Iterative Methods for Eigenvalue and Eigenvector Computation: Power and QR Methods

Barry Zheng, Ziduo Yi

December 7, 2024

Abstract

Eigenvalue and eigenvector computations are essential in various scientific and engineering fields, and iterative methods offer practical solutions for large or ill-conditioned matrices. This paper compares two widely used iterative methods: the Power method and the QR method. The Power method focuses on calculating the dominant eigenvalue and its corresponding eigenvector, iteratively refining an initial guess. Its convergence is influenced by the spectral gap, with faster convergence for matrices where the largest eigenvalue is significantly larger than the second-largest. In contrast, the QR method computes all eigenvalues of a matrix through repeated QR decompositions, converging reliably for both symmetric and non-symmetric matrices. While the Power method is efficient for matrices with a large spectral gap, the QR method is more versatile and computationally intensive but suitable for complete eigenvalue calculations. Experimental results demonstrate that both methods converge faster for matrices with a larger spectral gap, though the Power method struggles with small gaps, while the QR method shows more consistent convergence. This paper provides a detailed analysis of the convergence properties and performance of both methods, offering insights into their appropriate use in practical applications.

1 Introduction

Eigenvectors are fundamental in linear algebra, revealing how transformations scale along specific directions. They are crucial across fields like machine learning for dimensionality reduction, physics for solving differential equations, and engineering for stability analysis. By simplifying complex systems and uncovering intrinsic structures, eigenvectors play a key role in understanding and solving real-world problems.

Two popular iterative methods for eigenvalue computation are the Power method and the QR method. These methods are widely used due to their efficiency and robustness. This paper aims to provide a thorough analysis of both methods, offering insight into their derivations, behavior, and speed of convergence.

2 Iterative Methods

2.1 The Power Method

The Power method is a simple iterative algorithm designed to compute the dominant eigenvalue and its corresponding eigenvector of a square matrix.

2.1.1 The Power Method

Given a matrix A, the Power method begins with an initial guess for an eigenvector, say $\mathbf{x}_0$, and iteratively computes the next approximation:

$$\mathbf{x}_{k+1} = \frac{A\mathbf{x}_k}{\|A\mathbf{x}_k\|}$$

where $\|\cdot\|$ denotes the magnitude of the vector. The idea behind the Power method is that, for most matrices, the vector $\mathbf{x}_k$ will eventually converge to the eigenvector associated with the largest eigenvalue λ_1 of A, and the corresponding eigenvalue can be estimated as:

$$\lambda_k = \frac{\mathbf{x}_k^T A\mathbf{x}_k}{\mathbf{x}_k^T\mathbf{x}_k}.$$

2.1.2 Derivation of the Power Method

Assume A is diagonalizable with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. Any initial vector $\mathbf{x}_0$ can be expressed as a linear combination of these eigenvectors:

$$\mathbf{x}_0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n$$

The next approximation $\mathbf{x}_1 = A\mathbf{x}_0$ will be:

$$\mathbf{x}_1 = c_1\lambda_1\mathbf{v}_1 + c_2\lambda_2\mathbf{v}_2 + \cdots + c_n\lambda_n\mathbf{v}_n.$$

When A is applied repeatedly to $\mathbf{x}_0$ using $Ax_0 = \lambda_i v_i$ where λ_i and v_i are the eigenvalues and eigenvectors of the matrix, respectively:

$$A^k\mathbf{x}_0 = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + \cdots + c_n\lambda_n^k\mathbf{v}_n$$

In order to prevent the vector v_1 from becoming computationally inefficient (i.e. too small or large), we normalize $A^k\mathbf{x}_0$ at each step. That way the actual values of the matrix are not affected too heavily by overflow. This diminishes the influence of the smaller eigenvalues, leaving only the direction of $\mathbf{v}_1$. If $|\lambda_1| > |\lambda_2|$, the term $c_1\lambda_1^k\mathbf{v}_1$ dominates as $k \to \infty$. By repeatedly applying the matrix A, the term involving the dominant eigenvalue λ_1 will dominate, as $|\lambda_1| > |\lambda_2|$. After enough iterations, the vector $\mathbf{x}_k$ will converge to the eigenvector $\mathbf{v}_1$, and the corresponding eigenvalue can be computed.

