64(1), node, node)
#create the matrix B (n times n matrix where all of its entries are 1)
V = fill(1/node, node)
#initial state vector
M_hat = d * Markov + ((1-d)/node)*matrix1
#creates the new Markov matrix with respect to the dampening factor that models the random surf model
eigmatrix = eigvecs(M_hat)
#calculates the eigenvectors of M hat
coeff = inv(eigmatrix)*V
#caculates the coefficients of the eigenbasis representation of v via multiplying by the inverrse
c1 = coeff[node]
#gets the coefficient for the eigenvector with eigenvalue of 1
rank = c1*eigmatrix[:,node]
#calculates the vector that the Markov chain converges to

newrank = fill(Float64(1), node)

for i in eachindex(rank)
    newrank[i] = real.(rank[i])
end
```

```
bestwebsite = reverse(sortperm(newrank))
#sorts based on rank

for i in eachindex(newrank)
    println("Website ", i , " has rank ", newrank[i])
end
println("The websites are as follows (sorted based on
   pagerank) ")
for i in  eachindex(bestwebsite)
    println("Website ", bestwebsite[i], " is ranked
   number ", i)
end
```

Input

$[2,3,4][1,3,4][4][1,3]$ which represents the following network:



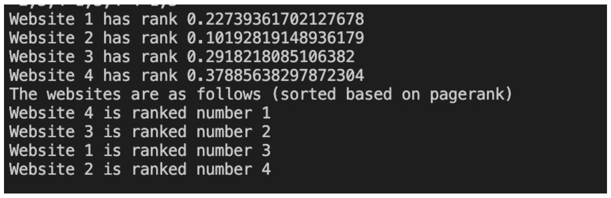
Which is represented by the transition matrix:

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

and the Markov Matrix:

$$\hat{M} = \begin{pmatrix} 0.0375 & 0.320833 & 0.0375 & 0.48125 \\ 0.320833 & 0.0375 & 0.0375 & 0.0375 \\ 0.320833 & 0.320833 & 0.0375 & 0.48125 \\ 0.320833 & 0.320833 & 0.85 & 0.0375 \end{pmatrix}$$

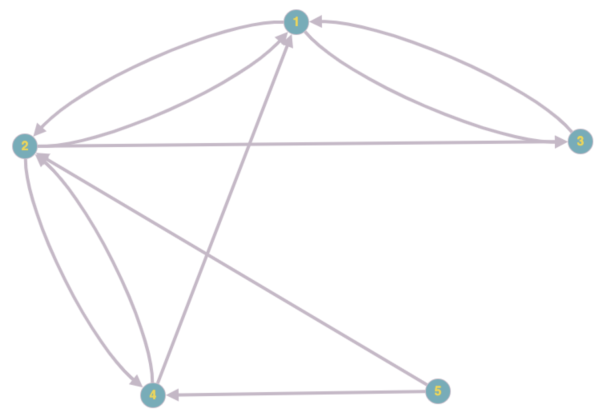
Output



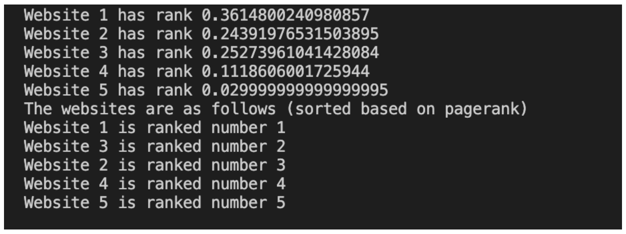
```
Website 1 has rank 0.22739361702127678
Website 2 has rank 0.10192819148936179
Website 3 has rank 0.2918218085106382
Website 4 has rank 0.37885638297872304
The websites are as follows (sorted based on pagerank)
Website 4 is ranked number 1
Website 3 is ranked number 2
Website 1 is ranked number 3
Website 2 is ranked number 4
```

Input

$[2,3][3,4,1][1][1,2][2,4]$ which represents the following network:



Output



