Software Assignment: Finding Eigenvalues

EE24BTECH11007 - Arnav Makarand Yadnopavit

I. Introduction

There are various algorithms present to approximate eigenvalues of matrices. The best algorithms considering general $n \times n$ matrix are as following.

Algorithm	Advantages	Disadvantages
QR Algorithm with Householder technique and Wilkinson Shift	• Fast convergence for symmetric/Hermitian matrices. • Wilkinson shift accelerates convergence. • Householder transformations ensure numerical stability and efficiency.	• Computationally intensive for large matrices due to repeated QR decompositions.
Divide-and-Conquer	• Highly efficient for symmetric matrices. • Scales well with matrix size for parallel computing.	• Less numerically stable for small eigenvalues. • Preprocessing required for non-symmetric matrices.
Jacobi Algorithm	• Simple to implement. • Numerically stable for symmetric matrices.	• Slower convergence compared to QR with shifts. • Inefficient for large matrices.
Bisection Method	• Reliable for computing eigenvalues in a specific interval. • Works well with tridiagonal matrices.	• Only provides eigenvalues, not eigenvectors. • Convergence can be slow without good initial bounds.
Standard QR Algorithm (without Shift)	• Robust for small matrices. • Simple structure and implementation.	• Slower convergence compared to the shifted version. • Requires significantly more iterations.

Hence QR Algorithm with Householder technique and Wilkinson Shift stands out for being generally better for any general $n \times n$ matrix due to its fast convergence rate, Numerical stability.

II. QR Algorithm with Householder technique and Wilkinson Shift

A. QR Decomposition

QR decomposition factors a given matrix A into:

$$A = QR,$$

where Q is an orthogonal matrix ($Q^\top Q = I$), and R is an upper triangular matrix.

B. Householder Transformations

Householder transformations are used to zero out elements below the diagonal of a matrix column. Given a vector v, the Householder matrix is:

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

The way it works is:
Initialize Q as Identity matrix. Let $\mathbf{x}$ be the the first column of A, and $\alpha = ||\mathbf{x}||$.

$$\mathbf{u} = \mathbf{x} - \alpha\mathbf{e_1}$$
$$\mathbf{v} = \frac{\mathbf{u}}{||\mathbf{u}||}$$
$$Q = I - 2vv^H$$

By this we obtain Q_1 such that:

$$Q_1 A = \begin{pmatrix} \alpha_1 & * & \dots & * \\ 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{pmatrix}$$

This can be repeated for A' (obtained from $Q_1 A$ by deleting the first row and first column), resulting in a Householder matrix Q'_2. Note that $Q\prime_2$ is smaller than Q_1. Since we want it really to operate on $Q_1 A$ instead of A' we need to expand it to the upper left, filling in a 1, or in general:

$$Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & Q'_k \end{pmatrix}$$

After $n - 1$ iterations of this process.
$R = Q_{n-1} \dots Q_2 Q_1 A$
$Q^\top = Q_{n-1} \dots Q_2 Q_1$
$Q = Q_1 Q_2 \dots Q_{n-1}$

C. QR Algorithm for Eigenvalues

The QR algorithm iteratively applies QR decomposition to a shifted matrix $A - \mu I$ and reconstructs it as:

$$A = RQ + \mu I$$

converging to an upper triangular form with eigenvalues on the diagonal.
where μ can be calculated by:

$$\mu = a_m - \frac{\delta}{|\delta|} \frac{b_{m-1}^2}{|\delta| + \sqrt{\delta^2 + b_{m-1}^2}}$$

where B is the lower rightmost 2×2 matrix of A, B= $\begin{pmatrix} a_{m-1} & b'_{m-1} \\ b'_{m-1} & a_m \end{pmatrix}$
$\delta = \frac{a_{m-1} - a_m}{2}$
If $\delta = 0$, then $\mu = a_m - b_{m-1}$

D. Complex Eigenvalues

In case a matrix has complex eigenvalues a hessenberg matrix (2×2) will be formed along the diagonal of the triangularised matrix A such that:

$$A = \begin{pmatrix} \lambda_1 & \dots & \dots & \dots \\ 0 & a & b & \dots \\ \vdots & c & d & \dots \\ 0 & \dots & \dots & \lambda_n \end{pmatrix}$$

then:

$$\lambda_2 = \frac{a + d + \sqrt{(a+d)^2 - 4\,(ad - bc)}}{2}$$
$$\lambda_3 = \frac{a + d - \sqrt{(a+d)^2 - 4\,(ad - bc)}}{2}$$