2.1.3 Example of the Power Method

Let:

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

1. INITIALIZATION

   Choose $\mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

2. ITERATIONS

$$\mathbf{y}_1 = A\mathbf{x}_0 = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\mathbf{x}_1 = \frac{\mathbf{y}_1}{\|\mathbf{y}_1\|} = \frac{\begin{bmatrix} 5 \\ 5 \end{bmatrix}}{\sqrt{5^2 + 5^2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\mathbf{y}_2 = A\mathbf{x}_1 = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{5}{\sqrt{2}} \\ \frac{5}{\sqrt{2}} \end{bmatrix}$$

$$\mathbf{x}_2 = \frac{\mathbf{y}_2}{\|\mathbf{y}_2\|} = \frac{\begin{bmatrix} \frac{5}{\sqrt{2}} \\ \frac{5}{\sqrt{2}} \end{bmatrix}}{\sqrt{(\frac{5}{\sqrt{2}})^2 + (\frac{5}{\sqrt{2}})^2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

The process terminates because $x_1 = x_2$ so the normalized vector is constant, so $x_1 = x_2 = v_1$. We can now use the following equation to solve for eigenvalue

$$\lambda_1 = \frac{x_1^T A x_1}{x_1^T x_1} = \frac{\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{5}{\sqrt{2}} \\ \frac{5}{\sqrt{2}} \end{bmatrix}}{1}$$

which we get

$$\lambda_1 \approx 5, \quad \mathbf{v}_1 \approx \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

2.1.4 Convergence of the Power Method

The convergence of the Power method depends on the spectral gap, defined as the ratio of the largest eigenvalue to the second-largest eigenvalue:

$$\text{Spectral Gap} = \frac{|\lambda_1|}{|\lambda_2|}.$$

For convergence, it is essential that $|\lambda_1| > |\lambda_2|$. If the spectral gap is large, the Power method will converge quickly. However, if the gap is small, the convergence will be slow, and the method may struggle to isolate the dominant eigenvalue.

As we stated earlier, by normalizing $A^k\mathbf{x}_0$ at each step, the influence of the smaller eigenvalues is diminished, leaving only the direction of $\mathbf{v}_1$. If $|\lambda_1| > |\lambda_2|$, the term $c_1\lambda_1^k\mathbf{v}_1$ dominates as $k \to \infty$, and thus a large gap allows faster convergence.

2.2 The QR Method

Unlike the Power method, which only computes the dominant eigenvalue and its corresponding eigenvector, the QR method can compute all eigenvalues of a matrix simultaneously. The QR method works by iteratively decomposing the matrix into orthogonal and upper triangular matrices using the QR decomposition and then reassembling the matrix to gradually reveal its eigenvalues.

2.2.1 QR Decomposition and Basic Idea

The QR method is based on the QR decomposition, which is the factorization of a matrix A into two matrices: Q, an orthogonal matrix, and R, an upper triangular matrix. Specifically, given a matrix A, the decomposition is expressed as:

$$A = QR$$

where: Q is an orthogonal matrix, meaning $Q^TQ = I$, R is an upper triangular matrix.

The QR method iterates the following steps:

1. Perform QR decomposition on the matrix A_k to obtain Q_k and R_k, i.e., $A_k = Q_kR_k$.
2. Compute the next matrix $A_{k+1} = R_kQ_k$.
3. Repeat this process until A_k converges to an upper triangular matrix, whose diagonal elements are the eigenvalues of A.

2.2.2 Example of the QR Method

Consider:

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

1. INITIALIZATION

   Set $A_0 = A$.

2. QR FACTORIZATION

   Factor $A_0 = Q_0 R_0$, where:

$$Q_0 = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \quad R_0 = \begin{bmatrix} \sqrt{5} & \sqrt{2} \\ 0 & \sqrt{2} \end{bmatrix}$$

3. RECOMBINE

   Compute:

$$A_1 = R_0 Q_0 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

We now have that the eigenvalues are $\lambda_1 = 3$ and $\lambda_2 = 1$, and they appear on the diagonal of A_1.

