Computing Eigenvectors and Eigenvalues: QR method, Power method and Deflation

Janavi Gupta (janavig), Aditi Rao (aditirao)

1 Introduction

This project focuses on comprehending and investigating various methods for computing eigenvalues and eigenvectors of symmetric matrices. Additionally, it aims to compute singular values and singular vectors, not only for symmetric but also non-symmetric matrices. The methods under exploration include the QR Method, Power Method, and Deflation. Furthermore, this project delves into combining these techniques to achieve the intended results effectively.

2 Table of Contents

3 The QR Method

3.1 Introduction

To understand this method it is important to understand 2 concepts related to matrices

- Similar Matrices : Two matrices A and B are said to be similar if there exists a matrix X such that $A = X^{-1}BX$. We also have that A, B have the same eigenvalues

- QR method requires us to do the QR decomposition of a matrix A. This means A = QR, where Q is an orthogonal matrix and R is a upper triangular matrix. Since Q is an orthogonal matrix $Q^{-1} = Q^T$. From this we have that $Q^{-1}Q = Q^TQ = I$

Knowing these two concepts we can understand the QR method. Lets say we need to find the eigenvalues for a matrix A_0. starting at k = 0 we can do the QR decomposition to get $A_k = Q_kR_k$, and we define $A_{k+1} = R_kQ_k$. We see that

$$\begin{aligned} A_{k+1} &= R_kQ_k \\ &= Q_k^{-1}Q_kR_kQ_k \qquad \text{(since } Q_k \text{ is orthogonal } Q_k^{-1}Q_k = \text{I, IM = M for any matrix M)} \\ &= Q_k^{-1}AQ_k \qquad (A_k = Q_kR_k) \end{aligned}$$

This means A_{k+1} is similar to A_k. This means for all $k >= 0$, all A_k are similar to each other and have the same eigenvalues.

As k approaches infinity (iterating k times) we have that A_k converges to an upper triangular matrix, with all the eigenvalues on the diagonal. The element in the first row and the first column gives us the leading eigenvalue.

3.2 Example

To understand the application of the QR Method, let's determine the leading eigenvalue for the matrix A:

$$A = \begin{bmatrix} 7 & 1 \\ 1 & 7 \end{bmatrix}$$

In our code, by executing the iterative QR method on matrix A, we observe that it converges towards:

$$\begin{bmatrix} 8 & 0 \\ 0 & 6 \end{bmatrix}$$

From this outcome, we determine that 8 is the leading eigenvalue of matrix A. This conclusion is drawn from the fact that it resides in the first row and the first column of the resultant matrix.

3.3 Pseudo code

- Define the QR Eigenvalue Function `qr_method` that accepts the following:
  a. A symmetric matrix `A` with real values
  b. The number of times to iterate through the algorithm `max_iter`
- Create a copy of the matrix `A` and name it A_k.
- For `max_iter` times perform QR decomposition of A_k and set A_k `= R * Q`
- Lastly, return the first diagonal element of A_k as the leading eigenvalue of `A`.

4 The Power Method

4.1 Introduction

We use the power method to find the leading eigenvector and eigenvalue of a symmetric matrix A ($n \times n$). By the Spectral Theorem we have that A is a diagonalizable matrix. Since A is diagonalizable, it has n independent eigenvectors $\{x_1,...x_n\}$ with eigenvalues $\{\lambda_1,..,\lambda_n\}$. $\{x_1,...x_n\}$ forms the basis for $\mathbb{R}^n$ since they are all independent vectors.

Let x_0 be an arbitrary non-zero vector such that $x_0 \in \mathbb{R}^n$. Since $\{x_1,...x_n\}$ is a basis of $\mathbb{R}^n$ we can write x_0 as a linear combination of $\{x_1,...x_n\}$, with scalars $c_1, c_2...c_n$.

$$x_0 = c_1x_1 + ... + c_nx_n \tag{1}$$

Now we multiply (1) with matrix A on both the sides

$$\begin{aligned} Ax_0 &= A(c_1x_1 + c_2x_2 + ... + c_nx_n) && \text{(by substitution)} \\ &= c_1(Ax_1) + c_2(Ax_2) + ... + c_n(Ax_n) && \text{(by linearity)} \\ &= c_1(\lambda_1x_1) + c_2(\lambda_2x_2) + ... + c_n(\lambda_nx_n) && \text{(by eigenvalue defn)} \end{aligned}$$

Repeated multiplication (say k times) by Matrix A on both the sides

$$\begin{aligned} A^kx_0 &= c_1(\lambda_1^kx_1) + c_2(\lambda_2^kx_2) + ... + c_n(\lambda_n^kx_n) && \text{(by powers of matrix)} \\ &= \lambda_1^k[c_1x_1 + c_2(\tfrac{\lambda_2}{\lambda_1})^kx_2 + ... + c_n(\tfrac{\lambda_n}{\lambda_1})^kx_n] && \text{(by factoring out } \lambda_1^k) \end{aligned}$$

Let i be such that $1 < i <= n$. Since λ_1 is the leading vector, we have that $\lambda_1 > \lambda_i$ for all i. This implies $\frac{\lambda_i}{\lambda_1}$ is less than 1 for all i. So, as k approaches infinity, $(\frac{\lambda_i}{\lambda_1})^k$ approaches 0 for all i.

