Software Assignment

Sai Akhila

November 18, 2024

1 Introduction

Eigen values: Eigenvalues are scalar values that describe the factor by which the corresponding eigenvectors are stretched or compressed during a linear transformation. They are fundamental to understanding how a matrix acts on a vector space.

Mathematically, for a given square matrix A and a non-zero vector $\vec{v}$, if the matrix A transforms vv into a scalar multiple of vv, the scalar is called the eigenvalue corresponding to the eigenvector $\vec{v}$. This relationship is expressed as:

$$A\vec{v} = \lambda\vec{v} \tag{1}$$

where
A is a square matrix (of size n×nn×n).
$\vec{v}$ is an eigenvector of AA (a non-zero vector).
λ is the eigenvalue corresponding to vv.

2 Various algorithms widely used for computing eigen values:

2.1 Power iteration

1. Use case:

- To find the largest eigenvalue (in magnitude) and its corresponding eigenvector.

- Works well for sparse matrices.

2. Pros:

- Simple and easy to implement.

- Computationally inexpensive for large matrices.

3. **Cons:**
    - Converges slowly, especially for the eigen values that are close in magnitude.
    - Only finds one eigenvalue (the largest in magnitude).
    - May fail if the matrix is defective.

2.2 Inverse Power Iteration

1. **Use case:**
    - To find the smallest eigenvalue or eigenvalues closest to a specific value.
    - Useful for matrices where small eigenvalues are of interest.
2. **Pros:**
    - Effective for finding eigenvalues near a specific "shift."
    - Faster convergence as compared to standard power iteration.
3. **Cons:**
    - Requires solving a linear system at each step, which can be computationally expensive.
    - Needs a good initial guess for effective convergence.

2.3 Rayleigh Quotient Iteration

1. **Use case:**
    - To refine approximations of eigenvalues and eigenvectors.
    - Useful for problems requiring high precision.
2. **Pros:**
    - Cubic convergence (extremely fast for well-behaved problems).
    - Accurate for symmetric and Hermitian matrices.
3. **Cons:**
    - Requires solving a linear system in each iteration, which again can be computationally expensive.
    - Highly sensitive to the initial guess.

2.4 QR Algorithm

1. Use case:

- To compute all eigenvalues of a dense matrix (general-purpose algorithm).
- Works for both real and complex eigenvalues.

2. Pros:

- Robust and widely applicable.
- Efficient for small to medium-sized dense matrices.
- Handles both symmetric and non-symmetric matrices.

3. Cons:

- Computationally expensive for large matrices.
- Requires preprocessing for efficiency (e.g., reduction to Hessenberg form).

2.5 Jacobi Method

1. Use case:

- To compute all eigenvalues of symmetric matrices.
- Best suited for small to medium-sized matrices.

2. Pros:

- Simple and numerically stable.
- Explicitly constructs eigenvalues and eigenvectors.

3. Cons:

- Computationally expensive for large matrices.
- Converges slowly compared to other methods like QR.

2.6 Arnoldi Iteration

1. Use case:

- To compute a subset of eigenvalues for large, sparse, non-symmetric matrices.
- Popular in scientific computing and numerical simulations.

2. Pros:

- Efficient for large matrices.

- Constructs an approximation of the eigenvalues in Krylov subspaces.

3. **Cons:**
   - Requires storing Krylov subspaces, which can become memory-intensive.
   - May not converge for specific matrices.

2.7 Lanczos Algorithm

1. **Use case:**
   - To compute a subset of eigenvalues for large, sparse, symmetric (or Hermitian) matrices.
   - Used in physics, quantum mechanics, and machine learning.
2. **Pros:**
   - Memory-efficient compared to Arnoldi.
   - Highly efficient for symmetric or Hermitian matrices.
3. **Cons:**
   - May suffer from loss of orthogonality in the Krylov basis.
   - Requires reorthogonalization for accurate results.

2.8 Divide-and-Conquer Method

1. **Use case:**
   - To compute eigenvalues and eigenvectors of symmetric matrices.
   - Efficient for dense matrices.
2. **Pros:**
   - Faster than QR for large symmetric matrices.
   - Can compute all eigenvalues and eigenvectors.
3. **Cons:**
   - Limited to symmetric matrices.
   - Implementation is more complex than QR.

2.9 Faddeev-LeVerrier Algorithm

1. Use case:

- To compute the characteristic polynomial of a matrix, from which eigenvalues can be derived.

2. Pros:

- Theoretical importance in algebra.

3. Cons:

- Numerically unstable for large matrices.
- Rarely used in practical applications.

3 Chosen Algorithm

The QR Algorithm: The QR algorithm is an iterative method used to find the eigenvalues of a square matrix. It is based on decomposing a matrix into its QR decomposition (where Q is an orthogonal matrix and R is an upper triangular matrix), and then iteratively improving the approximation of the eigenvalues.