2.2.3 Derivation of the QR Method

The QR method relies on the fact that similarity transformations preserve the eigenvalues of a matrix. A similarity transformation of a matrix A is obtained by applying a sequence of orthogonal matrices:

$$A' = QAQ^T$$

where Q is orthogonal. Since orthogonal matrices preserve eigenvalues, applying this transformation repeatedly will eventually reveal the eigenvalues of A.

The iterative process starts with the matrix $A_0 = A$. In each step, we compute the QR decomposition of A_k, and then form the new matrix $A_{k+1} = R_k Q_k$, which is similar to A_k. As the process proceeds, the matrix A_k converges to an upper triangular matrix. The eigenvalues of A are the diagonal elements of this upper triangular matrix, and the method continues to refine the estimates of the eigenvalues with each iteration.

The QR method does not require an initial guess for an eigenvector, unlike the Power method. This makes it more suitable for finding all eigenvalues of a matrix, even when the matrix has multiple eigenvalues.

2.2.4 Convergence of the QR Method

The convergence of the QR method depends on the matrix's structure. For general matrices, the QR algorithm converges to a diagonal matrix whose diagonal elements are the eigenvalues. However, the rate of convergence can vary. For matrices with well-separated eigenvalues, the convergence is typically fast, and the eigenvalues are revealed in a few iterations. For matrices with closely spaced eigenvalues, the convergence may be slower, and the algorithm may require a larger number of iterations.

An important modification to improve the speed of convergence for the QR method is called "shifting." To speed up convergence, the QR Method often incorporates shifts. In this approach, at each iteration, we apply a shift to the matrix before performing the QR decomposition, which accelerates the convergence of eigenvalues, especially for matrices with multiple eigenvalues.

1. Subtract a scalar μ (an approximation of an eigenvalue) from A_k:

$$A_k - \mu I$$

2. Perform the QR decomposition on the shifted matrix:

$$A_k - \mu I = Q_k R_k$$

3. Reconstruct A_{k+1} by adding the shift back:

$$A_{k+1} = R_k Q_k + \mu I$$

2.3 Differences between the Power Method and the QR Method

While both the Power and QR methods are iterative techniques for finding eigenvalues, they differ significantly in their approach and scope:

- **Power Method:** The Power method is simpler and computes only the dominant eigenvalue and its corresponding eigenvector. Its convergence speed depends on the spectral gap between the largest and second-largest eigenvalues of the matrix.
- **QR Method:** The QR method computes all the eigenvalues of a matrix and works by iteratively decomposing the matrix into orthogonal and upper triangular components. It is more computationally intensive than the Power method but provides a comprehensive solution to the eigenvalue problem.
- **Convergence:** The Power method is faster when the spectral gap between the largest and second-largest eigenvalues is large. However, it can be slow for matrices with small spectral gaps. The QR method converges more reliably for all types of matrices, and its convergence speed can be improved with shifting techniques.
- **Scope:** The Power method is suitable for finding just the largest eigenvalue and eigenvector, making it efficient for applications where only the dominant eigenvalue is of interest. The QR method is more versatile and can find all eigenvalues of a matrix, which is beneficial for problems requiring the full spectrum.

3 Experimental Results

3.1 Code Explanation

3.1.1 Power Method

The Power Method is implemented to find the dominant eigenvalue and eigenvector of a given matrix A. It begins by initializing a random vector x and normalizing it. Then, for each iteration, the method performs matrix-vector multiplication ($A \cdot x$) and normalizes the resulting vector. The eigenvalue is estimated using the following equation:

$$\lambda = \frac{x^T A x}{x^T x}$$

If the difference between the current and previous eigenvalues is smaller than a given tolerance (tol), the method is considered to have converged. If the method fails to converge within a set number of iterations (max_iter), it returns a flag indicating non-convergence. This is done in the function `power_method(A::AbstractMatrix, max_iter::Int=100, tol::Float64=1e-6)` The check for convergence occurs at:

$$\texttt{if abs}(\lambda\_new - \lambda\_old) < tol$$

This ensures that the method terminates early if convergence is achieved.