```
Website 1 has rank 0.3614800240980857
Website 2 has rank 0.24391976531503895
Website 3 has rank 0.25273961041428084
Website 4 has rank 0.1118606001725944
Website 5 has rank 0.029999999999999995
The websites are as follows (sorted based on pagerank)
Website 1 is ranked number 1
Website 3 is ranked number 2
Website 2 is ranked number 3
Website 4 is ranked number 4
Website 5 is ranked number 5
```

Which is represented by the transition matrix:

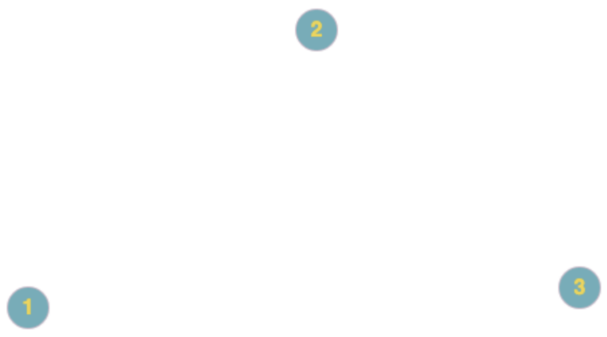
$$M = \begin{pmatrix} 0 & \frac{1}{3} & 1 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and the Markov Matrix:

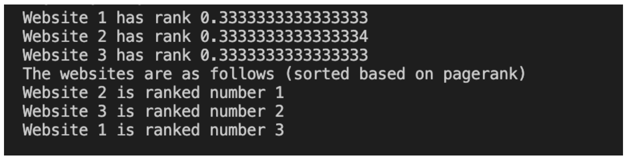
$$\hat{M} = \begin{pmatrix} 0.03 & 0.3133 & 0.88 & 0.425 & 0.03 \\ 0.425 & 0.03 & 0.03 & 0.425 & 0.425 \\ 0.425 & 0.3133 & 0.03 & 0.03 & 0.03 \\ 0.03 & 0.3133 & 0.03 & 0.03 & 0.425 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 \end{pmatrix}$$

Input

NULL, NULL, NULL which represents the following network:



Output



```
Website 1 has rank 0.3333333333333333
Website 2 has rank 0.3333333333333334
Website 3 has rank 0.3333333333333333
The websites are as follows (sorted based on pagerank)
Website 2 is ranked number 1
Website 3 is ranked number 2
Website 1 is ranked number 3
```

Which is represented by the transition matrix:

$$M = \begin{pmatrix} 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \end{pmatrix}$$

and the Markov Matrix:

$$\hat{M} = \begin{pmatrix} 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \end{pmatrix}$$

What we learned

Extensions

Overview

We also implemented the HITS algorithm. Similar to PageRank, HITS is a way to rank a list of website based on relevancy. However, in this case, HITS actually has two standards of ranking websites. It assigns and calculates two scores for each website, a **hub** score and a **authority** score. In the end, the algorithm returns two lists: one list ranks the websites depending on their hub score, and the other ranks the websites on their authority score[1][2].

Linear Algebra Background and Definitions

To implement the HITS algorithm, we mainly utilized the definition of matrix-vector multiplication, the properties of eigenvalues and the definition of their corresponding eigenvector, the fact that vectors can be written as a linear combination, and the steady state of a vector.

Additional Definitions

Vectors as a linear combination

Vectors can be represented as a linear combination of other vectors. In other words, $\mathbf{v}_0 = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n$, where c is a scalar. This, by the definition of eigenvectors and eigenvalues, $A^k\mathbf{v} = \lambda_1^k c_1\mathbf{x}_1 + \lambda_2^k c_2\mathbf{x}_2 + ... + \lambda_n^k c_n\mathbf{x}_n$.

What will be utilized later on is the fact that as $k \to \text{inf}$, $A^k\mathbf{v}_0$ will mainly consist of $\lambda_i c_i \mathbf{x}_i$, where λ_i is the largest eigenvalue, and $\mathbf{x}_i$ is the corresponding eigenvector. This is because as $k \to \text{inf}$, λ_i will grow the fastest.

HITS Algorithm Explained

Pseudocode:

```
Initialize adjacency matrix

Initialize hub and authority vectors

loop
```

```
    calculate new authority vector
    calculate new hub vector

return ranking of websites based on authority values,
   ranking of websites based on hub values.
```

To start off, we create an adjacency matrix that tracks which websites are connected to which. For example, the following matrix: $\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ for three websites, represented by a, b, c means:
$a \to b, c$
$b \to c$
$c \to a$
with the arrows representing direction of the graph. In other words, if $A_{ij} = 1, i \to j$.

After initializing the matrix, we initialize the authority and hub values for our websites, which we will represent by vectors $\mathbf{a}$ and $\mathbf{h}$. At the start, the vectors will all have an equal authority and hub score: $\mathbf{a} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \mathbf{h} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. And each value in these vector corresponds to a website, so in this case, the values for website a is the first value in each vectors, b is the second, c is the third.

After initializing our hub and authority scores, we begin the computations to find the equilibrium values for the hub and authority scores. The computation method is the same for each iteration.
To calculate the authority score for a website, we add up all of the hub scores of the websites that leads to it. In matrix form, this is represented by: $\mathbf{a} = A^T\mathbf{h}$.
To calculate the hub score, we add up all of the authority scores of the websites that it leads to. In matrix form, this is represented by: $\mathbf{h} = A\mathbf{a}$. We notice that we can simplify the equations to $\mathbf{a}_n = A^TA\mathbf{a}_{n-1}$ and $\mathbf{h}_n = AA^T\mathbf{h}_{n-1}$

From here we apply these equations, normalizing $\mathbf{a}$ and $\mathbf{h}$ after each computation to ensure that they converge, until they reach an equilibrium.

However, there's actually an alternative approach, which utilizes the property of eigenvectors and eigenvectors to not have to continuously multiply matrices together. The explanation and implementation for this method follows:

Our Implementation

```
using LinearAlgebra

adjMatrix = (any matrix made of 0s and 1s)

authValMat = adjMatrix' * adjMatrix # generate the
    matrix for the auth values
hubValMat = adjMatrix * adjMatrix' # generate the
    matrix for the hub values

authValEV = eigen(authValMat) # obtain the largest
    eigenvalue
tmp = argmax(authValEV.values) # placeholder for the
    index of the largest eigenvalue
authVals = authValEV.vectors[:, tmp] # find the
    eigenvector corresponding to the largest eigenvalue

hubValEV = eigen(hubValMat) # repeat steps for hub
    value matrix
tmp = argmax(hubValEV.values)
hubVals = hubValEV.vectors[:, tmp]

println()
print(reverse(sortperm(authVals))) # print out the
    websites
print(reverse(sortperm(hubVals)))
```

Input: adjMatrix $= \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$

Output: [3, 2, 1][1, 2, 3]

In our implementation, we only had to initialize the adjacency matrix, calculate A^TA and AA^T, and find the largest eigenvalue for each new matrix. Afterwards, we find the corresponding eigenvectors, which are the steady state representations of $\mathbf{a}_n$ and $\mathbf{h}_n$ as $n \to \inf$. We utilize **sortperm** to obtain an array of the incidices of the vector sorted from least to greatest based on the value at that indice. Then, we reverse the array to sort the list from greatest to least.

We end up with two arrays. The first one ranks the websites in order of greatest authority value to least authority value. The second ranks the websites in order from greatest hub value to least hub value.

It's important to note that the steady state obtained from the eigenvalue method differs slightly from the power iteration method, however the errors are roughly $1 * 10^{-5}$, thus they do not affect our overall answer.

Conclusion

On this project, we learned the secrets of success in Google's earlier years as we dissected the inner mechanics of the PageRank algorithm. It was very interesting to see how Markov matrices were utilized in real life as a way to rank web pages based on popularity. The dampening factor part was particularly ingenious as it introduces a new model that forces the transition matrix from the naive version of PageRank to become a positive Markov matrix where its properties were taken advantage of due to the existence of a steady state which can be interpreted as the rankings of each webpage. In addition, valuable experience was gained in working with Julia and applying concepts and techniques learned in class to some genuine real world situations was accomplishing and satisfying.

Works Cited

[1] https://www.math.ucdavis.edu/ saito/courses/167.s17/Lecture24.pdf
[2] https://www.youtube.com/watch?v=-kiKUYM9Qq8
[3] https://en.wikipedia.org/wiki/PageRank
[4] https://people.math.harvard.edu/ knill/teaching/math19b_2011/handouts/lecture34.pdf
[5] https://en.wikipedia.org/wiki/Random_surfing_model
[6] http://blog.kleinproject.org/?p=280
[7] https://stanford.edu/class/ee363/lectures/pf.pdf [8] https://en.ryte.com/wiki/Random_Surfer_Model
[9] https://en.wikipedia.org/wiki/Markov_chain