AMATH 584: Numerical Linear Algebra: Exercise 28.1
Sid Meka

What happens if you apply the unshifted QR algorithm to an orthogonal matrix? Figure out the answer, and then explain how it relates to Theorem 28.4.

When A is orthogonal, the unshifted QR algorithm to compute eigenvalues does not make progress and simply returns the diagonal entries of A as the eigenvalues if stopped after a fixed number of iterations. This behavior arises because the QR decomposition of an orthogonal matrix A yields $Q^{(k)} = A$ and $R^{(k)} = I$ since A is already orthogonal and satisfies $A = Q^{(k)}R^{(k)}$. Additionally, without a shift, as in the unshifted QR Algorithm, the algorithm does not separate the eigenvalues by magnitude as it would in the case of a symmetric matrix with distinct eigenvalues. Instead $A^{(k)}$ will converge to a block diagonal matrix containing rotations or signs depending on the eigenvalues of A. We will generally not see this process happen rapidly as there is no mechanism here to accelerate separation of eigenvalues as we don't have a shift as this is unshifted. Consequently, from Algorithm 28.1, we now have that the iterative update $A^{(k+1)} = R^{(k)}Q^{(k)}$. Notice that $R^{(k)}Q^{(k)} = IA = A$ leaving our matrix, A, unchanged.

Theorem 28.4 states that the convergence of the QR algorithm requires the leading $n + 1$ eigenvalues to be distinct in absolute value. However, this condition is not satisfied when A is an orthogonal matrix, as all its eigenvalues lie on the unit circle in the complex plane. Thus, all these eigenvalues have a magnitude of 1 by definition of lying on the unit circle in the complex plane. For orthogonal matrices, the eigenvalues of A are of equal magnitude ($|\lambda_i| = 1$), violating the strict inequality $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_m|$ mentioned in Theorem 28.4. Specifically, all eigenvalues have the same magnitude of 1, and thus, there is the prevention of the algorithm converging. Thus, Theorem 28.4 is not satisfied.

30.1. Derive $\tan(2\theta) = \frac{2d}{b-a}$, and give a precise geometric interpretation of the transformation $J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{bmatrix}$ based on this choice of θ.

Given a symmetric matrix: $A = \begin{bmatrix} a & d \\ d & b \end{bmatrix}$.

We also aim to find $A' = J^T A J$.

$$J = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad J^T = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$A' =$

$J^T A J =$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a & d \\ d & b \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} =$$

$$\begin{bmatrix} a\cos\theta - d\sin\theta & -b\sin\theta + d\cos\theta \\ a\sin\theta + d\cos\theta & b\cos\theta + d\sin\theta \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta(a\cos\theta - d\sin\theta) - \sin\theta(-b\sin\theta + d\cos\theta) & \sin\theta(a\cos\theta - d\sin\theta) + \cos\theta(-b\sin\theta + d\cos\theta) \\ \cos\theta(a\sin\theta + d\cos\theta) - \sin\theta(b\cos\theta + d\sin\theta) & \sin\theta(a\sin\theta + d\cos\theta) + \cos\theta(b\cos\theta + d\sin\theta) \end{bmatrix} =$$

$$\begin{bmatrix} (a\cos^2\theta - d\cos\theta\sin\theta) + (b\sin^2\theta - d\cos\theta\sin\theta) & (a\cos\theta\sin\theta - d\sin^2\theta) + (-b\cos\theta\sin\theta + d\cos^2\theta) \\ (a\sin\theta\cos\theta + d\cos^2\theta) + (-b\cos\theta\sin\theta - d\sin^2\theta) & (a\sin^2\theta + d\cos\theta\sin\theta) + (b\cos^2\theta + d\cos\theta\sin\theta) \end{bmatrix} =$$

$$\begin{bmatrix} a\cos^2\theta + b\sin^2\theta - 2d\cos\theta\sin\theta & a\cos\theta\sin\theta - b\cos\theta\sin\theta + d(\cos^2\theta - \sin^2\theta) \\ a\cos\theta\sin\theta - b\cos\theta\sin\theta + d(\cos^2\theta - \sin^2\theta) & a\sin^2\theta + b\cos^2\theta + 2d\cos\theta\sin\theta \end{bmatrix} =$$

$$\begin{bmatrix} a\cos^2\theta + b\sin^2\theta - 2d\cos\theta\sin\theta & (a-b)\cos\theta\sin\theta + d(\cos^2\theta - \sin^2\theta) \\ (a-b)\cos\theta\sin\theta + d(\cos^2\theta - \sin^2\theta) & a\sin^2\theta + b\cos^2\theta + 2d\cos\theta\sin\theta \end{bmatrix}$$

