Efficient Eigensolvers & Their Applications

Erica Liu*, Kelly Rivera†, Jordan Fox‡

Summer 2020

Abstract

In this paper, we examine several numerical methods for solving matrix eigenvalue problems, efficient eigensolvers and their variants: Power Iteration, Inverse Iteration, QR Algorithm, concerning techniques as iteration method, inverse iteration, shifted inverse iteration method, Gram-Schmidt process, Householder transformations, and shifted QR decomposition. We discuss the normalization's necessity and conditional convergence property for Power Iteration Method and compare various eigensolvers' performance. We computationally experiment on the PageRank Algorithm and its variants.

*University of Michigan Ann Arbor, yuqingl@umich.edu
†St. Thomas University, kellymrivera4@gmail.com
‡Western Carolina University, jfox3@catamount.wcu.edu

Contents

1 Introduction

Scientific computation rises after 1960s, taking a fundamental role to understand complex system numerically, where calculating eigenvalues is a representative subject in this area. Solving eigenvalues for a given matrix is systematically required in multiple fields, such as Spectral graph theory, Quantum field theory. The corresponding scientific computing algorithms are referred as eigensolvers. Eigensolvers mainly are designed from two perspectives: sequences converging to eigenvalues, which is utilized in e.g. Power Iteration Method and its variants, and eigenvalue-revealing factorizations, which is applied in QR algorithms.

In the nineteenth century, Abel and others proved that there is no closed formula for roots of polynomials with degree 5 or more. This theorem gives us an insight about the difficulty of designing eigensolvers. Finding the roots of a polynomial could be equivalently transformed into solving the eigenvalues of its conpanion matrix, which implies any eigensolver must be iterative[18].

In this way, the run time for eigensolvers are largely determined by two factors: the convergence rate – how many iterations we need to reduce the error to a satisfying level, and the computational cost for each iteration – which is closely related to the matrix's structure and the underlying principle we choose. To improve the convergence rate, multiple eigenstructure-preserving transformations are applied involving shift, inverse, deflation, etc. To reduce the computation cost, we always apply some form of reduction on the given matrix before feeding it to eigensolvers, in which Hessenberg reduction is a popular preprocessing techinique.

Eigensolvers are used in a multitude of applications, such as the PageRank Algorithm, quantum mechanics, etc. Eigensolvers are based on the methods that solve for eigenvalues and their corresponding eigenvectors. Considering that there are many variations of eigensolvers, depending on the decomposition method, we have explored three that are widely used, Power Iteration method, QR decomposition, and inverse iteration. The three methods explored have individually shown to be more efficient in certain mathematical situations.

This paper outlines six Eigensolver processes and summarizes their procedures, when each is most efficient, and application results. The methods that were examined include the Power Iteration method, the gram-schmidt process, householder transformations, shifted QR decomposition, inverse iteration, and shifted inverse iteration method.

The organization of this paper is as follows. Section 2 begins with an example in Page Rank Algorithms. Section 3 covers the theorems and definitions that are essential to understanding the Eigensolvers investigated throughout this research. Section 4 discusses and explains each variation of Eigensolvers implemented, comprising of all methods, algorithms, and definitions required. Section 5 concludes the results from applying Power Iteration Method to the actual Page Rank problem.

2 A Motivating Example

For Page Rank, a collection of web pages can be represented by a directed graph, where the edges represent links between pages. In Figure 1, we have a directed graph with four nodes, representing four connected web pages. Note that an edge going from node four to node one is a link from page four to page one.

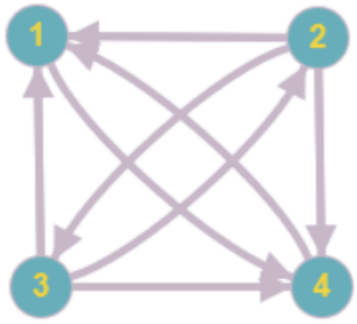

Figure 1: Four-page network

A web crawl of these pages would generate a matrix based on the importance scores of each page. However, it is easier to understand this matrix when first considering it in the form of a linear system based on Equation 1 below.

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j} \tag{1}$$

where x_k represents the importance score of page k, L_k is the set of pages with links to page k, and n_j is the number of links from page j [2].

The system of equations resulting from Equation 1 when constructed based on the graph in Figure 1 is as follows

$$\begin{aligned} x_1 &= \frac{x_2}{3} + \frac{x_3}{3} + \frac{x_4}{1} \\ x_2 &= \frac{x_3}{3} \\ x_3 &= \frac{x_2}{3} \\ x_4 &= \frac{x_1}{1} + \frac{x_2}{3} + \frac{x_3}{3} \end{aligned}$$

As we can now clearly see, the importance score of any one page is based on the importance score of other pages linked to it. Consider the equation for x_1: $x_1 = \frac{x_2}{3} + \frac{x_3}{3} + \frac{x_4}{1}$. All three of the other pages have a link to page one, so the importance scores of those pages will have an affect on the importance score of page one. Page two links to three different pages, so in the equation for x_1 we divide x_2 by three. Page three also links to three pages, so x_3 is divided by three as well. Page four has a link to page one but this is its only link, so x_4 is divided by one. The sum of these three terms is the importance score of page one.

Now we can take a look at the resulting matrix, matrix A.

$$A = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 1 & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

The importance scores of these webpages are now the elements in the eigenvector that corresponds to the dominant eigenvalue of this matrix. In the case of Page Rank, that is the eigenvector x from the equation $Ax = \lambda x$; where $\lambda = 1$. (Recall that the dominant eigenvalue for Page Rank adjacency matrices is always exactly one.) For our example, this is what the equation looks like.

$$\begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 1 & \frac{1}{3} & \frac{1}{3} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

To solve for this eigenvector x, we would utilize an eigensolver. Power Iteration would be the natural choice. However, the eigenvector x of this matrix A does not contain values simple enough to be useful in a basic example. Instead, consider a case where this was this was the resulting eigenvector:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 6 \\ 3 \\ 9 \end{bmatrix}$$

Therefore, the ranking from most important to least important pages is: page one, page four, page two, page three. This is useful, but even more useful would be these rankings as probabilities. To calculate these probabilities, we divide each element by the sum of all of the elements.

$$\frac{1}{30} * \begin{bmatrix} 12 \\ 6 \\ 3 \\ 9 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.2 \\ 0.1 \\ 0.3 \end{bmatrix}$$

If you were to take a random walk on the four web pages, these values are the probabilities that the walk would end on each respective page; ie. there is a ten percent chance you would end on page three and a forty percent chance you would end on page one.

3 Important Mathematical Facts

Definition 1. An eigenvalue decomposition of a square matrix A is a factorization

$$A = X\Lambda X^{-1}.$$

Theorem 1 (Jordan Canonical Form)**.** *For any square matrix A, there exists invertible matrix P which satisfies $PAP^{-1} = J$, where J is a block diagonal matrix*

$$J = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_p \end{bmatrix}$$

where each block J_i is a square matrix of the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix}.$$

Theorem 2. *[Theorem of Convergence] If the matrix has a unique dominant eigenvalue, the sequence*

$$\frac{x}{\|x\|}, \frac{Ax}{\|Ax\|}, \frac{A^2x}{\|A^2x\|}, \frac{A^3x}{\|A^3x\|}, \dots \tag{2}$$

converges to the eigenvector corresponding to the dominant eigenvalue of A.

Theorem 3 (QR Decomposition)**.** *If there exists a matrix A, $A \in \mathbb{R}^{n\times n}$, there exists an orthogonal matrix Q ($Q^TQ = QQ^T = I$) and an upper triangular matrix R, such that the product of Q and R equals A.*

4 Efficient Eigensolvers

4.1 Techniques in Efficient Eigensolvers

4.1.1 Eigenstructure-preserving transformations

The following techniques are applied with the intent to transform the matrix A so to increase convergence and efficiency when computed with an Eigensolvers. Each of the following methods of transformation are implemented in one or more of the Eigensolvers we explored. Throughout each of these transformations the eigenvectors remain constant to their original matrix A.