3.1.2 QR Method

The **QR Method** computes all eigenvalues of a matrix by iterating through QR decompositions. Initially, the matrix A is decomposed into orthogonal (Q) and upper triangular (R) matrices. The matrix is then updated by multiplying R and Q to form a new matrix, $A_k = R \cdot Q$. The process repeats until the matrix converges to an upper triangular form, at which point the diagonal elements represent the eigenvalues. Convergence is checked by ensuring that the difference between the current matrix and its diagonal form is below a specified tolerance:

```
if maximum(abs.(A_k .- Diagonal(diag(A_k)))) < tol
```

This is done in the function `qr_method(A::AbstractMatrix, max_iter::Int=100, tol::Float64=1e-6)` If convergence isn't achieved within the allowed iterations, the method returns the diagonal elements of the final matrix, which are approximate eigenvalues.

3.1.3 Testing

The test cases provide a framework to evaluate the performance of both the Power and QR methods on symmetric and non-symmetric matrices. The function `test_methods()` generates symmetric matrices with specified eigenvalue gaps and non-symmetric random matrices. For each matrix, it applies both methods and checks whether they converge within the allowed number of iterations. Note that eigen_gap refers to the spectral gap, and in our

matrix generation, the largest eigenvalue for each is 1. Results are printed for each test case in the following way:

```
println("Testing Power and QR Methods")
for size in sizes
    for eigen_gap in eigen_gaps
        A_sym = generate_symmetric_matrix(size, eigen_gap)
        println(@sprintf("Symmetric Matrix:  size=size, eigen_gap)))
        power_converged, power_vec, power_val = power_method(A_sym, iterations, toleranc
        println(" Power Method converged?  ", power_converged))
        qr_converged, qr_vals = qr_method(A_sym, iterations, tolerance)
        println(" QR Method converged?  ", qr_converged)
```

This file tests the methods on matrices of sizes 10x10, 100x100, and 1000x1000. and eigenvalue gaps of 25.0 and 5.0, and 1.0 for symmetric matrices, and it tested an arbitrary non-symmetric matrix of those same sizes as well. It prints whether each method converged or not, as seen in the output of the execution.

3.2 Code Results

The Julia implementation of the Power and QR methods was tested on matrices with varying properties. The results, shown in Table 1, reveal several interesting patterns regarding the convergence behavior of these methods.

Matrix Type	**Size**	**Power Method**	**QR Method**
Symmetric (eigen_gap = 25.0)	10	Converged to 1	Converged
Symmetric (eigen_gap = 5.0)	10	Converged to 1	Converged
Symmetric (eigen_gap = 1.0)	10	Converged to 1	Converged
Symmetric (eigen_gap = 25.0)	100	Converged to 1	Converged
Symmetric (eigen_gap = 5.0)	100	Converged to 1	Converged
Symmetric (eigen_gap = 1.0)	100	Converged to 1	Converged
Symmetric (eigen_gap = 25.0)	1000	Converged to 1	Converged
Symmetric (eigen_gap = 5.0)	1000	Converged to 1	Converged
Symmetric (eigen_gap = 1.0)	1000	Converged to 1	Converged
Non-symmetric	10	Did not converge	Did not converge
Non-symmetric	100	Did not converge	Did not converge
Non-symmetric	1000	Did not converge	Did not converge

Table 1: Convergence results of the Power and QR methods for symmetric and non-symmetric matrices.

From the experimental results, we observe the following:

- For symmetric matrices, both the Power and QR methods converge reliably, with the Power method succeeding in computing the dominant eigenvalue and the QR method successfully computing all eigenvalues.

- For non-symmetric matrices, neither method converges to a valid solution in this test case. This highlights the limitation of the Power method when applied to non-symmetric matrices and suggests that the QR method also struggles with non-symmetric matrices in some cases.

