Software Assignment Report

Pushkar Gudla-AI24BTECH11012

November 17, 2024

Contents

Chapter 1

Introduction

This report presents a comprehensive discussion of an algorithm implemented in C for efficiently computing the eigenvalues of a square matrix. Eigenvalue computation is a fundamental problem in numerical linear algebra, with applications spanning diverse fields such as physics, engineering, computer science, and data analysis. The proposed algorithm combines two pivotal numerical techniques—Hessenberg decomposition and the QR algorithm with shifts—to achieve a balance between computational efficiency and numerical robustness.

Hessenberg decomposition serves as an essential preprocessing step, reducing the input matrix to a nearly triangular form while preserving its eigenvalues. This reduction simplifies subsequent computations and significantly enhances the overall performance of the algorithm. The QR algorithm with shifts, a widely recognized iterative method, is then employed to extract eigenvalues with improved convergence rates. By incorporating shifts, the algorithm accelerates convergence and mitigates numerical instability, making it well-suited for practical applications.

This document provides a detailed exploration of the algorithm's theoretical foundations, elucidating how each component contributes to the solution of the eigenvalue problem. It also examines the C code implementation, offering insights into the design choices, data structures, and optimization strategies employed to ensure high performance. Furthermore, the report critically evaluates the algorithm's strengths, such as its computational efficiency and adaptability, while addressing potential limitations and areas for improvement.

Chapter 2

Algorithm Overview

2.1 Step 1: Hessenberg Decomposition

Hessenberg decomposition is an essential preprocessing step for eigenvalue computation. It simplifies a general square matrix A into a Hessenberg matrix H, which retains the same eigenvalues but has a specific structure:
- Elements below the first subdiagonal are zero, while the upper part of the matrix is preserved.
- For example, for a 4×4 matrix:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ 0 & h_{32} & h_{33} & h_{34} \\ 0 & 0 & h_{43} & h_{44} \end{bmatrix}$$

The goal of Hessenberg reduction is to transform a square matrix A into a Hessenberg matrix H while retaining the eigenvalues of the original matrix. This is achieved through similarity transformations, a mathematical approach defined as:

$$H = Q^\top A Q$$

where Q is an orthogonal matrix.

Since similarity transformations involve only pre and post multiplication of matrix A by Q and $Q^\top$, the eigenvalues of A remain unchanged. This property ensures that we can work with a simpler matrix H without altering the final result.

Role of Householder Reflections

Householder reflections are a technique used to perform similarity transformations efficiently. A householer reflection creates zeroes in a specified column below the diagnol by applying a transformation of the form:

$$P = I - 2vv^\top$$

where:

- v is a vector chosen such that P eliminates the desired elements
- v is normalized to ensure P is orthogonal

For Hessenberg reduction:

1. For each column j, a Householder vector v is computed to zero out the elements below the first subdiagnol of that column.

2. The matrix A is updates using P_j:

$$A \leftarrow P_j A P_j^\top$$

   This process is repeated iteratively for each column, gradually transforming the entire matric into Hessenberg form.

2.2 Step 2: QR Algorithm

The QR algorithm is an iterative method widely used to compute the eigenvalues of a matrix. It operates as follows:

1. Factorization: At each iteration, the current matrix H is decomposed into two matrices:
   - Q: An orthogonal matrix, whose columns are orthonormal vectors.
   - R: An upper triangular matrix. This factorization satisfies the relationship $H = QR$, leveraging the orthogonality of Q to preserve numerical stability during computations.

2. Matrix Update: The matrix is updated by reversing the roles of Q and R, forming a new matrix H' as $H' = RQ$. This transformation maintains similarity to the original matrix, ensuring the eigenvalues remain unchanged. With each iteration, this process progressively drives H closer to an upper triangular form.

