Eigenvalue Calculation

Eshan Ray
Dept. of EE, IITH
EE24BTECH11021

November 18, 2024

Abstract

This report examines key algorithms for eigenvalue computation, including **Power Iteration**, **Hessenberg Reduction**, and the **QR Algorithm with Shift**. It highlights their strengths, limitations, and suitability for different types of matrices, providing insights into selecting the appropriate method based on matrix size, structure, and computational efficiency.

Contents

1 Introduction

The German prefix "eigen" roughly translates to "self" or "own". An eigenvector of A is a vector that is taken to a multiple of itself by the matrix transformation $T(x) = Ax$, so it is a vector that remains in the same direction but gets scaled by some value.
This value which a eigenvector gets scaled with is known as an eigenvalue.
Eigenvalues can be real as well as complex depending on the matrix A.

2 Objective of the Study

The objective of the study is to explore and compare different numerical methods for eigenvalue computation, focusing on their efficiency, accuracy, and suitability for different types of matrices. This will mainly involve the following algorithms:-

- **Gram-schmidt Process**
- **QR with shift algorithm**
- **Hessenberg Decomposition**
- **Power Iteration Method**

3 QR decomposition

QR Decomposition is used to factor a matrix A into two components: an orthogonal matrix Q and an upper triangular matrix R.
For a given matrix A of size $n \times n$, QR decomposition represents it as:

$$A = Q \cdot R$$

where:
Q is an $n \times n$ orthogonal matrix (or unitary if dealing with complex matrices). This means that the columns of Q are orthogonal (i.e., their dot product is zero), and $Q^\top Q = I$(the identity matrix).
R is an $n \times n$ upper triangular matrix. For real matrices, this means that all entries below the main diagonal are zero, i.e., $R_{ij} = 0$ for $i > j$.

QR decomposition is used in various algorithms for finding solutions to systems of linear equations of the form $Ax = b$, and also for computing of eigenvalues of matrices.

Some of the algorithms for finding eigenvalues which uses QR is given below :-

3.1 Gram-schmidt Process

The Gram-Schmidt process is a fundamental algorithm in linear algebra used to orthogonalize a set of vectors in an inner product space, most commonly in Euclidean space. It takes a set

of linearly independent vectors and transforms them into an orthogonal or orthonormal set.
Steps for Gram-Schmidt process

1) We start by producing an orthogonal set of vectors $\{q_1, q_2, \ldots, q_n\}$ from a set of linearly independent vectors $\{a_1, a_2, \ldots, a_n\}$.

2) For orthogonalization we subtract each vector a_i with the projections of all previously obtained orthogonal vectors $q_1, q_2, \ldots, q_{i-1}$ to make v_i orthogonal to them.
The projection of a_i onto a vector q_j is calculated as:

$$proj_{q_j}(a_i) = \frac{\langle a_i, q_j \rangle}{\langle q_j, q_j \rangle} q_j$$

Then q_i is computed as:

$$q_i = a_i - \sum_{j=1}^{i-1} proj_{q_j}(a_i)$$

Then all the q_i's are normalized by :

$$q_i = \frac{q_i}{||q_i||}$$

After the process, $q_1, q_2, \ldots, q_n$ form an orthonormal basis for the vectors $(a_1, a_2, \ldots, a_n)$

3) Modified algorithm : when we encounter $q_j = 0$, we skip to the next vector a_{j+1} and continue.
We also compute the projection and orthogonalization of the vector at a single step unlike the normal process which increases its efficiency and stability.

3.2 QR With shift

Basic QR algorithm
In the basic QR algorithm, the matrix A is decomposed into matrices Q and R as:

$$A = QR$$

Then, the new matrix A_{new} is computed as:

$$A_{new} = RQ$$

This process is repeated until the off-diagonal elements of the matrix become negligibly small, at which point the diagonal elements approximate the eigenvalues of the original matrix.

The shifted QR iteration is as follows:

1) Modify the matrix by subtracting it with μI where μ is a shift parameter, thus giving the following matrix:

$$A' = A - \mu I$$

2) Compute the QR decomposition of A' and revert the shift in the next matrix as $A'' = RQ + \mu I$.

3) Repeat the process until the matrix becomes nearly upper triangular, i.e., the off-diagonal elements are sufficiently small, indicating that the process is completed and eiganvalues of the matrix is computed.

This process's ability to converge is very fast but it increases the computational cost of it in each iteration.
it increases numerical instability, particularly in the case of matrices with closely spaced eigenvalues.

3.3 Hessenberg Algorithm

1) **Householder reflection**

- First a column vector x is taken which is generally the first column of the sub-matrix.
- A vector $v = x - \alpha e_1$ is defined where e_1 is the standard basis vector/impulse vectors and α is chosen such that after reflection all the elements in x are zero except the first.
- The vector v is normalised and Householder reflection matrix is given by:

$$Q = I - vv^\top$$

  The matrix Q formed is symmetric and orthogonal.

2) Q is used to transform A into upper hessenberg matrix H where:

$$A = QHQ^\top$$

3) Apply QR decomposition on H and update the matrix as $H_{new} = RQ$ on every iteration until it converges to upper triangular matrix whose diagonal elements are eigenvalues of matrix A.

This process is very efficient for large matrices and reduces the computational complexity a lot compared to regular QR algorithm.
But it may cause loss of precision while using QR on hessenberg matrix and it might show slow convergence for certain eigenvalue problems.

