Software Assignment: Eigenvalue Calculation

EE24BTECH11064 - Harshil Rathan

Contents

1 Eigenvalue

1.1 Definition

- Eigenvalues are a special set of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots.
- Let $\mathbf{A}$ be a linear transformation represented by a matrix $\mathbf{A}$. If there is a vector $\mathbf{X} \in \mathbb{R}^n$ with $\mathbf{X} \neq 0$ such that

$$\mathbf{AX} = \lambda \mathbf{X}$$

- for some scalar λ, then λ is called the eigenvalue of $\mathbf{A}$ with corresponding (right) eigenvector $\mathbf{X}$.
- Letting $\mathbf{A}$ be a $k \times k$ square matrix:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

- This can be written as:

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

- where I is the identity matrix.
- As shown in Cramer's rule, a linear system of equations has nontrivial solutions if and only if the determinant vanishes. Therefore, the solutions are given by:

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

2 Eigenvalue finding algorithms

1. QR Algorithm
2. Power Iteration Method
3. Jacobi Method

*2.1 **QR Algorithm***

The QR algorithm uses **QR decomposition** iteratively to transform a matrix into a form where its eigenvalues become evident

2.1.1 Matrix Decomposition

Given a square matrix $\mathbf{A}$, the matrix $\mathbf{A}$ can be decomposed like: $\mathbf{A} = \mathbf{QR}$, where

- $\mathbf{Q}$ is an orthogonal matrix
- $\mathbf{R}$ is an upper triangular matrix

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = 1$$

- This decomposition can be done using methods like Gram-Schmidt *etc*.

2.1.2 Iterative Process

We construct a new matrix $\mathbf{A_1}$, by reversing the order of $\mathbf{Q}$ and $\mathbf{R}$

$$\mathbf{A_1} = \mathbf{RQ}$$

- The new matrix $\mathbf{A_1}$ has the same eigenvalues as the matrix $\mathbf{A}$. We now repeat this process
- Take $\mathbf{A_1}$ and decompose it again to

$$\mathbf{A_1} = \mathbf{Q_1R_1}$$

- Compute $\mathbf{A_2}$ by reversing $\mathbf{Q_1}$ and $\mathbf{R_1}$

$$\mathbf{A_2} = \mathbf{R_1Q_1}$$

- Repeating this k times gives us

$$\mathbf{A_k} = \mathbf{Q_kR_k},\ \mathbf{A_{k+1}} = \mathbf{R_kQ_k}$$

2.1.3 Convergence

- The diagonal elements of $\mathbf{A_k}$ approach the eigenvalues of the original matrix $\mathbf{A}$ as $k \to \infty$
- The matrix $\mathbf{A_k}$ will converge to an upper triangular matrix with the eigenvalues on the diagonal, where k is the iteration count in the QR algorithm
- The iteration process is repeated till $\mathbf{A_k}$ becomes an upper triangular matrix

2.1.4 PROs

- It can compute all eigenvalues of a matrix simultaneously
- It can be adapted to matrices having real, complex entries
- Efficient for dense and symmetric matrices, converges rapidly for symmetric matrices

2.1.5 CONs

- Computation cost for Large matrices, for a $n \times n$ matrix each iteration has a time complexity of $O(n^3)$. (Time Complexity)
- It requires storing multiple matrices which can consume high memory, which makes it less practical for very large matrices.
- Limited efficiency for Sparse Matrices, suitable for dense and symmetric matrices only

*2.2 **Power-Iteration Method***

This Method is simple and is used to find the dominant eigenvalue (The eigenvalue with the largest absolute value) of a given matrix $\mathbf{A}$. The given matrix $\mathbf{A}$ has to be a square matrix.

2.2.1 Initialization

- Given a square matrix $\mathbf{A}$ and a random vector $\mathbf{x_0}$. Choose $\mathbf{x_0}$ as the starting point. This vector will be iteratively transformed to approximate the eigenvector associated with the dominant eigenvalue of $\mathbf{A}$.

- The initial vector $\mathbf{x_0}$ can be randomly chosen but it should have atleast some component in the direction of the dominant eigenvector.
- Check for Orthogonality at each iteration.

2.2.2 Iterate

- This process iterates repeatedly by multiplying the matrix $\mathbf{A}$ by the vector $\mathbf{x_k}$, where $\mathbf{x_k}$ is the vector at each step k. For k = 1,2,3 $\cdots$
- compute the next vector $\mathbf{x_{k+1}} = \mathbf{Ax_k}$
- Normalize $\mathbf{x_{k+1}}$ by dividing by it's norm

$$\mathbf{x}_{k+1} = \frac{\mathbf{A}\mathbf{x}_k}{|\mathbf{A}\mathbf{x}_k|}$$

- This multiplication and normalization process is repeated until $\mathbf{x_k}$ converges

2.2.3 Convergence

- The vector $\mathbf{x_k}$ converges to the eigenvector associated with the dominant eigenvalue, as $k \to \infty$
- The eigenvalue λ can be approximated as

$$\lambda_k = \frac{\mathbf{x}_k^T \mathbf{A}\mathbf{x}_k}{\mathbf{x}_k^T \mathbf{x}_k}$$

- If the matrix has a dominant eigenvalue, the convergence is usually fast, but if the difference between the magnitude of the eigenvalues is less the convergence is slow

2.2.4 PROs

- Simple to implement and computationally inexpensive, making it efficient for large matrices when only the dominant eigenvalue is needed.
- Suitable for sparse matrices, as it requires only matrix-vector multiplication.
- Works well when the matrix has a unique dominant eigenvalue.

