数值分析第二次大作业

张晋

学号：15091060

最后更新于：May 20, 2017

目录 | 0

题目 1

试求矩阵 $\boldsymbol{A}=\left[a_{ij}\right]_{10\times10}$ 的全部特征值，并对其中的每一个特征值求相应的特征向量，已知：

$$a_{ij}=\begin{cases}\sin(0.5i+0.2j) & i\neq j\\ 1.52\cos(i+1.2j) & i=j\end{cases}\qquad (i,j=1,2,\cdots,10)$$

说明：

1. 用双步位移 QR 法求矩阵特征值时，要求迭代精度水平为 $\epsilon=10^{-12}$。
2. 打印以下内容：
   (a) 全部源程序;
   (b) 矩阵 $\boldsymbol{A}$ 经过拟上三角化后所得的拟上三角阵 $\boldsymbol{A}^{(n-1)}$;
   (c) 对矩阵 $\boldsymbol{A}^{(n-1)}$ 使用双步位移 QR 法迭代结束后所得的矩阵 $\boldsymbol{Q}$ 和 $\boldsymbol{R}$，并把 $R\times Q$ 打印，看是不是拟上三角阵;
   (d) 矩阵 $\boldsymbol{A}$ 的全部特征值 $\lambda_i=(R_i,I_i)\qquad(i=1,2,\cdots,10)$;
   (e) 矩阵 $\boldsymbol{A}$ 相对应于实特征值的特征向量。
3. 采用 e 型输出实型数，并且至少显示 12 位有效数字。

算法设计方案 | 2

2.1 Householder 变换

Theorem 2.1.1. 我们称矩阵

$$H = I - 2vv^*, \qquad v \in \mathbb{C}^n \quad s.t. \quad ||v||_2 = \sqrt{v^*v} = 1, \tag{2.1}$$

为 Householder **矩阵** (或 Householder 变换)，向量 v 称为 Householder **向量**. 我们通常将矩阵记为 $H(v)$。

Householder 变换又称为反射变换或镜像变换，有明显的几何意义。在 $\mathbb{R}^3$ 中，给定一个向量 α，令 β 表示 α 关于平面 π（以 ω 为法向量）的反射变换所得像，如图所示，

记

$$\omega = \frac{\alpha - \beta}{|\alpha - \beta|} \in \mathbb{R}^3$$

$$H(\omega) = I - 2\omega\omega^T$$

则 $H(\omega)\alpha = \beta$

即:该变换将向量 α 变成了以 β 为法向量的平面的对称向量 β。因此,Householder 变换也称为反射变换.

下面是关于 Householder 矩阵的几个基本性质.

Theorem 2.1.2. 设 $H \in \mathbb{C}^{n\times n}$ 是一个 *Householder* 矩阵, 则

(1) $H^* = H$, 即 H 是 Hermite 的;

(2) $H^*H = I$, 即 H 是酉矩阵;

(3) $H^2 = I$, 所以 $H^{-1} = H$;

(4) $det(H) = -1$;

(5) H 有两个互异的特征值:$\lambda = 1, \lambda = -1$, 其中 $\lambda = 1$ 的代数重数为 $n-1$.