3.4 Power Iteration Method

Power iteration is one of the simplest iterative methods for computing the largest eigenvalue of a matrix A, and the corresponding eigenvector. It is based on the idea that repeated multiplication of a random vector by the matrix will eventually converge to the eigenvector with largest eigenvalue.
Steps of the algorithm

1) We start with an initial random vector b_0 and compute the sequence of vectors:

$$b_{k+1} = Ab_k$$

2) Normalize the vector b_{k+1} to avoid overflow or underflow

3) After several iterations, estimate the eigenvalue corresponding to the eigenvector b_k by using the Rayleigh quotient:

$$\lambda_k = \frac{b_k^\top A b_k}{b_k^\top b_k}$$

The iteration continues until b_k converges to the dominant eigenvector, and λ_k stabilizes to the dominant eigenvalue.

Rayleigh Quotient Iteration (RQI):
Rayleigh quotient iteration is a more sophisticated and faster method for finding an eigenvalue and eigenvector of a matrix A. It improves upon power iteration by refining the eigenvalue estimate at each step using the Rayleigh quotient, and it typically converges quadratically when starting near an eigenvalue.

1) s tart with a initial guess vector y_0 and compute the Rayleigh quotient

$$\lambda_0 = \frac{y_0^\top A y_0}{y_0^\top y_0}$$

2) Solve the linear system:

$$(A - \lambda_k I) y_{k+1} = y_k$$

where λ_k is the current estimate of the eigenvalue.
Update the eigenvalue estimate using the Rayleigh quotient and repeat the process until y_k and λ_k converge to the eigenvector and eigenvalue, respectively.

4 Algorithm Selection and Justification

For this report, I chose the QR algorithm with shifts to compute the eigenvalues of a matrix. This algorithm was selected because it is efficient and robust for eigenvalue problems. The inclusion of shifts accelerates convergence by dynamically adjusting the algorithm during iterations, improving its speed and accuracy. This method strikes a good balance between computational efficiency and precision, making it an ideal choice for the task at hand.

4.1 Python Code for Real Matrices

This is the python code used for the calculation of eigenvalues of real matrices using QR algorithm with shifts:
Path to the python code:
`https://github.com/eshan810/EE1030/blob/main/softAssign/codes/real.py`

Some of the results obtained by giving various types of inputs to the code and matching the precision and accuracy with the expected output have been shown in Table 1,

Input	Computed output	Expected output
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 16.116844 \\ 0.0 \\ -1.116844 \end{bmatrix}$	$\begin{bmatrix} 16.116844 \\ 0.0 \\ -1.116844 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 6 \\ 4 & 6 & 9 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ -0.745967 \\ 14.745967 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ -0.745967 \\ 14.745967 \end{bmatrix}$
$\begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.499999 + 0.866025j \\ 0.499999 - 0.866025j \end{bmatrix}$	$\begin{bmatrix} 0.5 + 0.866025j \\ 0.5 - 0.866025j \end{bmatrix}$

Table 1: Output Table

4.2 C code for complex Matrices

This is the c code used for the calculation of eigenvalues of complex matrices using QR algorithm with shifts:
Path to the c code:
`https://github.com/eshan810/EE1030/blob/main/softAssign/codes/complex.c`
This code helps in faster computation of complex matrices as python code without external libraries can be computationally very slow for complex matrices and might incur inaccuracies in the output. Some of the results obtained by giving various types of complex inputs(where atleast one entry should have non zero imaginary part) to the code and matching the precision and accuracy with the expected output have been shown in Table 2,

Input	Computed output	Expected output
$\begin{bmatrix} 2+1j & -1+7j \\ 1+3j & 1+6j \end{bmatrix}$	$\begin{bmatrix} 1.358314 - 1.793399j \\ 1.641685 + 8.793399j \end{bmatrix}$	$\begin{bmatrix} 1.358314 - 1.793399j \\ 1.641685 + 8.793399j \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 \\ 4+3j & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 16.128356 + 0.412322j \\ -1.327054 + 0.372370j \\ 0.198698 + -0.784692j \end{bmatrix}$	$\begin{bmatrix} 16.128356 + 0.412322j \\ -1.327054 + 0.372370j \\ 0.198698 + -0.784692j \end{bmatrix}$

Table 2: Complex Output Table

5 Limitations

This algorithm is very good in dealing with dense matrices but shows some limitations too, they are as follows:-

- High Computational Cost as it requires $O(n^3)$ operations per iteration due to matrix factorizations, making it expensive for large matrices.
- Dependence on Shift Choice: The algorithm's performance is sensitive to the choice of shift, and a poor shift can slow convergence or lead to failure.

- Not Efficient for Sparse Matrices: Does not exploit matrix sparsity, leading to inefficiencies in large sparse systems.

6 Conclusion

In conclusion, eigenvalue calculation methods such as **Power Iteration**, the **Hessenberg Reduction**, and the **QR Algorithm with Shift** each have distinct advantages based on matrix characteristics and computational needs.
Power Iteration is efficient for finding the dominant eigenvalue, particularly in sparse or well-conditioned matrices.
Hessenberg Reduction simplifies the problem by transforming a matrix into a Hessenberg form, improving the efficiency of subsequent eigenvalue methods like QR.
The **QR algorithm with shift** is versatile and suitable for general matrices, offering faster convergence with shifts, but it comes with higher computational cost. The choice of algorithm depends on factors such as matrix size, structure, and the specific eigenvalue problem.

7 References

The references section lists all the sources you referred to while writing the report. You can use the `cite` package for citations. For example:

- Stephen Boyd and Lieven Vandenberghe.Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares, 5.4, 97-103
- Dinh Phuoc Vinh, Theory and problems of mathematics for machine learning, chap-2
- Dan Margalit, Joseph Rabinoff. Interactive Linear Algebra
- Wikipedia
- Other online sources like youtube, github, etc..