Computation of eigen values

EE24BTECH11056 - S.Kavya Anvitha

I. What are eigen values?

1) If there is a matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. this matrix is transforming $\hat{i}$ and $\hat{j}$ with $\hat{i}$ to (a,c) and $\hat{j}$ to (b,d)
2) most vectors will be knocked out of their span during transformation.but some special vectors do remain on their own span meaning the effect that the matrix has on such a vector is just to stretch it.
3) these vectors are called eigen vectors
4) The factor by which the eigen vector is stretched during the transformation is called eigen value.

II. Algorithm

QR Algorithm with Hessenberg Reduction: This method is used to find eigen values of a matrix

1) Reduce the Matrix to Hessenberg Form
   - reduce the matrix to upper Hessenberg form.
   - this step simplifies the matrix structure while retaining its eigenvalues,making the QR steps more efficient.
2) Iterate with QR steps
   - Apply the QR algorithm.
   - Continue iterating until the off-diagonal elements are close to zero,indicating convergence.
3) Extract EigenValues
   - Once the matrix is nearly diagonal, the eigenvalues are approximated by the diagonal elements.

III. Time Complexity

1) for constructing householder vector v:$O(n-k)$
   for applying transformation to update rows and columns of A:$O((n-k)^2)$
   Summing over all columns:$\Sigma(n-k)^2$ approx $\frac{n^3}{3}$
2) QR decomposition for a Hessenberg matrix takes $O(n^2)$ so per iteration $O(n^2)$.since there are n iterations total cost is $O(n^2) \times O(n) = O(n^3)$
3) the overall time complexity is:

$$T_{\text{Total}} = T_{\text{Hessenberg}} + T_{QR} = O(n^3) + O(n^3) = O(n^3)$$

IV. OTHER INSIGHTS

1) This method is suitable for symmetric, non-symmetric and large matrices also.
   - Handles large and small matrices by reducing computation through matrix reduction.
2) By reducing the matrix to Hessenberg form, fewer elements need to be considered for each QR iteration, leading to faster convergence.
   - The number of operations required for QR decomposition and subsequent iterations is smaller compared to directly applying the QR algorithm to a general matrix.
3) The Hessenberg reduction reduces the number of elements in the matrix, which helps in reducing memory requirements compared to methods that do not utilize Hessenberg form, making it more memory-efficient for large matrices.

V. COMPARISON WITH OTHER ALGORITHMS

1) This method have fast convergence rate while compared to other methods like LU Decomposition Power Iteration(for eigen values with close magnitude),Jacobi method... etc.
2) Efficient for finding all eigenvalues of dense matrices.Where as power iteration is not suitable for finding all eigenvalues.
3) This method works for both symmetric and non-symmetric matrices.While Jacobi method is Limited to symmetric matrices.
4) Unlike other methods like power iteration, the QR algorithm can handle matrices that may be defective (not diagonalizable).
5) Time complexity of this method is $O(n^3)$,Where as Time complexity of Jacobi method is $O(n^4)$

VI. WORKING WITH ALGORITHM

Hessenberg reduction:

1) We need to reduce the matrix to Hessenberg form which helps QR algorithm to perform easily
2) To do this we need to use householder reflections.
   - Using householder reflections we need to zero out elements below the subdiagonal of the matrix.
   - To do this we construct a vector v and a reflection matrix p.
     consider a matrix A,

$$P = I - \frac{2vv^T}{v^T v}$$

$$v = x + sign(x)\,||x||\,e_1$$

     where x is the 1st column below A[1,1].and a_2, a_3, a_4, a_5 are elements of first column below A[1,1]
3) after finding v we need to normalise it

4) now, $PA = A - 2v(v^T A)$
   $A_{new} = A - 2v(v^T A)$.
5) similarly we need to apply this for all rows and columns.
6) these together implies $A = PAP_T$ without explicitly forming P_T.Finally the transformation to Hessenberg form preserves eigenvalues because the Householder reflections are orthogonal transformations.

QR Algorithm:

1) Start with a square matrix $A \in \mathbb{R}^{n \times n}$
2) Set $A_0 = A$
3) For each iteration k, compute the QR decomposition of the matrix A_k:

$$A_k = Q_k R_k$$

   where:

   - Q_k is Orthogonal meaning $Q_k^T Q_k = I$
   - R_k is an upper triangular matrix

   QR decomposition can be done using methods like Gram-Schmidt
4) After performing the QR decomposition, update the matrix as:

$$A_{k+1} = R_k Q_k$$

   This step essentially reorders the matrix, and the process is repeated for subsequent iterations.

- Repeat the QR decomposition and matrix update steps iteratively until the matrix A_k converges to a diagonal matrix, i.e., the off-diagonal elements become very small (close to machine precision).
- The diagonal elements of the resulting matrix A_k after sufficient iterations are the eigenvalues of the original matrix A.