Shift A motivation to implement the Shift Transformation technique in a numerical algorithm is the possibility of improvement in both convergence speed and stability. In reference to the Power Method and the Inverse Iteration Method there are limitations to which eigenvalues and eigenvectors can be found. The Power method efficiently solves for the dominant eigenvalue while the Inverse Method solves for the smallest eigenvalue. If one where to attempt to find eigenpairs between the extremes it would require the implementation of a Shift. [3] In the case of Power Method we must first add the chosen σ to each diagonal entry in matrix A, to do so by adding the product of σ and $I_{n\times n}$ to matrix A resulting in the a new matrix to which one may now implement the algorithm on. $A_k = A - \sigma I \to (A - \sigma I)v = (\lambda - \sigma)v$.

Figure 2 shows the better performance of shifted QR algorithm than unshifted version regarding of the convergence rate.

Figure 2: Shifted QR Algorithm vs Unshifted

Inversion The inverse transformation technique is used to output the smallest Eigenvalue, $\frac{1}{\lambda_n}$, of matrix A and the corresponding eigenvector. $A^{-1}v = \lambda^{-1}v$. [17] This is evident in the Inverse Power Iteration Method, this eigensolver implements the transformation of inverse to matrix A before applying the Power Method. Refer back to **Algorithm 2** for the implementation of inverse in one of the researched eigensolvers.

Shifted Inverse The Shifted Inverse transformation method is the application of both a shift and inversion, $A_k = (A - \sigma I)^{-1} \rightarrow (A - \sigma I)^{-1}v = (\lambda - \sigma)^{-1}v$. The shifted inverse method is extremely useful when looking to find an eigenvalue between the two extremes. [17] The implementation of inverse with shift is present in **Algorithm 3**. The Inverse Iteration with Shift method is simply an application of a shifted inverse transformation onto the Power Iteration Method.

Figure 3: Shited Inverse: PowerMethod vs Rayleigh Quotient Iteration

4.1.2 Hessenberg Reduction and shift transformation experiment

As we introduced before, converting a matrix to Hessenberg form before applying eigensolvers to it would reduce the time complexity for each iteration. Here we conducted performance experiments on QR Algorithms with and without shift with controlled variable Hessenberg reduction.

Figure 4: Hessenberg Reduction and Shifts

4.2 Eigensolvers

4.2.1 Power Iteration Method

Inspired by Theorem 2, we could converge to the eigenvector corresponding to the dominant eigenvalue by calculating the limit of Sequence 2. Practically, we would iteratively calculate each term until the k^{th} term is close enough to the actual eigenvector corresponding to the dominant eigenvalue. Deciding until which iteration we should stop is tricky, as a result, we introduce Rayleigh Quotient.

Definition 2. The Rayleigh quotient of a vector $x \in R^m$ is the scalar

$$r(x) = \frac{x^T A x}{x^T x}.$$

$r(x)$ would minimize $\|Ax - \lambda x\|$, which functions like a natural eigenvalue estimate which makes x act like an eigenvector.

Algorithm 1 (Power Iteration Method)**.** *Given matrix $A \in R^{n \times n}$, we randomly initialize a unit vector $x \in R^n$. For each iteration, apply the matrix operator to previous vector with normalization. To measure how close this newly-generated vector is close to an eigenvector, we calculate its Rayleigh quotient.*

Algorithm 1 Power Iteration Method

Initialize a unit vector $x^{(0)}$
$\lambda_0 \leftarrow r(x^{(0)})$
while termination condition not satisfied **do**
  $x^{(k+1)} \leftarrow Ax^{(k)} / \left\|Ax^{(k)}\right\|$
  $\lambda_{k+1} \leftarrow r(x^{(k+1)})$
  if *Condition* **then**
    break
  end if
end while
return $x^{(k+1)}, \lambda_{k+1}$

Necessity of the normalization The normalization process is noticeable in Power Method. We scrutinized the necessity of applying normalization in each iteration and realized the normalization is related to the dominant eigenvalue's norm and algebraic multiplicity. Then we have the following claim.

Claim 1 (Normalization Necessity)**.** For any matrix A with dominant eigenvalue λ_1, satisfying

$$\lambda_1 = 1, \mu_A(\lambda_1) = 1,$$

where μ_A represents the algebraic multiplicity, the normalization in Power Iteration method is not necessary, i.e. the sequence

$$x, Ax, A^2x, \cdots \tag{3}$$

converges to the eigenvector corresponding to the dominant eigenvalue.
Further more, when $\lambda_1 > 1$, Sequence (3) diverges. When $\lambda_1 < 1$, Sequence (3) converges.

Proof. The proof is similar to the one for Theorem (2). First we would go through a simple case leveraging the diagonalizable matrix A by its eigenvalue decomposition.

Consider orthonormal eigenvectors $v_1, v_2, \cdots, v_n$ in X where X comes from the eigenvalue decomposition of A, we can write the initial vector x as

$$x = c_1v_1 + c_2v_2 + \cdots + c_nv_n.$$

The $(k+1)^{th}$ term of the sequence could be expressed as

$$A^kx = c_1\lambda_1^kv_1 + c_2\lambda_2^kv_2 + \cdots + c_n\lambda_n^kv_n$$

$$= \lambda_1^k(c_1v_1 + c_2(\frac{\lambda_2}{\lambda_1})^k + \cdots + c_n(\frac{\lambda_n}{\lambda_1})^k)$$

When k goes to infinity, this sequence would have different converge/diverge behaviors.

$$\lim_{k\to+\infty} A^kx = \begin{cases} \infty, & \lambda_1 > 1 \\ c_1v_1, & \lambda_1 = 1 \\ 0, & \lambda_1 < 1 \end{cases}$$

□

Example 1 (A Non-normalized Example)**.** We applied Power Method to a matrix A,

$$A = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix},$$

with dominant eigenvalue 2, which doesn't satisfy the condition in Claim 1, then the non-normalized Power Iteration will not converge. The Rayleigh Quotient difference for each iteration is plotted in Fig 5.

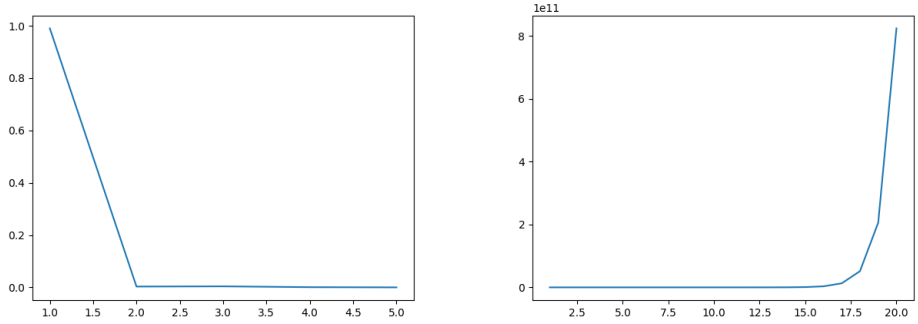

Figure 5: With and without Normalization with matrix [[1.5, .5],[.5, 1.5]]

Conditions for applying the Power Iteration Based on the proof for Power Iteration Method, we noticed that only when the dominant eigenvalue's geometric multiplicity is one, could the Sequence (2) converge to a single vector.

Claim 2 (Prerequisite)**.** When the dominant eigenvalue is simple, Power Iteration method could converge to the desired eigenvector.