$A'_{12} = A'_{21} =$

$(a-b)\cos\theta\sin\theta + d(\cos^2\theta - \sin^2\theta) =$

$(a-b)\frac{\sin(2\theta)}{2} + d(\cos(2\theta)) =$

$\frac{a-b}{2}\sin(2\theta) + d\cos(2\theta)$

Note that from the equation, we know that

$$J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{bmatrix}.$$

This means that the off diagonal terms are equal to 0. So, we have that $\frac{a-b}{2}\sin(2\theta) + d\cos(2\theta) = 0$. Note that we can divide by $\cos(2\theta)$ as $a \neq b$ for the formula we are trying to derive for $\tan(2\theta)$. If $a=b$, $\tan(2\theta)$ would equal $\frac{2d}{0}$ which would be undefined and not what we are trying to show. Also, if $a=b$, $\cos(2\theta)=0$ from using trigonometric definitions as $b-a$ would be the adjacent part in trigonometry and thus the numerator of $\cos(2\theta)$ and denominator of $\tan(2\theta)$.

Thus, we proceed by saying that $\cos(2\theta) \neq 0$.

Now, we have that $\frac{a-b}{2}\sin(2\theta) + d\cos(2\theta) = 0$.

Dividing by our $\cos(2\theta)$, which we can do as $\cos(2\theta) \neq 0$, we have $\frac{a-b}{2}\tan(2\theta) + d = 0$.

$$\frac{a-b}{2}\tan(2\theta) + d = 0$$

$$\frac{a-b}{2}\tan(2\theta) = -d$$

$$(a-b)\tan(2\theta) = -2d$$

$$\tan(2\theta) = -\frac{2d}{a-b}$$

$$\tan(2\theta) = \frac{2d}{b-a}$$

Thus, we have derived $\tan(2\theta) = \frac{2d}{b-a}$.

The transformation $A' = J^T A J$ applies a rotation to the matrix A using the Givens rotation matrix J:

$$J = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

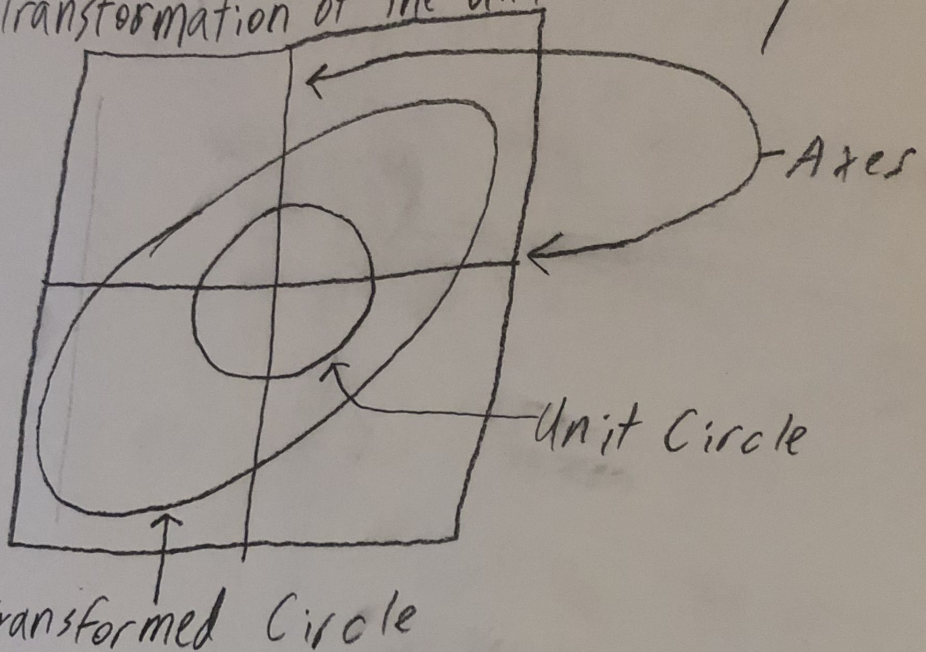
The geometric interpretation is as follows:

The transformation described where $J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{bmatrix}$ represents the transformation of A in the basis of its eigenvectors. The matrix A transforms the unit circle into an ellipse E with its axes aligned to the eigenvectors of A. The matrix $J^T A J$ is a diagonal matrix that transforms the unit circle, in the eigenvector coordinates, into an ellipse E'.

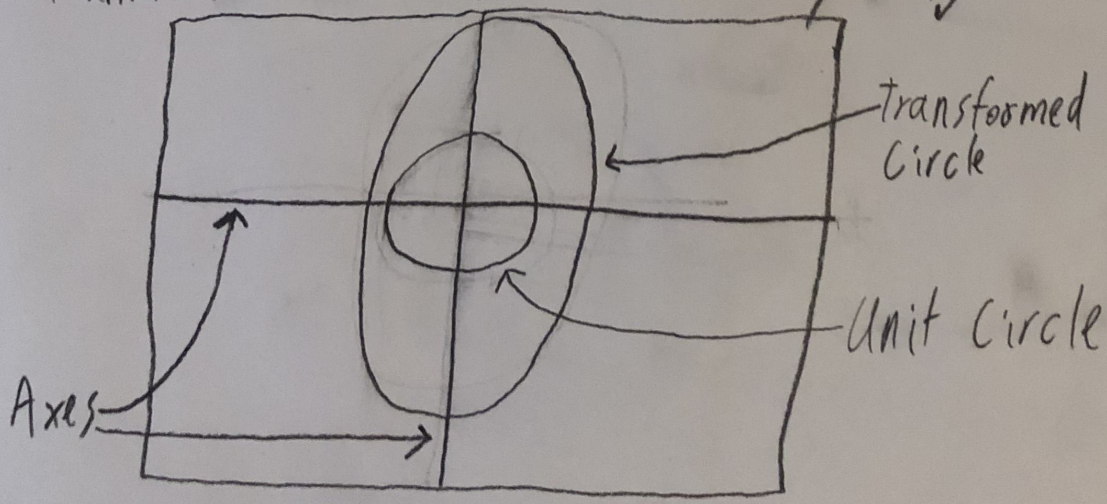
The ellipse E' is aligned with the eigenvectors coordinate system effectively representing a geometric rotation by an angle θ of our ellipse E created by our original matrix A. Additionally, the rotation matrix J aligns the principal axes of the quadratic form represented by A with the standard basis of $\mathbb{R}^2$.

Here are some diagrams to help visualize this geometrically:

Transformation of the Unit Circle by A



Transformation of the Unit Circle by J^TAJ



Note that there aren't specific values associated with the Transformed Circle as the Transformed Circle in both images is supposed to be general.

AMATH 584: Numerical Linear Algebra: Exercise 30.3
Sid Meka

Show that if the largest off diagonal entry is annihilated at each step of the Jacobi algorithm, then the sum of the squares of the off diagonal entries decreases by at least the factor $1 - \frac{2}{m^2-m}$ at each step.

We will say that $A' = J^T A J$ where A is the symmetric matrix whose eigenvalues we want to find and J is the Jacobi rotation matrix, a unitary matrix, designed to have the off diagonal terms equal to 0 while the terms on the diagonal are not equal to 0 for A'.

$$\text{We can say } A = \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix}.$$

Note that by applying our A and J appropriately we have:

$$a'_{pp} = \cos^2(\theta)a_{pp} + \sin^2(\theta)a_{qq} - 2\cos(\theta)\sin(\theta)a_{pq}$$

and

$$a'_{qq} = \sin^2(\theta)a_{pp} + \cos^2(\theta)a_{qq} - 2\cos(\theta)\sin(\theta)a_{pq}$$

We also have $\tan(2\theta) = \dfrac{2a_{pq}}{a_{qq} - a_{pp}}$.

Let $\text{Off}(K)$ denote the sum of squares of all off diagonal entries of an arbitrary matrix K.

