భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Report
Software Assignment: Eigenvalue Calculation

AI24BTECH11015 - Harshvardhan Patidar
Department of Artificial Intelligence

Indian Institute of Technology, Hyderabad
November 18, 2024

Contents

1 What are Eigenvalues?

[1] For a $n \times n$ matrix, multiplying it, or simply applying the matrix to a $n \times 1$ vector gives a transformed vector. If the transformed vector is a scalar multiple of the original vector, then that vector is called an eigenvector of the matrix and the scalar is called an eigenvalue of the matrix. Let A be a square matrix of order n, and

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

then λ is an eigenvalue of the matrix A corresponding to the eigenvector $\mathbf{v}$. The simplest way to calculate the eigenvalues of a matrix is to solve

$$|A - \lambda I| = 0$$

to get all the eigenvalues.

2 Algorithms to calculate Eigenvalues

[4] There are lots of algorithms already discovered to calculate the eigenvalues of a given matrix. Let's discuss some of them :

2.1 Lanczos Algorithm

It is an iterative algorithm that calculates the m most useful eigenvalues of a matrix. It does so by using the Krylov Subspaces. It creates a smaller matrix that approximates the original matrix's eigenvalues. Its time complexity is $\mathcal{O}(n^2k)$ where n is the order of the matrix and k is the number of iterations.

Algorithm 1 Lanczos Algorithm

Given: Hermitian Matrix A, initial vector v_0, iterations k

Normalize v_0 : v_1 = v_0 / $||v_0||$

Take $w_1 = Av_1 - v_1^* Av_1$

for $i = 2$ to k **do**

  Let $\beta_j = ||w_{j-1}||$

  if $\beta_j \neq 0$ **then**

    $v_j = w_{j-1}/\beta_j$

  else

    pick v_j as an arbitrary vector such that $||v_j|| = 1$ and is orthogonal to $v_1, \ldots, v_{j-1}$

  end if

  $w'_j = Av_1$

  $\alpha_j = w'^*_j v_j$

  $w_j = w'_j - \alpha_j v_j - \beta_j v_{j-1}$

end for

Let V be matrix with columns $v_1, \ldots, v_m$. T =

$$\begin{bmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_k & \alpha_m \end{bmatrix}$$

If λ is an eigenvalue of T with x as eigenvector, then λ is also an eigenvalue of A with eigenvector as $y = Vx$.

Pros	Cons
It is greatly efficient for large sparse matrices, i.e. the matrices with many of the entries as zero. It's convergence time is also low, causing it to quickly converge to the eigenvalues.	It lacks numerical stability. The vectors on which it is working may become less accurate over time. Thus adding extra steps to restore their accuracy, and decreasing the efficiency.

2.2 Power Iteration

This is also an iterative algorithm which approximates the numerically largest eigenvalues. The higher is the number of iterations better is the convergence. Its time complexity is $\mathcal{O}(n^2)$ where n is the order of the matrix.

Algorithm 2 Power Iteration

Given Matrix A, max iterations k, tolerance ϵ

Let v_0 be an arbitrary vector such that $||v_0|| = 1$
for $i = 1$ to k **do**
    Calculate $v_i = \dfrac{Av_{i-1}}{||Av_{i-1||}}$
    if $||Av_{i-1} - Av_i|| < \epsilon$ **then**
        Break.
    end if
end for
Compute $\lambda = \dfrac{v^\top Av}{v^\top v}$

Pros	Cons
• Efficient to find the dominant eigenvalue. Low memory requirements. Fast convergence for matrices with well separated dominant eigenvalue. • Easy to implement. Can be parallelized easily to make it faster. Remains same for all types of matrices.	• If there are multiple dominant eigenvalues, it may not converge to a unique eigenvector. • Doesn't find the other eigenvalues. • Convergence depends upon the choice of initial vector v_0.

2.3 Jacobi Method

This algorithm is a solid choice for calculating the eigenvalues of a symmetric matrix. It applies rotations to the matrix iteratively, leading to the eigenvalues and the eigenvectors. Its time complexity is $\mathcal{O}(n^3)$ per iteration.

Algorithm 3 Jacobi Method

Given Symmetric matrix A, tolerance ϵ, max iterations k, $a_{p,q}$ denotes $A[p][q]$

Initialize V as identity matrix of same order as A

repeat

  take $a_{p,q}$ = largest element in A

  if $a_{p,q} < \epsilon$ **then**

    Break (converged)

  end if

  Calculate rotation angle θ

    $\tan(2\theta) = \frac{2a_{p,q}}{a_{p,p} - a_{q,q}}$

  Calculate the rotation matrix P such that

    $p_{p,p} = \cos\theta,\ p_{q,q} = \cos\theta$

    $p_{p,q} = -\sin\theta,\ p_{q,p} = \sin\theta$

  Update A and V using P

    $A = P^\top AP$

    $V = VP$

until convergence or max iterations

Output:

    Diagonal elements of A are the eigenvalues

    Columns of V are the eigenvectors

Pros	Cons
• Simple, easy to implement and parallelizable.	• It may not always converge quickly. Convergence also depends upon the initial guess.
• works well for large system, where other methods might be computationally intensive	• even for symmetric matrices it may take a large amount of iterations.