4 Convergence Speed and the Role of Eigenvalue Ratios

The speed of convergence in iterative eigenvalue methods, such as the Power Method and the QR Method, is significantly influenced by the spectral properties of the matrix, particularly the ratio of the largest eigenvalue to the second-largest eigenvalue. This ratio plays a critical role in determining how quickly the iterative process will converge to the dominant eigenvalue, and consequently, how effective the method will be for a given problem.

Eigen_gap	**Size**	**Iterations within Power Method**	**Iterations within QR Method**
10.0	10	4	20
5.0	10	5	21
2.0	10	10	22
1.0	10	12	25
10.0	100	5	22
5.0	100	6	23
2.0	100	8	25
1.0	100	14	28
10.0	1000	5	18
5.0	1000	6	19
2.0	1000	10	21
1.0	1000	15	25

Table 2: Results of the speed of convergence for the Power and QR methods for symmetric matrices.

4.1 Impact of the Eigenvalue Ratio on the Power Method

The Power Method is highly sensitive to spectral gap. If $|\lambda_1|$ is significantly larger than $|\lambda_2|$, the Power Method will converge rapidly, as the influence of the smaller eigenvalues will decay quickly after each iteration. However, when the spectral gap is small (i.e., when $|\lambda_1|$ and $|\lambda_2|$ are close in magnitude), the convergence of the Power Method slows down.

4.1.1 Convergence Rate for the Power Method

The convergence rate of the Power Method can be quantitatively understood by considering the dominant eigenvalue λ_1 and the second-largest eigenvalue λ_2. After k iterations, the error in the approximation to the dominant eigenvector decays as:[2]

$$\text{Error at iteration } k \sim \left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k$$

Thus, the rate of convergence is determined by the ratio $\frac{|\lambda_2|}{|\lambda_1|}$. If the spectral gap is large, the method converges quickly because the error reduces exponentially with each iteration. If the spectral gap is small, the convergence is slow because the error diminishes at a much slower rate.

For instance, if $|\lambda_1| = 2$ and $|\lambda_2| = 1$, the error will decrease by a factor of 0.5 after each iteration. However, if $|\lambda_1| = 2$ and $|\lambda_2| = 1.9$, the error will decrease by only a factor of 0.95 per iteration, which results in much slower convergence.

4.1.2 Example: Power Method Convergence with Varying Eigenvalue Ratios

To illustrate the effect of the spectral gap on convergence, consider the following experiments on a matrix with largest eigenvalue $\lambda_1 = 1.0$. The convergence behavior for these different values of λ_2 is summarized in Table 2 displayed earlier.

The table clearly shows that as the spectral gap decreases (i.e., as $|\lambda_1|$ and $|\lambda_2|$ become closer in magnitude), the number of iterations required for the Power Method to converge increases significantly.

4.2 Impact of the Eigenvalue Ratio on the QR Method

The QR Method, unlike the Power Method, is not as directly sensitive to the ratio of the largest to the second-largest eigenvalue. This is because the QR method computes all eigenvalues simultaneously by transforming the matrix into an upper triangular form. However, the spectral properties still affect the rate of convergence.

4.2.1 Convergence Rate for the QR Method

For the QR method, the convergence rate can be understood in terms of the distribution of the eigenvalues. If the eigenvalues are well-separated, the matrix A_k produced during the QR iterations will quickly converge to a diagonal matrix, with the diagonal entries corresponding to the eigenvalues of A. However, if the eigenvalues are clustered together (i.e., when the spectral gap is small), the convergence will be slower, as the iterations will have difficulty distinguishing between closely spaced eigenvalues.

The convergence rate of the QR method can also be related to the spectral gap of the matrix, which is defined as:

$$\text{spectral gap} = \frac{|\lambda_1|}{|\lambda_2|}$$

For matrices with a small spectral gap (i.e., when the eigenvalues are closer together), the convergence of the QR method will be slower. For matrices with a large spectral gap, the method will converge faster.

4.2.2 Example: QR Method Convergence with Varying Eigenvalue Ratios

The QR method's convergence was tested on matrices with the largest eigenvalue $\lambda_1 = 1$. The convergence behavior is summarized in Table 2 as well.

From the results, we can observe that while the QR method is less sensitive to the spectral gap than the Power method, the number of iterations still increases as the eigenvalues become closer in magnitude.

