Software Assignment Report

Pushkar Gudla-AI24BTECH11012
Department of AI
ai24btech11012@iith.ac.in

1 Introduction

This report presents an algorithm implemented in C for computing the eigenvalues of a square matrix. The algorithm leverages two key numerical techniques: Hessenberg decomposition and the QR algorithm with shifts. By combining these methods, it achieves computational efficiency and robustness in eigenvalue computation. This document provides a detailed explanation of the underlying algorithm, evaluates its strengths and weaknesses, and offers a clear understanding of the C code implementation.

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

Preprint. Under review.

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 process to find the eigenvalues of a matrix:

1. Factorization: The current matrix H is decomposed into two matrices:
   - Q: An orthogonal matrix.
   - R: An upper triangular matrix.

   This decomposition satisfies $H = QR$.
2. Matrix Update: The next matrix in the sequence is computed as $H' = RQ$. Iteratively applying this transformation drives H toward an upper triangular form.
3. Eigenvalue Extraction: When H converges to an upper triangular matrix, the diagonal entries of H represent the eigenvalues of the original matrix.

2.3 Step 3: Accelerated Convergence with Shifts

A shift σ is introduced to accelerate the algorithm. Instead of directly working with H, the algorithm processes $H - \sigma I$, where I is the identity matrix. After each iteration, the shift is adjusted based on the current matrix properties, significantly improving convergence speed for matrices with close eigenvalues.

2.4 Handling Complex Eigenvalues

If the matrix has complex eigenvalues, the algorithm identifies them through convergence patterns. These values are extracted and represented using a structure for complex numbers.

3 Implementation Details

3.1 Matrix Initialization

The code begins by dynamically allocating memory for the input matrix using a double pointer (2D array). The matrix size $n \times n$ is specified by the user. The dynamic memory allocation allows flexibility in handling matrices of different sizes:

- **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 reduces the input matrix to Hessenberg form by iteratively applying similarity transformations. Each transformation eliminates elements below the first subdiagonal in a column: 1. For column j, a Householder vector is computed to zero out elements below $h_{j+1,j}$. 2. The matrix is updated using this vector, ensuring numerical stability while preserving eigenvalues.

This reduction significantly simplifies the subsequent QR iterations.

3.3 QR Iteration with Shifts

Once the matrix is in Hessenberg form, the QR algorithm is applied. The iterative process involves:

1. QR Factorization:
   - The Hessenberg matrix H is decomposed into Q and R using Gram-Schmidt orthogonalization or a similar method.
2. Shifted Update:
   - A shift σ is applied to modify the matrix as $H - \sigma I$ before factorization. After the QR iteration, the shift is removed by reversing the operation.
3. Convergence Check:
   - The iteration continues until the off-diagonal elements are sufficiently small (determined by a tolerance parameter 'EPSILON') or until the maximum iteration limit ('MAX_ITER') is reached.

3.4 Complex Eigenvalue Handling

The code includes a custom data structure ('ComplexNum') to handle complex eigenvalues. This structure stores the real and imaginary parts separately. The algorithm identifies complex eigenvalues by analyzing convergence patterns in the QR iterations.

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.

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.

5 Conclusion

The algorithm implemented in this code combines Hessenberg decomposition and QR shifts to provide an efficient and versatile method for eigenvalue computation. While the implementation is robust for most practical applications, addressing numerical stability and optimizing memory usage could further enhance its utility. This code offers a strong foundation for exploring eigenvalue problems in numerical linear algebra.