SOFTWARE ASSIGNMENT

EE24BTECH11010 - BALAJI B

November 18, 2024

1 Introduction

Eigenvalues are fundamental concept in linear algebra, appearing in various applications in physics, engineering and computer science. My work implements and analyse the QR algorithm enhanced with Wilkinson's shift, aimed at finding the eigenvalues of real and complex matrices with improved efficiency and accuracy.

2 Definition of Eigenvalues

Given a square matrix A of order n, a scalar λ is called an eigenvalue of A if there exists a non-zero vector $\mathbf{v}$ (called an eigenvector) such that:

$$A\mathbf{v} = \lambda\mathbf{v} \tag{0.1}$$

3 Overview

Eigenvalues and eigenvectors feature prominently in analysis of linear transformation. Originally used to study principle axes of rotational bodies, eigenvalue and eigenvectors find wide range of application, for example atomic orbitals, face recognition and matrix diagonalization.

4 History

Eigenvalues are often introduced in the contex of linear algebra or matrix theory.

In the 18th century, Leonhard Euler studied the rotational motion of a rigid body, and discovered the importance of the principal axes. Joseph-Louis Lagrange realized that the principal axes are the eigenvectors of the inertia matrix.

In the early 19th century, Augustin-Louis Cauchy saw how their work could be used to classify the quadric surfaces, and generalized it to arbitrary dimensions. Cauchy also coined the term *racine caracteriqtique* (characteristic root), for what is now called eigenvalue; his term survives in characteristic equation.

Cauchy arrived at the fact that the real symmetric matrices have real eigenvalues. This was later extended by Charles Hermite in 1855 to what now called Hermitian matrices

The first numerical algorithm for computing eigenvalues and eigenvectors appeared in 1929, when Richard von Mises published the power-Iteration method. One of the most popular methods today, the QR algorithm, was proposed independently by John G. F. Francis and Vera Kublanovskaya in 1961

5 QR with Wilkinson's Shift

The QR algorithm with Wilkinson's shift was developed by James H. Wilkinson. He introduced the concept of the shift to accelerate the convergence of the QR algorithm, which was originally developed by John G. F. Francis and Vera N. Kublanovskaya in the late 1950s.

The QR algorithm with Wilkinson's shift works on the foundation of the QR decomposition algorithm, combined with a strategic enhancement through Wilkinson's Shift.

QR decomposition method:
This method starts with a matrix A, we decompose it into a product of an orthogonal matrix Q and an upper triangular matrix R.

Steps to find the eigenvalues:

1) Initial decomposition:
   - Start with the matrix A
   - Decompose A into Q and R such that $A = QR$, where Q is an orthogonal matrix and R is upper triangle matrix.
2) Iteration:
   - Form the new matrix $A_1 = RQ$
   - Repeat the QR decomposition on A_1 to get Q_1 and R_1, then form $A_2 = R_1 Q_1$
   - Continue this iterative process: $A_{i+1} = R_i Q_i$
3) Convergence :
   - The process converges when A_n becomes an upper triangular matrix T.
   - The diagonal elements of T are the eigenvalues of the original matrix A .

QR combined with Wilkinson's Shift:

- Before each QR decomposition, apply a shift μ to improve convergence.
- Calculate the shift μ based on the eigenvalues of a smaller submatrix. For a matrix A of size N, consider the lower 2×2 submatrix.
$$T = \begin{pmatrix} a_{N-1,N-1} & a_{N-1,N} \\ a_{N,N-1} & a_{N,N} \end{pmatrix}$$
- Calculate the Discriminant δ
$$\delta = \left(\frac{a_{N-1,N-1} - a_{N,N}}{2}\right)^2 + a_{N,N-1} \cdot a_{N-1,N}$$
- So the shift μ is
$$\mu = a_{N,N} - \operatorname{sign}\left(\frac{a_{N-1,N-1} - a_{N,N}}{2}\right) \cdot \sqrt{\delta}$$