From this we get that

$$A^kx_0 \approx \lambda_1^kc_1x_1 \qquad \text{(all other terms go to 0)}$$

λ_1^k and c_1 are both scalars, and hence normalizing A^kx_0 gives us the leading eigenvector

For an eigenvector x, by definition of eigenvalue we have that

$$Ax = \lambda x \qquad \text{(for eigenvalue } \lambda)$$

So, we can write

$$\begin{aligned}\frac{Ax\cdot x}{x\cdot x} &= \frac{\lambda x\cdot x}{x\cdot x}\\ &= \frac{\lambda(x\cdot x)}{x\cdot x}\\ &= \lambda\end{aligned}$$

So, we have that for a given eigenvector x, eigenvalue λ is

$$\lambda = \frac{Ax\cdot x}{x\cdot x}$$

This is called Rayleigh quotient, and can be used to calculate the leading vector by plugging in the leading eigenvector in the place of x.

This is how we can compute the leading eigenvector and eigenvalue using the Power Method

4.2 Example

To grasp the workings of the Power Method, let's determine the leading eigenvalue and eigenvector of the matrix A =

$$A = \begin{bmatrix} 7 & 3 & 5 \\ 3 & 5 & 1 \\ 5 & 1 & 1 \end{bmatrix}$$

In our code, we initialize x_0 as

$$x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

By executing the Power Method on matrix A, as demonstrated in the code, we derive the leading eigenvector of the matrix. Subsequently, we leverage this eigenvector along with matrix A to compute the associated eigenvalue using the Rayleigh quotient, as previously explained.

4.3 Pseudo code

- Define the Power Method `power_method` that accepts the following inputs:
  a. A symmetric matrix `A` with real values
  b. The number of times to iterate through the algorithm `max_iter`
- Determine the size of A and store the number of rows in n.
- Initialize a random vector v of size n
- Normalize v (set $v = \frac{v}{\|v\|}$)

- For `max_iter` times:
  a. Multiply v by A to get a new vector v_{new} (set $v_{\text{new}} = Av$)
  b. Normalize v_{new} (set $v_{\text{new}} = \frac{v_{\text{new}}}{\|v_{\text{new}}\|}$) and update $v = v_{\text{new}}$
- Compute the eigenvalue using the Rayleigh quotient: eigenvalue $= \frac{\langle Av, v\rangle}{\langle v, v\rangle}$
- Lastly, return the leading eigenvalue and the converged eigenvector v

5 The Deflation Method

5.1 Introduction

We can use the deflation method along with the power method to find the eigenvalues and eigenvectors of a symmetric matrix A $(n \times n)$.

The deflation method follows iteration and we start this method finding the largest eigenvalue of the $(n \times n)$ matrix and then reducing the matrix to an $(n-1) \times (n-1)$ matrix, finding the largest eigenvalue of this matrix and further reducing this matrix to $(n-2) \times (n-2)$ and so on. We do this to eliminate the influence of the current largest eigenvalue in the next step. This is the general idea of the deflation method.

Now going into the details to understand how and why this works, lets define matrix A such that

- it symmetric
- it is real and an $(n \times n)$ matrix
- with eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$ and eigenvectors $\{v_1, \ldots, v_n\}$

The main goal of deflation as described before is to get a $(n-1) \times (n-1)$ matrix with eigenvalues $\{\lambda_2, \ldots, \lambda_n\}$

Lets define another matrix $B = A - \lambda_1 v_1 v_1^T$ where λ_1 is the leading eigenvalue of A and v_1 is orthonormal (for simplicity), and the leading eigenvector of A.

We want to show how 0 is the eigenvalue corresponding to the eigenvector v_1 of B

$$\begin{aligned}
B &= A - \lambda_1 v_1 v_1^T && \text{(by definition)} \\
Bv_1 &= (A - \lambda_1 v_1 v_1^T) v_1 && \text{(multiplying by } v_1 \text{ on bts)} \\
Bv_1 &= Av_1 - \lambda_1 v_1 (v_1^T v_1) && \text{(expanding RHS, linearity)} \\
Bv_1 &= Av_1 - \lambda_1 v_1 (1) && (v_1 \text{ is orthonormal)} \\
Bv_1 &= \lambda_1 v_1 - \lambda_1 v_1 && (Av_1 = \lambda_1 v_1 \text{ since } \lambda_1 \text{ is eigenvalue)} \\
Bv_1 &= 0 v_1 && \text{(by math)}
\end{aligned}$$

Therefore, 0 is the eigenvalue corresponding to eigenvector v_1 for matrix B

Given this we also want to show that the remaining eigenvectors of A $\{v_2, \ldots, v_n\}$ are still eigenvectors for matrix B with the corresponding eigenvalues $\{\lambda_2, \ldots, \lambda_n\}$.