4.3 Comparison of Convergence Rates: Power vs. QR Methods

While both methods converge to the eigenvalues, their convergence rates depend on different factors:

- **Power Method:** The convergence rate is strongly dependent on the spectral gap. A larger spectral gap results in faster convergence, while a smaller gap results in slower convergence.
- **QR Method:** The QR method is less sensitive to the spectral gap, but still exhibits slower convergence for matrices with closely spaced eigenvalues. The method's convergence speed is influenced by the spectral gap and the distribution of eigenvalues.
- Note that as the size of the matrix increases, both methods slow down, but based on the data in Table 2, in general, the QR method is worse off, since the power method is more optimized in matrix-vector multiplications.

In cases where the spectral gap is large, the Power method may converge more quickly, especially when only the dominant eigenvalue is needed. However, for matrices with multiple eigenvalues or closely spaced eigenvalues, the QR method is generally more reliable and will compute all eigenvalues simultaneously.

4.4 Section Conclusion

The spectral gap is a crucial factor in determining the speed of convergence for the Power method. When this ratio is large, the Power method converges rapidly. However, for matrices with small spectral gaps, convergence is slower. On the other hand, the QR method is less sensitive to the spectral gap and can compute all eigenvalues simultaneously, making it more reliable for matrices with multiple eigenvalues or when all eigenvalues are required. Understanding these factors allows us to choose the appropriate method based on the matrix properties and the computational resources available.

5 Overall Conclusion

The Power and QR methods are both valuable tools for computing eigenvalues, each with its strengths and limitations. The Power method is fast and efficient for matrices with a large spectral gap, and is computationally more efficient for larger matrices, but it is limited in its scope, as it only computes the dominant eigenvalue and eigenvector. On the other hand, the QR method is more versatile, providing a full set of eigenvalues for any matrix. However, it is computationally more intensive and may require more iterations for convergence.

6 Julia Code

Here is the Julia code as a PDF, using a .ipynb to .pdf converter by iioiktoye:

iioiktoye

December 7, 2024

[2]:
```
function power_method(A::AbstractMatrix, max_iter::Int=100, tol::Float64=1e-6)
    n = size(A, 1)
    x = rand(n)  # Start with a random initial vector
    x /= norm(x)  # Normalize
     _old = 0.0
    for iter in 1:max_iter
        x = A * x
        x /= norm(x)  # Normalize
         _new = dot(x, A * x) / dot(x, x)  # Rayleigh quotient
        if abs( _new -  _old) < tol
            return true, x,  _new  # Converged
        end
         _old =  _new
    end
    return false, x,  _old  # Did not converge within max_iter
end
```

[2]: power_method (generic function with 3 methods)

[3]:
```
function qr_method(A::AbstractMatrix, max_iter::Int=100, tol::Float64=1e-6)
    Ak = A
    for iter in 1:max_iter
        Q, R = qr(Ak)  # QR decomposition
        Ak = R * Q  # Update Ak
        if maximum(abs.(Ak .- Diagonal(diag(Ak)))) < tol
            return true, diag(Ak)  # Converged: Return eigenvalues
        end
    end
    return false, diag(Ak)  # Did not converge within max_iter
end
```

[3]: qr_method (generic function with 3 methods)