III. Code

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <stdlib.h>
//TO DO LIST
//QR DECOMPOSITION DONE
//QR Algo DONE
//MATRIX SUB DONE
//MATRIX SCALARMULT DONE
//MATRIX MULT DONE
//WILKINSON SHIFT DONE
//VECTOR NORM DONE
#define MAX_ITER 10000
#define ORDER 3
typedef struct matrix{
double complex mat[ORDER][ORDER];
}matrix;
typedef struct QR{
matrix Q;
matrix R;
}QR;
double complex VectorInnerProduct(double complex* vector1,double complex* vector2){
double complex ip=0;
for (int i=0;i<ORDER;i++){
                ip+=vector1[i]*conj(vector2[i]);
        }
return ip;
}
double complex VectorNorm(double complex* vector){
return csqrt(VectorInnerProduct(vector,vector));
}
matrix Identity(){
```

```
matrix Id;

for (int i=0;i<ORDER;i++){
for(int j=0;j<ORDER;j++){
if (i==j){
Id.mat[i][j]=1;
}
else{
Id.mat[i][j]=0;
}
}
}
return Id;
}

matrix MatScalMult(matrix mat,double complex scal){
matrix result;
for (int i=0;i<ORDER;i++){
for (int j=0;j<ORDER;j++){
result.mat[i][j]=mat.mat[i][j]*scal;
}
}
return result;
}
matrix MatSub(matrix mat1,matrix mat2){
matrix result;

    for (int i=0;i<ORDER;i++){
            for (int j=0;j<ORDER;j++){
                result.mat[i][j]=mat1.mat[i][j]-mat2.mat[i][j];
            }
    }
        return result;
}
matrix MatMult(matrix mat1,matrix mat2) {
    matrix result;
    for (int i=0;i<ORDER;i++){
        for (int j=0;j<ORDER;j++){
            result.mat[i][j]=0;
        }
    }
    for (int i=0;i<ORDER;i++){
        for (int j=0;j<ORDER;j++){
            for (int k=0;k<ORDER;k++){
                result.mat[i][j]+=mat1.mat[i][k]*mat2.mat[k][j];
            }
        }
    }

    return result;
```

```
}
matrix trans(matrix mat){
    matrix result;
    for (int i=0;i<ORDER;i++){
        for (int j=0;j<ORDER;j++){
        result.mat[i][j]=conj(mat.mat[j][i]);
        }
    }
    return result;
}
double complex WilkinsonShift(matrix A){
    double complex delta = (A.mat[ORDER-2][ORDER-2]-A.mat[ORDER-1][ORDER-1])/2;
    if (cabs(delta)==0 && cabs(A.mat[ORDER-1][ORDER-2])==0){
        return A.mat[ORDER-1][ORDER-1]+1;
    }
    else if (cabs(delta)==0){
        return A.mat[ORDER-1][ORDER-1]-A.mat[ORDER-1][ORDER-2]+1;
    }
    else if(cabs(A.mat[ORDER-1][ORDER-2])==0){
        return A.mat[ORDER-1][ORDER-1]+1;
    }
    else{
        return A.mat[ORDER-1][ORDER-1]-((delta/cabs(delta))*(A.mat[ORDER-1][ORDER-2]*A.mat
    }
}
int tolcheck(matrix A){
    int t=0;
    double complex tol=1e-12;
    for (int i=1;i<ORDER;i++){
        for (int j=0;j<i;j++){
            if (cabs(A.mat[i][j])>cabs(tol)){
                t++;
                break;
            }
        }
    }
    if (t>0){
        return 1;
    }
    else{return 0;}
}