VII. C CODE

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

#define MAX_ITER 1000//here we are defining global variables for further uses
#define EPS 1e-10

void hessenbergReduction(int n, double A[n][n]);
void QR_Decomposition(int n, double A[n][n], double Q[n][n], double R[n][n]);//these
    are the function prototypes
void matrixMultiply(int n, double A[n][n], double B[n][n], double C[n][n]);
double Norm(int n, double A[n][n]);
void finding_eigenvalues(int n, double A[n][n], double eigenvalues[n]);

int main() {
    int n;
    printf("Enter the size of the matrix: ");
    scanf("%d", &n);

```

```c
    double A[n][n];
    printf("Enter the matrix elements row by row:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%lf", &A[i][j]);
        }
    }

    printf("Performing Hessenberg reduction...\n");//we are calling all functions required for finding eigenvalues
    hessenbergReduction(n, A);

    double eigenvalues[n];
    finding_eigenvalues(n, A, eigenvalues);
    printf("Eigenvalues:\n");
    for (int i = 0; i < n; i++) {
        printf("%lf\n", eigenvalues[i]);
    }

    return 0;
}

void hessenbergReduction(int n, double A[n][n]) {
    for (int p = 0; p < n - 2; p++) {
        double norm = 0.0;

        for (int i = p + 1; i < n; i++) {
            norm += A[i][p] * A[i][p];
        }
        norm = sqrt(norm);

        double alpha = (A[p + 1][p] > 0) ? -norm : norm;

        double v[n];
        for (int i = 0; i < n; i++) v[i] = 0.0;//we are defining a householder vector v

        v[p + 1] = A[p + 1][p] - alpha;
        for (int i = p + 2; i < n; i++) {
            v[i] = A[i][p];
        }

        double v_norm = 0.0;
        for (int i = p + 1; i < n; i++) {//here we are finding norm of v to normalise the vector v
            v_norm += v[i] * v[i];
        }
        v_norm = sqrt(v_norm);

        if (fabs(v_norm) > EPS) {
            for (int i = p + 1; i < n; i++) {//here we are comparing with very small value EPS to avoid division by a very small number, which could cause numerical instability
                v[i] /= v_norm;
            }
        }

        for (int j = p; j < n; j++) {
            double dot = 0.0;
```

```
            for (int i = p + 1; i < n; i++) {
                dot += v[i] * A[i][j];
            }
            for (int i = p + 1; i < n; i++) {//by using householder reflections we are turning the matrix into hessenberg form
                A[i][j] -= 2 * v[i] * dot;
            }
        }

        for (int i = 0; i < n; i++) {
            double dot = 0.0;
            for (int j = p + 1; j < n; j++) {
                dot += v[j] * A[i][j];
            }
            for (int j = p + 1; j < n; j++) {
                A[i][j] -= 2 * v[j] * dot;
            }
        }
    }
}

void QR_Decomposition(int n, double A[n][n], double Q[n][n], double R[n][n]) {
    double V[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            V[i][j] = A[i][j];
            Q[i][j] = 0.0;
            R[i][j] = 0.0;
        }
    }

    for (int k = 0; k < n; k++) {
        double norm = 0.0;
        for (int i = 0; i < n; i++) {
            norm += V[i][k] * V[i][k];
        }
        norm = sqrt(norm);
        R[k][k] = norm;

        for (int i = 0; i < n; i++) {//here we are using Gram-Schmidt process to perform QR decomposition
            Q[i][k] = V[i][k] / norm;
        }

        for (int j = k + 1; j < n; j++) {
            double dot = 0.0;
            for (int i = 0; i < n; i++) {
                dot += Q[i][k] * V[i][j];
            }
            R[k][j] = dot;
            for (int i = 0; i < n; i++) {
                V[i][j] -= dot * Q[i][k];
            }
        }
    }
}

void matrixMultiply(int n, double A[n][n], double B[n][n], double C[n][n]) {
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
            C[i][j] = 0.0;
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];//here we are finding multiplication of two matrices
            }
        }
    }
}

double Norm(int n, double A[n][n]) {
    double sum = 0.0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += A[i][j] * A[i][j];//we are finding norm of a matrix
        }
    }
    return sqrt(sum);
}

void finding_eigenvalues(int n, double A[n][n], double eigenvalues[n]) {
    double Q[n][n], R[n][n], temp[n][n];
    int iter = 0;

    while (iter < MAX_ITER) {

        QR_Decomposition(n, A, Q, R);//MAX_ITER is a predefined maximum number of iterations to avoid infinite loops if convergence fails
        matrixMultiply(n, R, Q, temp);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                A[i][j] = temp[i][j];
            }
        }

        int converged = 1;
        for (int i = 1; i < n; i++) {
            if (fabs(A[i][i - 1]) > EPS) {
                converged = 0;
                break;
            }
        }
        if (converged) break;  //check if the subdiagonal elements are close to zero. if they are the algorithm has converged.

        iter++;
    }

    for (int i = 0; i < n; i++) {
        eigenvalues[i] = A[i][i];
    }
}
```