Let v_i be a random eigenvector of A with eigenvalue λ_i such that $1 < i <= n$. Then,

$$
\begin{aligned}
Bv_i &= (A - \lambda_1 v_1 v_1^T) v_i && \text{(definition of B, multiplying by } v_i\text{)} \\
Bv_i &= Av_i - \lambda_1 v_1 v_1^T v_i && \text{(linearity)} \\
Bv_i &= Av_i && (\lambda_1 = 0, \text{ for B}) \\
Bv_i &= \lambda_i v_i && \text{(definition of eigenvalue } Av_i = \lambda_i v_i\text{)}
\end{aligned}
$$

Hence, we have successfully created a $(n-1) \times (n-1)$ matrix B, with eigenvalues $\{\lambda_2,...,\lambda_n\}$, and eigenvectors $\{v_2,...,v_n\}$ as desired.

We use the power method on matrix B to get the leading eigenvalue and eigenvector. After this, we continue to do the matrix deflation process to get all the eigenvalues and eigenvectors of the original matrix A in this manner.

5.2 Example

To comprehend the deflation method, we aim to compute the eigenvectors and eigenvalues of a matrix. We apply this approach to matrix A:

$$A = \begin{bmatrix} 7 & 3 & 5 \\ 3 & 5 & 1 \\ 5 & 1 & 1 \end{bmatrix}$$

The complete process, including the computation of eigenvalues and eigenvectors of matrix A along with the intermediary steps, is explicitly detailed in the provided code (section 9)

5.3 Pseudo code

- Define a Deflation method `deflation` that accepts:
  a. A symmetric matrix `A` with real values
  b. The number of iterations `max_iter`

- Determine the size of `A` and store the number of rows in n

- Initialize empty lists for eigenvalues and eigenvectors

- For each eigenvalue and eigenvector pair to be found:
  a. Use `power_method` as defined above with `A` and `max_iter` to obtain the leading eigenvalue and eigenvector
  b. Add the found eigenvalue and eigenvector to their respective lists *b.* Deflate `A` using the formula: $A = A - \text{eigenvalue} \times \text{eigenvector} \times \text{eigenvector}^T$

- Return the lists of eigenvalues and eigenvectors.

6 Singular values and Singular vectors

6.1 Introduction

To obtain the singular vectors of a matrix A, we calculate the eigenvectors of both the matrices A^TA and AA^T. This involves utilizing the previously mentioned deflation and power methods. These resulting eigenvectors provide us with all the singular vectors of matrix A, meeting our objective.

To determine the singular values of matrix A, we employ a method that involves computing the eigenvalues of either A^TA or AA^T matrices. This process is conducted through deflation and the power method, as previously explained. Once we have these eigenvalues, we obtain the singular values of matrix A by taking the square root of these eigenvalues, thereby achieving the desired result of singular values for matrix A.

6.2 Example

To comprehend the computation of singular vectors and singular values for a matrix, we'll demonstrate this process for matrix A:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 5 \\ 4 & 0 \end{bmatrix}$$

In the provided code (section 9) , we'll determine the singular values and singular vectors of matrix A. This will involve employing the power method and deflation method for accurate computation.

6.3 Pseudo code

- Define a function to obtain the singular values and vectors `singular_values_vectors` that accepts:
  a. A matrix `A`.
  b.The number of iterations `max_iter`.
- Compute and store A^TA and AA^T
- Apply `power_method` on A^TA to obtain the leading eigenvalue and eigenvector.
- Apply `deflation` on A^TA to obtain all right singular vectors and singular values.
- Compute the square root of each eigenvalue to get singular values.
- Apply `power_method` on AA^T to obtain the leading eigenvalue and eigenvector.
- Apply `deflation` on AA^T to obtain all left singular vectors.
- Return the singular values, right singular vectors, and left singular vectors.

7 Code Design

The code consists of the following methods:

- QR method: The `qr_method` function approximates the leading eigenvalue of a matrix using the QR algorithm. It is an iterative method involving repeated QR decompositions and matrix multiplications. The leading eigenvalue is estimated as the first diagonal element of the resulting matrix after a specified number of iterations.
- Power method: The `power_method` function computes the leading eigenvalue and corresponding eigenvector of a matrix. It starts with a random vector, which is iteratively updated by multiplying with the matrix and normalizing. The leading eigenvalue is calculated using the Rayleigh quotient.
- Deflation method: The `deflation` function finds all eigenvalues and eigenvectors of a symmetric matrix. It applies the power method and then performs matrix deflation by subtracting the outer product of the found eigenvector and its corresponding eigenvalue. This process is repeated until all eigenvalues and eigenvectors are found.
- Singular values and vectors: The `singular_values_vectors` function computes the singular values and singular vectors of a matrix. It applies the power method and deflation to both $A^T A$ and AA^T to find the right and left singular vectors, respectively. The singular values are the square roots of the eigenvalues of $A^T A$.
- Random matrix generator: The `rand_matrix_generator` function generates a random symmetric matrix. It creates a matrix with random values and forms a symmetric matrix by multiplying it with its transpose.
- Displaying computations and calling all the methods: The `display_computations` function integrates the above methods to compute and display the leading eigenvalue, all eigenvalues and eigenvectors, and singular values/vectors. It also compares these results with Julia's built-in functions for eigenvalues/eigenvectors and singular value decomposition.

Each method is written separately to make it easier to see what is being calculated in each function. Additionally, functions that are unaffected by symmetric matrices don't necessarily have to then be given symmetric matrices allowing for a wide variety of calculations.

8 Bibliography

While doing our project we used different sources, as listed below

1. 10.3 POWER METHOD FOR APPROXIMATING EIGENVALUES
2. Power Method - an overview — ScienceDirect Topics.
3. Lecture 14: Eigenvalue Computations The power method - symmetric matrices
4. The Power Method

5. The QR Method

6. The Deflation Method for Finding Eigenvalues and Eigenvectors

7. Math 361S Lecture notes Finding eigenvalues: The power method

8. Manipulating Matrices

9 Code

The code begins on the next page along with the results!

[131]:
```
using LinearAlgebra
using PrettyTables
```

[132]:
```
# Function to approximate the leading eigenvalue of a matrix using the QR method
function qr_method(A, max_iter)
    # Initialize A_k with the input matrix A
    A_k = copy(A)

    # Iterate up to a maximum number of iterations
    for k = 1:max_iter
        # Perform QR decomposition on A_k
        Q, R = qr(A_k)
        # Form the next matrix A_k by multiplying R and Q
        A_k = R * Q
    end

    # # Return the leading eigenvalue, which is the first diagonal element of A_k
    return A_k[1,1]
end
```

[132]:
```
qr_method (generic function with 1 method)
```

[133]:
```
# Define a function to compute the leading eigenvalue using the Power Method
function power_method(A, max_iter)
    # Determine the size of the matrix A and initialize a random vector of this␣
 ↪size
    n, _ = size(A)
    v = rand(n)  # Start with a random vector
    v = v / norm(v)  # Normalize the vector

    # Iterate for a maximum number of iterations or until convergence
    for _ = 1:max_iter
        # Multiply the vector by the matrix A
        v_new = A * v
        # Normalize the resulting vector
        v_new = v_new / norm(v_new)
        # Update the vector for the next iteration
        v = v_new
    end

    # Calculate the leading eigenvalue which is done by the Rayleigh quotient:␣
 ↪(v'Av) / (v'v)
    eigenvalue = dot(A * v, v) / dot(v, v)
    return eigenvalue, v
end
```

[133]:
```
power_method (generic function with 1 method)
```

[134]:
```
# Function to find all eigenvalues and eigenvectors of a symmetric matrix using␣
 ↪deflation
function deflation(A, max_iter)
    k = size(A, 1)
    eigenvalues = []
    eigenvectors = []

    # Main loop for finding all eigenvalues and eigenvectors
    for i = 1:k
        eigenvalue, eigenvector = power_method(A, max_iter)
        push!(eigenvalues, eigenvalue)
        push!(eigenvectors, eigenvector)
        A = A - eigenvalue * eigenvector * transpose(eigenvector)
    end

    return eigenvalues, eigenvectors
end
```

[134]:
```
deflation (generic function with 1 method)
```

[135]:
```
#Function to obtain all the singular values and singular vectors
function singular_values_vectors(A, max_iter)

v_eigenvalue, v_eigenvector = power_method(transpose(A) * A, max_iter)
singular_values, v_singular_vectors = deflation(transpose(A) * A, max_iter)
singular_values = sqrt.(singular_values)

u_eigenvalue, u_eigenvector = power_method(A * transpose(A), max_iter)
_, u_singular_vectors = deflation(A * transpose(A), max_iter)

return singular_values, v_singular_vectors, u_singular_vectors
end
```

[135]:
```
singular_values_vectors (generic function with 1 method)
```

[136]:
```
#Function to generate a random symmetric matrix A and A^TA
function rand_matrix_generator(size, min_val, max_val)
    X = rand(min_val : max_val, size, size)
    A = transpose(X) * X
    return A
end
```