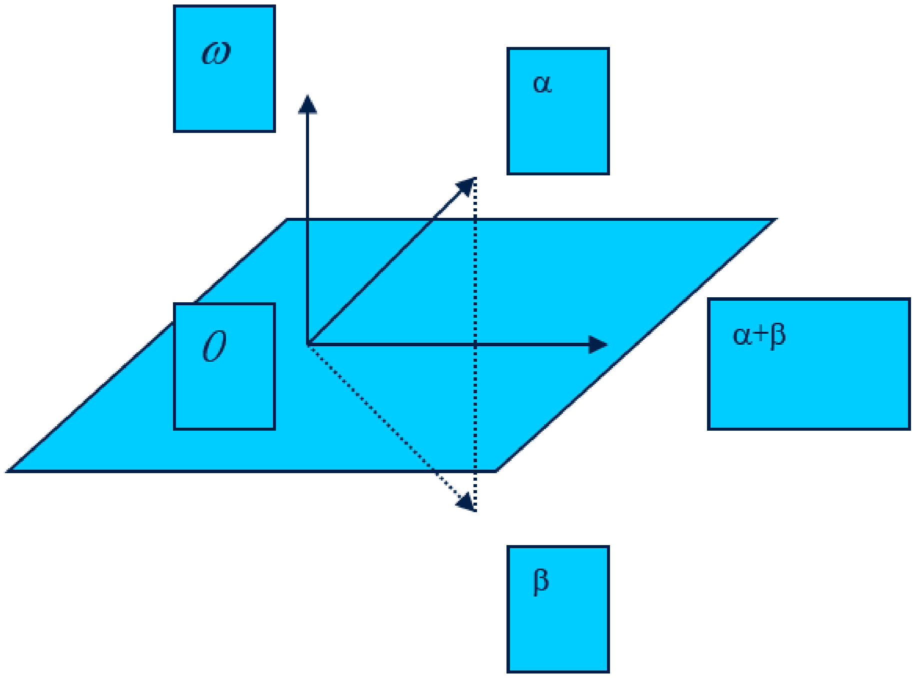

图 2.1: Householder

2.2 QR 分解

QR 分解也称为正交三角分解

QR 分解将一个矩阵分解一个正交矩阵 (酉矩阵) 和一个三角矩阵的乘积. QR 分解被广泛应用于线性最小二乘问题的求解和矩阵特征值的计算

Algorithm 1 QR Iteration

```
1: Set A_1 = A and k = 1
2: while not convergence do
3:     A_k = Q_k R_k
4:     Compute A_{k+1} = R_k Q_k
5:     k=k+1
6: end while
```

在 QR 迭代算法中, 我们有

$$A_{k+1} = R_k Q_k = (Q_k^T Q_k) R_k Q_k = Q_k^T (Q_k R_k) Q_k = Q_k^T A_k Q_k$$

由这个递推关系可得

$$\begin{aligned} A_{k+1} = Q_k^T A_k Q_k &= Q_k^T Q_{k-1}^T A_{k-1} Q_{k-1} Q_k \\ &= \cdots \\ &= Q_k^T Q_{k-1}^T \cdots Q_1^T A Q_1 \cdots Q_{k-1} Q_k. \end{aligned}$$

记 $\boxed{\tilde{Q}_k = Q_1 \cdots Q_{k-1} Q_k}$, 则

$$A_{k+1} = \tilde{Q}_k^T A \tilde{Q}_k$$

即 A_{k+1} 于 A 正交相似

Theorem 2.2.1–QR **迭代的收敛性**. 设 $A = VJV^{-1} \in \mathbb{R}^{n\times n}$, 其中 $J = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$，且 $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$。若 V^{-1} 的所有主子矩阵都非奇异 (即 V^{-1} 存在 LU 分解), 则 A_k 的对角线以下的元素均收敛到 0.

2.3 带位移的 QR 迭代

为了加快 QR 迭代的收敛速度, 我们可以采用位移策略和反迭代思想.

Algorithm 2 QR Iteration with shift

1: Set $A_1 = A$ and $k = 1$
2: while not convergence do
3:     Choose a shift σ_k
4:     $A_k - \sigma_k I = Q_k R_k$
5:     Compute $A_{k+1} = R_k Q_k + \sigma_k I$
6:     k=k+1
7: end while

我们有

$$\begin{aligned} A_{k+1} = R_k Q_k + \sigma_k I &= (Q_k^T Q_k) R_k Q_k + \sigma_k I \\ &= Q_k^T (A_k - \sigma_k I) Q_k + \sigma_k I \\ &= Q_k^T A_k Q_k \end{aligned}$$

所以, 带位移的 QR 算法中所得到的所有矩阵 A_k 都与 A 正交相似.

