Analysis of Software Assignment

EE24BTECH11043

November 18, 2024

Introduction

The provided C code implements the QR algorithm to compute the eigenvalues of a square matrix. The algorithm utilizes QR decomposition, which factors a matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R. The eigenvalues are obtained iteratively from the diagonal elements of the matrix.

Code Description

The key components of the code are as follows:

Matrix Operations

The code includes utility functions for essential matrix operations:

- **Matrix Multiplication:** Multiplies two matrices A and B to produce the result matrix.
- **Matrix Transposition:** Transposes the input matrix.
- **Matrix Subtraction:** Subtracts one matrix from another.
- **Matrix Norm:** Computes the Frobenius norm of a matrix, defined as:

$$\|A\|_F = \sqrt{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}^2}$$

Code Analysis

The key functions in the code are as follows:

Matrix Multiplication

The `multiply_matrices` function multiplies two square matrices A and B of size $N \times N$ and stores the result in the `result` matrix.

```
void multiply_matrices(double A[N][N], double B[N][N], double result[N][N],
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = 0;
            for (int k = 0; k < n; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

This function iterates through each row of A and each column of B, performing the dot product to compute the result.

Matrix Transposition

The `transpose` function computes the transpose of a matrix.

```
void transpose(double matrix[N][N], double result[N][N], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = matrix[j][i];
        }
    }
}
```

The function swaps the rows and columns of the input matrix.

Matrix Subtraction

The `subtract_matrices` function performs element-wise subtraction of two matrices.

```
void subtract_matrices(double A[N][N], double B[N][N], double result[N][N],
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }
}
```

This function computes result$[i][j] = A[i][j] - B[i][j]$ for each element.

Matrix Norm

The `norm` function computes the Frobenius norm of a square matrix `matrix` of size $N \times N$. The Frobenius norm is defined as:

$$\|A\|_F = \sqrt{\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij}^2}$$

```c
double norm(double matrix[N][N], int n) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += matrix[i][j] * matrix[i][j];
        }
    }
    return sqrt(sum);
}
```

The function iteratively sums the squares of all matrix elements and takes the square root of the total to compute the Frobenius norm.

QR Decomposition

The QR decomposition is performed using the Gram-Schmidt method:

1. For each column of the matrix A, the algorithm computes:

$$R[j][j] = \|A_{\text{col } j}\|$$

2. The column is normalized to produce the corresponding column in Q.

3. Remaining columns are orthogonalized by subtracting projections.

QR Decomposition

The `qr_decompose` function performs the QR decomposition of a square matrix A of size $N \times N$ into an orthogonal matrix Q and an upper triangular matrix R, such that $A = QR$. The function uses the Gram-Schmidt process to compute the decomposition.

```c
void qr_decompose(double A[N][N], double Q[N][N], double R[N][N], int n) {
    double temp[N];

    // Initialize Q and R
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            Q[i][j] = 0;
            R[i][j] = 0;
        }
    }

    for (int j = 0; j < n; j++) {
        // R[j][j] = norm of column j
        double norm_col = 0;
        for (int i = 0; i < n; i++) {
            norm_col += A[i][j] * A[i][j];
        }
        R[j][j] = sqrt(norm_col);

        // Q column vector is A column vector normalized
```

```
        for (int i = 0; i < n; i++) {
            Q[i][j] = A[i][j] / R[j][j];
        }

        // Orthogonalize remaining columns
        for (int k = j + 1; k < n; k++) {
            double dot_product = 0;
            for (int i = 0; i < n; i++) {
                dot_product += A[i][k] * Q[i][j];
            }
            R[j][k] = dot_product;

