Software assignment

Srihaas Gunda - EE24BTECH11026

Introduction

Eigenvalues are used in a variety of fields like engineering and machine learning.For symmetric matrices, the 'Jacobi method' is a well-known algorithm that can compute eigenvalues by diagonalizing the matrix through a series of orthogonal transformations.

1 Jacobi Method Overview

The Jacobi method diagonalizes a symmetric matrix A by successively applying orthogonal rotations. In each iteration, the largest off-diagonal element is targeted, and a rotation is applied to eliminate it. The goal is to transform the matrix into a diagonal form, where the diagonal elements will be the eigenvalues of the original matrix.

Given a symmetric matrix A, the Jacobi method follows these key steps:

1) Find the largest off-diagonal element A_{pq}, where $p \neq q$.
2) Compute the rotation angle θ using the formula:

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2A_{pq}}{A_{pp} - A_{qq}}\right) \tag{1}$$

   where A_{pp} and A_{qq} are the diagonal elements of the matrix.
3) Construct the rotation matrix $P(\theta)$, which is a matrix that performs the orthogonal rotation. The matrix $P(\theta)$ represents a rotation in the $p-q$ plane and has the following structure:

$$P(\theta) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \cos(\theta) & \sin(\theta) & \cdots & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{2}$$

4) Update the matrix A: The matrix A' is updated by applying the rotation matrix P. The update is done in two steps:

$$A' = P^T A P \tag{3}$$

   After this update, the off-diagonal element A_{pq} is reduced to zero, and the matrix A is closer to being diagonal.
5) Repeat the process until the matrix becomes sufficiently diagonal, i.e., when the off-diagonal elements are smaller than a given tolerance.

This matrix is orthogonal, meaning $P^T P = I$, where I is the identity matrix. Each such rotation reduces the magnitude of the off-diagonal elements, while preserving the diagonal elements.

2 Convergence of the Jacobi Method

One of the key properties of the Jacobi method is its convergence. The method converges to the eigenvalues of the matrix as long as the matrix is symmetric. The main factors that affect the convergence rate are:

1) The magnitude of the off-diagonal elements.The algorithm works by zeroing out the largest off-diagonal elements. As these elements become smaller, the matrix gets closer to diagonal form.
2) The number of iterations: The method typically requires more iterations for larger matrices or matrices with smaller off-diagonal elements. The method converges when all off-diagonal elements fall below a pre-specified tolerance.
3) The matrix's condition number: A matrix with a large condition number (i.e., a matrix that is ill-conditioned) might require more iterations for convergence.

In practice, the Jacobi method converges relatively slowly, especially for large matrices. It is generally more efficient for small to medium-sized matrices or for problems where other more sophisticated methods (e.g., QR algorithm) are not applicable.

Code

The following code does just that

```
#include<stdio.h>
#include<math.h>

void findpq(int n, double a[n][n], int *p, int *q) {
   *p=-1;
   *q=-1;
   double maxvalue = 0.0;
   for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
         if (fabs(a[i][j]) > maxvalue) {
             maxvalue = fabs(a[i][j]);
             *p = i;
             *q = j;
         }
      }
   }
}

void jacobi(int n, double a[n][n], double P[n][n], int p, int q) {
   double t;
   double temp1[n][n], temp2[n][n];
   t = 0.5 * atan((2 * a[p][q]) / (a[p][p] - a[q][q]));
   double c, s;
   c = cos(t);
   s = sin(t);
```

```
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         if (i == j) {
            P[i][j] = 1.0;
         } else {
            P[i][j] = 0.0;
         }
      }
   }
   P[p][p] = c;
   P[q][q] = c;
   P[p][q] = -s;
   P[q][p] = s;

   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         temp1[i][j] = 0.0;
         for (int k = 0; k < n; k++) {
            temp1[i][j] += P[k][i] * a[k][j];
         }
      }
   }

   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         temp2[i][j] = 0.0;
         for (int k = 0; k < n; k++) {
            temp2[i][j] += temp1[i][k] * P[k][j];
         }
      }
   }

   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         a[i][j] = temp2[i][j];
      }
   }
}

int main() {
   int n;
   double tolerance = 0.00001;
   int maxpass = 100;
   int pass;
   printf("Enter the order of the matrix:\n");
```

```
    scanf("%d", &n);
    double a[n][n], P[n][n];
    printf("Enter the elements of the symmetric matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%lf", &a[i][j]);
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i][j] != a[j][i]) {
                printf("Enter a proper symmetric matrix\n");
                return 0;
            }
        }
    }

    for (pass = 0; pass <= maxpass; pass++) {
        int p = 0, q = 0;
        findpq(n, a, &p, &q);
        if (fabs(a[p][q]) < tolerance) {
            break;
        }
        jacobi(n, a, P, p, q);
    }

    printf("Eigen values of a are:\n");
    for (int i = 0; i < n; i++) {
        printf("%.5lf\n", a[i][i]);
    }

    return 0;
}
```

3 Computational Complexity

The computational complexity of the Jacobi method can be analyzed by looking at the operations performed in each iteration. Each iteration involves finding the largest off-diagonal element and applying the rotation matrix P, which requires:

1) Finding the largest off-diagonal element: This involves scanning all $\frac{n(n-1)}{2}$ off-diagonal elements, which takes $O(n^2)$ time.
2) Constructing the rotation matrix and applying it: Constructing the matrix P and performing matrix multiplications take $O(n^3)$ operations.

Therefore, the overall time complexity per iteration is $O(n^3)$. If the method requires k iterations to converge, the total time complexity is approximately $O(kn^3)$.

The number of iterations k depends on the size and condition of the matrix, and can vary.

4 Advantages and Disadvantages

The Jacobi method has several advantages:

1) It guarantees real eigenvalues for symmetric matrices.
2) It works well for small to medium-sized matrices and in scenarios where other methods might not be applicable.

However, there are also disadvantages:

1) It is relatively slow for large matrices.
2) It requires $O(n^3)$ operations per iteration, making it inefficient for large-scale problems.
3) The convergence rate is slow compared to other algorithms such as the QR algorithm.

5 Comparison of Algorithms

Algorithm	**Time Complexity**	**Suitability**
Jacobi	$O(n^3)$	Symmetric matrices
QR Decomposition	$O(n^3)$	General matrices
Power Iteration	$O(kn^2)$	Sparse matrices
Lanczos	$O(kn^2)$	Large, sparse matrices

Conclusion

The Jacobi method is a reliable algorithm for computing eigenvalues and eigenvectors of symmetric matrices, particularly when high precision is required. However, for larger or sparse matrices, alternative methods like Lanczos or QR decomposition may be more suitable.