- Subtract the shift μ from the diagonal elements:
$$A - \mu I$$
- Perform the QR decomposition on the shifted matrix:
$$A - \mu I = QR$$
- Form the new matrix:
$$A_{i+1} = RQ + \mu I$$
- Repeat the process until the matrix A_i converges to an upper triangular matrix, where the eigenvalues are found on the diagonal

6 C code for QR algorithm with Shift

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

void print_mat(int size, complex double mat[size][size]) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("(%4.6f + %4.6fI) ", creal(mat[i][j]), cimag(mat[i][j]));
        }
        printf("\n");
    }
}

void matmul(int size, complex double mat1[size][size], complex double mat2[size][size
    ], complex double result[size][size]) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            result[i][j] = 0 + 0*I;
            for (int k = 0; k < size; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

complex double norm_complex(int size, complex double vec_1[size][1]) {
    complex double norm = 0 + 0*I;
    for (int i = 0; i < size; i++) {
        norm += vec_1[i][0] * conj(vec_1[i][0]);
    }
    norm = csqrt(norm);
    return norm;
}

void normalized_mat(int size, complex double mat_A[size][1], complex double mat_Q[size
    ][1], complex double norm) {
    for (int i = 0; i < size; i++) {
        mat_Q[i][0] = mat_A[i][0] / norm;
    }
}

void initialize_QR(int size, complex double Q[size][size], complex double R[size][size
    ]) {
```

```
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            Q[i][j] = R[i][j] = 0 + 0*I;
        }
    }
}

void QRdecomposition(int size, complex double A[size][size], complex double Q[size][
     size], complex double R[size][size]) {
    initialize_QR(size, Q, R);
    for (int k = 0; k < size; k++) {
        complex double norm = 0 + 0*I;
        for (int i = 0; i < size; i++) {
            norm += A[i][k] * conj(A[i][k]);
        }
        norm = csqrt(norm);
        for (int i = 0; i < size; i++) {
            Q[i][k] = A[i][k] / norm;
        }
        for (int j = k; j < size; j++) {
            complex double dot = 0 + 0*I;
            for (int i = 0; i < size; i++) {
                dot += conj(Q[i][k]) * A[i][j];
            }
            R[k][j] = dot;
            for (int i = 0; i < size; i++) {
                A[i][j] -= Q[i][k] * R[k][j];
            }
        }
    }
}

void QRwithShift(int size, complex double matrix[size][size], complex double
     eigenvalues[size]) {
    complex double Q[size][size], R[size][size];
    complex double shift;
    int converged;
    for (int iter = 0; iter < 1000; iter++) {
        if (size > 1) {
            complex double d = (matrix[size-2][size-2] - matrix[size-1][size-1]) / 2.0;
            complex double sign = (creal(d) >= 0) ? 1.0 + 0*I : -1.0 + 0*I;
            shift = matrix[size-1][size-1] - sign * csqrt(d * d + matrix[size-1][size-2]
                 * matrix[size-2][size-1]);
        } else {
            shift = matrix[size-1][size-1];
        }
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                matrix[i][j] -= shift * (i == j);
            }
        }
        QRdecomposition(size, matrix, Q, R);
        matmul(size, R, Q, matrix);
        for (int i = 0; i < size; i++) {
            matrix[i][i] += shift;
        }
        converged = 1;
        for (int i = 0; i < size - 1; i++) {
            if (cabs(matrix[i + 1][i]) > 1e-9) {
```

```
                converged = 0;
                break;
            }
        }
        if (converged) {
            break;
        }
    }
    for (int i = 0; i < size; i++) {
        eigenvalues[i] = matrix[i][i];
    }
}

int main(void) {
    int size;
    printf("Enter the size of the matrix: ");
    scanf("%d", &size);

    complex double matrix[size][size];
    complex double eigenvalues[size];
    double real, imaginary;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("Enter the real part and imaginary part of the matrix[%d][%d]: ", i +
                  1, j + 1);
            scanf("%lf %lf", &real, &imaginary);
            matrix[i][j] = real + imaginary * I;
        }
    }

    printf("\nOriginal Matrix:\n");
    print_mat(size, matrix);

