Eigenvalue Computation Using the QR Algorithm and Alternative Methods

Introduction

Eigenvalues are fundamental concepts in linear algebra and have applications in a wide range of fields such as physics, engineering, computer science, and data analysis. Given a square matrix A, an eigenvalue λ is defined as a scalar such that there exists a non-zero vector v (called the eigenvector) satisfying:

$$Av = \lambda v. \tag{1}$$

This equation implies that multiplying the matrix A by the eigenvector v scales v by the factor λ without changing its direction.

Computing eigenvalues is often a challenging numerical task, especially for large matrices, and is a core topic in computational linear algebra. This report focuses on computing eigenvalues using the **QR Algorithm**, a widely used iterative method, and briefly discusses other algorithms and their comparative advantages.

Chosen Algorithm

The algorithm implemented for eigenvalue computation is the **QR Algorithm**. It works iteratively by factorizing a matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R, then updating A as RQ. Over successive iterations, the off-diagonal elements of A converge to zero, and the diagonal elements approximate the eigenvalues of the original matrix. The key steps of the algorithm can be summarized as:

$$A = QR, \quad \text{(QR decomposition of } A\text{)} \tag{2}$$

$$A' = RQ, \quad \text{(Recompute } A \text{ for the next iteration)} \tag{3}$$

After k iterations, the matrix $A^{(k)}$ converges to a quasi-diagonal form:

$$A^{(k)} \approx \begin{bmatrix} \lambda_1 & \cdots & \cdots & \vdots \\ 0 & \lambda_2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}, \tag{4}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of the original matrix A.

QR Decomposition Using the Householder Algorithm

For the QR decomposition, the **Householder Algorithm** is used. The Householder method is a numerically stable and efficient approach to compute Q and R. It achieves the decomposition by iteratively applying orthogonal transformations (Householder reflections) to eliminate elements below the diagonal.

The transformation is defined as:

$$H = I - 2\frac{vv^T}{v^Tv}, \tag{5}$$

where v is a carefully chosen vector that zeroes out all elements below the pivot in a column of A.

Benefits of the Householder Algorithm

1. Numerical Stability: Householder reflections maintain orthogonality of Q and ensure numerical stability, unlike the classical Gram-Schmidt process, which can accumulate rounding errors.
2. Efficiency: The algorithm is computationally efficient with a complexity of $O(n^3)$, making it suitable for dense matrices.
3. Simplicity for Parallelization: The structure of Householder transformations is conducive to parallel computation, especially for large matrices.
4. Robustness for Complex Matrices: The method generalizes easily to complex matrices, as required in the current implementation.

Workflow of the Householder Algorithm

1. Compute the Householder matrix H to zero out sub-diagonal entries in the first column.
2. Apply H to the matrix A to compute R.
3. Repeat the process on the reduced submatrix for subsequent columns.

This choice of QR decomposition ensures that the algorithm performs reliably and accurately for eigenvalue computation.

Key Steps in the QR Algorithm

1. Perform QR decomposition of $A^{(k)}$ into $Q^{(k)}$ and $R^{(k)}$. 2. Update $A^{(k+1)} = R^{(k)}Q^{(k)}$.
3. Repeat until the off-diagonal elements of $A^{(k)}$ converge to zero.

Performance and Applications

The QR Algorithm works well for dense matrices and converges faster for symmetric or Hermitian matrices. Its computational cost per iteration is $O(n^3)$, with overall complexity dependent on the number of iterations k.

Other Eigenvalue Computation Algorithms

In addition to the QR Algorithm, other methods are commonly used, depending on the matrix properties and computational requirements:

1. Power Iteration

This is an iterative method that computes the dominant eigenvalue and its corresponding eigenvector. It works by repeatedly applying the matrix to a vector until convergence.

Features: - Computationally inexpensive ($O(kn^2)$). - Suitable for large sparse matrices. - Limited to finding the largest eigenvalue.

2. Jacobi Method

This method computes all eigenvalues for symmetric matrices by iteratively zeroing out off-diagonal elements using Givens rotations.

Features: - Accurate for symmetric matrices. - Computational cost: $O(n^3)$. - Convergence may be slow for large matrices.

3. Divide-and-Conquer Method

This is a specialized algorithm for symmetric matrices that divides the problem into smaller subproblems and combines their solutions.

Features: - Efficient for large dense symmetric matrices. - Complexity: $O(n^3)$. - Suitable for parallelization.

4. Arnoldi and Lanczos Methods

These are iterative methods optimized for large sparse matrices. They compute a subset of eigenvalues (usually the largest or smallest).

Features: - Computationally efficient ($O(kn^2)$). - Well-suited for large sparse matrices. - Particularly useful for matrices with a few dominant eigenvalues.

Comparison of Algorithms

The following table summarizes the characteristics of these methods:

Algorithm	**Time**	**Accuracy**	**Suitability**
QR Algorithm	$O(kn^3)$	High	General; complex/real matrices
Power Iteration	$O(kn^2)$	Low (dominant)	Simple; large sparse matrices
Jacobi Method	$O(n^3)$	High	Symmetric matrices
Divide-and-Conquer	$O(n^3)$	High	Large dense symmetric matrices
Arnoldi/Lanczos	$O(kn^2)$	High	Sparse matrices; dominant eigenvalues

Table 1: Comparison of Eigenvalue Computation Algorithms

Conclusion

The QR Algorithm is a versatile and accurate method for eigenvalue computation, particularly for dense and small-to-medium-sized matrices. However, for very large or sparse matrices, iterative methods such as Arnoldi or Lanczos are preferable due to their efficiency and scalability. Symmetric matrices benefit from the Jacobi or Divide-and-Conquer methods, which leverage their structure for faster convergence.

In practice, the choice of algorithm depends on the matrix properties, desired accuracy, and computational resources available.