After applying the transformation $A' = J^T A J$, the matrix A' has updated entries. Importantly:

- The diagonal entries a_{ii} for $i \neq p, q$ remain unchanged.
- The off-diagonal entry a_{pq} is set to 0, and similarly a_{qp} is 0 because the transformation annihilates these terms.
- The Frobenius norm, or the sum of the squares of all entries, is preserved because J is unitary.

The sum of the squares of the off-diagonal elements in A', $\text{Off}(A')$, can be expressed as:

$$\text{Off}(A') = \|A'\|_F^2 - \sum_i (a'_{ii})^2 \ ,$$

where $\|A'\|_F^2$ is the Frobenius norm squared, summing over all entries of A'.

Now, for a **Step by Step Derivation of Off**(A'):

1. Frobenius Norm Preservation: The Frobenius norm of A is preserved under the unitary transformation J, so $||A'||_F^2 = ||A||_F^2$.

   This means:
   $$||A'||_F^2 = \sum_{i,j} (a'_{ij})^2 = ||A||_F^2 = \sum_{i,j} (a_{ij})^2$$

2. Changes to the Off Diagonal Entries:
   Among the off diagonal entries:

   - The largest off diagonal entry a_{pq}^2 is eliminated, so its contribution is removed.
   - The remaining off diagonal entries are modified slightly due to the Jacobi rotation, but their squared contributions decrease because the rotation is designed to reduce the off diagonal sum.

We subtract $2a_{pq}^2$ because the Jacobi algorithm annihilates both a_{pq} and a_{qp} directly reducing $\text{Off}(A)$ by the sum of their squared contributions. This reduction forms the foundation for showing how the off diagonal sum decreases iteratively. Thus, the reduction in $\text{Off}(A)$ comes primarily from the term $-2a_{pq}^2$ reflecting the removal of a_{pq} and a_{qp}.

3. Thus, we have that
$$\text{Off}(A') = \text{Off}(A) - 2a_{pq}^2$$
Note how this shows how the sum of the squares of the off diagonal entries decreases by $2a_{pq}^2$ with a_{pq} being the largest off diagonal entry.

   This can also be written as
$$\text{Off}(A) = \text{Off}(A') + 2a_{pq}^2$$

Now, we consider the inequality portion. Specifically, to connect this with the desired inequality, note that a_{pq}^2, as the largest off diagonal entry satisfies:

$$a_{pq}^2 \geq \frac{\text{Off}(A)}{m^2 - m}$$

We can also say this as:

$$a_{pq}^2 \geq \frac{\text{Off}(A') + 2a_{pq}^2}{m^2 - m}$$

We have that

$$\text{Off}(A) - 2a_{pq}^2 \leq \text{Off}(A)\left(1 - \frac{2}{m^2 - m}\right)$$

Or more simply

$$\text{Off}(A') \leq \text{Off}(A)\left(1 - \frac{2}{m^2 - m}\right)$$

Thus, we have shown with this inequality that if the largest off diagonal entry is annihilated at each step of the Jacobi algorithm, then the sum of the squares of the off diagonal entries decreases by at least the factor $1 - \frac{2}{m^2-m}$ at each step.

AMATH 584: Numerical Linear Algebra: Homework 8 A1 Part a
Sid Meka

Consider the 10×10 matrix:

$$A = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}$$

What information does the Gerschgorin's theorem tell you about this matrix?

Gerschgorin's Theorem: Let $A = [a_{ij}]$ be an $n \times n$ matrix. For each $i \in \{1, 2, \ldots, n\}$, define the Gerschgorin disk $D(a_{ii}, R_i)$ in the complex plane, where

$$D(a_{ii}, R_i) = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \right\}$$

Moreover,

$$R_i = \sum_{j \neq i} |a_{ij}|$$

Thus, we have every eigenvalue of A lies within at least one of the Gerschgorin disks:

$$\bigcup_{i=1}^{n} D(a_{ii}, R_i)$$

In our case, all the diagonal entries of A are 2, and the Gerschgorin disks are centered at 2. The radius of each disk is given by the sum of the absolute values of the off diagonal entries in the corresponding row. Note that we have our superdiagonal and subdiagonal entries all equal to -1. So, we have our entries for A as:

- 2 if our entry is on the diagonal
- -1 if our entry is on the superdiagonal or subdiagonal.
- 0 for all other entries

So, our radius is at maximum 2.