3 问题求解

1. 初始化定义 A 矩阵
2. 采用 $Householder$ 变化将 A 矩阵拟上三角化得到矩阵 A^{n-1}
3. 采用 QR 分解将矩阵 A^{n-1} 分解为矩阵 Q 与 R，并打印 RQ
4. 用带双步位移的 QR 方法求解所有特征值
5. 用 Gauss 消去法求解 $(A-\lambda I)X=0$, 得到对应的特征向量
6. 输出

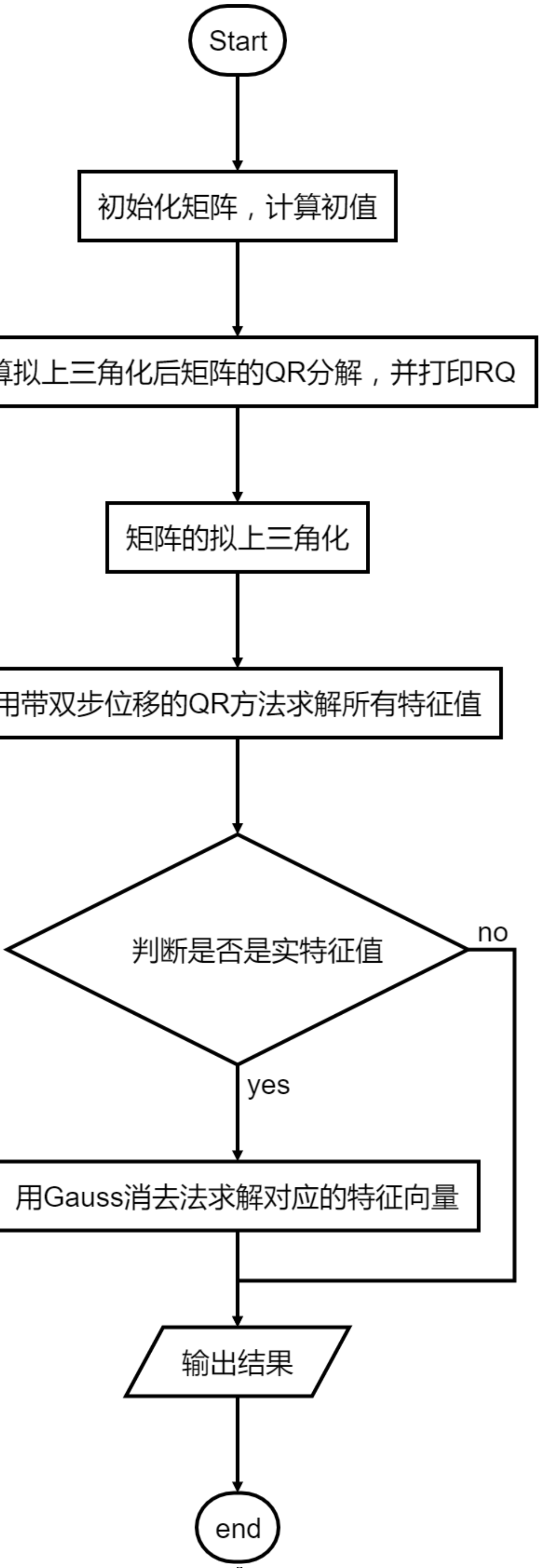
Start
初始化矩阵，计算初值
计算拟上三角化后矩阵的QR分解，并打印RQ
矩阵的拟上三角化
用带双步位移的QR方法求解所有特征值
判断是否是实特征值
no
yes
用Gauss消去法求解对应的特征向量
输出结果
end