3. Eigenvalue Extraction: Once H converges to an upper triangular matrix (or nearly so), its diagonal entries approximate the eigenvalues of the original matrix. For matrices with real eigenvalues, this triangular form is direct. For complex eigenvalues, patterns such as 2x2 blocks represent conjugate pairs.

This iterative refinement enables the QR algorithm to efficiently compute eigenvalues for a broad range of matrices, including non-symmetric and complex cases, while maintaining numerical robustness.

2.3 Step 3: Accelerated Convergence with Shifts

To accelerate the convergence of the QR algorithm, a strategically chosen shift σ is introduced, transforming the matrix H into $H - \sigma I$, where I is the identity matrix of the same size as H. This shift serves to improve the numerical properties of the matrix during iteration, effectively targeting eigenvalues and enhancing the convergence rate. By focusing the algorithm on specific regions of the spectrum, the shift reduces the number of iterations required to achieve eigenvalue separation, particularly for matrices with closely spaced eigenvalues. After each iteration, the value of σ is dynamically adjusted based on the properties of the current matrix, such as the entries in its subdiagonal or its largest eigenvalue estimates. This adaptability is critical for dealing with matrices where eigenvalues are clustered, as it prevents stagnation and ensures steady progress toward convergence. The use of shifts not only accelerates the process but also contributes to numerical stability, as it minimizes the accumulation of rounding errors that can occur in floating-point computations. This approach is a cornerstone of modern eigenvalue algorithms, enabling them to efficiently handle complex and high-dimensional matrices in practical applications.

2.4 Handling Complex Eigenvalues

When a matrix has complex eigenvalues, the algorithm detects them through specific convergence patterns, such as oscillations in subdiagonal entries or the formation of 2x2 blocks in the Hessenberg form. These blocks are used to compute the eigenvalues directly, often as complex conjugate pairs. To represent these values, the algorithm utilizes a structure for complex numbers, storing their real and imaginary components. This approach ensures accurate computation and broad applicability, particularly for non-symmetric matrices encountered in various scientific and engineering fields.

Chapter 3

Implementation Details

3.1 Matrix Initialization

The code starts by dynamically allocating memory for the input matrix, implemented as a 2D array using a double pointer. The size of the matrix, $n \times n$, is determined by user input, providing flexibility to work with matrices of varying dimensions. Dynamic memory allocation ensures efficient use of resources, as memory is allocated only as needed at runtime, avoiding limitations of fixed-size arrays. This approach not only accommodates matrices of different sizes but also allows the algorithm to handle large matrices efficiently within the constraints of available system memory. By organizing the matrix dynamically, the implementation remains scalable and adaptable for diverse applications.

- **createMatrix**: Allocates memory for the matrix rows and initializes all elements to zero using 'calloc'.
- **printMatrix**: Prints the matrix in a human-readable format, useful for debugging or output purposes.

Memory management is carefully handled to prevent leaks. If allocation fails at any stage, the program terminates gracefully.

3.2 Hessenberg Reduction

The code begins by reducing the input matrix to Hessenberg form through a series of similarity transformations, which iteratively simplify the matrix while preserving its eigenvalues. For each column j, the algorithm computes a Householder vector designed to eliminate all elements below the subdiagonal entry $h_{j+1,j}$. This vector is then used to construct a Householder transformation, which is applied to both the rows and columns of the matrix. These transformations ensure numerical stability and maintain the matrix's similarity property, which is critical for accurate eigenvalue computation. By reducing the matrix to Hessenberg form—where all elements below the first subdiagonal are zero—the code optimizes the subsequent QR iterations, significantly improving their efficiency and convergence properties.

3.3 QR Iteration with Shifts

Once a matrix is transformed into Hessenberg form, the QR algorithm is applied iteratively. This method is used to compute the eigenvalues of a matrix with improved numerical efficiency and stability. Below is a detailed breakdown of the process: 1. QR Factorization

- Objective: Decompose the Hessenberg matrix H into the product of two matrices:

$$H = Q \cdot R$$

  where:

  - Q: An orthogonal matrix ($Q^T Q = I$), which ensures the numerical stability of the process.
  - R: An upper triangular matrix, retaining the structure essential for iteration.