Proof. Suppose $dim(Ker(A - \lambda_1 I)) = k$. Accordingly, matrix A's Jordan Canonical form is $A = VJV^{-1}$, where J is a block diagonal matrix,

$$J = \begin{bmatrix} J_1 & & & & \\ & \ddots & & & \\ & & J_k & & \\ & & & \ddots & \\ & & & & J_p \end{bmatrix}$$

where the first k blocks are one-dimensional square with the dominant eigenvalue λ_1

$$J_i = \begin{bmatrix} \lambda_1 \end{bmatrix}, i = 1, 2, \cdots, k,$$

where V contains all the eigenvectors,

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}, v_i \in^n .$$

For $x \in^n$, express x as a linear combination of $v_1, v_2, \cdots, v_n$ as

$$x = Va,$$

where $a \in^n$,

$$a = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}^T .$$

Then, the k^{th} iteration should be

$$\begin{aligned} \frac{A^k x}{\|A^k x\|} &= c(VJV^{-1})^k x \\ &= cVJ^k V^{-1} Va \\ &= cVJ^k a \\ &= c\lambda_1^k \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & \ddots \\ & & & & & J_p^k \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \\ \vdots \\ \\ a_n \end{bmatrix} \end{aligned} \tag{4}$$

where

$$J_i^k = \begin{bmatrix} (\frac{\lambda_j}{\lambda_1})^k & p(\frac{\lambda_j}{\lambda_1}) & & \\ & (\frac{\lambda_j}{\lambda_1})^k & p(\frac{\lambda_j}{\lambda_1}) & \\ & & & \\ & & \ddots & p(\frac{\lambda_j}{\lambda_1}) \\ & & & (\frac{\lambda_j}{\lambda_1})^k \end{bmatrix}, j = k+1, \cdots, p. \tag{5}$$

Thus, when k increases, we have

$$Eq(4) = C(a_1 v_1 + a_2 v_2 + \cdots a_k v_k).$$

Eq (4) will converge to the eigenvector corresponding to λ_1 since

$$v_i \in Ker(A - \lambda_1 I), i = 1, 2, \cdots, k.$$

□

Time complexity For each iteration, Power Method does matrix-vector multiplication with cost $O(n^2)$. The convergence rate is determined by h ow fast the error is reduced to nearly zero, which is indicated by Matrix (5). We could notice that the diagonal values are reaching zero linearly with rate $|\frac{\lambda_2}{\lambda_1}|$. Accordingly, the convergence rate is $O(|\frac{\lambda_2}{\lambda_1}|^k)$.

Power Iterartion Method has two main drawbacks. On the one hand, it doesn't tell us the information about eigenvalues other than the dominant one. On the other hand, its convergence rate is relatively low. We would solve those problems by introducing Inverse Iteration, which reveals the least dominant eigenvalue, and the first eigenstructure-preserving transformation technique we will be exposed to – shifting.

4.2.2 Inverse Iteration Method

There is more than one eigensolver that makes use of Theorem 2 as Power Iteration Method does. One such eigensolver is an adaptation of Power Method.

Algorithm 2 (Inverse Power Iteration Method)**.** *[14] If an invertible matrix $A \in \mathbb{R}^{n \times n}$ has eigenvalues $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ such that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$, then the matrix A^{-1} has eigenvalues $\{\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots \frac{1}{\lambda_n}\}$ such that $|\frac{1}{\lambda_1}| \leq |\frac{1}{\lambda_2}| \leq \ldots \leq |\frac{1}{\lambda_n}|$.*

When one applies the Power Iteration Method to A^{-1}, called Inverse Iteration Method, it converges to the eigenvector corresponding to the inverse of the eigenvalue of smallest magnitude, $\frac{1}{\lambda_n}$. This eigenvector is the same eigenvector that corresponds to λ_n.

The following pseudocode for Inverse Iteration[iv] is designed such that it outputs the eigenvalue of smallest magnitude of matrix A and its corresponding eigenvector, rather than the inverse of said eigenvalue. This is achieved in the computation of λ_k where matrix A is used instead of A^{-1}.

[iv] To see the convergence of Inverse Iteration when applied to an example matrix, see Appendix B

Algorithm 2 Inverse Iteration Method

Initialize a unit vector $x^{(0)}$
$\lambda_0 \leftarrow (x^{(0)})A(x^{(0)})$
while termination condition not satisfied **do**
  Solve $A(x^{(k)}) = x^{(k-1)}$ for x^k
  $x^k \leftarrow x^{(k)} / \left\|x^{(k)}\right\|$
  $\lambda_k \leftarrow (x^k)A(x^k)$
  $k \leftarrow k + 1$
  if *Condition* **then**
    break
  end if
end while
return x^k, λ_k

The Inverse Iteration and Power Iteration Methods are both useful for finding the individual eigenvalue and eigenvector pairs, eigenpairs, that they are designed to find; however, when one needs a different eigenpair, another iterative method needs to be used. One of these methods is an adaptation of Inverse Iteration, called Inverse Iteration with Shift.

Algorithm 3 (Inverse Iteration with Shift)**.** *[14] Note that for any $\mu \in \mathbb{R}$, the matrix $B = A - \mu I$ has eigenvalues $\{\lambda_1 - \mu, \lambda_2 - \mu, \ldots \lambda_n - \mu\}$. If a μ close to an eigenvalue λ_k of A is chosen, it ensures that $\lambda_k - \mu$ is the eigenvalue of smallest magnitude of matrix B. Then by applying inverse iteration on B, an approximation of λ_k and its corresponding eigenvector can be found.*

When one knows or has an idea of the eigenvalue they are interested in, they can use this method quite effectively. One just needs to choose a μ close to the desired eigenvalue, inverse iteration with shift will converge to the corresponding eigenvector, and they will have their desired eigenpair.

Algorithm 3 Inverse Iteration with Shift

Initialize a unit vector $x^{(0)}$
$\lambda_0 \leftarrow (x^{(0)})A(x^{(0)})$
$B \leftarrow$(A - μI)
while termination condition not satisfied **do**
  Solve $Bx^{(k)} = x^{(k-1)}$ for $x^{(k)}$
  $x^{(k)} \leftarrow x^{(k)} / \left\|x^{(k)}\right\|$
  $\lambda_k \leftarrow (x^k)A(x^k)$
  if *Condition* **then**
    break
  end if
  $k \leftarrow k + 1$
end while
return $x^{(k)}, \lambda_k$

The problem arises that one needs an eigenpair other than those including the eigenvalues of smallest or largest magnitude, but they do not have an idea of what shift to choose. The following theorem allows one to find valuable information about the eigenvalues of a matrix that will assist in choosing an appropriate shift.

Theorem 4 (Gershgorin Circle Theorem)**.** *[1] Given an square matrix $A \in \mathbb{C}^{n \times n}$ there exists $i \in \{1, 2, \ldots n\}$ such that every eigenvalue of A satisfies:*

$$|\lambda - a_{ii}| \leq \sum_{i \neq j} |a_{ij}| \qquad i \in \{1, 2, \ldots n\} \ [1]$$

Remark (Gershgorin Circle Theorem). Though the Gershgorin Circle Theorem applies to complex matrices as well as real matrices, within this paper we are only interested in real matrices; thus, all examples will include real matrices.

From Theorem 4 arises the following definition of a Gershgorin disk.

