Eigenvalue Computation in Matrices: Implementation and Analysis

EE1030 - Matrix Theory

November 18, 2024

Vijaya Sreyas Badde

AI24BTECH11003

Contents

1 Objective

To develop a program that computes all distinct eigenvalues of a square matrix $A \in \mathbb{C}^{n \times n}$.

2 Algorithm Selection

While several approaches to this question exist, each has it's trade-offs in terms of speed, accuracy, and generality.

2.1 Direct Methods

Direct methods like Gaussian Elimination and Full Eigendecomposition solve the characteristic equation: $\det(A - \lambda I) = 0$
While providing exact results, these methods have computational complexity $O(n^4)$, rendering them impractical for large matrices.

2.2 The Power Method

The power method approximates the dominant eigenvalue $\lambda_{\max}$ through iterative transformation:

$$v_{k+1} = \frac{Av_k}{|Av_k|}$$

Limitations include:

- Restricted to largest eigenvalue
- Poor performance with complex eigenvalues
- Convergence rate $O(1 - \frac{\lambda_2}{\lambda_1})$

2.3 The Jacobi Method

For symmetric matrices $A^T = A$, the Jacobi method uses orthogonal similarity transformations:

$$A_{k+1} = P_k^T A_k P_k$$

Where P_k represents a sequence of rotation matrices designed to zero off-diagonal elements.

Characteristics:

- Time complexity: $O(n^3)$
- Excellent for symmetric matrices
- Iterative orthogonal transformations

2.4 QR Algorithm

The QR algorithm decomposes matrices through iterative QR decompositions:

$A_0 = A$ and $A_{k+1} = R_k Q_k$

With a shift strategy:

$A_k \leftarrow (A_k - \mu I)$ and $A_k \leftarrow Q_k R_k + \mu I$

Although it becomes prohibitively expensive for larger matrices, it demonstrates robust performance for a wide range of matrix types and provides a comprehensive eigenvalue extraction approach.

3 Current Implementation

Our current implementation takes a hybrid approach, combining sophisticated techniques to address the inherent challenges of eigenvalue computation.

3.1 Computational Strategy

The implementation supports two primary computational strategies:

- A Jacobi rotation method for symmetric matrices
- A complex QR shift algorithm for non-symmetric matrices

3.2 Complex Number Support

A critical change in our approach is comprehensive complex number support. The custom-made library ⟨complex_utils.h⟩ allows

- Explicit control over complex number representation.
- Custom printing functions.
- Specific mathematical operations tailored to our implementation.

These have been placed in a separate library and not in the code itself to make the code easier on the eyes.

3.3 Method Selection Mechanism

The accompanying bash script ⟨eigval.sh⟩ serves as an intelligent router, automatically detecting matrix symmetry, and selecting the most appropriate computational method. This ensures optimal algorithm selection without manual intervention.

Symmetric Matrix Handling

For symmetric matrices, the Jacobi method is employed. This approach leverages the matrix's structural properties to compute eigenvalues through iterative orthogonal transformations.

For the symmetric matrix A, the Jacobi method uses:

$A \to O^T AO$, where O represents orthogonal transformation matrices.

Non-Symmetric Matrix Handling

Non-symmetric matrices are processed using the QR shift algorithm. This method implements complex number operations, shift strategies, and iterative matrix transformations to converge on eigenvalues.

The QR shift algorithm implements:

- Shift Selection: $\mu = \frac{\lambda_{n-1}+\lambda_n}{2}$
- Matrix Transformation: $A_{k+1} = Q_k^{-1}(A_k - \mu I)Q_k + \mu I$

3.4 Computational Characteristics

Some basic and notable characteristics of these codes are:

- Maximum Iteration Limit: 1000
- Convergence Tolerance: 10^{-6}
- Flexible input parsing supporting comma-separated complex entries
- Stopping Criterion: |Off-Diagonal Elements| $< \tau$

4 Performance Analysis

Computational Complexity

The implementation maintains a consistent computational complexity:

- •Time Complexity: $O(n^3)$
- •Space Complexity: $O(n^2)$

Numerical Stability

Error bound estimation:

$|\lambda_{\text{computed}} - \lambda_{\text{exact}}| \leq \epsilon$

Where ϵ represents machine precision.

These complexity metrics reflect the fundamental challenges of matrix eigenvalue computation, balancing computational efficiency with numerical precision.

5 Conclusions

Our eigenvalue computation framework represents a flexible, robust solution capable of handling diverse matrix types. By combining sophisticated numerical techniques with intelligent method selection, we've developed a computational tool that balances efficiency, accuracy, and generality. While there's room for optimization, the current implementation provides a robust solution for practical applications.