2.4 Divide and Conquer Algorithm

This is another algorithm that is commonly used to compute the eigenvalues of symmetric matrices. The time complexity of this algorithm is nearly $\mathcal{O}(n^3)$. It breaks down the matrix into smaller matrices, and then calculates their eigenvalues separately thus finding them efficiently.

Algorithm 4 Divide and Conquer Algorithm

Given: Matrix A

DivideAndConquer(A):

  If matrix A is small enough:

    Return eigenvalues of A by calculating them.

  Else:

    Divide A into two smaller submatrices A_1 and A_2.

    eigenvalues1 = DivideAndConquer(A_1)

    eigenvalues2 = DivideAndConquer(A_2)

    Combine eigenvalues1 and eigenvalues2 to get the eigenvalues of A

    Return the combined eigenvalues

Pros	Cons
• It is very fast for symmetric matrices than other methods. • It's recursive nature allows it to be parallelized.	• This approach is more complex to implement than simpler methods. • This approach may not effectively handle the sparse matrices.

2.5 Folded Spectrum Method

The Folded Spectrum method is also an iterative technique which helps in finding a particular eigenvalue close to a specific target value. Its complexity can range from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$ depending on the implementation.

Algorithm 5 Jacobi Method

Given: Real, symmetric matrix A, target value λ_t, tolerance ϵ

Initialize a random vector v such that $||v|| = 1$

repeat

  Compute $w = (A - \lambda I)v$

  Let $v_{\text{new}} = w/||w||$

  If $|v_{\text{new}} - v| < \epsilon$, Converged, Break

  Update $v \leftarrow v_{\text{new}}$

until convergence

Calculate associate eigenvalue as $\lambda = v^\top Av$

Output: converged eigenvector v and eigenvalue λ

Pros	Cons
• Can find eigenvalues close to a required target value. • Works well for large, sparse matrices.	• May take time to converge if it requires many iterations. • Convergence depends upon the choice of initial vector.

3 QR Algorithm : A Suitable Choice

[2] QR Algorithm is the most popular method for finding the eigenvalues of almost all types of matrices. It works by repeatedly decomposing the given matrix into its QR factors (Q is an orthogonal matrix, and R is an upper triangular matrix). It then updates the original matrix as RQ, slowly bringing it closer to a form where the eigenvalues can be easily found.

The algorithm finally converges to a diagonal-type matrix whose diagonal elements are the eigenvalues of the original matrix. This makes the QR algorithm a solid choice for eigenvalue calculation, especially when working with large and dense matrices.

Its applicability for all types of matrices makes it a perfect and suitable choice for my eigenvalue calculation software. Its algorithm is easy to understand and implement. The memory requirement of this algorithm are also not high as for every iteration, we only carry forward the updated matrix. which makes it quite efficient on memory as well.

The complexity of QR Algorithm with Householder Transformation is $\mathcal{O}(n^3)$. Generation of each householder reflection matrix H takes operations proportional to n^2 and the process is repeated n times for one time full decomposition of the matrix, thus making its complexity as $\mathcal{O}(n^3)$ per iteration.

Algorithm 6 QR Algorithm

Given: Square Matrix A, tolerance ϵ, max iterations k

initialize $A_0 = A$
for $i = 1$ to k **do**
    Perform QR decomposition: $A_k = Q_k R_k$
    Update $A_{k+1} = R_k Q_k$
    If $|A_{k+1} - A_k| > \epsilon$, Break (Converged)
end for

Output: The diagonal elements of A_{k+1} as the eigenvalues.

3.1 Decomposition Methods

For performing the decomposition, there are several methods. Some of them are:

3.1.1 Gram-Schmidt Process

It is a common algorithm used for orthogonalizing a set of vectors in an inner product space. In QR decomposition, it transforms the columns of matrix A into an orthogonal set of vector (columns of Q), while maintaining a matrix R such that $A = QR$.

Steps:

1. For each column vector a_i of A, subtract the projections of a_i onto the previous orthogonal vectors to make it orthogonal to all the previous vectors.
2. Normalize each of the vector a_i to get the corresponding column of the matrix Q.
3. The matrix R contains the coefficients from the projections of the original vectors beofre normalization on the previously orthogonalised vectors.

It is conceptually simple and easy to implement. It works well for small matrices, but it might be numerically unstable for larger matrices. This makes it a less preferred choice for large matrices.