图 3.1: 流程图

源程序 4

4.1 C 语言版大作业

```c
#include<stdio.h>
#include<math.h>
#include<string.h>
#define N 20
const double eps=1e-12;
double a[N][N],B[N][N],C[N][N];
int n=10;

typedef struct{
//定义复数结构体
    double Re;
    double Im;
}ComplexNumber;

void zeroMat(double a[N][N]){
    for(int i= 1; i<=10; i++) {
        for(int j = 1; j<=10; j++) {
            if (fabs(a[i][j])<eps)
                a[i][j] = 0;
        }
    }
}

void def(){
    //初始化A数组
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++){
            if(i==j)
```

```
                a[i][j]=1.52*cos(i+1.2*j);
            else if(i!=j)
                a[i][j]=sin(0.5*i+0.2*j);
        }
}

void eye(double Q[N][N]){
    //定义单位阵
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            Q[i][j]=(i==j);
}

int Is_zero(double Q[N][N],int r,int k){
    //K>0表示在QR中调用, K<0表示在 DQR中调用
    int kk=n;//更改数组大小
    if(k<0){kk=-k;k=1;}
    for(int i=r+k;i<=kk;i++)
        if(fabs(Q[i][r])>eps)
            return 0;
    return 1;
}


int sgn(double x){
    if(x>0)
        return 1;
    return -1;
}

void Matirix_M(double A[N][N],double x[N],double y[N],double h){
    for(int i=1;i<=n;i++)y[i]=0;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            y[i]+=A[i][j]*x[j]/h;
}

double Vector_M(double x[N],double y[N]){
    double sum=0;
    for(int i=1;i<=n;i++)
        sum+=x[i]*y[i];
    return sum;
}

void MatirixM_M(double X[N][N],double Y[N][N]){
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            for(int k=1;k<=n;k++)
                B[i][j]+=X[i][k]*Y[k][j];
```

```
}

void copy(double X[N][N],double Y[N][N]){
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            X[i][j]=Y[i][j];
}

void output(double A[N][N]){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++)
            printf("%.12e ",A[i][j]);
        printf("\n");
    }
}

void QR(double A[N][N],double Q[N][N]){
    double u[N],w[N],p[N],d,c,h;
    eye(Q);
    for(int r=1;r<n;r++){
        d=0;
        if(Is_zero(A,r,1))continue;
        for(int i=r;i<=n;i++)
            d+=A[i][r]*A[i][r];
        d=sqrt(d);
        c=-sgn(A[r][r])*d;
        h=c*c-c*A[r][r];
        for(int i=1;i<=n;i++){
            if(i<r)
                u[i]=0;
            else if(i==r)
                u[i]=A[r][r]-c;
            else if(i>r)
                u[i]=A[i][r];
        }
        memset(w,0,sizeof(w));
        Matirix_M(Q,u,w,1);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                Q[i][j]-=(w[i]*u[j]/h);
        memset(p,0,sizeof(p));
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                p[i]+=(A[j][i]*u[j]/h);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                A[i][j]-=(u[i]*p[j]);
    }
}
```

```
void Householder_Triangularization(double A[N][N]){
    double u[N],p[N],q[N],d,c,h;
    for(int r=1;r<n-1;r++){
        d=0;
        if(Is_zero(A,r,2))continue;
        for(int i=r+1;i<=n;i++)
            d+=A[i][r]*A[i][r];
        d=sqrt(d);
        c=-sgn(A[r+1][r])*d;
        h=c*c-c*A[r+1][r];
        for(int i=1;i<=n;i++){
            if(i<=r)
                u[i]=0;
            else if(i==(r+1))
                u[i]=A[i][r]-c;
            else if(i>r)
                u[i]=A[i][r];
        }
        memset(p,0,sizeof(p));
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                p[i]+=(A[j][i]*u[j]/h);
        Matirix_M(A,u,q,h);
        c=Vector_M(p,u)/h;
        for(int i=1;i<=n;i++)
            q[i]-=u[i]*c;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                A[i][j]-=(q[i]*u[j]+u[i]*p[j]);
    }
}


void select(int k,double b[N][N]){
    double max,c;
    int m=k;
    max=b[k][k];
    for(int i=k+1;i<=n;i++)
        if(fabs(b[i][k])>fabs(max)){
            max=b[i][k];
            m=i;
        }