So, Gerschgorin's theorem tells us that all the eigenvalues of A lie inside a disk of radius 2 centered at 2 in the complex plane.

```
warning('off', 'all');
%This ensures that warnings don't show in my PDF file.
```

Part a Answered on previous page

Part b: Implementing the Power Method

```
function [eigenvalue, eigenvector] = power_method(A, max_iterations, tol)
    % POWER_METHOD Computes the largest eigenvalue and corresponding
eigenvector
    % using the Power Method.
    % Inputs:
    %   A               - Square matrix
    %   max_iterations  - Maximum number of iterations (default: 1000)
    %   tol             - Convergence tolerance (default: 1e-10)
    % Outputs:
    %   eigenvalue      - Approximation of the largest eigenvalue
    %   eigenvector     - Approximation of the corresponding eigenvector

    % Default arguments
    if nargin < 2
        max_iterations = 1000;
    end
    if nargin < 3
        tol = 1e-10;
    end

    % Initialize a random vector
    n = size(A, 1);
    v = rand(n, 1); % Random vector
    v = v / norm(v); % Normalize the vector
    eigenvalue_old = 0; % Placeholder for the previous eigenvalue

    for k = 1:max_iterations
        % Multiply A with the current vector
        w = A * v;

        % Approximate the eigenvalue
        eigenvalue = dot(w, v);

        % Normalize the resulting vector
        v = w / norm(w);

        % Convergence check using the stopping criteria
        if norm(A * v - eigenvalue * v) < tol
            eigenvector = v;
            return;
        end

        % Convergence check for eigenvalue
```

```
        if abs(eigenvalue - eigenvalue_old) < tol
            eigenvector = v;
            return;
        end

        % Update eigenvalue
        eigenvalue_old = eigenvalue;
    end

    error('Power method did not converge within the maximum number of
iterations.');
end

% Example: Define the 10x10 tridiagonal matrix A
size = 10;
A = 2 * eye(size) - diag(ones(size-1, 1), 1) - diag(ones(size-1, 1), -1);

% Apply the power method
[eigenvalue, eigenvector] = power_method(A);

% Display the results
disp('Largest Eigenvalue:');
```

```
Largest Eigenvalue:
```

```
disp(eigenvalue);
```

```
    3.9190
```

```
disp('Corresponding Eigenvector:');
```

```
Corresponding Eigenvector:
```

```
disp(eigenvector);
```

```
    0.1201
   -0.2305
    0.3222
   -0.3879
    0.4221
   -0.4221
    0.3879
   -0.3223
    0.2305
   -0.1201
```

