Software Assignment:Eigenvalue calculation

EE24BTECH11032- John Bobby

I. What are eigen values?

Given a square matrix $\mathbf{A}$, an eigenvalue λ is a number such that there exists a non-zero vector $\mathbf{v}$ which is called the eigen vector such that

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

II. Jacobi's Algorithm

A. *Theory*

Jacobi's algorithm is an iterative algorithm that calculates the eigen values of a real symmetric matrix. The algorithm is known for its simplicity and precision for small or medium sized matrices. The algorithm consists of applying the Jacobi rotation till the matrix becomes diagonal matrix or close to a diagonal matrix.
A Brief overview of the algorithm is given below

1) Find the largest non-diagonal element $\left(a_{ij}\right)$
2) compute $\theta = \frac{1}{2}\arctan\frac{2a_{ij}}{a_{ii}-a_{jj}}$
3) create a matrix $\mathbf{P}$ such that
   $P_{ii} = \cos\theta$
   $P_{jj} = \cos\theta$
   $P_{ij} = -\sin\theta$
   $P_{ji} = \sin\theta$
   and the rest all elements are similar to the identity matrix.

4) apply the rotation by computing $\mathbf{A_i} = \mathbf{P}^\top\mathbf{A}\mathbf{P}$
   where $\mathbf{A_i}$ is the matrix after ith rotation of the given matrix $\mathbf{A}$
   Applying the above rotation makes the maximum off diagonal elements $\left(a_{ij}, a_{ji}\right)$ zeroed and also maintains the symmetry. The process is continued and finally a diagonal matrix is achieved whose diagonal elements are the eigen values.

The Advantages of the algorithm are

1) **Numerical Stability for Symmetric matrices**
   The algorithm works very well for symmetric matrices. As symmetric matrices have real and orthogonal eigen values the mathematical behavior of the algorithm is simplified.

2) **Suitable for Small or Medium sized matrices**
   The algorithms converge in few iterations itself for small or medium sized matrices.
   For example
   3×3 symmetric matrix $\begin{pmatrix} 4 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{pmatrix}$ converges approximately (tolerance=10^{-8}) to a diagonal matrix in

8 iterations.

4×4 symmetric matrix $\begin{pmatrix}7 & 5 & 6 & 4\\5 & 3 & 5 & 7\\6 & 5 & 8 & 6\\4 & 7 & 6 & 2\end{pmatrix}$ converges approximately (tolerance=10^{-8}) to a diagonal matrix in 14 iterations.

3) **Accuracy for Small or Medium sized matrices**
For the 3×3 matrix $\begin{pmatrix}4 & 1 & 2\\1 & 3 & 0\\2 & 0 & 5\end{pmatrix}$ the eigen values are: $6, 3, 1.85$
the eigen values given by the code are: $6.66, 3.47, 1.85$
the values are close with real ones within an error of1.

The Disadvantages of Jacobi's Algorithm are

1) **Not Efficient for Large matrices**
As the convergence speed of a symmetric matrix is proportional to the size of the matrix. Large matrices need large no of iterations.

2) **Convergence Speed**
On comparing with other algorithms like QR decomposition the convergence speed is slow.

3) **Only for Symmetric matrices**
The algorithm only finds the eigen values of symmetric matrices.

B. Time Complexity

Code is available in Codes/jacobi_iteration.c

Time complexity analysis of the code is given below

1) **larges_off_diag()**
The function returns the largest off diagonal element from the matrix by iterating through each element. 2 loops are present which run for n times where n is the order of the matrix.
Thus the overall complexity is $O\left(n^2\right)$

2) **check_diagonal()**
The function iterates through diagonal elements and check they are close to zero, by iterating through 2 loops
Thus the overall complexity is $O\left(n^2\right)$

3) **mult()**
Multiplies 2 matrices and stores the result in a third matrix. The function iterates through 3 loops.
Thus the overall complexity is $O\left(n^3\right)$

4) **transpose()**
Calculates the transpose and stores in another matrix. Iteration is done through 2 loops.
Thus the overall complexity is $O\left(n^2\right)$

5) **main()**
   Complexity of 1 iteration

   - check_diagonal() runs in $O\left(n^2\right)$
   - largest_off_diag() runs in $O\left(n^2\right)$
   - mult() runs 2 times in $O\left(n^3\right)$

   Thus the total complexity per iteration is $O\left(n^3\right)$, assuming it takes i iterations to converge
   Overall Complexity=$O\left(i \times n^3\right)$
   The most computationally expensive part is the mult() function which contributes more to the overall complexity.

C. Memory Usage

The memory usage analysis of the code ignoring all the stack variables is given below

1) **Static Memory**
   The static variables used are
   - int (size=4bytes) n, max_iter, iter. Total=12bytes
   - double (size=8bytes), A[n][n], S[n][n], St[n][n], temp[n][n], temp2[n][n], theta. Total=$40n^2$+8bytes
2) **Dynamic Memory**
   The dynamic variable used are
   - int * (size=2×8) result. Total=16bytes

Thus the overall memory usage = $12 + 40n^2 + 8 + 16 = 40n^2 + 36$ which is of $O\left(n^2\right)$

D. Comparison

On comparing with QR algorithm, Power Iteration Method and Lanczo's Algorithm

1) **Similarities**

   - All 3 algorithms are iterative in nature, and the solutions are approximate.
   - All the algorithms consists of repetitive matrix multiplication until a specific condition is reached.
   - The rate of convergence depends on the properties of the matrix like size and symmetry.

2) **Differences**

   - Lanczo's and Jacobi's are only applied to symmetric matrices whereas Power Iteration and QR is applied to all matrices.
   - Power iteration only finds the largest eigen value in magnitude whereas others find all the eigen values. Also it is a very simple and efficient algorithm when we need only the largest eigen value.
   - Lanczo's algorithm is useful for large sparse(most of elements are 0) as the complexity is of $O(m \times n)$, where m is the number of non-zero elements and n is the order.

- In terms of convergence speed Jacobi's take time to converge for large matrices which is overcome by QR algorithm. The convergence of the Power iteration depends on the difference between the largest and the second largest, less the gap more the multiplications need to be done so that the term due to largest eigen value dominates.
- Even though QR can handle assymetric matrices the number of iterations needed is too much large.