[136]:
```
rand_matrix_generator (generic function with 1 method)
```

[137]:
```
#Function to compute and display all the eigenvalues and eigenvectors
function display_computations(A, max_iter_qr, max_iter_pm)
    #get leading eigenvalue from QR method
    eigenvalue_qr = qr_method(A, max_iter_qr)
    print("A's leading eigenvalue (QR method)")
    display(eigenvalue_qr)

    #get leading eigenvector and eigenvalue from the power method
    eigenvalue_pm, eigenvector_pm = power_method(A, max_iter_pm)
    print("A's leading eigenvector (Power method):")
    display(eigenvector_pm)
    print("A's leading eigenvalue (Power method and Raleigh quotient):")
    display(eigenvalue_pm)

    #get all eigenvalues and eigenvectors
    eigenvalues, eigenvectors = deflation(A, max_iter_pm)
    println("A's eigenvectors and corresponding eigenvalues (Deflation):")
    for i in range(1, length(eigenvectors))
        print("Eigenvector number " * string(i) * " is:")
        display(eigenvectors[i, 1])
        println("And has corresponding eigenvalue:")
        display(eigenvalues[i, 1])
        println("")
    end

    #get all singular vectors and singular values
    singular_values, left_singular_vectors, right_singular_vectors =␣
↪singular_values_vectors(A, max_iter_pm)
    println("A's singular vectors for matrix V in A's SVD are:")
    for i in range(1, length(left_singular_vectors))
        println("Singular vector number " * string(i) * " is:")
        display(left_singular_vectors[i, 1])
    end
    println("A's singular vectors for matrix U in A's SVD are:")
    for i in range(1, length(right_singular_vectors))
        println("Singular vector number " * string(i) * " is:")
        display(right_singular_vectors[i, 1])
    end
    println("A's singular values are:")
    display(singular_values)

    println("-----------------------------------------------------")
    println("Julia's Output to compare")
    display(eigvals(A))
    display(eigvecs(A))
    display(svd(A))
    end
```

```
[137]: display_computations (generic function with 1 method)
```

```
[139]: #Running our functions
A = rand_matrix_generator(8, 1, 14)
display_computations(A, 11, 11)
```

```
A's leading eigenvalue (QR method)

3706.6197377438066

A's leading eigenvector (Power method):

8-element Vector{Float64}:
 0.3513608224753594
 0.37638183990226576
 0.3237371058940461
 0.32864052652044595
 0.28750607067702877
 0.2945645790232591
 0.3102680156972951
 0.5063375358008766

A's leading eigenvalue (Power method and Raleigh quotient):

3706.6197377438125

A's eigenvectors and corresponding eigenvalues (Deflation):
Eigenvector number 1 is:

8-element Vector{Float64}:
 0.351360822475992
 0.3763818399045702
 0.3237371058940415
 0.32864052652215336
 0.28750607067694306
 0.2945645790215287
 0.31026801569583873
 0.5063375357995669

And has corresponding eigenvalue:

3706.619737743812


Eigenvector number 2 is:

8-element Vector{Float64}:
  0.14571551613931755
  0.6033476030755105
 -0.03655637427851586
  0.4279101425701151
  0.015203163364386573
 -0.4202180831804913
 -0.370047458248081
 -0.3413879093355321
```

```
And has corresponding eigenvalue:

380.82371257775554


Eigenvector number 3 is:

8-element Vector{Float64}:
 -0.25360418740957874
  0.27324405248271455
 -0.5867753620222746
 -0.11108035160359431
  0.6112811260590814
  0.3374063824440383
  0.010179994276021861
 -0.1294879131382918

And has corresponding eigenvalue:

316.6112888725979


Eigenvector number 4 is:

8-element Vector{Float64}:
 -0.14479027505085648
  0.28515556005123127
  0.38619850935107053
 -0.4310838100757833
  0.19973712688819506
 -0.30792437131169925
  0.5520123908990922
 -0.35115455981196486

And has corresponding eigenvalue:

185.83444412027958


Eigenvector number 5 is:

8-element Vector{Float64}:
 -0.017664417859922416
  0.4253602654085433
 -0.4078677800266019
 -0.06503699991839602
 -0.6853129067817778
  0.16333342881167237
  0.38547950712171036
  0.05696170670867869

And has corresponding eigenvalue:

74.94648539345816
```

```
Eigenvector number 6 is:

8-element Vector{Float64}:
 -0.5014680359460327
 -0.2388432661000973
 -0.1215065561743243
  0.6357019236098919
  0.05897685800826313
 -0.28315930677305284
  0.43081039424468226
  0.05786074408806446

And has corresponding eigenvalue:

45.27278198455453


Eigenvector number 7 is:

8-element Vector{Float64}:
  0.6021546427501092
 -0.3104373074456427
 -0.16400780699191936
  0.2273149350154011
  0.04986095031691271
  0.17667849492608365
  0.33207741763012194
 -0.5643500754755785

And has corresponding eigenvalue:

23.701241446165888


Eigenvector number 8 is:

8-element Vector{Float64}:
 -0.3942384538645349
  0.09202856509942431
  0.4361751313517652
  0.2240837200299576
 -0.1717457742789772
  0.6214834143606667
 -0.12927103460524353
 -0.4039743558842078

And has corresponding eigenvalue:

13.225754959699774


A's singular vectors for matrix V in A's SVD are:
Singular vector number 1 is:
```

```
8-element Vector{Float64}:
 0.35136082247487804
 0.3763818399002095
 0.3237371058941957
 0.3286405265190035
 0.2875060706769486
 0.2945645790246673
 0.31026801569854606
 0.5063375358020394
```

```
Singular vector number 2 is:
```

```
8-element Vector{Float64}:
 -0.13446246714199986
 -0.6145149653100952
  0.06165940230197924
 -0.42226936888861333
 -0.041741132335195885
  0.40523438581096294
  0.3687783520891321
  0.34673217042034815
```

```
Singular vector number 3 is:
```

```
8-element Vector{Float64}:
 -0.2618641573347747
  0.2444004526946695
 -0.5815705398649463
 -0.13592696382362113
  0.6114480151789907
  0.3562360870043977
  0.03305751175149655
 -0.11458414139462526
```

```