[4]:
```
using LinearAlgebra
using Random
using Printf
```

```julia
# Function to create symmetric matrices with specific properties
function generate_symmetric_matrix(size::Int, eigen_gap::Float64;␣
 ↪random_state=42)
    Random.seed!(random_state)
    D = Diagonal([1.0, 1.0 / eigen_gap, fill(1.0 / (2 * eigen_gap), size - 2)...
 ↪]) #Largest Eigenvalue is 1
    Q = qr(randn(size, size)).Q  # Random orthogonal matrix
    return Q * D * Q'  # Symmetric matrix
end

# Function to create a non-symmetric matrix
function generate_nonsymmetric_matrix(size::Int; random_state=42)
    Random.seed!(random_state)
    return randn(size, size)
end

# Function to test the Power and QR methods
function test_methods()
    sizes = [10, 100, 1000]
    eigen_gaps = [25.0, 5.0, 1.0]
    iterations = 100
    tolerance = 1e-6

    println("Testing Power and QR Methods\n")

    for size in sizes
        for eigen_gap in eigen_gaps
            # Generate a symmetric matrix
            A_sym = generate_symmetric_matrix(size, eigen_gap)
            println(@sprintf("Symmetric Matrix: size=%d, eigen_gap=%.2f", size,␣
 ↪eigen_gap))

            # Test Power Method
            power_converged, power_vec, power_val = power_method(A_sym,␣
 ↪iterations, tolerance)
            println("  Power Method converged? ", power_converged)
            println("  Largest eigenvalue (Power Method): ", power_val)

            # Test QR Method
            qr_converged, qr_vals = qr_method(A_sym, iterations, tolerance)
            println("  QR Method converged? ", qr_converged)
        end

        # Generate a non-symmetric matrix
        A_nonsym = generate_nonsymmetric_matrix(size)
        println(@sprintf("Non-symmetric Matrix: size=%d", size))
```

```
        # Test Power Method on non-symmetric matrix
        power_converged, _, _ = power_method(A_nonsym, iterations, tolerance)
        println("  Power Method converged? ", power_converged)

        # Test QR Method on non-symmetric matrix
        qr_converged, _ = qr_method(A_nonsym, iterations, tolerance)
        println("  QR Method converged? ", qr_converged)
    end
end

test_methods()
```

```
Testing Power and QR Methods

Symmetric Matrix: size=10, eigen_gap=25.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999999968708
  QR Method converged? true
Symmetric Matrix: size=10, eigen_gap=5.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999603381332
  QR Method converged? true
Symmetric Matrix: size=10, eigen_gap=1.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999069879087
  QR Method converged? true
Non-symmetric Matrix: size=10
  Power Method converged? false
  QR Method converged? false
Symmetric Matrix: size=100, eigen_gap=25.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999999957403
  QR Method converged? true
Symmetric Matrix: size=100, eigen_gap=5.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999998417708
  QR Method converged? true
Symmetric Matrix: size=100, eigen_gap=1.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999996851552266
  QR Method converged? true
Non-symmetric Matrix: size=100
  Power Method converged? false
  QR Method converged? false
Symmetric Matrix: size=1000, eigen_gap=25.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999999846713
```

```
  QR Method converged? true
Symmetric Matrix: size=1000, eigen_gap=5.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999999979488923
  QR Method converged? true
Symmetric Matrix: size=1000, eigen_gap=1.00
  Power Method converged? true
  Largest eigenvalue (Power Method): 0.9999998478094786
  QR Method converged? true
Non-symmetric Matrix: size=1000
  Power Method converged? false
  QR Method converged? false
```

[5]:
```
function power_method_speed(A, max_iter=100, tol=1e-6)
    n = size(A, 1)
    x = rand(n)
    x /= norm(x)
     _old = 0.0
    iter_count = 0

    for k in 1:max_iter
        x = A * x
        x /= norm(x)
         _new = dot(x, A * x) / dot(x, x)
        iter_count += 1
        if abs( _new -  _old) < tol
            return iter_count, x,  _new
        end
         _old =  _new
    end
    return max_iter, x,  _old
end
```

[5]:
```
power_method_speed (generic function with 3 methods)
```

[6]:
```
function qr_method_speed(A, max_iter=100, tol=1e-6)
    Ak = A
    n = size(A, 1)
    iter_count = 0

    for k in 1:max_iter
        Q, R = qr(Ak)
        Ak = R * Q
        iter_count += 1
        if maximum(abs.(Ak .- Diagonal(diag(Ak)))) < tol
            return iter_count, diag(Ak)
        end
```

```
    end
    return max_iter, diag(Ak)
end
```