Definition 3 (Gershgorin Disk)**.** [1] The Gershgorin disks of a matrix $A \in \mathbb{C}^{n\times n}$ define the area in which the eigenvalues of A exist. Matrix A has n Gershgorin disks, each one centered at a diagonal element a_{ii} with radius $r_i = \sum_{i\neq j} |a_{ij}|$.
The i-th Gershgorin disk is given by the equation

$$D_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \leq r_i\}$$

Remark. The center of a Gershgorin disk is (real component of a_{ii}, complex component of a_{ii}). Therefore, if a_{ii} is real, the center of the corresponding disk will be $(a_{ii}, 0)$

Example 2 (Gershgorin Theorem)**.** Given the following matrix $A \in \mathbb{R}^{n\times n}$, the Gershgorin disks are computed as follows.

$$A = \begin{bmatrix} 14 & 1 & -8 \\ 0 & 0.2 & 0 \\ -5 & -2 & -8 \end{bmatrix}$$

The given matrix A is a 3×3 matrix; therefore, there will be three Gershgorin disks, each centered at a diagonal entry. The radii of these disks will be the sum of the absolute values of the non-diagonal entries in each row. The center and radius of each disk are outlined below.

$$\begin{aligned} &D_1\text{: } center = 14,\ radius = |1| + |-8| = 9 \\ &D_2\text{: } center = 0.2,\ radius = |0| + |0| = 0 \\ &D_3\text{: } center = -8,\ radius = |-5| + |-2| = 7 \end{aligned}$$

Figure 6 depicts the Gershgorin disks of A and valuable information about the eigenvalues is clearly visible. The three disks are distinct; therefore, there will be exactly one eigenvalue in each disk. The disk centered at 0.2 has a radius of zero, so the only option for an eigenvalue in this disk is 0.2.
With this information about the eigenvalues, one can easy choose an efficient shift for use in Inverse Iteration with Shift.

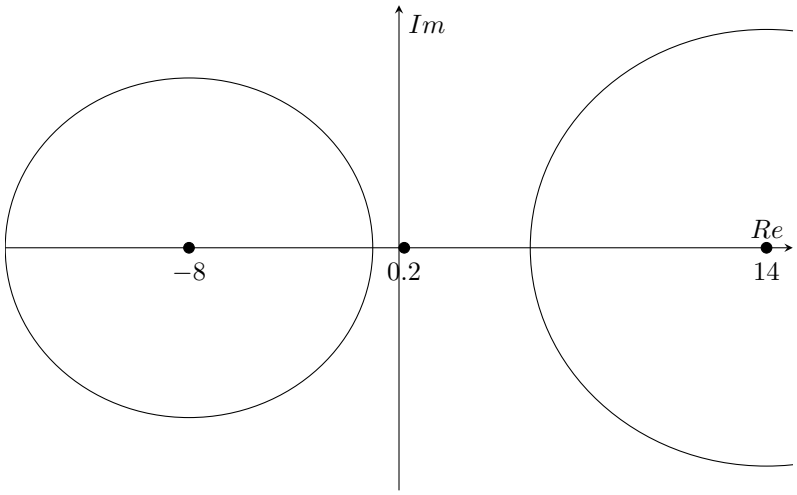

Figure 6: Gershgorin disks of matrix A

Remark. The example above had three distinct Gershgorin disks, but this is not always the case. In the case that are overlapping disks, the number of eigenvalues within the union of the disks will be exactly equal to the number of overlapping [12]. A general example can be seen in Figure 7. This figure depicts two overlapping Gershgorin disks, each is centered on the real axis. There will be exactly two eigenvalues within the entire, combined area of those disks.

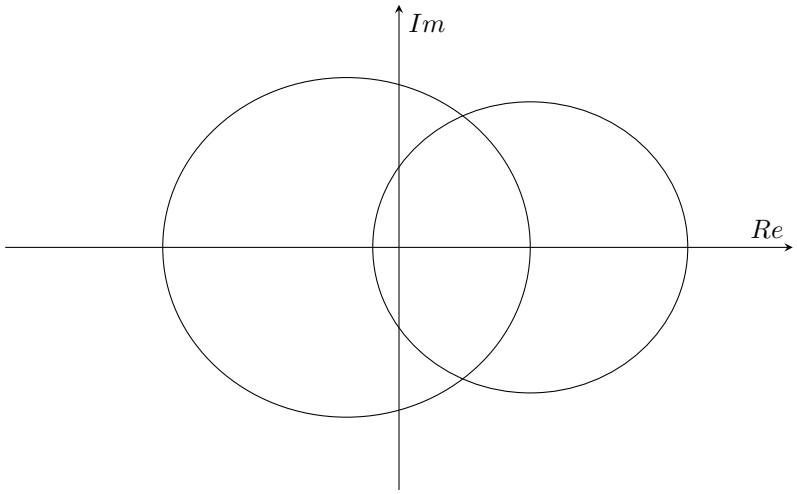

Figure 7: Two overlapping Gershgorin disks

4.2.3 QR Algorithm

The QR Algorithm implements Theorem 3, by executing a loop of the following steps: compute the QR decomposition of matrix A_k, producing Q_k and R_k (whose product is equal to matrix A_k), set A_{k+1} equal to the product of R_kQ_k, and iterate till the convergence of A_k into an upper triangular matrix [16]. This convergence is possible because each iteration on matrix A_k is similar, therefore their eigenvalues are equal. Conversely to the Power Iteration Method and the Inverse Iteration Method, the QR algorithm instead computes all eigenvalues and eigenvectors of a given matrix A_k. We can note that once the algorithm converges to the final matrix A_k, the eigenvalues will be the elements contained within the main diagonal. We explored three different QR decomposition computational methods for the implementation of the QR algorithm. We introduce the Gram-Schmidt process, the Householder transformations, and the use of Shift in a QR decomposition [20].

Algorithm 4 QR Algorithm

Set $A = A_0$
for k = 1, 2, 3, ...(til convergence) **do**
  $A_k = Q_kR_k$
  $A_{k+1} = R_kQ_k$
end for
return x, λ

Definition 4. The Gram-Schmidt (GS) process is one of several methods employed to compute QR decomposition [20]. The GS process functions by obtaining an orthogonal projection q_n for each column vector a_n in a matrix, then subtracts each projection from the prior projection. This results in a vector that must then be divided by its length, producing the desired unit vector. We must keep in mind that for the first column vector there is not previous projection, resulting in $a_1 = v_1$ [13].

Algorithm 5 Modified Gram-Schmidt [16]

for j = 1, 2, ..., n **do**
  Set $v = A(:, j)$
  for i = 1, 2, ..., $j - 1$ **do**
    $R(i, j) = Q(:, i)^T \cdot v$
    $v = v - R(i, j) \cdot Q(:, i)$
  end for
  $R(j, j) = \|v\|$
  $Q(:, j) = v/R(j, j)$
end for
return Q, R

$$v_{k+1} = a_{k+1} - (a_{k+1} \cdot e_1)e_1 \cdots (a_{k+1} \cdot e_k)e_k \Longleftarrow e_{k+1} = \frac{u_{k+1}}{||u_{k+1}||}$$

The produced Q matrix is orthogonal and a set of all orthogonal vectors computed, seen in the equation above represented as e_k. By computing each orthogonal vector e_k, we proportionately compute each column of Q. The upper triangular R matrix is the product of the given matrix A and the matrix Q transposed [16].

$$A = [a_1|a_2|\cdots|a_k] = QR = [e_1|e_2|\cdots|e_k] \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_k \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_k \cdot e_2 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & a_k \cdot e_k \end{bmatrix}$$

Definition 5. The Householder transformations implements orthogonal transformations that convert a vector A into a unit vector B parallel to A. The Householder reflection matrix form is as follows, $H_k = I - 2v_k v_k^T$, v denotes the unit vector previously mentioned. This method computes Q as a total product of all H reflections, Q is required to continue iterating until the R is produced as an upper triangular matrix. Conversely to the Gram-Schmidt method Householder computes R as a product of Q matrices produced. [10] The implementation of the Householder transformation in the QR Algorithm may prove to be less computationally expensive; note that this method transforms all elements under the main diagonal element (a_{ii}) into zeros by multiplication [7].

$$H_k = I - 2v_k v_k^T$$
$$^{\text{v}}R = (H_k)A$$
$$Q = H_n \cdots H_2 H_1$$

Definition 6. The application of shift is intended to increase the convergence speed and lessen the computational expense. A shift is applied to the QR algorithm via subtraction when computing for the Q and R matrices. The shift is determined to be $\mu_k = a_{nn}^k$, an eigenvalue estimation, so to allow $A_{kn,n1}$ to swiftly converge to zero. I represents an identity matrix the same size as A_k. This in turn leaves the A_k main diagonal to contain all eigenvalues. The eigenvalues are shifted by the values of μ, so to avoid errors we add μ to the product of RQ when computing A_{k+1}. The Eigenvectors remain constant throughout this process.