            // Subtract the projection of A's k-th column onto Q's j-th colu
            for (int i = 0; i < n; i++) {
                A[i][k] -= Q[i][j] * R[j][k];
            }
        }
    }
}
```

Function Description

The `qr_decompose` function computes the QR decomposition of matrix A and returns two matrices: Q, which is orthogonal, and R, which is upper triangular. The decomposition is performed using the Gram-Schmidt process, which involves orthogonalizing the columns of A to generate Q and then computing the corresponding entries of R.

Steps in QR Decomposition

- The function first initializes the matrices Q and R to zero.
- For each column j of A, it computes the norm of the column, stores it in $R[j][j]$, and normalizes the column to form the corresponding column in Q.
- The remaining columns of A are orthogonalized with respect to the columns of Q by subtracting the projection of each column of A onto the columns of Q.
- The upper triangular matrix R is updated as the dot product of the corresponding column of A with the columns of Q.

QR Algorithm

The QR algorithm iteratively applies the following steps:

1. Decompose the matrix A into Q and R such that $A = QR$.
2. Compute a new matrix $A_{\text{new}} = RQ$.
3. Check for convergence by evaluating the norm of A_{new}.
4. Repeat until convergence or the maximum number of iterations is reached.

The eigenvalues are extracted from the diagonal elements of A_{new} after convergence.

QR Algorithm for Eigenvalues

The `qr_algorithm` function implements the QR algorithm to compute the eigenvalues of a given square matrix. The algorithm repeatedly performs QR decompositions and uses the resulting matrices to approximate the eigenvalues of the input matrix.

```c
void qr_algorithm(double matrix[N][N], int n, int max_iterations) {
    double A[N][N], Q[N][N], R[N][N], A_new[N][N];
    double eigenvalues[N];
    int iteration = 0;

    // Copy the matrix to A
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = matrix[i][j];
        }
    }

    while (iteration < max_iterations) {
        // Step 1: QR decomposition of A = QR
        qr_decompose(A, Q, R, n);

        // Step 2: Multiply R and Q to get A_new
        multiply_matrices(R, Q, A_new, n);

        // Step 3: Check for convergence (norm of A_new is small enough)
        if (norm(A_new, n) < 1e-6) {
            break;
        }

        // Update A for the next iteration
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                A[i][j] = A_new[i][j];
            }
        }

        iteration++;
    }

    // Extract eigenvalues from the diagonal of the matrix A
    for (int i = 0; i < n; i++) {
        eigenvalues[i] = A[i][i];
    }

    // Print eigenvalues
    printf("Eigenvalues:\n");
    for (int i = 0; i < n; i++) {
        printf("%lf ", eigenvalues[i]);
    }
    printf("\n");
}
```

Function Description

The `qr_algorithm` function computes the eigenvalues of a square matrix using the QR decomposition and iteration process. The basic idea is that after several iterations of QR decompositions, the matrix A becomes nearly diagonal, and the diagonal elements of A converge to the eigenvalues of the matrix.

Algorithm Steps

- **Step 1: QR Decomposition of A**
  The matrix A is decomposed into the product of an orthogonal matrix Q and an upper triangular matrix R, i.e., $A = QR$.
- **Step 2: Multiply R and Q to obtain A_{new}**
  The matrices R and Q are multiplied to form the updated matrix $A_{\text{new}} = RQ$. This step is crucial for the convergence of the algorithm.
- **Step 3: Check for Convergence**
  The norm of A_{new} is calculated. If the norm is sufficiently small (below a predefined threshold such as $1e-6$), the algorithm has converged, and the loop is terminated.
- **Step 4: Eigenvalue Extraction**
  Once the algorithm converges, the eigenvalues of the matrix are approximated by the diagonal elements of A_{new}.

Function Output

The eigenvalues of the input matrix are extracted from the diagonal elements of the matrix A after convergence. These values are printed to the console.

Analysis

Strengths

- The implementation is self-contained and includes all necessary utility functions for matrix operations.
- It employs the Gram-Schmidt process, which is straightforward and easy to implement.
- Iterative convergence ensures the accuracy of the eigenvalues within a defined tolerance.

Limitations

- **Fixed Size:** The matrix size is limited by the compile-time constant N, reducing flexibility.
- **Numerical Stability:** The Gram-Schmidt method can be numerically unstable. Alternative approaches like Householder transformations may provide better results.

- **Convergence Criterion:** The convergence check is based on the norm of the matrix rather than focusing on off-diagonal elements.
- **Eigenvectors:** The implementation computes only eigenvalues. To compute eigenvectors, additional steps are required.

Conclusion

The QR algorithm is a powerful iterative method for computing the eigenvalues of a square matrix. While the provided implementation is effective and accurate, it can be improved for greater flexibility, stability, and functionality. These enhancements would make it suitable for a broader range of applications in numerical linear algebra.