2.2.5 CONs

- Only finds the dominant eigenvalue, additional techniques are required to find other eigenvalues.
- May converge slowly or not at all if the dominant eigenvalue is not significantly larger than other eigenvalues.
- Limited to matrices with a clear dominant eigenvalue
- Does not work for matrices with complex values

2.3 Jacobi Method

This method is an iterative process designed for computing the eigenvalues and eigenvectors of real symmetric matrices. It transforms the diagonal elements of the matrix to be it's eigenvalues. It is based on a series of rotations called Jacobi or given rotations.

2.3.1 Initial Transformations

- It applies a series of Jacobi rotations (orthogonal transformations) to the zero out off-diagonal elements iteratively
- A rotation matrix $\mathbf{J}(p, q, \theta)$ is constructed for each iteration, targeting the off-diagonal element $\mathbf{A_{pq}}$
- The new matrix $\mathbf{A'}$ minimizes the off-diagonal entries as to $\mathbf{A}$

2.3.2 Constraints

- As we want to make the off element of the new matrix $\mathbf{A'}$ zero. We write the condition

$$0 = (c^2 - s^2)\mathbf{a_{pq}} + cs(\mathbf{a_{pp}} - \mathbf{a_{qq}})$$

- As $c = \cos\theta$ and $s = \sin\theta$, if $\mathbf{a_{pq}} \neq 0$ we get

$$\phi = \cot 2\theta = \frac{(a_{qq} - a_{pp})}{2a_{pq}}$$

- Say $\tan\theta = t$, then

$$t = \tan\theta = \phi \pm \sqrt{\phi^2 + 1}$$

- c and s are computed as

$$c = \frac{1}{\sqrt{1 + t^2}}, \quad s = t \cdot c$$

2.3.3 Iterative Process

- Identify the largest off-diagonal element $\mathbf{a_{pq}}$
- Perform operations using $\mathbf{J}(p, q, \theta)$ and transform $\mathbf{A'} = \mathbf{J^T A J}$
- Update the new matrix $\mathbf{A'}$ to $\mathbf{A}$ and repeat the process until the off-diagonal elements are smaller than the tolerance ϵ

2.3.4 Convergence

- Convergence is guaranteed for symmetric matrices due to its reliance on symmetric transformations
- It iteratively reduces the magnitude of the off-diagonal elements, leading to diagonalization
- Convergence can be fast for matrices when the eigenvalues are well-separated, for matrices with close eigenvalues it requires more iterations
- The algorithm stops when the largest off-diagonal element is smaller than the pre-defined tolerance ϵ

2.3.5 PROs

- The algorithm can be easily implemented since it does not require complex matrix factorization like the QR

- It also yield the eigenvectors of the given matrix. The columns of the transformed matrix are its eigenvectors and the diagonal elements are its eigenvalues

2.3.6 CONs

- High cost of computation, the overall complexity is $O(n^3)$, making it inefficient for large matrices
- Is efficient only for real symmetric matrices
- For matrices with close eigenvalues, convergence may be slow and require more iterations

*2.4 **Comparision***

Criteria	**QR Algorithm**	**Power Iteration Method**	**Jacobi Method**
Purpose	Computes all eigenvalues and eigenvectors.	Computes the dominant eigenvalue.	Computes all eigenvalues of symmetric matrices.
Matrix Suitability	Dense Matrices and Symmetric Matrices	Large, Sparse matrices.	Symmetric matrices.
Complexity (Per Iteration)	$O(n^3)$ for dense matrices, $O(kn^2)$ for sparse matrices (k = iterations).	$O(n^2)$ for large sparse matrices (scales well with size).	$O(n^3)$, with slower convergence compared to QR.
Convergence	Quadratic convergence for symmetric matrices slower for general matrices.	Linear convergence requires well-separated eigenvalues.	Quadratic convergence for symmetric matrices.
Efficiency	Efficient for small to medium-sized matrices costly for very large matrices.	Highly efficient for sparse, large matrices focusing on a single eigenvalue.	Inefficient for large matrices due to iterative pairwise rotations.
Key Features	- Computes all eigenvalues at once. - Suitable for iterative refinements (shifted QR).	- Simplest method. - Focuses on the largest eigenvalue. - Memory-efficient.	- Simple to implement for symmetric matrices. - Explicitly diagonalizes the matrix.

2.5 Chosen Algorithm

QR Algorithm

- QR algorithm is more efficient as it generates all the eigenvalues at once
- Power Iteration Algorithm only computes the largest eigenvalue hence is not suitable
- Jacobi method is more efficient for symmetric matrices and also has a slower convergence as compared to QR
- QR also computes eigenvalues for matrices with complex entries and fatser convergence for small-medium size matrices

3 References

- Eigenvalue
- QR Algorithm
- Power Iteration
- Jacobi Method
- VMLS - Python Companion
- VMLS.pdf
- Householder Algorithm