Algorithm 6 Shifted QR Algorithm

Set $A = A_0$
for k = 1, 2, 3, ...(til convergence) **do**
  $\mu_k = a_{nn}^k$
  $Q_k R_k = A_k - \mu I$
  $A_{k+1} = R_k Q_k + \mu I$
end for
return x, λ

[v]In practice R is the product of the final H_k and A, so that R becomes an upper triangular matrix.

Notes for Application For QR Algorithm implementing the Gram-Schmidt process, matrix linear independence is required [20]. For implementations of the QR Algorithm with Shift, the convergence is dependent of $\frac{|\lambda_n|}{|\lambda_{n-1}|}$. The QR algorithm may be applied to any matrix A with linear independence, though take note that not every matrix A may return an invertible R [4]. For a given matrix A that is square, then Q^T must equal Q^{-1} [16].

4.3 Hilbert Matrix: Accuracy Dilemma

In 1894, David Hilbert introduces a special type of square matrix, with entries being the unit fractions.

$$H_{ij} = \frac{1}{i+j-1}.$$

It's obvious that Hilbert matrix is symmetric and it also has some good properties. The entries of each Hilbert matrix inverse are integer, with closed form,

$$H_{ij}^{-1} = (-1)^{i+j}(i+j+1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2.$$

Hilbert matrix is well-known ill-conditioned for scientific computation. Specifically, for eigensolvers, the common phenomenon is that for eigenvectors x returned by conputational eigensolver, the residue has converged under the specified convergence condition, while the actual distance for estimated x and real eigenvector v is still proportionally large, i.e.

$$\left\| \frac{Hx}{\|Hx\|} - \frac{\lambda x}{\|\lambda x\|} \right\|_2 << \left\| \frac{x}{\|x\|} - \frac{v}{\|v\|} \right\|_2$$

According to Henry E. Fettis and James C. Caslin [5], we obtained the actual eigenvectors for Hilbert matrices with order 3 to 10. To measure how far the estimated eigenvector is from the actual eigenvector, we calculated residue $\|Hx - \lambda x\|$, distance $\left\| \frac{x}{\|x\|} - \frac{v}{\|v\|} \right\|$, residue_real $\|Hx - \lambda_v x\|$, where x is the eigenvector returned by our eigensolver, λ_v is the actual dominant eigenvalue, and v is the eigenvector corresponding to λ_v.

In Figure 8, it is well-noticed that the residue is much smaller than distance or real-valued residue. Also note that the y axis in Fig 8 is log-scaled, which emphasizes how large the accuracy error is when feeding hilbert matrices to those eigensolvers.

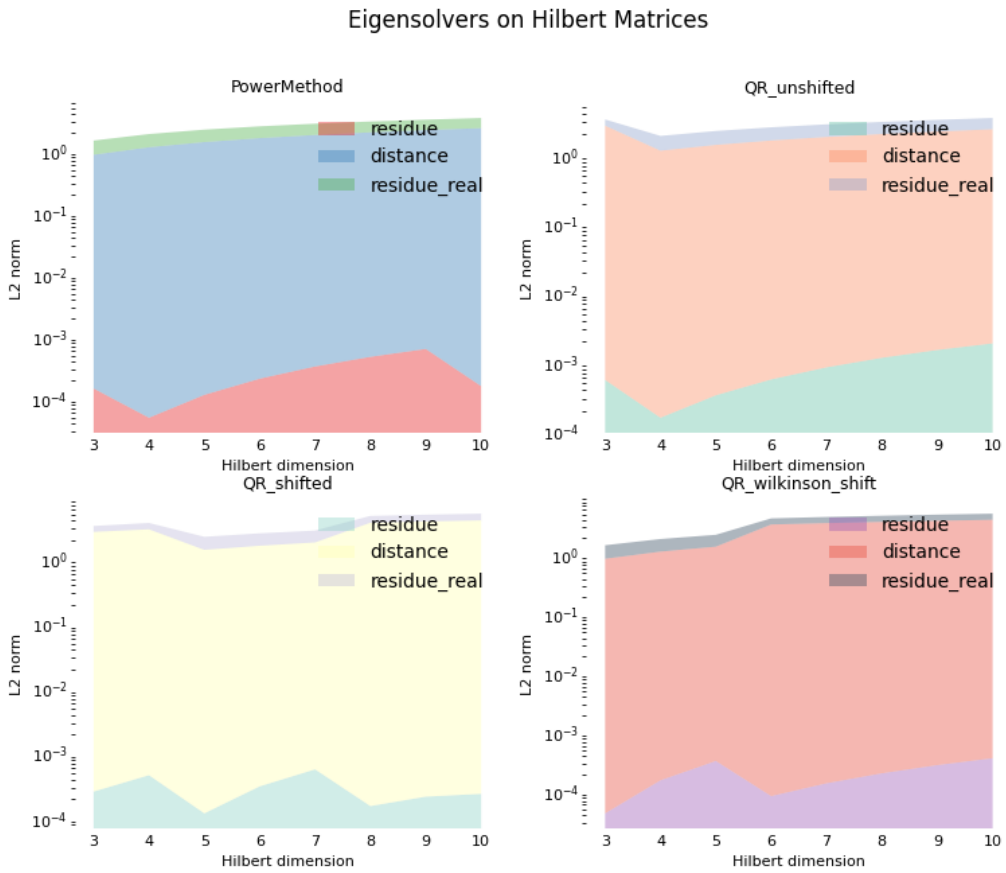

Figure 8: Applying eigensolvers on Hilbert matrices

5 Page Rank Algorithms

Page Rank is an application of eigenvalues and eigenvectors that is used to rank the importance of web pages, like Google's PageRank. This process begins with a chosen collection or number of web pages. A crawl is performed on these pages to generate an adjacency matrix from which the the importance scores are found. This matrix is a stochastic matrix with the dominant eigenvalue equal to exactly one. The eigenvector corresponding to that dominant eigenvalue is used to rank the chosen web pages. If one divides each element of the eigenvector by the sum of all of the elements, one gets a very useful vector. This new vector is a vector of probabilities; if one takes a random walk on the chosen web pages, each element is the probability that one ends up on the corresponding page [2].

An eigensolver is used in this application to find the relevant eigenvector. Power Iteration is a natural choice of eigensolver for this application. As was covered in 4.2.1, this method returns only the dominant eigenvalue and its corresponding eigenvector. These are the only eigenthings necessary for Page Rank, making Power Iteration an efficient eigensolver for Page Rank.

QR Algorithm can be used as it returns all eigenvalues and their corresponding eigenvalues, including the relevant eigenpair. The QR Algorithm may not be the most efficient eigensolver for PageRank for two reasons. Considering that this application requires only the dominant eigenpair, the QR Algorithm computes unused eigenpairs. In addition, the QR Algorithm can at times be unstable when implementing the Gram-Schimdt process because of the multitude of divisions. On the other hand, Inverse Iteration is not at

all useful for this application. While it is possible to chose a shift that results in Inverse Iteration with Shift converging to the desired eigenvector, but this would be extremely inefficient.

5.1 Network Markov Chain

Applying Power Iteration Method requires dominant eigenvalue to be somehow "unique", which reminds us of Perron-Frobenius theorem.

