Eigen Value Decomposition

G Sarvajith
ai24btech11008@iith.ac.in

1 Eigen Value Decomposition

Eigenvalue decomposition (EVD) is a method of decomposing a square matrix into its eigenvalues and eigenvectors.
It is one of the cornerstones of linear algebra, used extensively in mathematical modeling, numerical computation, and machine learning.
Eigenvalues(λ) are scalars associated with the transformation described by matrix.
Eigenvectors(v) are vectors that remain in the same direction(or opposite) after the transformation of matrix.
The relationship between a matrix(A), its eigenvalues and eigenvectors is

$$Av = \lambda v$$

The goal of Eigenvalue Decomposition is to express matrix A as:

$$A = V\Lambda V^{-1}$$

Where

- A: Original $n \times n$ matrix.
- V: A square $n \times n$ matrix whose columns are the eigenvectors of A.
- Λ: A diagonal $n \times n$ matirx where each diagonal entry is an eigenvalue of A.
- V^{-1}: The inverse of V.

Conditions for Eigenvalue Decomposition

- A matrix **A** should be diagonalizable(i.e it has a full set of n linearly independent eigenvectors).
- If A doesn't have n linearly independent eigenvectors, it cannot be decomposed.

Eigenvalue decomposition provides a way to understand how the matrix A scales and rotates vectors. Eigenvectors determine the directions in which the transformation occurs. Eigenvalues determine the scaling factors along those directions.

2 QR algorithm

The QR algorithm for eigenvalue decomposition relies on repeatedly factoring a matrix into an orthogonal matrix **Q** and an upper triangular matrix **R**.

QR decomposition is the process of decomposing a matrix A into:

$$A = QR$$

Preprint. Under review.

- Q is an orthogonal matrix, and columns of Q are orthonormal vectors.
- R is an upper triangular matrix.

There are many ways to compute the matrices Q, R. **Gram-Schmidt orthogonalization** process can be used to orthogonal matrix Q and Upper triangular matrix R.

3 Gram-Schmidt orthogonlization

Gram-Schmidt orthogonalization is a process to convert a set of linearly independent vectors into an orthonormal set of vectors, meaning the vectors are:

- Orthogonal: Each pair of vectors is perpendicular (their dot product is zero).
- Normalized: Each vector has unit length (magnitude of 1).

3.1 Upper Triangular matrix(R) and Orthogonal Matrix(Q)

A matrix A with columns as $a_1, a_2, a_3, ..., a_n$ is given as an input.

1. In the first step the first column of A(a_1) is normalized to give the vector q_1.
$$q_1 = \frac{a_1}{||a_1||}$$
2. For the other columns, it follows the following process:
for j=2,3,...,n:
$$q_j = \frac{u_j}{||u_j||}$$
where
$$u_j = a_j - \sum_{i=1}^{j-1}(q_i^T a_j)q_i$$
3. $$R_{ij} = \begin{cases} q_i^T a_j & \text{if } i \leq j, \\ 0 & \text{if } i > j. \end{cases}$$
4. The matrix with columns $q_1, q_2, \ldots, q_n$ is Q.

4 Iterative QR algorithm for Eigenvalues

Using QR decomposition iteratively transforms the matrix A into a nearly diagonal matrix, revealing its Eigenvalues.

- We will start with a matrix A_0 = A.
- QR Decomposition :
$$A_k = Q_k R_k$$
- Update A_{k+1} :
$$A_{k+1} = R_k Q_k$$
.
- Repeat these steps until A_k converges to a diagonal matrix Λ, whose diagonal entries are the eigen values of A.

The matrix A_k(updated after each iteration) converges to an upper triangle matrix whose diagonal elements are the Eigenvalues of the original matrix A. This matrix is referred as **Schur form** when A is asymmetric and **diagonal matrix** when A is symmetric.

4.1 Details of convergence

1. If A is symmetric, then it always converges to a diagonal matrix A_k.
2. If A is not symmetric, then the QR algorithm converges to an upper triangular matrix. The diagonal elements of this matrix are Eigenvalues.

The iterative step in the QR algorithm stops when the matrix A_k satisfies convergence criterion. This occurs when the off-diagonal elements of A_k become sufficiently small or within chosen tolerance. Stopping criteria in the QR algorithm:

- The algorithm stops when the off-diagonal elements of A_k are smaller than a predefined threshold ϵ
- The algorithm can also stop when the difference between successive iterates becomes negligible:

$$||A_{k+1} - A_k|| < \epsilon$$

- To avoid infinite loops in cases of slow convergence or divergence a maximum number of iterations N_{max} is often imposed.

5 Advantages of the QR Algorithm

The QR algorithm, has several advantages for computing eigenvalues of a matrix. Here are some of them :

1. **General Applicability**
   - The QR algorithm is applicable to any square matrix, whether symmetric, non-symmetric, real, or complex.
   - It does not require the matrix to have specific properties like diagonalizability or positive definiteness.
2. **Convergence to Eigenvalues**
   - The QR algorithm iteratively reduces the matrix to an upper triangular form (or Schur form for complex matrices).
   - The eigenvalues of the matrix emerge on the diagonal, making it an effective method for eigenvalue computation.
3. **Stability**
   - The QR algorithm is numerically stable, especially when combined with shifts (not present in your code but can be added).
   - Orthogonal transformations (QR decomposition) preserve the eigenvalues and improve numerical robustness.
4. **Iterative Nature**
   - The iterative approach makes the QR algorithm flexible in terms of precision. You can control the number of iterations or convergence tolerance (ϵ) to balance accuracy and computation time.
5. **Memory Efficiency**
   - The algorithm operates in-place on the matrix A, reducing the need for additional memory allocation for intermediate matrices.
6. **No Need for Direct Matrix Inversion**
   - The algorithm avoids matrix inversion, which can be computationally expensive and numerically unstable for large matrices.

