Software Assignment- Report

EE24BTECH11039 - Ranjith

1 Eigen values

Eigenvalues are a fundamental concept in linear algebra. They are associated with square matrices and provide insight into the properties of these matrices.

Definition

Given a square matrix A, an eigenvalue λ is a scalar such that there exists a nonzero vector $\mathbf{v}$ (called the eigenvector)satisfying the equation:

$$A\mathbf{v} = \lambda\mathbf{v}.$$

Here:

- A is the square matrix.
- λ is the eigenvalue.
- $\mathbf{v}$ is the eigenvector, a vector that does not change direction under the transformation defined by A, only its magnitude is scaled by λ.

Finding Eigenvalues

2 Algorithms to calculate

2.1 Characteristic Polynomial Method

Steps

1) Let A be an $n \times n$ square matrix.
2) Form the characteristic equation:

$$\det(A - \lambda I) = 0,$$

   where λ is the eigenvalue, and I is the identity matrix.
3) Compute the determinant of $A - \lambda I$ to obtain a polynomial in λ, called the characteristic polynomial.
4) Solve the roots of the polynomial to find the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$.
5) For each eigenvalue λ, solve the equation:

$$(A - \lambda I)\mathbf{v} = 0,$$

   to find the eigenvector $\mathbf{v}$ corresponding to λ.

Example

Let:

$$A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}.$$

1) Form the characteristic equation:

$$\det\left(\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0.$$

2) Compute $A - \lambda I$:

$$A - \lambda I = \begin{bmatrix} 4-\lambda & 2 \\ 1 & 3-\lambda \end{bmatrix}.$$

3) Compute the determinant:

$$\det \begin{bmatrix} 4-\lambda & 2 \\ 1 & 3-\lambda \end{bmatrix} = (4-\lambda)(3-\lambda) - (2 \cdot 1).$$

4) Expand and simplify:

$$\lambda^2 - 7\lambda + 10 = 0.$$

5) Solve for λ:

$$\lambda = 5, \quad \lambda = 2.$$

Pros

- **Exact solution**: Provides an algebraic solution for small matrices.
- **Simple for small matrices**: Directly computes eigenvalues for 2×2 or 3×3 matrices.

Cons

- **Not scalable**: Determinant computation becomes computationally expensive for larger matrices ($n > 4$).
- **Numerical instability**: Solving high-degree polynomials is prone to rounding errors.

2.2 Power Iteration Method

Steps

1) Start with a random nonzero vector $\mathbf{v}_0$.
2) Normalize $\mathbf{v}_0$ by dividing it by its norm:

$$\mathbf{v}_0 = \frac{\mathbf{v}_0}{\|\mathbf{v}_0\|}.$$

3) Iteratively compute:

$$\mathbf{v}_{k+1} = A\mathbf{v}_k,$$

and normalize the vector:

$$\mathbf{v}_{k+1} = \frac{\mathbf{v}_{k+1}}{\|\mathbf{v}_{k+1}\|}.$$

4) After sufficient iterations, $\mathbf{v}_k$ approximates the eigenvector associated with the largest eigenvalue $\lambda_{\max}$.

5) Compute the eigenvalue using:

$$\lambda_{\max} \approx \frac{\mathbf{v}_k^\top A \mathbf{v}_k}{\mathbf{v}_k^\top \mathbf{v}_k}.$$

Example

Let:

$$A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix},$$

and let the initial vector be:

$$\mathbf{v}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

1) Normalize the initial vector:

$$\mathbf{v}_0 = \frac{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\sqrt{1^2 + 1^2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}.$$

2) Perform matrix multiplication:

$$\mathbf{v}_1 = A\mathbf{v}_0 = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 3\sqrt{2} \\ 2\sqrt{2} \end{bmatrix}.$$

3) Normalize $\mathbf{v}_1$:

$$\mathbf{v}_1 = \frac{\begin{bmatrix} 3\sqrt{2} \\ 2\sqrt{2} \end{bmatrix}}{\sqrt{(3\sqrt{2})^2 + (2\sqrt{2})^2}} = \begin{bmatrix} \frac{3}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} \end{bmatrix}.$$

4) Repeat until the vector stabilizes. The eigenvalue is then computed using the formula:

$$\lambda_{\max} \approx \frac{\mathbf{v}_k^\top A \mathbf{v}_k}{\mathbf{v}_k^\top \mathbf{v}_k}.$$

Pros

- **Efficient for large sparse matrices**: Focuses on the dominant eigenvalue.
- **Memory-efficient**: Requires minimal storage.

Cons

- **Finds only the largest eigenvalue**: Cannot compute all eigenvalues.
- **Slow convergence**: May require many iterations, especially if eigenvalues are close in magnitude.
- **Does not work well if the matrix is defective**: Fails for matrices with degenerate eigenvalues.