Theorem 5 (Perron-Frobenius theorem)**.** *A real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector can be chosen to have strictly positive components.*

The requirements "real positive matrix" mentioned above provides a great connection to probability matrices, i.e. Markov transition matrix.

Obtaining the adjacency matrix A of the network graph, we could calculate the markov transition matrix P accordingly as

$$P_{ji} = A_{ij}/outdegree(j).$$

Only when all the entries are positive, could Perron-Frobenius theorem hold. If we have nodes with no outdegree, the whole column would be 0, which violates the condition so that we cannot guarantee the existance of unique largest eigenvalue 1. Here comes a problem.

Problem. How to deal with dangling nodes?

If we have dangling nodes, does the eigenvector corresponding to the dominant eigenvalue still fit here? Dangling nodes would form a whole-zero column, which makes the matrix not stochastic anymore. We would do the following modification to maintain the stochastic-ness property of P.

$$d_i = \begin{cases} 1, & \text{if } outdeg(i) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$D = \vec{d} \cdot \vec{v}^T$$

$$P' = P + D$$

After we modify our matrix to make it stochastic again, we are not clear about whether the matrix could converge, i.e. has a unique stationary probability distribution. This concern is taken care of by Ergodic Theorem.

Theorem 6 (Ergodic Theorem for Markov chains)**.** *The Markov chain defined by P' has a unique stationary probability distribution if P' is aperiodic and irreducible.*

Since matrix P' is derived from the directed graph, P' is aperiodic. To make P' irreducible, we have to add a new set of complete outgoing transitions, with small transition probabilities, to all nodes, creating a complete and strong connected transition graph.

$$\vec{e} = [1]_{n \times 1}$$

$$E = \vec{e} \cdot \vec{v}^T$$

$$P'' = cP' + (1 - c)E$$

5.2 Adaptive Page Rank Algorithm

It has been observed that the run time of the Power Iteration method can be significantly reduced by eliminating redundant computation. According to Sepandar, Taher, and GeneGoluba [11], the convergence rates for different page is different during the application of Power Method, in which most pages converge in 15 iterations.

We don't want to recompute the PageRanks of the pages that have already converged, nor its contribution to other pages. We would refactor the matrix by grouping the converged pages and not-yet converged pages together.

$$\vec{x}^{(k)} = (\vec{x}_N^{(k)} \vec{x}_C^{(k)})^T$$

$$A = (A_N A_C)^T$$

$$(\vec{x}_N^{(k+1)} \vec{x}_C^{(k+1)})^T = (A_N A_C)^T \cdot (\vec{x}_N^{(k)} \vec{x}_C^{(k)})^T$$

But we don't need to recompute $\vec{x}_C^{(k+1)}$. Now the computation for each iteration could be simplified as:

$$\vec{x}_N^{(k+1)} = A_N \vec{x}_N^{(k)}$$

$$\vec{x}_C^{k+1} = \vec{x}_C^{(k)}$$

A_C is not actually used in computing $\vec{x}^{(k+1)}$.

Also, the cost of the matrix-vector multiplication is essentially given by the number of nonzero entries in the matrix, not the matrix dimensions.

$$A'_{ij} = \begin{cases} 0, & \text{if } i \in C \\ A_{ij}, & \text{otherwise} \end{cases} \tag{7}$$

$$(x'_C{}^{(k)})_i = \begin{cases} (x^{(k)})_i, & \text{if } i \in C \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

After those computation simplifications, we reached the adaptive version of Page Rank Algorithm.

Algorithm 7 Adaptive Page Rank Algorithm

Repeat
- $\vec{x}^{k+1} \leftarrow A' \vec{x}^{(k)} + \vec{x}'_C$
- Periodically,
  - $[N, C] =$ **detectConverged**$(\vec{x}^k, \vec{x}^{k+1}, \epsilon)$
  - $[A'] =$ **filter**(A', N, C)
  - $[\vec{x}'_C] =$ **filter**$(\vec{x}^{(k)}, C)$
- Periodically, $\delta = \left\| A\vec{x}^k \right\| - \vec{x}^k$

Until $\delta < \epsilon$

return $x^{(k+1)}$

5.3 Experiments

Curious about page rank scores for specific webpage clusters, we applied Power Iteration Method on different web domains to rank external pages by calculating their corresponding importance scores. The dominant eigenvector returned by Power Iteration Algorithm represents the scores for web pages involved. Figures 9, 10, 11, 12 show how the web pages are linked to each other. If the square at the i^{th} row and the j^{th} column is colored, then it means there is a link from the i^{th} page to the j^{th} page.

For *https* : *//icerm.brown.edu*, we have the adjacency matrix shown as follows.

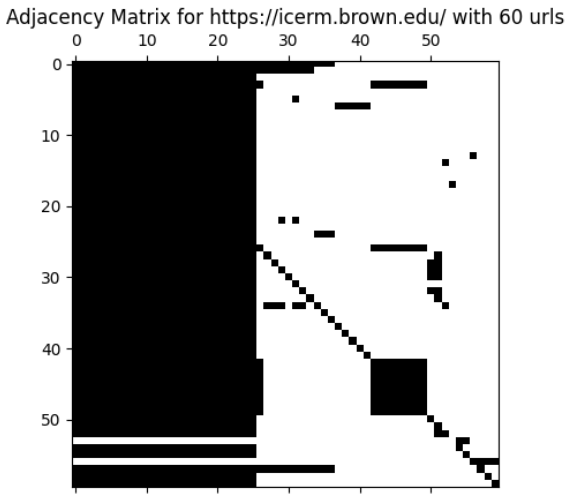

Figure 9: First 60 urls under icerm.brown.edu

The first 25 columns are links on navigation bar, which appears on each subpage under icerm.browm.edu. This explains that why the first 25 columns are almost full, since they are reached by all the pages.

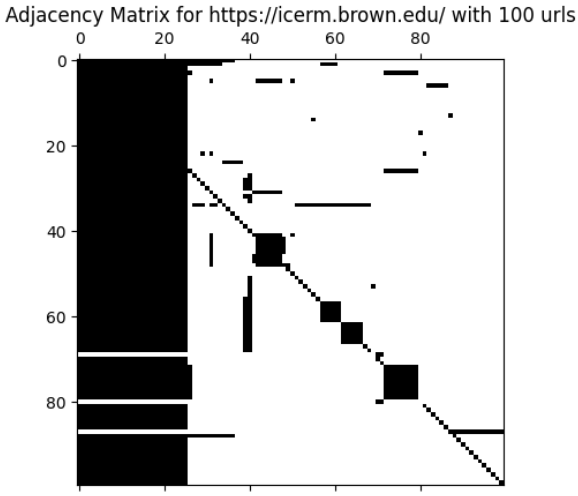

Figure 10: First 100 urls under icerm.brown.edu

The first 100 urls' adjacency matrix takes the first 60's as its submatrix. The page rank score is not revealing much information for this domain since the amount of all the pages starting with *https* : *//icerm.brown.edu* but not ending with *.pdf* is relatively small.

We crawled Wikipedia and Mathworld Wolfram. We focused on the first 220 urls of *https* : *//en.wikipedia.org/wiki/* as shown in Fig 11 and the first 200 urls of *https* : *//mathworld.wolfram.com/* in Fig 12. The gathered black columns are navigation bar group. By feeding the eigensolver with the raw data, we have the following interesting results. Based on what is ranked higher, we could conclude which wiki page is on trend recently.

5.3.1 Page rank results

Wikipedia Page Rank Results In Figure 11, the location where the fulfilled columns shows up is different from the one in Figure 10. This is the result of the difference places the navigation bars are located in *https* : *//en.wikipedia.org/wiki/* and *https* : *//icerm.brown.edu/*.