6 C-Code

6.1 Helper Functions

```
    double** allocateMatrix(int n) {
    double** matrix = (double**)malloc(n * sizeof(double*));
    for (int i = 0; i < n; i++) {
        matrix[i] = (double*)malloc(n * sizeof(double));
    }
    return matrix;
}
```

- This function allocates a square matrix with double data type values, where the dimension of the to be created matrix (n) is taken as input.

```
    void freeMatrix(double** matrix, int n) {
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
```

- This function frees the dynamically allocated matrix, by the previous function. This is done to increase the memory efficiency.

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

As the function name suggests, it performs matrix multiplication.

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

- This function performs the transpose of a matrix.
- It takes a matrix as an input and stores it in another matrix.

6.2 QR-Algorithm's Functions

```
    void qrDecomposition(double** A, double** Q, double** R, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            Q[i][j] = A[i][j];
        }
    }


    for (int j = 0; j < n; j++) {
        R[j][j] = 0;
        for (int i = 0; i < n; i++) {
            R[j][j] += Q[i][j] * Q[i][j];
        }
        R[j][j] = sqrt(R[j][j]);

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

        for (int k = j + 1; k < n; k++) {
            R[j][k] = 0;
            for (int i = 0; i < n; i++) {
                R[j][k] += Q[i][j] * Q[i][k];
            }
            for (int i = 0; i < n; i++) {
                Q[i][k] -= R[j][k] * Q[i][j];
            }
        }
    }
}
```

- This block of code implements the Gram-Schmidt process for QR decomposition.
- It takes an input matrix A and decomposes it into an orthogonal matrix Q and an upper triangular matrix R.
- Initially all the elements in A are copied into Q, and then the matrix Q gets modified.
- The values in Q gets modified into orthonormal vetcors.
- **Note:** This is not the iterativ process.
- After the completion of loop, Q becomes a matrix of orthogonal unit vectors.
- R accumulates the norms and projection coefficients and forms an upper triangular matrix.

```
    void qrAlgorithm(double** A, int n) {
    double** Q = allocateMatrix(n);
    double** R = allocateMatrix(n);
    double** temp = allocateMatrix(n);

    for (int iter = 0; iter < MAX_ITER; iter++) {
        qrDecomposition(A, Q, R, n);  // QR decomposition of A
        multiplyMatrices(R, Q, temp, n);  // A = R * Q

        // Check convergence by looking at off-diagonal elements
        int converged = 1;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && fabs(A[i][j]) > EPSILON) {
                    converged = 0;
                    break;
                }
            }
            if (!converged) break;
        }

        if (converged) break;

        // Update A with the new values
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                A[i][j] = temp[i][j];
            }
        }
    }

    printf("Eigenvalues:\n");
    for (int i = 0; i < n; i++) {
        printf("%lf␣", A[i][i]);  // Eigenvalues are on the diagonal
    }
    printf("\n");

    freeMatrix(Q, n);
    freeMatrix(R, n);
    freeMatrix(temp, n);
}
```

- The actual QR algorithm is performed by this code.
- It takes the targeted matrix A as its input, and starts with allocating matrices for Q,R.
- Then starts the iterative process for QR Decomposition.
- The loop goes on until it reaches Max number of iterations or it reaches the convergence criterion.

7 Time Complexity

1. **Matrix Allocations** (`allocateMatrix`)
   - Allocating memory for matrices involves $O(n^2)$ operations for each matrix.
   - For three matrices Q, R, and temp, this totals $O(3n^2)$, but allocation is a one-time cost.
2. **QR Decomposition** (`qrDecomposition`)
   - For $R[j][j]$ calculation: $O(n)$ operations for each j.
   - For normalization of $Q[i][j]$: $O(n)$ operations for each i and j, hence $O(n^2)$ per column.
   - $R[j][k]$ and updating $Q[i][k]$: $O(n^2)$ per pair of j and k.

- Total for QR decomposition: $O(n^3)$ for all columns.

3. **Matrix Multiplication (`multiplyMatrices`)**
   - Computing temp$[i][j] = \sum A[i][k] \cdot B[k][j]$: $O(n^2)$ for each i, j, hence $O(n^3)$ overall.
4. **Convergence Check**
   - Checking off-diagonal elements: $O(n^2)$.
5. **Overall Iterations**
   - The QR algorithm requires up to `MAX_ITER` iterations. In practice, convergence is faster and depends on the spectral gap of the matrix. For k iterations:
     - Total cost per iteration: $O(n^3)$ (QR decomposition + matrix multiplication + checks).
     - Overall: $O(k \cdot n^3)$, where k is the number of iterations required to converge.

8 Space Complexity

1. **Matrix Storage**
   - Three matrices (Q, R, temp) require $3 \times O(n^2) = O(n^2)$ space.
   - Input matrix A: $O(n^2)$.
   - Total: $O(4n^2) = O(n^2)$.
2. **Other Storage**
   - Scalar variables like i, j, k, converged, $R[j][j]$: $O(1)$.
   - Total additional space is negligible compared to matrix storage.