3 QR ALGORITHM

I prefer the QR Algorithm because it balances accuracy, generality, and efficiency. The QR Algorithm is a gold standard for finding eigenvalues in computational mathematics. The following are the deeper explanation of why it is often the preferred choice over other methods:

1) **Comprehensive Eigenvalue Computation**
   - **Why It's Important:** Many applications require *all* eigenvalues (and sometimes eigenvectors) of a matrix, not just the dominant one.
   - **Advantage of QR:**
     - Computes all eigenvalues in a single iterative process.
     - Unlike the Power Iteration Method, it is not limited to the largest eigenvalue.
     - Avoids the instability issues of the Characteristic Polynomial Method for large or poorly conditioned matrices.
2) **Numerical Stability**
   - **Why It's Important:** Computational errors due to floating-point arithmetic can accumulate, especially with high-dimensional matrices.
   - **Advantage of QR:**
     - Uses orthogonal transformations, which are inherently stable and prevent the amplification of rounding errors.
     - Avoids determinant or root-finding operations, which are prone to numerical instability.
3) **Applicability to Complex Eigenvalues**
   - **Why It's Important:** Many matrices, especially in physics, control theory, and signal processing, have complex eigenvalues.
   - **Advantage of QR:** Handles complex eigenvalues naturally using techniques like the *complex Schur form*. In contrast, the Power Iteration Method often fails for non-dominant or complex eigenvalues.
4) **Iterative Refinement**
   - **Why It's Important:** Large-scale problems often require iterative solutions for accuracy.
   - **Advantage of QR:**
     - Iteratively refines approximations to eigenvalues, ensuring highly accurate results.
     - Converges faster than simple iterative methods like Power Iteration, especially with shift strategies.
5) **Preprocessing Enhancements**
   - **Why It's Important:** Preprocessing can significantly reduce computational cost.
   - **Advantage of QR:**
     - Matrices can be reduced to Hessenberg or tridiagonal form (for symmetric matrices), speeding up computation while maintaining accuracy.
6) **Practical Applications**

- **Why It's Important:** Eigenvalues are essential in various fields:
  - **Physics:** Quantum mechanics, stability analysis.
  - **Engineering:** Vibrations, structural analysis, control systems.
  - **Machine Learning:** Principal Component Analysis (PCA).
- **Advantage of QR:** Universally applicable for both small systems in theoretical studies and large, sparse systems in machine learning.

Time complexity:

the total time complexity of the QR algorithm is: $O(k.n3)$

4 Gram-Schmidt

The Gram-Schmidt process is an orthogonalization procedure used to generate an orthogonal (or orthonormal) basis for a subspace spanned by a set of vectors. This process takes a set of linearly independent vectors and produces a set of orthogonal (or orthonormal) vectors that span the same subspace.

In the context of the QR decomposition, the Gram-Schmidt process is used to decompose a matrix AA (which is generally not orthogonal) into a product of two matrices:

QQ, an orthogonal matrix (whose columns are orthonormal vectors), RR, an upper triangular matrix.

5 Implementation of the QR Algorithm

Matrix Allocation

Matrix allocation is crucial in any algorithm that involves matrix manipulation. In this case, we dynamically allocate a 2D matrix of size $n \times n$ using malloc. This ensures that memory is allocated for the matrix based on the user-defined size of the matrix.

The functions createMatrix and freeMatrix handle memory allocation and deallocation, respectively. createMatrix creates a new matrix, allocating space for the rows and columns, while freeMatrix ensures that the allocated memory is properly released at the end of the computation, preventing memory leaks.

Input Handling

The scanMatrix function allows the user to input the elements of the square matrix. The user is prompted to enter each element of the matrix, which is then stored in the matrix array. This input is necessary as the QR algorithm requires a specific matrix to operate on. The matrix elements are stored in a 2D array, allowing the QR decomposition and eigenvalue extraction processes to operate on the matrix.

QR Decomposition (Gram-Schmidt)

QR decomposition is a process where a matrix A is decomposed into two matrices: Q (orthogonal matrix) and R (upper triangular matrix). In this implementation, the gramSchmidt function applies the classical Gram-Schmidt process to orthogonalize the columns of the matrix A.

- First, the columns of A are copied to Q. - Then, each column of Q is orthogonalized with respect to the previously computed columns. - This is done by subtracting projections of the current column on the previous columns and normalizing the current column to unit length. - The resulting matrices Q and R are used in the next steps of the QR algorithm.

This Gram-Schmidt process ensures that the columns of Q are orthogonal and that the matrix R is upper triangular.

QR Iteration

In the QR algorithm, the matrix A is repeatedly decomposed using QR decomposition in order to approximate the eigenvalues of the matrix. In the qrAlgorithm function, we repeatedly apply the QR decomposition to the matrix A as follows:

1) Apply Gram-Schmidt to decompose A into Q and R.
2) Update A by setting $A = R \cdot Q$, preserving the eigenvalues.
3) Repeat this process for a set number of iterations (1000 in this case).

Each iteration refines the approximation of the eigenvalues, and as the iterations continue, the matrix A converges to a diagonal matrix where the diagonal elements are the eigenvalues of the original matrix.

Eigenvalue Extraction

After a sufficient number of iterations, the matrix A converges. The diagonal elements of A represent the approximated eigenvalues of the original matrix. These values are extracted by simply reading the diagonal elements of the matrix A. This process yields the eigenvalues of the matrix, which are printed as output.

Main Function

The main function of the program performs the following tasks:

1) It takes the matrix size as input from the user.
2) It dynamically allocates memory for the matrix using the createMatrix function.
3) It prompts the user to input the matrix elements using the scanMatrix function.
4) It calls the qrAlgorithm function to compute the eigenvalues of the matrix through repeated QR decompositions.
5) Finally, it prints the approximated eigenvalues (the diagonal elements of matrix A) to the console.

The program also ensures proper memory management by calling `freeMatrix` at the end to deallocate the memory used by the matrix.

6 Conclusion

This project enhanced my understanding of numerical algorithms for linear algebra, particularly eigenvalue computation, and gave me valuable hands-on experience in implementing such algorithms. It also underscored the importance of efficiency, stability, and optimization in numerical methods. Most importantly, I now appreciate the broader applications of these mathematical techniques and their relevance in solving complex real-world problems.