- How It's Done:

  - Algorithms such as Gram-Schmidt orthogonalization, Householder reflections, or Givens rotations are employed to perform the factorization.
  - For Hessenberg matrices, the sparsity of the lower part (mostly zero) allows for computational efficiency, significantly reducing the number of operations required.

- Outcome:

  - The QR factorization provides matrices Q and R, which are then used to update the Hessenberg matrix H in subsequent iterations.

2. Shifted QR Iteration

- Purpose of the Shift:

  - To accelerate convergence, a scalar shift σ is introduced before the QR factorization. The shift improves numerical efficiency by targeting eigenvalues and reducing the number of iterations needed.

- Shift Selection:

  - A common choice for σ is based on the Rayleigh quotient or Wilkinson's shift, which approximate an eigenvalue near the bottom-right corner of the matrix. These strategies ensure faster localization of eigenvalues.

- Modified QR Decomposition:

  - The shift modifies the matrix as follows:

$$\tilde{H} = H - \sigma I$$

    where I is the identity matrix.

- Reversing the Shift:

  - After performing QR factorization on $\tilde{H}$, the shift is removed by reversing the transformation:

$$H' = R \cdot Q + \sigma I$$

    Here, Q and R are the orthogonal and triangular matrices obtained from the factorization of $\tilde{H}$.

- Preservation of Structure:

- The matrix H' remains in Hessenberg form after each iteration, which is a crucial property of the QR algorithm. This ensures that subsequent steps are computationally efficient.

3. Convergence Check

- Objective: Determine whether the algorithm has converged to the desired result, i.e., whether the matrix is close enough to an upper triangular form.
- Criteria:
  - Off-Diagonal Tolerance:
    - The algorithm monitors the size of the off-diagonal elements (particularly those below the main diagonal).
    - If the absolute value of these elements is smaller than a pre-specified tolerance EPSILON (e.g., 10^{-8}), they are considered negligible, indicating convergence.
  - Maximum Iterations:
    - A maximum iteration count MAX_ITER is set to prevent infinite loops in cases where convergence is slow or does not occur.
- Result:
  - If convergence criteria are met, the algorithm terminates. At this point, the diagonal elements of H approximate the eigenvalues of the original matrix.
  - If not, the process is repeated with a new shift σ and the updated matrix H'.

3.4 Complex Eigenvalue Handling

The QR algorithm incorporates a custom data structure, 'ComplexNum', to efficiently handle complex eigenvalues that arise during iterations. This structure separates the real and imaginary components, providing a clear and organized representation. Complex eigenvalues are identified through convergence patterns observed in the subdiagonal elements of the Hessenberg matrix during QR iterations. Specifically, when these elements stabilize at nonzero values, they indicate the presence of a complex conjugate pair. The real part of the eigenvalue is computed as the average of the diagonal elements of a 2×2 block, while the imaginary part is derived from the product of the off-diagonal elements. The 'ComplexNum' structure simplifies storing and processing such eigenvalues, allowing for precise numerical operations and easy integration into the algorithm, enhancing its robustness in handling general matrices.

3.5 Output

The eigenvalues are printed in either real or complex form. Real eigenvalues are directly extracted from the diagonal of the converged upper triangular matrix. Complex eigenvalues are output using the 'ComplexNum' structure.