From the page rank score, we could notice that more people care about *Deaths in 2020* than *2020 Twiter bitcoin scam* and COVID-related wiki pages call out more attention.

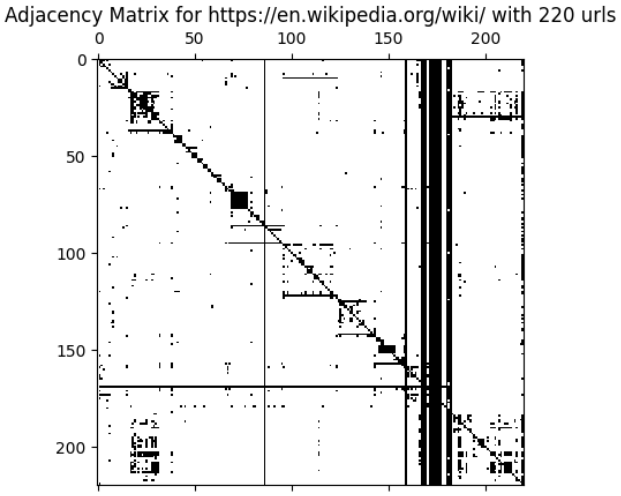

Figure 11: First 220 urls under wikipedia.org

Page	Rank
https://en.wikipedia.org/wiki/Deaths_in_2020	0.028629160277534114
https://en.wikipedia.org/wiki/California	0.025893256903712636
https://en.wikipedia.org/wiki/July_17	0.02227403984319967
https://en.wikipedia.org/wiki/COVID-19_pandemic	0.020537702316887802
https://en.wikipedia.org/wiki/Portal:Coronavirus_disease_2019	0.01656484624983901
https://en.wikipedia.org/wiki/List_of_ deaths due_to_COVID-19	0.01639792923859829
https://en.wikipedia.org/wiki/Coronavirus_disease_2019	0.015480623120122779
https://en.wikipedia.org/wiki/Severe_acute_respiratory_syndrome_coronavirus_2	0.015463315287237326
https://en.wikipedia.org/wiki/COVID-19_testing	0.015194270696435312
https://en.wikipedia.org/wiki/Timeline_of_the_COVID-19_pandemic	0.015194270696435312
https://en.wikipedia.org/wiki/COVID-19_pandemic_by_country_and_territory	0.015194270696435312
https://en.wikipedia.org/wiki/Impact_of_the_COVID-19_pandemic	0.015194270696435312
https://en.wikipedia.org/wiki/World_War_I	0.014436228831460307
https://en.wikipedia.org/wiki/SMS_Derfflinger	0.013901178029826601
https://en.wikipedia.org/wiki/High_Seas_Fleet	0.012441670371025016
https://en.wikipedia.org/wiki/Wikipedia:In_the_news/Candidates	0.012384588905511574
https://en.wikipedia.org/wiki/Imperial_German_Navy	0.012345100308246922
https://en.wikipedia.org/wiki/Joanna_Cole]_(author)	0.012201273284717616
https://en.wikipedia.org/wiki/2020_Twitter_bitcoin_scam	0.012105299451118816

Mathworld Wolfram Page Rank Results From the page rank scores, we could see in "topics" subsection, Interactive Demonstrations has more important links connected.

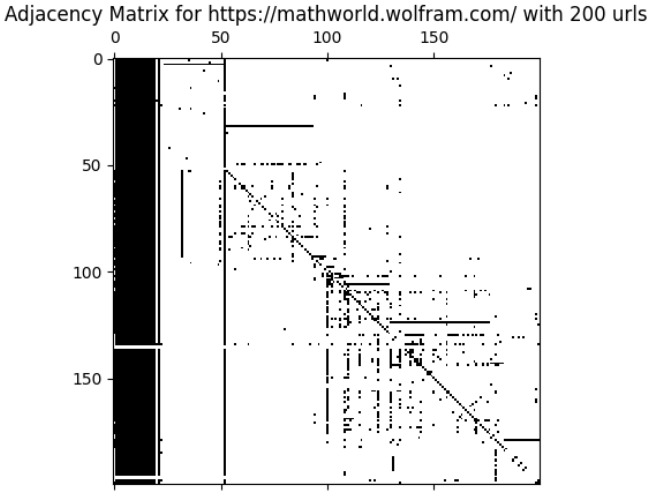

Figure 12: First 200 urls under mathworld

Page	Rank
https://mathworld.wolfram.com/topics/InteractiveDemonstrations.html	0.018454442986652088
https://mathworld.wolfram.com/topics/FiniteGroups.html	0.017501499518318534
https://mathworld.wolfram.com/topics/DifferentialGeometry.html	0.017008252546676733
https://mathworld.wolfram.com/topics/DifferentialEquations.html	0.01694654631617086
https://mathworld.wolfram.com/topics/Manifolds.html	0.016869553656892754
https://mathworld.wolfram.com/topics/DynamicalSystems.html	0.016664930770821353
https://mathworld.wolfram.com/topics/Polynomials.html	0.016470012578170302
https://mathworld.wolfram.com/topics/OperatorTheory.html	0.01511727687798641
https://mathworld.wolfram.com/topics/MathematicsintheArts.html	0.014543261872502258
https://mathworld.wolfram.com/FiniteGroup.html	0.01441299104750234
https://mathworld.wolfram.com/topics/NUMB3RS.html	0.013744920220832117
https://mathworld.wolfram.com/topics/PrimeNumbers.html	0.013544828492553762
https://mathworld.wolfram.com/PrimeNumber.html	0.013516112703471611
https://mathworld.wolfram.com/topics/Combinatorics.html	0.013163840998935994
https://mathworld.wolfram.com/topics/Mnemonics.html	0.013133333257223328
https://mathworld.wolfram.com/topics/MathematicalHumor.html	0.013024943466366769
https://mathworld.wolfram.com/topics/MathematicalRecords.html	0.012648912204085093
https://mathworld.wolfram.com/topics/FunctionalAnalysis.html	0.011329293474258123

5.3.2 Row-stochastic-oriented Page Rank Algorithm

Page rank algorithm is dealing with a specific type matrix – stochastic matrix, calculating the page rank score, the limiting probability distribution. There are two perspectives to tackle this problem. To simplify the description, we only deal with the non-dangling situation. The adjacency matrix A for the page network graph is defined as

$$A_{ij} = \begin{cases} 1, \text{there is a link from page i to page j} \\ 0, otherwise \end{cases}$$

The weighted adjacency matrix M is defined as

$$M_{ij} = A_{ji}/outdegree(j).$$

Solving linear system $Mx = x$ would return the page rank score endorsed by Perron-Frobinus theorem, which is the motivation we apply eigensolvers to M. Under this kind of interpretation, we are calculating the eigenvector corresponding to the dominant eigenvalue for a column-stochastic matrix.

While another interpretation could be calculating the limiting probability distribution iteratively. This is based on the Theory of Ergodic Markov chains: for any aperiodic and irreducible row-stochastic matrix A, the limit exists and it is a rank one stochastic matrix, i.e.

$$\lim_{k->\infty} A^k = vx^T,$$

where v is the eigenvecor of A, $x \in R^n$. Based on that, we came up with another Page Rank Algorithm leveraging the multiplication of row-stochastic matrices.

Specifically, in this Row-stochastic-oriented Page Rank Algorithm, we are calculating $\lim_{k->\infty}(M^T)^k$, where M^T is the transpose of the weighted adjacency matrix introduced above. According to the rank one

stochastic result, we set the converge measurement as the distance between the first two normalized column vectors of M^k.

