Eigenvalue Computation

EE24BTECH11042-SRUJANA

1 WHAT ARE EIGEN VALUES

For a square matrix A there exits an eigenvalue λ such that for any non-zero vector v this equation will satisfy

$$Av = \lambda v$$

In general we compute it by following method

$$Av = \lambda v$$
$$Av - \lambda vI = 0$$
$$|A - \lambda I|v = 0$$
$$|A - \lambda I| = 0$$

By computing the determinent we can find the values of λ
But this method is no a good method to solve matrices of order greater than 3
For this we have to chose an algoritm

2 CHOSEN ALGORITHM

$*$ QR decomposition without hessenberg reduction

This method is used when the size of the matrix is less.

$*$ QR decomposition followed by hessenberg reduction

This methood is used when the size of the matrix is more.

3 Time Complexity

The time complexity of the QR algorithm is dominated by the QR decomposition, which takes $O(n^3)$ time.

4 Comparison of Algorithms

Algorithm	Order	Accuracy	Suitability
QR Algorithm	$O(n^5)$	Finds all eigenvalues	Small to medium matrices
Power Iteration	$O(n^2)$	Largest eigenvalue only	Large sparse matrices
Jacobi Method	$O(n^3)$	Symmetric matrices	Symmetric matrices
Lanczos	$O(n^2)$	Approximate eigenvalues	Large sparse matrices

TABLE I: Comparison of Eigenvalue Computation Algorithms

Even though the order for Power Iteration and Lanczos method are of order n^2 we wont get all the eigenvalues The QR algorithm is effective for computing all eigenvalues of dense matrices, but its time complexity can be prohibitive for large matrices. Power iteration is much faster for large matrices but only computes one eigenvalue at a time. Hence we use Hessenberg reduction for large matrices

5 HOW THIS WORKS ??

$*$ without hessenberg reduction

The QR algorithm is an iterative algorithm for finding the eigenvalues of the matrix . We get Q and R based on the formula A=QR.

$*$How to find Q and R

− Q is an orthogonal matrix and R is an upper triangular matrix

−vectors in Q are derived from the vectors of A

− let $q_1, q_2, q_3, q_4, ...$ are vectors in Q and $a_1, a_2, a_2, a_3, a_4, ...$ are vectors in A

− assume q_1 be unit vector in the direction of a_1

− as q_2 should be orthogonal with q_1 we can say that q_2 is the residue of the projection of a_2 on q_1.

− In the same way we can coonstruct the vectors $q_3, q_4, q_5, ...$

General Formula

$$q_i = v_i - \sum_{j=1}^{i-1} \mathrm{proj}_{q_j}(v_i)$$

where

$$\mathrm{proj}_{q_j}(v_i) = \frac{\langle v_i, q_j \rangle}{\langle q_j, q_j \rangle} q_j$$

− We can get R from the formula $Q^T A = R$

−By performing QR decompositions on the matrix and updating it using the formula $A' = RQ$, where Q is orthogonal and R is upper triangular. Repeat the process until you end up with a matrix which has entries (excluding diagonal elements) approximately equal to zero .

−The diagonal elements in the final matrix are the eigenvalues .

$*$ with hessenberg reduction

– This process is similar to the QR decomposition, but instead of directly applying QR, we first reduce the matrix A to a Hessenberg form H (H_{ij}=0 where i>j+1), where H is obtained through a similarity transformation:

– We can get B from the formula $1 - 2\frac{vv^T}{v^T v}$

The vector v is constructed from the column below the diagonal that you want to zero out. This is done by first extracting the subcolumn and computing the Householder vector v, which is designed to make the desired element of the matrix zero.

$$v = x + \text{sign}(x_1)||x||_2 \cdot e_1$$

where x is the subcolumn of the matrix

Preservation of eigenvalues

– eigenvalues A remains even though we transformed it into hessenberg matrix ,because B is an orthogonal matrix

$$\begin{aligned} &|B^T AB - \lambda I| \\ &= \frac{|AB - B\lambda I|}{|B|} \\ &= \frac{|A - \lambda I|}{|BB^T|} \\ &= |A - \lambda I| \end{aligned}$$

Hence the eigenvalues are not changing

6 PYTHON CODE

Listing 1: QR Algorithm

```python
def QR_decomposition(matrix):
    import math
    n = len(matrix)
    Q = [[0.0] * n for _ in range(n)]
    R = [[0.0] * n for _ in range(n)]
    for j in range(n):
        q = [matrix[i][j] for i in range(n)]
        for k in range(j):
            R[k][j] = sum(Q[i][k] * matrix[i][j] for i in range(n))
            for i in range(n):
                q[i] -= R[k][j] * Q[i][k]
        norm = math.sqrt(sum(q[i] ** 2 for i in range(n)))
        R[j][j] = norm
        for i in range(n):
            Q[i][j] = q[i] / norm
    return Q, R

```

```
def Hessenberg_reduction(matrix):
    import math
    n = len(matrix)
    H = [row[:] for row in matrix]
    for k in range(n - 2):
        x = [H[i][k] for i in range(k + 1, n)]
        norm_x = math.sqrt(sum(val ** 2 for val in x))
        if norm_x == 0:
            continue
        alpha = -norm_x if x[0] > 0 else norm_x
        v = [x[0] + alpha] + x[1:]
        norm_v = math.sqrt(sum(val ** 2 for val in v))
        v = [val / norm_v for val in v]
        for i in range(k + 1, n):
            for j in range(k, n):
                H[i][j] -= 2 * v[i - (k + 1)] * sum(v[m - (k + 1)] * H[m][j]
                     for m in range(k + 1, n))
        for i in range(n):
            for j in range(k + 1, n):
                H[i][j] -= 2 * v[j - (k + 1)] * sum(H[i][m] * v[m - (k + 1)]
                     for m in range(k + 1, n))
    return H

def QR_algorithm(matrix, max_iterations=100, tolerance=1e-10):
    n = len(matrix)
    A = [row[:] for row in matrix]
    for _ in range(max_iterations):
        Q, R = QR_decomposition(A)
        A = [[sum(R[i][k] * Q[k][j] for k in range(n)) for j in range(n)] for
             i in range(n)]
        off_diagonal = sum(A[i][j] ** 2 for i in range(n) for j in range(i))
        if off_diagonal < tolerance:
            break
    return [A[i][i] for i in range(n)]

def main():
    print("Enter the size of the matrix (n x n):")
    n = int(input())
    if n < 1:
        print("Matrix size must be >=1.")
        return
    print(f"Enter the {n} x {n} matrix row by row, space separated:")
    matrix = []
    for _ in range(n):
        row = list(map(float, input().split()))
        if len(row) != n:
            print("Invalid input. Each row must have exactly n elements.")
            return
        matrix.append(row)

    if n < 50:
        eigenvalues = QR_algorithm(matrix)
    else:
        hessenberg_matrix = essenberg_reduction(matrix)
        eigenvalues = QR_algorithm(hessenberg_matrix)

    print("Eigenvalues␣are:")
    for eig in eigenvalues:
        print(eig)

main()
```

7 EXAMPLE

INPUT

Enter the size of the matrix ($n \times n$):

$$n = 3$$

Enter the 3×3 matrix row by row, space separated:

$$A = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Eigenvalues:

$$\lambda_1 = 7.287992138957324, \quad \lambda_2 = 2.1330744753515734, \quad \lambda_3 = 0.5789333856911023$$