[6]:
```
qr_method_speed (generic function with 3 methods)
```

[8]:
```
using LinearAlgebra
using Random
using Printf

# Function to create symmetric matrices with specific properties
function generate_symmetric_matrix(size::Int, eigen_gap::Float64;␣
 ↪random_state=42)
    Random.seed!(random_state)
    D = Diagonal([1.0, 1.0 / eigen_gap, fill(1.0 / (2 * eigen_gap), size - 2)...
 ↪]) #Largest Eigenvalue is 1
    Q = qr(randn(size, size)).Q  # Random orthogonal matrix
    return Q * D * Q'  # Symmetric matrix
end

# Function to test Power Method Speed and QR Method Speed
function test_speed_methods()
    sizes = [10, 100, 1000]
    eigen_gaps = [10.0, 5.0, 2.0, 1.0]
    iterations = 100
    tolerance = 1e-6

    println("Testing Power Method Speed and QR Method Speed\n")

    for size in sizes
        for eigen_gap in eigen_gaps
            # Generate a symmetric matrix
            A = generate_symmetric_matrix(size, eigen_gap)
            println(@sprintf("Symmetric Matrix: size=%d, eigen_gap=%.2f", size,␣
 ↪eigen_gap))

            # Test Power Method Speed
            power_iter_count, power_vec, power_val = power_method_speed(A,␣
 ↪iterations, tolerance)
            println(@sprintf("  Power Method: iterations=%d, eigenvalue=%.6f",␣
 ↪power_iter_count, power_val))

            # Test QR Method Speed
            qr_iter_count, qr_vals = qr_method_speed(A, iterations, tolerance)
            println(@sprintf("  QR Method: iterations=%d, eigenvalues=%.6f, ...
 ↪", qr_iter_count, qr_vals[1]))
```

```
        end
    end
end

# Call the test function
test_speed_methods()
```

```
Testing Power Method Speed and QR Method Speed

Symmetric Matrix: size=10, eigen_gap=10.00
  Power Method: iterations=4, eigenvalue=1.000000
  QR Method: iterations=20, eigenvalues=1.000000, …
Symmetric Matrix: size=10, eigen_gap=5.00
  Power Method: iterations=5, eigenvalue=1.000000
  QR Method: iterations=21, eigenvalues=1.000000, …
Symmetric Matrix: size=10, eigen_gap=2.00
  Power Method: iterations=10, eigenvalue=1.000000
  QR Method: iterations=22, eigenvalues=1.000000, …
Symmetric Matrix: size=10, eigen_gap=1.00
  Power Method: iterations=12, eigenvalue=1.000000
  QR Method: iterations=25, eigenvalues=1.000000, …
Symmetric Matrix: size=100, eigen_gap=10.00
  Power Method: iterations=5, eigenvalue=1.000000
  QR Method: iterations=22, eigenvalues=1.000000, …
Symmetric Matrix: size=100, eigen_gap=5.00
  Power Method: iterations=6, eigenvalue=1.000000
  QR Method: iterations=23, eigenvalues=1.000000, …
Symmetric Matrix: size=100, eigen_gap=2.00
  Power Method: iterations=8, eigenvalue=1.000000
  QR Method: iterations=25, eigenvalues=1.000000, …
Symmetric Matrix: size=100, eigen_gap=1.00
  Power Method: iterations=14, eigenvalue=1.000000
  QR Method: iterations=28, eigenvalues=1.000000, …
Symmetric Matrix: size=1000, eigen_gap=10.00
  Power Method: iterations=5, eigenvalue=1.000000
  QR Method: iterations=18, eigenvalues=1.000000, …
Symmetric Matrix: size=1000, eigen_gap=5.00
  Power Method: iterations=6, eigenvalue=1.000000
  QR Method: iterations=19, eigenvalues=1.000000, …
Symmetric Matrix: size=1000, eigen_gap=2.00
  Power Method: iterations=10, eigenvalue=1.000000
  QR Method: iterations=21, eigenvalues=1.000000, …
Symmetric Matrix: size=1000, eigen_gap=1.00
  Power Method: iterations=15, eigenvalue=1.000000
  QR Method: iterations=25, eigenvalues=1.000000, …
```

7 Bibliography

1. *https://www.cs.cornell.edu/jeh/book.pdf*
2. *https://math.mit.edu/∼gs/linearalgebra/ila5/linearalgebra5_11-3.pdf*