Algorithm 8 Row-stochastic-oriented Page Rank Algorithm

$v_0^{(0)} = M[:, 0]$
$v_1^{(0)} = M[:, 1]$
$d \leftarrow \left\| \frac{v_0^{(0)}}{\left\|v_0^{(0)}\right\|} - \frac{v_1^{(0)}}{\left\|v_1^{(0)}\right\|} \right\|$
while $d > 1e-4$ **do**
  $v_0^k \leftarrow M v_0^{(k-1)}$
  $v_2^k \leftarrow M v_2^{(k-1)}$
  $d \leftarrow \left\| \frac{v_0^{(0)}}{\left\|v_0^{(0)}\right\|} - \frac{v_1^{(0)}}{\left\|v_1^{(0)}\right\|} \right\|$
  $k \leftarrow k + 1$
end while
return v_1

To reduce the number of iterations, we could do divide and conquer in the opposite direction for calculating the power of the row-stochastic matrix. This cost more space and more time for each iteration, but would reduce the convergence rate to $O(logk)$.

Algorithm 9 Row-stochastic-oriented Page Rank Algorithm - Iteration Saving

$d \leftarrow \infty$
while $d > 1e-4$ **do**
  $S^{(k)} \leftarrow S^{(k-1)} S^{(k-1)}$
  $v_1 \leftarrow S^k[:, 0] / \left\|S^k[:, 0]\right\|$
  $v_2 \leftarrow S^k[:, 1] / \left\|S^k[:, 1]\right\|$
  $d \leftarrow \|v_1 - v_2\|$
  $k \leftarrow k + 1$
end while
return v_1

6 Acknowledgements

The authors thank Dr. Yanlai Chen, Dr. Minah Oh, Justin Baker, and Liu Yang for the many beneficial and valuable discussions; and for providing many informative materials that greatly assisted in the completion of our research and this paper. We gratefully acknowledge the support of the Brown University's ICERM Summer Research program.

Appendices

A Python Implementations

The following URLs are the link addresses to our GitHub repository and webpage that contain key elements of our research. The repository includes our written algorithms, their performance comparison, and the PageRank web crawler; all of which is written in Python Script language.
Github repository: `https://github.com/ICERM-Efficient-Eigensolvers-2020/EE_with_applications`
See our webpage: `https://icerm-efficient-eigensolvers-2020.github.io/EE_with_applications/`

B Convergence of Inverse Iteration

In the following example, the input matrix is the matrix A shown below and the convergence range was set at 0.0001 - meaning the iterations would cease when the difference between the current λ and the previous λ was less than 0.0001.

$$A = \begin{bmatrix} 2 & 2 & -1 \\ -5 & 9 & -3 \\ -4 & 4 & 1 \end{bmatrix}$$

The convergence can be seen below. The initial values are set before the iterations begin. The eigenvalue of smallest magnitude of matrix A is 3 and as we can see, this is converging to the normalized eigenvector that corresponds to the smallest eigenvalue three.

- Initial Values
  - eigenvalue: 1.0
  - eigenvector: [0, 0, 1]
- Iteration: 1
  - eigenvalue: 1.8380743982494527
  - eigenvector: [0.09923118, 0.36384766, 0.92615768]
- Iteration: 2
  - eigenvalue: 2.4326298081988718
  - eigenvector: [0.10917414, 0.45421832, 0.88417573]
- Iteration: 3
  - eigenvalue: 2.7121398904194773
  - eigenvector: [0.10039825, 0.48272843, 0.86999624]
- Iteration: 4
  - eigenvalue: 2.8521450153150854
  - eigenvector: [0.08662022, 0.49021612, 0.86728605]
- Iteration: 5
  - eigenvalue: 2.925233045118846
  - eigenvector: [0.0721433, 0.48916397, 0.86920306]
- Iteration: 6
  - eigenvalue: 2.963961174446346
  - eigenvector: [0.05867467, 0.48462713, 0.87275073]
- Iteration: 7
  - eigenvalue: 2.984379261124954
  - eigenvector: [0.04689181, 0.47896138, 0.87658266]

- Iteration: 8
  - eigenvalue: 2.9948873604832746
  - eigenvector: [0.03697491, 0.47330823, 0.88012055]
- Iteration: 9
  - eigenvalue: 3.000028980760065
  - eigenvector: [0.02885101 0.46819241 0.88315541]
- Iteration: 10
  - eigenvalue: 3.0022998032375607
  - eigenvector: [0.02232701, 0.46381064, 0.88565298]
- Iteration: 11
  - eigenvalue: 3.0030757900768994
  - eigenvector: [0.01716608 0.46018743 0.88765582]
- Iteration: 12
  - eigenvalue: 3.0031114367996437
  - eigenvector: [0.01313031, 0.45726312, 0.88923452]

The final results are shown below. Most of these results are the self explanatory, but note that the number labelled difference is the difference between the eigenvalue from iteration eleven and the eigenvalue from iteration twelve. This is how it is determined whether the convergence range has been reached.

Results:
Number of Iterations: 12
Smallest Eigenvalue: 3.0031
Corresponding Eigenvector: [0.01313031 0.45726312 0.88923452]
Difference: 3.564672274425362e-05

Note that it took twelve iterations to reach the desired convergence range, but to reach a convergence range equal to 0.00001 it takes thirty iterations.

References

[1] S. Brakken-Thal, *Gershgorin's theorem for estimating eigenvalues*, Sthal@ ups. edu, (2007).

[2] K. Bryan and T. Leise, *The $25,000,000,000 eigenvector: The linear algebra behind google*, SIAM review, 48 (2006), pp. 569–581.

[3] C.-Y. Chiang and M. M. Lin, *The eigenvalue shift technique and its eigenstructure analysis of a matrix*, Journal of Computational and Applied Mathematics, 253 (2013), pp. 235–248.

[4] G. Eraña Robles, *Implementing the qr algorithm for efficiently computing matrix eigenvalues and eigenvectors*, (2018).

[5] H. E. Fettis and J. C. Caslin, *Eigenvalues and eigenvectors of hilbert matrices of order 3 through 10*, 1967.

[6] W. Gander, *Algorithms for the qr decomposition*, Res. Rep, 80 (1980), pp. 1251–1268.

[7] A. George and J. W. Liu, *Householder reflections versus givens rotations in sparse orthogonal decomposition*, Linear Algebra and its Applications, 88 (1987), pp. 223–238.

[8] A. Grigorev, *Householder transformation*, 2017.

[9] I. C. Ipsen, *Computing an eigenvector with inverse iteration*, SIAM review, 39 (1997), pp. 254–291.

[10] T. Joffrain, T. M. Low, E. S. Quintana-Ortí, R. v. d. Geijn, and F. G. V. Zee, *Accumulating householder transformations, revisited*, ACM Transactions on Mathematical Software (TOMS), 32 (2006), pp. 169–179.

[11] S. Kamvar, T. Haveliwala, and G. Golub, *Adaptive methods for the computation of pagerank*, Linear Algebra and its Applications, 386 (2004), pp. 51–65.

[12] M. P. Math, *9: Gershgorin circle theorem - learning linear algebra.*

[13] P. J. Olver, *Orthogonal bases and the qr algorithm*, University of Minnesota, (2008).

[14] M. Panju, *Iterative methods for computing eigenvalues and eigenvectors*, arXiv preprint arXiv:1105.1185, (2011).

[15] Y. Saad, *Numerical methods for large eigenvalue problems: revised edition*, SIAM, 2011, ch. 10.

[16] G. Strang, *Introduction to Linear Algebra*, Wellesley - Cambridge Press, 2016.

[17] R. A. Tapia, J. E. Dennis Jr, and J. P. Schafermeyer, *Inverse, shifted inverse, and rayleigh quotient iteration as newton's method*, Siam Review, 60 (2018), pp. 3–55.

[18] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, Siam, 1997.

[19] R. van de Geijn and M. Myers, *Advanced linear algebra: Foundations to frontiers.*

[20] I. Yanovsky, *Qr decomposition with gram-schmidt*, University of California, Los Angeles, (2012).