    QRwithShift(size, matrix, eigenvalues);

    printf("\nMatrix after QR Algorithm with Shift:\n");
    print_mat(size, matrix);

    printf("\nEigenvalues:\n");
    for (int i = 0; i < size; i++) {
        printf("(%4.6f + %4.6fI)\n", creal(eigenvalues[i]), cimag(eigenvalues[i]));
    }

    return 0;
}
```

7 Time Complexity of the QR Algorithm with Wilkinson's Shift

The QR algorithm with Wilkinson's shift has a time complexity that can be described as follows:

QR Decomposition

Each QR decomposition of an $N \times N$ matrix typically has a time complexity of:

$$O(N^3)$$

Iterations

The number of iterations needed for convergence can vary, but in practice, it often converges in about:

$$O(N)$$

Overall Time Complexity

Given that each iteration involves a QR decomposition, the overall time complexity for the QR algorithm with shifts is approximately:

$$O(N \times N^3) = O(N^4)$$

8 Insights into QR Algorithm with Wilkinson's Shift

The QR algorithm with Wilkinson's shift is a robust and efficient method for eigenvalue computation. Below are some key insights into its advantages and applications:

Advantages

- **Faster Convergence**:
  $$\mu = a_{N,N} - \text{sign}\left(\frac{a_{N-1,N-1} - a_{N,N}}{2}\right) \cdot \sqrt{\delta}$$
  Wilkinson's shift accelerates the convergence by rapidly reducing off-diagonal elements.
- **Effective for Complex Matrices**: It efficiently handles complex arithmetic, ensuring precise calculations for complex matrices.
- **General Applicability**: Applicable to both real and complex matrices, making it versatile for various scientific and engineering problems.
- **Avoids Deflation Issues**: Helps in avoiding deflation issues that can occur with simpler methods, ensuring accurate computation of all

Applications

- **Large Matrices**: Efficient for large matrices due to faster convergence.
- **Non-Symmetric Matrices**: Handles non-symmetric matrices effectively.
- **Matrices with Close Eigenvalues**: Quickly isolates eigenvalues that are close to each other.
- **Dense Matrices**: Performs well with dense matrices, maintaining stability and accuracy.

9 Comparison of Eigenvalue Algorithms

Algorithm	Strengths	Weaknesses
QR Algorithm with Wilkinson's Shift	• Faster convergence due to shifts • Numerically stable • Effective for complex and non-symmetric matrices • Handles closely spaced eigenvalues well	• Higher complexity ($O(N^4)$ in worst case) compared to some simpler methods
Power Method	• Simple to implement • Effective for finding the largest eigenvalue and its corresponding eigenvector	• Slow convergence for eigenvalues of similar magnitude • Only finds one eigenvalue (the largest)
Inverse Iteration	• Effective for finding the smallest eigenvalue • Faster convergence for smallest eigenvalue compared to power method	• Requires matrix inversion • Not suitable for large matrices
Jacobi Method	• Highly accurate for symmetric matrices • Numerically stable	• Not efficient for non-symmetric or large matrices • Convergence can be slow for certain matrices
LU Decomposition	• Useful for solving linear systems • Efficient for certain matrix operations	• Not typically used for eigenvalue problems • Requires different methods for finding eigenvalues
Lanczos Algorithm	• Efficient for large sparse matrices • Finds a few eigenvalues quickly	• Not as effective for dense matrices • Requires orthogonalization to maintain numerical stability

10 Conclusion

QR algorithm with Wilkinson's shift is a robust and efficient method for computing eigenvalues, particularly for complex, non-symmetric, and dense matrices. Its ability to handle closely spaced eigenvalues and maintain numerical stability makes it superior to many other traditional methods such as the Power Method and Jacobi Method. While the Power Method and Inverse Iteration are simpler and can be effective for specific eigenvalue problems, they lack the general applicability and efficiency of the QR algorithm with shifts. The Jacobi Method is highly accurate for symmetric matrices but less efficient for large or non-symmetric matrices. Overall, for applications requiring high precision and efficiency, especially in complex and large matrices, the QR algorithm with Wilkinson's shift is the preferred choice. It strikes a balance between computational complexity and practical performance, making it indispensable in various scientific and engineering fields.