Part c: Implementing the pure QR Algorithm

```
function [Ak, eigenvalues] = qr_algorithm(A, max_iterations, tol)
    % QR_ALGORITHM Implements the pure QR algorithm (without shifts) to
diagonalize a matrix.
    %
    % Inputs:
    %   A               - The input square matrix (must be symmetric for
eigenvalues to converge)
```

```
    %   max_iterations  - Maximum number of iterations (default: 100000)
    %   tol             - Tolerance for convergence based on off-diagonal
entries (default: 1e-10)
    %
    % Outputs:
    %   Ak              - The diagonalized matrix.
    %   eigenvalues     - The eigenvalues of the matrix as a vertical vector.

    % Default parameters
    if nargin < 2
        max_iterations = 100000;
    end
    if nargin < 3
        tol = 1e-10;
    end

    % Initialize the matrix to iterate upon
    Ak = A;

    for iter = 1:max_iterations
        % Perform the QR decomposition
        [Q, R] = qr(Ak);

        % Update Ak = R * Q
        Ak = R * Q;

        % Check convergence: if the off-diagonal entries are small enough
        off_diagonal_sum = sum(sum(abs(Ak - diag(diag(Ak)))));
        if off_diagonal_sum < tol
            disp('Convergence achieved.');
            eigenvalues = diag(Ak); % Extract diagonal entries as eigenvalues
            return;
        end
    end

    error('QR algorithm did not converge within the maximum number of
iterations.');
end

% Example: Define the 10x10 tridiagonal matrix A
size = 10;
A = 2 * eye(size) - diag(ones(size-1, 1), 1) - diag(ones(size-1, 1), -1);

% Apply the QR algorithm
[diagonalized_matrix, eigenvalues] = qr_algorithm(A);
```

```
Convergence achieved.
```

```
% Display the results
```

```
disp('Diagonalized Matrix:');
```

```
Diagonalized Matrix:
```

```
disp(diagonalized_matrix);
```

```
    3.9190   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.0000
   -0.0000    3.6825   -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000
         0   -0.0000    3.3097   -0.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000
         0         0   -0.0000    2.8308   -0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000
         0         0         0   -0.0000    2.2846   -0.0000   -0.0000    0.0000   -0.0000   -0.0000
         0         0         0         0   -0.0000    1.7154   -0.0000   -0.0000    0.0000    0.0000
         0         0         0         0         0   -0.0000    1.1692    0.0000   -0.0000   -0.0000
         0         0         0         0         0         0   -0.0000    0.6903    0.0000   -0.0000
         0         0         0         0         0         0         0   -0.0000    0.3175   -0.0000
         0         0         0         0         0         0         0         0    0.0000    0.0810
```

```
disp('Eigenvalues as a vertical vector:');
```

```
Eigenvalues as a vertical vector:
```

```
disp(eigenvalues);
```

```
    3.9190
    3.6825
    3.3097
    2.8308
    2.2846
    1.7154
    1.1692
    0.6903
    0.3175
    0.0810
```

Part d: Picking the fifth Eigenvalue

```
function eigenvector = inverse_iteration_with_shift(A, shift,
max_iterations, tol)
    % INVERSE_ITERATION_WITH_SHIFT Computes the eigenvector corresponding to
a given shift
    % using the inverse iteration method.
    %
    % Inputs:
    %   A               - The input square matrix
    %   shift           - The shift (approximate eigenvalue)
    %   max_iterations  - Maximum number of iterations (default: 100)
    %   tol             - Convergence tolerance (default: 1e-6)
    %
    % Output:
    %   eigenvector     - The eigenvector corresponding to the eigenvalue
near the shift

    % Default arguments
    if nargin < 3
        max_iterations = 100;
    end
    if nargin < 4
```

```
        tol = 1e-6;
    end

    % Size of the matrix and initialization
    n = size(A, 1);
    I = eye(n); % Identity matrix
    A_shifted = A - shift * I; % Shifted matrix
    v = rand(n, 1); % Random initial vector
    v = v / norm(v); % Normalize the initial vector

    for iter = 1:max_iterations
        % Solve the linear system (A - shift * I) w = v
        try
            w = A_shifted \ v; % MATLAB's backslash operator solves linear
systems
        catch
            error('Matrix is singular or nearly singular with the given
shift.');
        end

        % Normalize the resulting vector
        v_next = w / norm(w);

        % Check for convergence
        if norm(v_next - v) < tol
            eigenvector = v_next; % Converged eigenvector
            return;
        end

        % Update for the next iteration
        v = v_next;
    end

    error('Inverse iteration did not converge within the maximum number of
iterations.');
end

% Example: Define the 10x10 tridiagonal matrix A
size = 10;
A = 2 * eye(size) - diag(ones(size-1, 1), 1) - diag(ones(size-1, 1), -1);

% Perform the QR algorithm to get the eigenvalues
[~, eigenvalues] = qr_algorithm(A);
```

```
Convergence achieved.
```

```
% Use the fifth eigenvalue as the shift
shift = eigenvalues(5);
disp(eigenvalues(5));
```

```
    2.2846
```

```
% Perform inverse iteration with the fifth eigenvalue as the shift
eigenvector = inverse_iteration_with_shift(A, shift);

% Display the result
disp('Eigenvector corresponding to the fifth eigenvalue:');
```

```
Eigenvector corresponding to the fifth eigenvalue:
```

```
disp(eigenvector);
```

```
   -0.4221
    0.1201
    0.3879
   -0.2305
   -0.3223
    0.3223
    0.2305
   -0.3879
   -0.1201
    0.4221
```