Eigenvalue Computation Using the QR Algorithm

Tarun Reddy

Introduction

Eigenvalue computation is a fundamental problem in linear algebra and is widely used in various fields such as physics, engineering, and computer science. The eigenvalues of a matrix are scalars that provide useful insight in many areas such as Quantum Physics (Shrodinger's Wave Equation), and much more. In this report, we will focus on the QR algorithm, a well-known numerical method for computing the eigenvalues of a square matrix.

What is an Eigenvalue?

In linear algebra, an eigenvalue λ of a square matrix A is a scalar for which there exists a nonzero vector $\overrightarrow{v}$ (called an eigenvector) such that:

$$A\overrightarrow{v} = \lambda\overrightarrow{v}$$

This equation means that applying the matrix A to the eigenvector $\mathbf{v}$ only scales it by the scalar λ without changing its direction. Eigenvalues have many applications, including solving systems of linear equations, analyzing dynamical systems, and performing dimensionality reduction techniques in machine learning. For a given $n \times n$ square matrix, there exist at-most n eigenvalues for which, each has a unique direction vector known as eigenvector.

Chosen Algorithm: QR Algorithm

The QR algorithm is an iterative method for finding the eigenvalues of a matrix. The idea is to perform a sequence of QR factorizations of the matrix and update it at each iteration. In each step, a matrix A is decomposed into an orthogonal matrix Q and an upper triangular matrix R:

$$A = QR$$

Then, the matrix A is updated as:

$$A_{\text{new}} = RQ$$

This process is repeated iteratively, and as the iterations progress, the matrix converges to a diagonal matrix whose diagonal elements are the eigenvalues of the original matrix. The eigenvalues can then be read off the diagonal of the resulting matrix after convergence.

Example of QR Algorithm Calculation

Consider the matrix:

$$A = \begin{bmatrix} 6 & 2 \\ 2 & 3 \end{bmatrix}.$$

1. Perform QR decomposition:
$$A = QR, \quad Q = \begin{bmatrix} 0.9487 & -0.3162 \\ 0.3162 & 0.9487 \end{bmatrix}, \quad R = \begin{bmatrix} 6.3246 & 3.1623 \\ 0 & 2.5298 \end{bmatrix}.$$
2. Update A:
$$A_{\text{new}} = RQ = \begin{bmatrix} 6.8 & 1.6 \\ 1.6 & 2.2 \end{bmatrix}.$$
3. Repeat the process: After a few iterations, A converges to:
$$\begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix},$$

where the eigenvalues are $\lambda_1 = 7$ and $\lambda_2 = 2$.

Time Complexity Analysis

The time complexity of the QR algorithm depends on the QR decomposition, which takes $O(n^3)$ for a matrix of size $n \times n$. The number of iterations k required for convergence varies depending on the matrix, but for well-conditioned matrices, it is often a constant.

Thus, the total time complexity of the QR algorithm is:
$$O(n^3 \cdot k)$$

For most matrices, k is relatively small, so the method is feasible for matrices up to moderate size. However, for extremely large matrices or those with ill-conditioned eigenvalues, the number of iterations could increase, making the algorithm computationally expensive.

Other Insights

Memory Usage

The QR algorithm requires $O(n^2)$ memory for storing the matrix A, and additional memory for storing the matrices Q and R during the decomposition. For large matrices, the memory requirements could become significant.

Convergence Rate

The QR algorithm is generally fast and converges quickly for well-conditioned matrices. The convergence is usually linear or superlinear, but the rate of convergence can be slower for matrices with eigenvalues that are close to each other or for poorly conditioned matrices.

Suitability for Different Types of Matrices

The QR algorithm is suitable for dense matrices and works well for non-symmetric matrices. However, for symmetric matrices, specialized methods such as the *Divide and*

Conquer algorithm can be more efficient. For sparse matrices, where most of the elements are zero, the QR algorithm is inefficient, and methods like the Arnoldi iteration or Lanczos method are preferred.

Aspect	**Description**
Memory Usage	$O(n^2)$ for storing the matrix A, plus additional storage for Q and R.
Convergence Rate	Generally linear or superlinear; may be slower for matrices with close eigenvalues or poor conditioning.
Suitability for Dense Matrices	Very suitable, works efficiently for general dense matrices.
Suitability for Sparse Matrices	Less efficient, alternative methods like Arnoldi or Lanczos are preferred for sparse matrices.
Suitability for Symmetric Matrices	Can be slower than specialized methods like Divide and Conquer for symmetric matrices.
Suitability for Non-Symmetric Matrices	Suitable for non-symmetric matrices with a broad range of eigenvalue distributions.
Computational Complexity	$O(n^3 \cdot k)$, where k is the number of iterations, typically small for well-conditioned matrices.

Table 1: Summary of Key Aspects of the QR Algorithm

Comparison of Algorithms

Several algorithms exist for eigenvalue computation. While each algorithm has its own strengths and weaknesses, the QR algorithm remains widely used due to its reliability and applicability to general dense matrices. For specialized matrices, like symmetric or sparse matrices, other algorithms may be more efficient.

Aspect	**QR Algorithm**	**Power Method**	**Jacobi Method**
Matrix Type	Suitable for dense matrices, symmetric or non-symmetric	Works best for large, sparse matrices	Limited to symmetric matrices
Convergence Rate	Linear to superlinear	Slow, especially for closely spaced eigenvalues	Quadratic for well-conditioned matrices
Eigenvalues Computed	All eigenvalues	Dominant eigenvalue only	All eigenvalues
Computational Complexity	$O(n^3 \cdot k)$, with moderate k	$O(n^2 \cdot k)$, with large k	$O(n^3)$, efficient for small n
Memory Usage	$O(n^2)$	$O(n)$	$O(n^2)$
Suitability for Sparse Matrices	Inefficient	Highly suitable	Inefficient
Suitability for Large Matrices	Moderate efficiency	Very suitable	Inefficient
Advantages	Handles complex and real eigenvalues effectively	Simple and scalable	High precision for symmetric matrices
Limitations	Computationally expensive for large matrices	Computes only the dominant eigenvalue	Limited to symmetric matrices

Table 2: Comparison of Eigenvalue Computation Algorithms

Conclusion

The QR algorithm is a powerful and versatile method for computing eigenvalues. It is particularly effective for general dense matrices and can handle both real and complex eigenvalues. However, for sparse or symmetric matrices, other algorithms may be more suitable. While the QR algorithm has a higher computational complexity, it remains one of the most widely used methods for eigenvalue computation due to its simplicity, reliability, and robustness across a wide range of matrices.