Finding Eigenvalues through QR Decomposition Algorithm

EE24BTECH11008

About eigenvalues: For a matrix, **A** the eigen values are the $\lambda\prime s$ which are obtained through solving the equation

$$\mathbf{AV}= \lambda \ \mathbf{V}$$

where **V** is the eigenvector.
In general, when a matrix is multiplied with another, it does not produce a scaled multiple of the second one, both for some special matrices, it happens.They are known as eigenvectros and the scaling factor is known as eigenvalue.
For an $n \times n$ matrix usually there exist n number of eigenvectors and eigenvalues.
There are so many algorithms to find out these eigenvalues.

I. Introduction

I used QR decomposition algorithm to find the eigenvalues of a square matrix. This algorithm works on the basis that a square matrix can be decomposed into product of an orthogonal matrix **Q** $\left(Q^T Q = I\right)$ and an upper triangular matrix **R**, such that:

$$\mathbf{A=QR}$$

II. Explanation of algorithm

* First compute **Q** and **R** using Gram-Schmidt.
* Then get A_1 **=RQ**
* Repeat the process for around 1000 times and figure out A_k
* $\because$ **A** and A_1 are similar, eigenvalues of them are also similar.
* Repeat the process for somany times,automatically we will find that all the non-diagonal elements ofA_k will tend to zero and eigenvalues of the matrix will be the diagonal elements of that matrix.
* This algorithm does not works for complex inputs.

III. Time Complexity of the QR Decomposition Algorithm

For a matrix of size $n \times n$, QR decomposition needs $\mathbf{O}(n^3)$ operations. $\therefore$ Computational cost of one iteration is $\mathbf{O}(n^3)$.For well-conditioned and diagonalizable matrices, the number of iterations required is typically $\mathbf{O}(n)$.The overall time complexity of the QR algorithm for eigenvalue computation is: $\mathbf{O}(n^4)$

A. Memory Usage in the Code

The code allocates memory dynamically for four matrices:

- Matrix A: The input matrix for which eigenvalues are to be computed.
- Matrix Q: The orthogonal matrix produced during the decomposition.
- Matrix R: The upper triangular matrix produced during the decomposition.
- Matrix A_1: A temporary matrix used to store the updated matrix after each iteration.

Each matrix is dynamically allocated as a $n \times n$ array, requiring $O(n^2)$ memory. With four matrices, the total memory usage is:

$$\text{Total Memory} = 4 \times O(n^2) = O(n^2)$$

For large matrices: For large matrices, this algorithm is not effieint, $\because$ the memory usage is of order $O\left(n^2\right)$

For Symmetric and Diagonal matrices: Other algorithms are efficient than this one.

Comparison with other algorithms: One of its key strengths is that it can compute all eigenvalues of a matrix, unlike the Power method algorithm, which only finds the largest eigenvalue, or Inverse Iteration method, which is better suited for finding eigenvalues near a integer.

But the QR algorithm is slow and computationally expensive, $\because$ it has a time complexity of $O(n^3)$.

Lanczos and Davidson or ARPACK are more more efficient for large matrices. These algorithms are for sparse matrices and can find a few eigenvalues much more quickly by taking advantage of the matrix′s sparsity. But, they are generally limited to symmetric matrices.

In comparison, the Jacobi method is another algorithm that works well for symmetric matrices, but it tends to be very slow and inefficient for large problems.

Overall, while the QR algorithm is a reliable choice for most situations, it may not always be the best option for large-scale problems. In such cases, methods like Lanczos or Davidson or ARPACK are often faster, and if you're just looking for specific eigenvalues, the Power method or Inverse Iteration might do the job more efficiently.

IV. Conclusion:

So, choosing the right algorithm depends on the size of your matrix, its properties, and the specific eigenvalues you're interested in finding.

V. References:

- Youtube Channel- $3Blue1Brown$
- Youtube Channel- $MITOpenCourseWare$