QR QRDecomposition(matrix A){
    QR result;
    matrix  H;
    result.Q=Identity();
    result.R=A;

    for (int k=0;k<ORDER-1;k++){
        double complex v[ORDER-k];
```

```
        double complex beta;
        for (int i=k;i<ORDER;i++){
            v[i-k]=result.R.mat[i][k];
        }
        double complex alpha;
        //printf("%lf %lf\n",creal(v[0]),creal(v[1]));
        if (creal(v[0])==0){
        alpha=VectorNorm(v);
        }
        else{
        alpha=(result.R.mat[k][k]/cabs(result.R.mat[k][k]))*VectorNorm(v);
        }
        v[0]+=alpha;
        double complex vc=VectorNorm(v);
        if (vc!=0){
        for (int i=0;i<ORDER-k;i++){
            v[i]/=vc;
            }
        }
        double complex qprime[ORDER-k][ORDER-k];
        for(int i=0;i<ORDER-k;i++){
            for(int j=0;j<ORDER-k;j++){
                if (i==j){
                    qprime[j][i]=1-(2*conj(v[i])*v[j]);
                }
                else{
                    qprime[j][i]=-(2*conj(v[i])*v[j]);
                }
            }
        }


        for(int i=0;i<ORDER;i++){
            for (int j=0;j<ORDER;j++){
                if (i==j && i<k){H.mat[i][j]=1;}
                else if(i>=k&&j>=k){
                    H.mat[i][j]=qprime[i-k][j-k];
                }
                else{H.mat[i][j]=0;}
            }
        }
        result.R=MatMult(H,result.R);


        result.Q=MatMult(result.Q,trans(H));
    }


    return result;
}
```

```
double complex* QRAlgorithm(matrix A){
    double complex* eigenv = (double complex*)malloc(ORDER * sizeof(double complex));
    double complex shift;

    for (int  i=0; i< MAX_ITER; i++){
        if (tolcheck(A)==0){
            break;
        }
         shift = WilkinsonShift(A);
         matrix identity = Identity();
         A = MatSub(A, MatScalMult(identity, shift));
         QR qr = QRDecomposition(A);
         A = MatMult(qr.R,qr.Q);
        A=MatSub(MatMult(qr.R, qr.Q),MatScalMult(identity, -shift));
    }

    int i;
    for (i=0;i<ORDER-1;i++){
        if (cabs(A.mat[i+1][i])>1e-5){
            double complex eig1,eig2,a=A.mat[i][i],b=A.mat[i][i+1],c=A.mat[i+1][i],d=A.mat
            eigenv[i]=(a+d+csqrt((a+d)*(a+d)-4*(a*d-b*c)))/2;
            eigenv[i+1]=(a+d-csqrt((a+d)*(a+d)-4*(a*d-b*c)))/2;
            ++i;

        }
        else{
        eigenv[i]=A.mat[i][i];
        }
    }
    if (i==ORDER-1){
    eigenv[ORDER-1]=A.mat[ORDER-1][ORDER-1];}
    return eigenv;
}

int main(){
double complex A[ORDER][ORDER] ={{1,2,5},
    {4,5,8},
    {2,4,6}};

    matrix Mat;

    for (int i=0;i<ORDER;i++){
        for (int j=0;j<ORDER;j++){
            Mat.mat[i][j]=A[i][j];
        }
    }

double complex* eigen=(double complex*)malloc(ORDER*sizeof(double complex));
    eigen=QRAlgorithm(Mat);
```

```
for (int i=0;i<ORDER;i++){
printf("(%lf + %lfi)\n",creal(eigen[i]),cimag(eigen[i]));
}

return 0;
}
```

IV. Properties of QR Algorithm with Householder technique and Wilkinson Shift

- Time complexity: $O\left(n^3\right)$
- Wilkinson shift quadratically converges the approx eigenvalues i.e convergence rate is quadratic:

$$\epsilon_k = \left|\lambda_i - \lambda_{i,k}\right|$$

$$\epsilon_{k+1} = C\epsilon_k^2$$

V. Conclusion

The QR decomposition and eigenvalue computation were successfully implemented using the Householder method. The results matched the expected theoretical outcomes.

VI. References

- Trefethen, L.N. and Bau, D. (1997) Numerical Linear Algebra. SIAM, Philadelphia.
- Strang, Gilbert, Linear Algebra and Its Applications. New York, Academic Press, 1976.
- Online resources: https://www.wikipedia.org

Thank You