3.1.2 Householder Reflections

This is a more numerically stable method for QR decomposition. It uses reflection matrices to zero out the subdiagonal elements of the given matrix leading to an upper triangular matrix R. These Reflections are simultaneously used to build the orthogonal matrix Q. Thus effectively decomposing the given matrix in the QR form.

Steps:

1. Initialize $R = A$, and $Q = I$
2. Calculate householder matrix H for each column to zero out the elements below the diagonal

$$H = I - 2\frac{vv^\top}{v^\top v}$$

3. Apply each reflection matrix H to the matrix R to ultimately converge to the upper triangular form.
4. Also apply each reflection H to the matrix Q to form the orthogonal matrix.

It is numerically stable unlike the gram-schmidt process for large matrices. It can be easily applied for both dense and sparse matrices. A shortcoming of this method is that it is computationally more expensive than the gram-schmidt process. So it clearly becomes the best choice for my implementation of the QR Algorithm.

3.1.3 Givens Rotations

In this method, we zero out the elements below the diagonal by applying sequence of plane rotations, thus forming the upper triangular matrix R. Each rotation zeroes out one element in the matrix, and all those rotations are combined to from the orthogonal matrix Q.

Steps:

1. At each step, choose two elements say a_{ij} and a_{kj} from the subdiagonal elements of the matrix.
2. Construct a rotation matrix G which eliminates either of the two elements by rotating the two rows in the plane.
3. Apply the rotation matrix G to A.
4. Repeat this process till A converges to an upper triangular form R.
5. The product of all the rotation matrices G form the orthogonal matrix Q.

This method is numerically very stable. This method is only suitable for sparse matrices, as it only modifies two elements at a time. But for denser matrices, it may be computationally expensive as it would require a very large number of rotations.

3.2 Implementation

To implement my Eigenvalue calculating software, i am using C as my low level language for programming it. The code can be accessed at

```
codes/main.c
```

Here is an outline of my implementation.

1. **Matrix Struct:** I have implemented a struct consisting of two elements. One stores the size of the matrix, and the other is a double pointer, which allows me to effectively store the matrix.
2. **Complex Data types:** I have used the `complex.h` library for handling complex entries. Every where i have used `long double` for precision and accuracy.
3. **Matrix Handling functions:** I have also created some matrix handling functions:
   - `createMat(size)` : for creating a matrix and allocating memory to it.
   - `freeMat(matrix)` : to free the allocated memory to the matrix when not needed.
   - `scanMat(matrix, isComplex)` : to effectively take input from the user.
   - `printMat(matrix)` to print the matrix.
4. **Householder Method** I have implemented the householder method for decomposition of the matrix in a modular fashion. The functions are:
   - `normCalculate()` : This function calculates and returns the norm of a complex vector.
   - `householderVector()` : This function takes in the columns of A and outputs the corresponding householder vector.
   - `householderApplicatioon()` This is the main decomposing function which takes the householder vector and calculates the reflection matrix H and updates the Q and R matrices.
5. **Wrapper function for QR Algorithm** : I have implemented the `qrAlgorithm()` function which takes in input as A, decomposes it into QR form using the householder functions and updates A as $A \leftarrow RQ$ while effectively managing the memory.
6. **Eigenvalue Printing**: I have also implemented the `printEigenValues()` function. It takes the modified matrix A and prints the eigenvalues depending upon if there are complex eigenvalues or not.

3.3 Handling Complex Eigenvalues in the case of Real Entries:

[3] In the case of real matrices, eigenvalues are not always guaranteed to be real numbers. In such cases only printing the diagonal elements is incorrect. It is essential to handle such cases.

To address this issue is the specific reason for implementing the `printEigenvalues()` function. In such cases the QR algorithm transforms the original matrix into a Block Diagonalizable matrix. The `printEigenvalues()` function then goes through all such 2×2 blocks and prints their eigenvalues by calculating by direct method (which is quite simple for a 2×2 matrix. This way this function handles the case when there are real entries but complex eigenvalues.

4 Conclusion

In this assignment I explored various eigenvalue computation algorithms and selected the QR algorithm according to my needs to build my software. I implemented it in a modular fashion, with the numerically stable Householder Reflections for decomposition. Also, I added a specific function to handle the case when the eigenvalues were complex for real entries.

References

[1] 3Blue1Brown. Eigenvectors and eigenvalues | chapter 14, essence of linear algebra, 2016. visit.

[2] Martijn Anthonissen. Qr algorithm for computing eigenvalues, 2021. visit.

[3] Dan Margalit and Joseph Rabinoff. Complex eigenvalues. visit.

[4] Wikipedia contributors. Eigenvalue algorithm - iterative algorithms, 2024. visit.