Software Assignment: Report

Kondi Akshay Teja
ai24btech11002@iith.ac.in

1 Introduction

In my exploration of eigenvalue computation, I focused on methods that can efficiently compute eigenvalues of matrices, particularly in the context of the QR algorithm, matrix balancing, and Hessenberg transformations. These methods are pivotal in fields like machine learning, physics, and computer science, providing insights into linear transformations. My report highlights these methods, their computational efficiency, and their ability to handle large matrices and both real and complex eigenvalues.

2 Eigenvalues and Eigenvectors

For a square matrix A, an eigenvalue λ and a corresponding eigenvector $\mathbf{v}$ satisfy the equation:

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

where $\mathbf{v} \neq 0$ is a non-zero vector. In my work, I focused on understanding how eigenvalues and eigenvectors affect transformations, with a specific interest in the computational methods for finding them.

3 Methods for Eigenvalue Computation

There are several approaches to computing eigenvalues, and I explored a few of them to understand their benefits and drawbacks:

3.1 Power Method

The Power Method is an iterative technique primarily used for finding the largest eigenvalue. By repeatedly multiplying a vector by the matrix and normalizing it, I could estimate the dominant eigenvalue using the Rayleigh quotient:

$$\lambda = \frac{\mathbf{v}^T A\mathbf{v}}{\mathbf{v}^T\mathbf{v}}$$

This method works well for large matrices but is limited to finding only the largest eigenvalue, which I found to be a drawback in some scenarios.

3.2 Jacobi Method

I also looked at the Jacobi Method, which works well for symmetric matrices by iteratively applying orthogonal transformations to reduce the matrix to diagonal form. Although accurate, this method proved computationally expensive for large matrices, which could be a limitation in practice.

3.3 QR Algorithm

The QR Algorithm is one of the most robust methods for computing all eigenvalues of a matrix. By performing QR decomposition at each step:

$$A = QR$$

and updating the matrix as:

$$A_{k+1} = R_k Q_k$$

I was able to compute all eigenvalues efficiently, including complex eigenvalues, making it ideal for general matrices. This method is central to my approach due to its versatility and computational efficiency.

3.4 Inverse Iteration (Rayleigh Quotient Iteration)

For finding specific eigenvalues, I employed Inverse Iteration, which involves solving the system:

$$(A - \lambda I)\mathbf{v} = \mathbf{w}$$

This method converged quickly to a given eigenvalue, and I found it particularly useful for targeting the smallest or largest eigenvalue.

3.5 Lanczos Algorithm

Lastly, the Lanczos Algorithm, which is effective for symmetric matrices, helped me understand how sparse matrices can be handled efficiently. By reducing the matrix to a tridiagonal form, it simplified computations for large-scale problems where only a few eigenvalues are needed.

3.6 Method Chosen: QR Algorithm with Hessenberg Transformation

For this assignment, I chose the QR algorithm combined with matrix balancing and Hessenberg transformation. This method improved both the computational efficiency and numerical stability, which were crucial for handling larger matrices.

4 QR Algorithm with Hessenberg Transformation

The QR algorithm, which involves decomposing the matrix A into QR at each step and updating the matrix as $A_{k+1} = R_k Q_k$, is an iterative process that converges to a triangular matrix. By applying the Hessenberg transformation beforehand, I was able to simplify the matrix structure and improve computational efficiency.

4.1 Matrix Balancing

To further improve the stability of the algorithm, I applied matrix balancing, which scales the rows and columns of the matrix, making it better conditioned for numerical computations. This step ensured that the QR algorithm could converge more quickly and with fewer numerical issues.

4.2 Transformation to Hessenberg Form

Before applying the QR algorithm, I transformed the matrix into Hessenberg form. This simplified the matrix by zeroing out elements below the first subdiagonal, making the QR decomposition process more efficient. It retained the eigenvalues and sped up convergence.

5 Complex Eigenvalues and QR Decomposition

The QR algorithm can compute both real and complex eigenvalues. When dealing with non-symmetric matrices, the eigenvalues may appear as complex conjugate pairs. The process of QR decomposition for complex eigenvalues follows the same iterative procedure, but when the matrix contains complex values, the QR algorithm works with complex arithmetic.

In each iteration, the matrix is decomposed as:

$$A = QR$$

where Q is an orthogonal (or unitary in the case of complex eigenvalues) matrix, and R is an upper triangular matrix. When dealing with complex eigenvalues, the off-diagonal elements of the matrix may become complex during the iterations. The algorithm converges to a triangular matrix where the complex eigenvalues appear as conjugate pairs on the diagonal.

For a matrix A with complex eigenvalues, the QR decomposition continues iterating until the matrix converges to a form with complex eigenvalues $\lambda = \alpha \pm \beta i$, where α is the real part, and β is the imaginary part of the eigenvalue.

6 Computational Complexity

The computational complexity of the QR algorithm, primarily driven by the QR decomposition, is $O(n^3)$ for an $n \times n$ matrix. While this cubic complexity might seem high, the Hessenberg transformation reduced the workload, making the overall process more efficient.

7 Comparison with Other Methods

I compared the QR algorithm with other common methods based on accuracy, efficiency, and its ability to handle complex eigenvalues:

Method	Key Features
Power Method	Limited to largest eigenvalue, Efficient for large matrices
Jacobi Method	Accurate for symmetric matrices, Computationally expensive
QR Algorithm	Accurate for all eigenvalues, Handles complex eigenvalues naturally
Inverse Iteration	Accurate for specific eigenvalues, Handles with a shift
Lanczos Algorithm	Accurate for symmetric matrices, Efficient for sparse matrices

8 Implementation

I implemented the QR algorithm with matrix balancing and Hessenberg transformation in C. The matrix was first balanced and transformed into Hessenberg form before the QR algorithm was applied. The implementation was tested on matrices of varying sizes, and the computed eigenvalues were compared with those obtained from established numerical libraries to ensure accuracy.

9 Conclusion

The QR algorithm with Hessenberg transformation proved to be an efficient and reliable method for computing eigenvalues of general matrices. It is capable of handling both real and complex eigenvalues, making it a versatile tool for a wide range of applications. Although its cubic time complexity may limit its scalability for very large matrices, the method offers a strong foundation for solving eigenvalue problems. Future work could explore parallelization to enhance performance for large-scale problems.