Singular vector number 4 is:
```

```
8-element Vector{Float64}:
  0.14168977911716651
 -0.2813837227396289
 -0.3935758979517547
  0.42992318919974126
 -0.19207110017692253
  0.31191827481249473
 -0.552078623662145
  0.3493367238365371
```

```
Singular vector number 5 is:
```

```
8-element Vector{Float64}:
  0.016279171892507566
 -0.42603601228306925
```

```
  0.4075128248518765
  0.06682537331644675
  0.6854630454405086
 -0.1641100949185373
 -0.38428981453502836
 -0.05679436933958514

8-element Vector{Float64}:
  0.5015891838415605
  0.23685729988522544
  0.1233790243566879
 -0.6353841632853213
 -0.05580897256448246
  0.28241750911344493
 -0.43256709496941037
 -0.05816278033605849

Singular vector number 6 is:
Singular vector number 7 is:

8-element Vector{Float64}:
  0.6022206153255303
 -0.31048401690461275
 -0.16418563883669587
  0.22732074756374882
  0.0499012843025556
  0.17643798761033988
  0.3321917950922549
 -0.5642042795627437

Singular vector number 8 is:

8-element Vector{Float64}:
 -0.3938768218942008
  0.09183966117273942
  0.4360753132162502
  0.22422435716256298
 -0.17171535869363252
  0.621588832271
 -0.12906804033445296
 -0.4043153284473653

A's singular vectors for matrix U in A's SVD are:
Singular vector number 1 is:

8-element Vector{Float64}:
 0.35136082247487804
 0.3763818399002095
 0.3237371058941957
 0.3286405265190035
 0.2875060706769486
 0.2945645790246673
 0.31026801569854606
 0.5063375358020394

Singular vector number 2 is:

8-element Vector{Float64}:
  0.10775382983545327
  0.6356244759336097
 -0.11911648826083002
  0.40658656978844104
  0.10226985221743627
 -0.3677539301662794
 -0.36359088384543137
 -0.3563268928963543

Singular vector number 3 is:

8-element Vector{Float64}:
 -0.27853379311488524
  0.1540105651328689
 -0.5668352883785468
 -0.1951390309809521
  0.5993972654035542
  0.41063938223714014
  0.08566497714236794
 -0.06385854704698624

Singular vector number 4 is:

8-element Vector{Float64}:
 -0.14150897886228334
  0.28128021660609265
  0.39394552047274967
 -0.4297981826037624
  0.19168033791736275
 -0.3121837417520839
  0.5520246645425246
 -0.34929332509455097

Singular vector number 5 is:

8-element Vector{Float64}:
 -0.016262189912691027
  0.42604403980048755
 -0.40750863703018686
 -0.06684690003569217
 -0.6854649303289657
  0.1641196507795538
  0.384275179863193
  0.056792391122013605
```

```
Singular vector number 6 is:

8-element Vector{Float64}:
  0.5015907494037682
  0.23681773135439324
  0.12341682697445912
 -0.635377936899505
 -0.0557453538219592
  0.2824022944299186
 -0.43260272491583596
 -0.05816810443005255

Singular vector number 7 is:

8-element Vector{Float64}:
  0.6022191959178412
 -0.3104829537785075
 -0.16418434972822868
  0.22732225992855612
  0.04990087109688192
  0.17643959910188997
  0.3321910331416273
 -0.5642061266586615

Singular vector number 8 is:

8-element Vector{Float64}:
 -0.3405179253335854
  0.29767210159497315
  0.3649656831408873
  0.3426645868690311
 -0.12033291313833257
  0.4720313360557701
 -0.24017315312850274
 -0.4998431332000915

A's singular values are:

8-element Vector{Float64}:
 3706.619737743812
  380.9711771879006
  316.44171865432446
  185.82225295647052
   74.94671524695414
   45.272401503073965
   23.701242414976036
   13.225752844573659

-----------------------------------------------------
Julia's Output to compare

8-element Vector{Float64}:
   13.225752844526445
   23.70124241494957
   45.272401501636026
   74.94671524727146
  185.8222528779295
  316.4393799764263
  380.97251739344813
 3706.619737743812

8×8 Matrix{Float64}:
  0.393879   -0.602221   0.501589   ...   0.262792    0.133212   -0.351361
 -0.0918407   0.310483   0.236862      -0.240159    0.615666   -0.376382
 -0.436076    0.164185   0.123375       0.581135   -0.0644282  -0.323737
 -0.224224   -0.22732   -0.635385       0.138846    0.421613   -0.328641
  0.171716   -0.049901  -0.0558159     -0.611157    0.0446528  -0.287506
 -0.621588   -0.176439   0.282419   ...  -0.359027   -0.403529   -0.294565
  0.129069   -0.332191  -0.432563      -0.0356102  -0.368613   -0.310268
  0.404314    0.564205  -0.0581619      0.112194   -0.347271   -0.506338

SVD{Float64, Float64, Matrix{Float64}, Vector{Float64}}
U factor:
8×8 Matrix{Float64}:
 -0.351361   0.133212    0.262792   ...   0.501589   -0.602221   0.393879
 -0.376382   0.615666   -0.240159       0.236862    0.310483  -0.0918407
 -0.323737  -0.0644282   0.581135       0.123375    0.164185  -0.436076
 -0.328641   0.421613    0.138846      -0.635385   -0.22732   -0.224224
 -0.287506   0.0446528  -0.611157      -0.0558159  -0.049901   0.171716
 -0.294565  -0.403529   -0.359027   ...   0.282419   -0.176439  -0.621588
 -0.310268  -0.368613   -0.0356102     -0.432563   -0.332191   0.129069
 -0.506338  -0.347271    0.112194      -0.0581619   0.564205   0.404314
singular values:
8-element Vector{Float64}:
 3706.6197377438125
  380.9725173934479
  316.4393799764264
  185.8222528779295
   74.9467152472715
   45.27240150163626
   23.70124241494977
   13.225752844526577
Vt factor:
8×8 Matrix{Float64}:
 -0.351361  -0.376382   -0.323737   ...  -0.294565  -0.310268   -0.506338
  0.133212   0.615666   -0.0644282     -0.403529  -0.368613   -0.347271
  0.262792  -0.240159    0.581135      -0.359027  -0.0356102   0.112194
 -0.14168    0.281375    0.393597      -0.311931   0.552078   -0.349332
 -0.016281   0.426035   -0.407513       0.164109   0.384291    0.0567946
  0.501589   0.236862    0.123375   ...   0.282419  -0.432563   -0.0581619
 -0.602221   0.310483    0.164185      -0.176439  -0.332191    0.564205
```

```
  0.393879  -0.0918407  -0.436076      -0.621588   0.129069    0.404314
```