3.1 The Gram-Schmidt orthogonalization

Consider a matrix A. It can be represented as

$$A = [\vec{a_1}\vec{a_2}....\vec{a_3}]$$

$\tilde{q_1} = \vec{a_1}$, $\vec{q_1} = \frac{\tilde{q_1}}{\|\tilde{q_1}\|}$
$\tilde{q_2} = \vec{a}_2 - (a_2^T\vec{q_1})\vec{q_1}$
$\vec{q_2} = \frac{\tilde{q_2}}{\|\tilde{q_2}\|}$
Generalising the algorithm:
Intialise $\tilde{q_1} = \vec{a_1}$, $\vec{q_1} = \frac{\tilde{q_1}}{\|\tilde{q_1}\|}$
for r=2 to k:

$$\tilde{q_r} = \vec{a_r} - \sum_{i=1}^{r-1}(a_r^T q_i)q_i$$
$$\vec{q_r} = \frac{\tilde{q_r}}{\|\tilde{q_r}\|}$$

Hance, we can write any matrix A as:

$$\text{A} = \text{QR}$$

where Q is the orthogonal matrix that consists of orthogonal basis of A, and image space of A = image space of Q. R is an upper triangular matrix, whose diagonal entries are the norms of the respective column vectors and the upper diagonal entries are dots products of the corresponding vectors with the orthogonal vectors in Q (in the indices less than the corresponding index).

3.2 Hessenburg reduction:

To reduce a matrix A to Hessenberg form, the Householder transformations are commonly used. The process works by successively applying orthogonal transformations to eliminate the elements below the first sub-diagonal.
Householder transformations are used to create zeros below the diagonal and the first subdiagonal of the matrix. The goal is to zero out the entries $A_{i,j}$ for $i > j + 1$.
The Householder transformation is a reflection operation that transforms the given matrix by applying orthogonal matrices to it. A Householder matrix H is constructed as:

$$H = I - 2\frac{\vec{v}\vec{v}^T}{\vec{v}^T\vec{v}}$$

where $\vec{v}$ is a vector that reflects the given column vector $\vec{x}$ (the portion of the matrix you want to zero out) to align with a scalar multiple of a unit vector.
This is done repeatedly for each column, creating the Hessenberg form.
Repeat for all columns.
After applying the transformations, the result will be an upper Hessenberg matrix where all elements below the first sub-diagonal are zero.

3.3 Iterating the QR decomposition:

1. Perform QR decomposition $A_k = Q_k R_k$
2. Compute the next matrix $A_k + 1 = R_k Q_k$
3. Repeat the process until A_k converges to a nearly upper triangular matrix
4. The diagonal entries of the converged matrix are the eigenvalues of A.

4 Time Complexity:

The time complexity of the QR algorithm primarily depends on two factors:

1. The number of iterations required for convergence
2. The computational cost of each iteration

The total time complexity of the QR algorithm is the product of the cost of one QR decomposition and the number of iterations:

1. **Per iteration**: QR decomposition costs O(n^3)(using Gram-Schmidt algorithm)
2. **Number of iterations**: The algorithm typically converges in O(n) iterations.

Thus, the overall time complexity of the QR algorithm is:

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

For optimized version**(Hessenburg reduction)**: The matrix is often reduced to Hessenberg form (H) using a pre-processing step that costs O(n^3).
This reduces the cost of the QR decomposition to O(n^2) for Hessenberg matrices.
Time complexity with Hessenberg reduction: O(n^3) for reduction + O(n^2) per iteration $\implies$ Overall complexity: O(n^3)

5 Why this algorithm?

The QR algorithm is considered one of the most efficient and widely used methods for finding eigenvalues because of its robustness, accuracy, and ability to handle a wide range of matrices.

5.1 Versatility:

- **General-purpose**: The QR algorithm works for both symmetric and non-symmetric matrices, as well as real and complex eigenvalues.
- **No special assumptions**: Unlike some algorithms (e.g., Jacobi, which requires symmetric matrices), the QR algorithm is applicable to most square matrices.

5.2 Efficiency

- **Hessenberg Reduction**: Before applying the QR iterations, the matrix is often reduced to Hessenberg form (O(n^3) complexity), simplifying subsequent QR steps.
- **Iterative Convergence**: Each QR iteration is efficient, with a complexity of O(n^2) for Hessenberg matrices, making it scalable for moderate-sized matrices.

5.3 Computes All Eigenvalues

- The QR algorithm computes all eigenvalues simultaneously, unlike iterative methods like Power Iteration or Lanczos, which only find one or a subset of eigenvalues at a time.
- This makes it highly suitable for dense matrices where a full spectrum of eigenvalues is required.