Chapter 4

Strengths and Weaknesses

4.1 Strengths

1. **Computational Efficiency:**
   - Matrix factorization, such as the QR decomposition, is computationally intensive for dense matrices. For a general $n \times n$ matrix, QR decomposition requires $O(n^3)$ operations.
   - When the matrix is in Hessenberg form, the QR decomposition is faster because the sparsity of H reduces the number of computations, for a hessenberg matrix, the complexity of QR decomposition reduces to approximately $O(n^2)$.
2. **Stability:**
   - Working with Hessenberg matrices minimizes round off errors in nuerical computations compared to directly applying QR iterations to original matrix
3. **General Applicability**
   - The algorithm works on any square matrix, whether it contains real or complex eigenvalues.
   - Hessenberg reduction preserves eigenvalues, ensuring the original problem is not altered. This property makes the algorithm broadly applicable across a wide range os problems in numerical linear algebra.
4. **Accuracy and Precision**
   - The iterative QR decomposition with shifts converges to eigenvalues with high precision. The stopping criteria based on tolerance parameter (EPSILON) allow fine control over level of accuracy.
   - Using orthogonal transformations helps maintain numerical stability during Hessenberg reduction and QR iterations, ensuring accurate results even for moderately large matrices.

4.2 Weaknesses

1. **Numerical Stability**: While the algorithm is numerically stable for many matrices, it can encounter issues with poorly conditioned or ill-posed problems:

- For many matrices with very small or very large eigenvalues, numerical errors may accumulate, leading to inaccuracies in convergence.
- Round-off errors can propagate during the QR iterations, especially if the matrix contains close or repeated eigenvalues.
- Small perturbations in the input matrix can sometimes lead to significant changes in the results due to inherent sensitivity of eigenvalue problems.

2. **Memory Usage**: The algorithm requires additional memory to store intermediate matrices and vectors during Hessenberg reduction and QR iterations:
   - Dynamic memory allocation, while flexible, may lead to fragmentation or memory leaks if not managed carefully.
   - For very large matrices, memory requirements can become a bottleneck, limiting the scalability of the algorithm on systems with constrained resources.
3. **Convergence Limitations**:

   Dependence on Shift strategy:
   - While shifts generally improve convergence, their effectiveness depends on the specific choice of shift values. Poorly chosen shifts can slow down convergence or cause the algorithm to stall.
   - The "simple shift" strategy (using the eigenvalue estimate from the bottom-right element of the matrix) is effective in many cases but may not work optimally for all matrices.

   Maximum Iteration Limit:
   - The algorithm enforces a cap on the number of iterations (MAX_ITER) to prevent infinite loops. However, for matrices that converge slowly, this limit may be insufficient, leading to incomplete or incorrect results.
4. **Sensitivity to Matrix Properties**: The QR algorithm is highly sensitive to the initial matrix properties:
   - For symmetric or well-conditioned matrices, convergence is rapid, and the results are accurate.
   - For non-symmetric or nearly defective matrices (matrices with repeated eigenvalues or very close eigenvalues), the algorithm may struggle to converge or produce results with reduced accuracy.

Chapter 5

Conclusion

The algorithm implemented in this code integrates Hessenberg decomposition with QR shifts to deliver an efficient and powerful method for computing eigenvalues of general matrices. By first reducing the input matrix to Hessenberg form, it simplifies the structure while preserving eigenvalues, significantly enhancing computational efficiency. The subsequent use of QR iterations, augmented by carefully chosen shifts, ensures rapid convergence, even for challenging matrices. This combination of techniques makes the implementation both versatile and effective, addressing a wide range of eigenvalue problems encountered in numerical linear algebra. The algorithm's structure inherently exploits the sparsity and symmetry of the Hessenberg form, minimizing unnecessary computations and reducing runtime.

While the implementation is robust for most practical applications, there is room for further refinement to improve performance and reliability. Enhancing numerical stability, especially for ill-conditioned matrices, could involve incorporating more advanced shift strategies or using higher-precision arithmetic where needed. Additionally, optimizing memory usage by leveraging in-place operations and reducing redundant data storage would make the algorithm more suitable for large-scale problems. Despite these potential improvements, the current implementation provides a solid and practical foundation for exploring eigenvalue computations, serving as a valuable tool in both academic and applied contexts of numerical analysis.