[140]:
```
A_QR = [[7 1]; [1 7]]
```

[140]:
```
2×2 Matrix{Int64}:
 7  1
 1  7
```

[141]:
```
# Function to approximate the leading eigenvalue of a matrix using the QR method␣
↪(example)
function qr_method_example(A, max_iter)
    # Initialize A_k with the input matrix A
    A_k = copy(A)

    # Iterate up to a maximum number of iterations
    for k = 1:max_iter
        # Perform QR decomposition on A_k
        Q, R = qr(A_k)
        print("Q: ")
        display(Q)
        print("R: ")
        display(R)
        # Form the next matrix A_k by multiplying R and Q
        A_k = R * Q
    end

    print("The leading eigevalue of A is: ")
    # # Return the leading eigenvalue, which is the first diagonal element of A_k
    display(A_k[1,1])
    return A_k[1,1]
end
```

[141]:
```
qr_method_example (generic function with 1 method)
```

[142]:
```
qr_method_example(A_QR, 5)
```

```
Q:

2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.989949  -0.141421
 -0.141421   0.989949

R:

2×2 Matrix{Float64}:
 -7.07107  -1.9799
  0.0       6.78823

Q:
```

```
2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.991417  0.130736
  0.130736  0.991417

R:

2×2 Matrix{Float64}:
 -7.34302  1.83031
  0.0      6.53682

Q:

2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.993603  -0.112927
 -0.112927   0.993603

R:

2×2 Matrix{Float64}:
 -7.5677  -1.58098
  0.0      6.34275

Q:

2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.995699   0.0926481
  0.0926481  0.995699

R:

2×2 Matrix{Float64}:
 -7.73108  1.29707
  0.0      6.20871

Q:

2×2 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.997304   -0.0733787
 -0.0733787   0.997304

R:

2×2 Matrix{Float64}:
 -7.83913  -1.0273
  0.0       6.12313

The leading eigevalue of A is:

7.893377271188352
```

