Eigenvalue Calculation

AI24BTECH11020 - Rishika Kotha

1 Introduction

Eigenvalue computation is a fundamental problem in linear algebra with numerous applications in physics, engineering, and computer science. Given a square matrix A, the goal is to find scalar values λ (eigenvalues) and corresponding vectors $\mathbf{v}$ (eigenvectors) such that:

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

In this report, we implement and analyze the **QR algorithm**, which computes the eigenvalues of a matrix through repeated QR decompositions. The QR algorithm is chosen for its numerical stability and general applicability to various types of matrices.

2 Chosen Algorithm: QR Algorithm

The **QR algorithm** is an iterative method for computing the eigenvalues of a matrix. It involves decomposing the matrix A into an orthogonal matrix Q and an upper triangular matrix R such that:

$$A = QR$$

The process is repeated by reconstructing a new matrix $A' = RQ$ from the product of R and Q. With each iteration, the matrix A' approaches an upper triangular form, and the eigenvalues become the diagonal elements of the matrix after sufficient iterations.

2.1 QR Decomposition: Basic Steps

The QR algorithm follows these basic steps:

1. Start with an initial matrix $A_0 = A$.
2. Perform the QR decomposition of $A_n = Q_n R_n$.
3. Reconstruct the new matrix $A_{n+1} = R_n Q_n$.
4. Repeat the process until convergence (i.e., when the off-diagonal elements of the matrix become negligible).

5. The eigenvalues of the original matrix A are approximated by the diagonal elements of the final matrix A_n.

The QR algorithm works well for general matrices and is especially efficient for all types of matrices.

3 Time Complexity Analysis

The time complexity of the QR algorithm is dominated by the cost of performing the QR decomposition at each iteration. The QR decomposition of an $n \times n$ matrix takes $O(n^3)$ operations. If we denote the number of iterations required for convergence as k, the overall time complexity of the algorithm is:

$$O(k \cdot n^3)$$

In practice, the number of iterations(k) required for convergence depends on the properties of the matrix, such as its condition number and spectral properties. For well-conditioned matrices, the QR algorithm typically converges within $O(n)$ iterations. However, in the worst case, it may take more iterations.

4 Other Insights

4.1 Memory Usage

The QR algorithm requires storing multiple matrices during each iteration. Specifically, for an $n \times n$ matrix, we need to store:
- The original matrix A and the reconstructed matrices A_{n+1}.
- The orthogonal matrix Q and the upper triangular matrix R.

Each of these matrices requires $O(n^2)$ memory. Therefore, the memory usage for the QR algorithm is $O(n^2)$.

4.2 Convergence Rate

The convergence rate of the QR algorithm depends on the matrix's spectral properties. For matrices with distinct eigenvalues, the QR algorithm converges rapidly. However, for matrices with clustered eigenvalues, convergence can be slower. In practice, the algorithm is quite efficient for many real-world matrices, especially when implemented with optimized linear algebra libraries.

4.3 Suitability for Different Types of Matrices

- **Symmetric Matrices**: The QR algorithm is highly effective for symmetric matrices, as it will quickly converge to the eigenvalues. The orthogonal matrix Q in each step retains the symmetry of the matrix.
- **Sparse Matrices**: For sparse matrices, the QR algorithm is not inherently efficient, as it requires dense matrix operations. Specialized methods, such as

the Lanczos algorithm, may be more suitable for sparse matrices.
- **Non-Symmetric Matrices**: The QR algorithm is well-suited for non-symmetric matrices, as it works for both real and complex eigenvalues.

5 Comparison of Algorithms

In addition to the QR algorithm, several other algorithms are commonly used for eigenvalue computation, including:

- **Power Iteration**: A simple and fast method for finding the largest eigenvalue. However, it only converges to the dominant eigenvalue and does not provide all eigenvalues.
- **Jacobi Method**: An iterative method for computing eigenvalues of symmetric matrices. It is slower than the QR algorithm and is typically used for small matrices.
- **Divide and Conquer**: A more advanced algorithm that is efficient for symmetric matrices, particularly when dealing with large matrices.

5.1 Comparison in Terms of Time Complexity

- **QR Algorithm**: $O(k \cdot n^3)$, where k is the number of iterations.
- **Power Iteration**: $O(n^2)$ for each iteration, but only works for the largest eigenvalue.
- **Jacobi Method**: $O(n^3)$, but it is more suitable for symmetric matrices.
- **Divide and Conquer**: $O(n^3)$ for symmetric matrices.

5.2 Comparison in Terms of Accuracy

- The **QR algorithm** is accurate and converges to the exact eigenvalues, provided that the matrix is not ill-conditioned.
- The **Power Iteration method** can fail to converge to all eigenvalues, especially when they are clustered or have similar magnitudes.
- The **Jacobi Method** is accurate for symmetric matrices but can be slow for large matrices.
- The **Divide and Conquer** method is accurate for symmetric matrices and is more efficient than the QR algorithm for very large matrices.

5.3 Suitability for Specific Types of Matrices

- **QR Algorithm**: Works well for general, non-symmetric matrices and is the method of choice for large-scale eigenvalue problems.
- **Power Iteration** : Best suited for finding the dominant eigenvalue of large,

sparse matrices.
-**Jacobi Method** : Suitable for small, symmetric matrices where accuracy is important.
- **Divide and Conquer**: Most effective for symmetric matrices, especially when the matrix is large.

6 Conclusion

The QR algorithm is an effective and general-purpose method for computing the eigenvalues of a matrix. It is well-suited for non-symmetric matrices and large-scale problems. The time complexity of the QR algorithm is $O(k \cdot n^3)$, where k is the number of iterations, and its memory usage is $O(n^2)$. The method is numerically stable and converges efficiently for most matrices. However, for specialized matrices, alternative algorithms such as **Power Iteration** or **Jacobi Method** may be more appropriate depending on the matrix's properties.