[142]:
```
7.893377271188352
```

[143]:
```
A_Power = [[7 3 5]; [3 5 1]; [5 1 1]]
```

[143]:
```
3×3 Matrix{Int64}:
 7  3  5
 3  5  1
 5  1  1
```

[144]:
```
# Define a function to compute the leading eigenvalue using the Power Method␣
↪(Example)
function power_method_example(A, max_iter)
    # Determine the size of the matrix A and initialize a random vector of this␣
↪size
    n, _ = size(A)
    v = rand(n)  # Start with a random vector
    v = v / norm(v)  # Normalize the vector

    # Iterate for a maximum number of iterations or until convergence
    for _ = 1:max_iter
        # Multiply the vector by the matrix A
        v_new = A * v
        print("The updated vector is ")
        display(v_new)
        # Normalize the resulting vector
        v_new = v_new / norm(v_new)
        # Update the vector for the next iteration
        v = v_new
    end

    # Calculate the leading eigenvalue which is done by the Rayleigh quotient:␣
↪(v'Av) / (v'v)
    eigenvalue = dot(A * v, v) / dot(v, v)
    print("The leading eigenvalue is ")
    display(eigenvalue)
    print("The leading eigenvector is ")
    display(v)
    return eigenvalue, v
end
```

[144]:
```
power_method_example (generic function with 1 method)
```

[153]:
```
power_method_example(A_Power, 5)
```

```
The updated vector is

3-element Vector{Float64}:
 8.161030539868374
 4.627094182971236
 3.3134414620023938

The updated vector is
```

```
3-element Vector{Float64}:
 8.802075512800766
 5.119083565097348
 4.899340934828883

The updated vector is

3-element Vector{Float64}:
 8.979684450238407
 5.035582211875657
 4.7814018188575576

The updated vector is

3-element Vector{Float64}:
 8.974382615152209
 5.012466428032355
 4.820158385228858

The updated vector is

3-element Vector{Float64}:
 8.980518953209506
 5.0034304465975765
 4.818365892732459

The leading eigenvalue is

11.35345127702423

The leading eigenvector is

3-element Vector{Float64}:
 0.7909951454764633
 0.4406971595526109
 0.42439685837071095
```

[153]:
```
(11.35345127702423, [0.7909951454764633, 0.4406971595526109,
 0.42439685837071095])
```

[145]:
```julia
A_Deflation = [[7 3 5]; [3 5 1]; [5 1 1]]
```

[145]:
```
3×3 Matrix{Int64}:
 7  3  5
 3  5  1
 5  1  1
```

[156]:
```julia
# Function to find all eigenvalues and eigenvectors of a symmetric matrix using␣
 ↪deflation (Example)
function deflation_example(A, max_iter)
    k = size(A, 1)
    eigenvalues = []
    eigenvectors = []
    # Main loop for finding all eigenvalues and eigenvectors
    for i = 1:k
        eigenvalue, eigenvector = power_method(A, max_iter)
        push!(eigenvalues, eigenvalue)
        push!(eigenvectors, eigenvector)
        A = A - eigenvalue * eigenvector * transpose(eigenvector)
        print("The new matrix is ")
        display(A)
    end

    print("The list of eigenvalues is")
    for i = 1:size(eigenvalues, 1)
        display(eigenvalues[i])
    end
    print("The list of eigenvectors is")
    for i = 1:size(eigenvectors, 1)
        display(eigenvectors[i])
    end
    return eigenvalues, eigenvectors
end
```

[156]:
```
deflation_example (generic function with 1 method)
```

[157]:
```
deflation_example(A_Deflation, 5)
```

```
The new matrix is

3×3 Matrix{Float64}:
 -0.107028  -0.953945   1.18611
 -0.953945   2.80025   -1.12183
  1.18611   -1.12183   -1.04667

The new matrix is

3×3 Matrix{Float64}:
 -0.149139  -0.680337   0.981404
 -0.680337   1.02252    0.208205
  0.981404   0.208205  -2.04176

The new matrix is

3×3 Matrix{Float64}:
  0.272566   -0.539916   0.0513521
 -0.539916    1.06927   -0.101489
  0.0513521  -0.101489   0.00943407

The list of eigenvalues is

11.353452085530611

2.8149303216918904
```

```
-2.5196568848768055

The list of eigenvectors is

3-element Vector{Float64}:
 0.7911886727089043
 0.4401721932271454
 0.42458088097172625

3-element Vector{Float64}:
 -0.12230960774710155
  0.7946934140708866
 -0.5945609619585772

3-element Vector{Float64}:
  0.40910363121697507
  0.13622557702085586
 -0.9022620523386873
```

```
[157]: (Any[11.353452085530611, 2.8149303216918904, -2.5196568848768055],
        Any[[0.7911886727089043, 0.4401721932271454, 0.42458088097172625],
        [-0.12230960774710155, 0.7946934140708866, -0.5945609619585772],
        [0.40910363121697507, 0.13622557702085586, -0.9022620523386873]])
```