    if(m!=k)
        for(int i=k;i<=n+1;i++){
        c=b[m][i];
        b[m][i]=b[k][i];
        b[k][i]=c;
        }
```

```
}

//Gauss消元法
void gauss(double lambda)
{
    double X[N];
    double m,sum,t;
    def();
    memset(X,0,sizeof(X));
    for(int i=1;i<=n;i++){
        a[i][i]-=lambda;
        a[i][n+1]=0;
    }
    for(int k=1;k<n;k++){
        select(k,a);
        for(int i=k+1;i<=n;i++){
            m=a[i][k]/a[k][k];
        for(int j=k;j<=n+1;j++)
            a[i][j]-=m*a[k][j];
        }
    }
    X[n]=1;
    for(int i=n-1;i>=1;i--){
        sum=0;
        for(int j=i+1;j<=n;j++)
            sum+=a[i][j]*X[j];
        if(fabs(a[i][i])>eps)
            X[i]=(a[i][n+1]-sum)/a[i][i];
        else
            X[i]=0;
    }
    t=0;
    for(int i=1;i<=n;i++)
        t+=X[i]*X[i];
    t=sqrt(t);
    printf("Eigenvector=(");
    for(int i=1;i<=n;i++)
        printf("%lf ", X[i]/t);
    printf(")\n");
}

void DQR(double A[N][N]){
    double Q[N][N],M[N][N],s,t,re,im,a,b,c,d;
    double u[N],v[N],p[N],q[N],h,det;
    ComplexNumber L[N];
    int m=n,LL=1000,r=1;
    for(int k=1;k<=100;k++){
        if (m == 1) {
            L[r].Re = A[m][m];
```

```
            L[r].Im = 0; break;
        }
        else if(m<=0)break;
        if(fabs(A[m][m-1])<=eps){
            L[r].Re = A[m][m];
            L[r].Im = 0;
            m--;r++;continue;
        }
        a=A[m-1][m-1];
        b=A[m-1][m];
        c=A[m][m-1];
        d=A[m][m];
        re=(a+d)/2;
        det=(a-d)*(a-d)+4*b*c;

        if((m==2)||(fabs(A[m-1][m-2])<=eps)){
            if(det>0){
                L[r].Re = re+sqrt(det)/2;
                L[r].Im = 0;
                L[r+1].Re = re-sqrt(det)/2;
                L[r+1].Im = 0;
            }
            else if(det<0){
                L[r].Re = re;
                L[r].Im=sqrt(fabs(det))/2;
                L[r+1].Re = re;
                L[r+1].Im = -L[r].Im;
            }
            m-=2;
            r+=2;
            continue;
        }
        if(k==LL)break;
        s=a+d;
        t=a*d-c*b;
        memset(M,0,sizeof(M));
        for(int i=1;i<=m;i++)
            for(int j=1;j<=m;j++){
                for(int l=1;l<=m;l++)
                    M[i][j]+=A[i][l]*A[l][j];
                M[i][j] -= s*A[i][j];
                M[i][j] += t*(i==j);
            }

        for(int r=1;r<m;r++){
            d=0;
            if(Is_zero(M,r,-m))continue;
            for(int i=r;i<=m;i++)
                d+=M[i][r]*M[i][r];
```

```
                d=sqrt(d);
                c=-sgn(M[r][r])*d;
                h=c*c-c*M[r][r];
                for(int i=1;i<=m;i++){
                    if(i<r)
                        u[i]=0;
                    else if(i==r)
                        u[i]=M[r][r]-c;
                    else if(i>r)
                        u[i]=M[i][r];
                }
                memset(p,0,sizeof(p));
                memset(q,0,sizeof(q));
                memset(v,0,sizeof(v));
                for(int i=1;i<=m;i++)
                    for(int j=1;j<=m;j++)
                        v[i]+=(M[j][i]*u[j]/h);
                for(int i=1;i<=m;i++)
                    for(int j=1;j<=m;j++){
                        M[i][j]-=(u[i]*v[j]);
                        p[i]+=(A[j][i]*u[j]/h);
                        q[i]+=(A[i][j]*u[j]/h);
                    }

                c=0;
                for(int i=1;i<=m;i++)
                    c+=p[i]*u[i]/h;
                for(int i=1;i<=m;i++)
                    q[i]-=u[i]*c;
                for(int i=1;i<=m;i++)
                    for(int j=1;j<=m;j++)
                        A[i][j]-=(q[i]*u[j]+u[i]*p[j]);
                zeroMat(A);
            }

        }
        zeroMat(A);
        for(int r = 1; r<=10; r++){
            printf("\n");
            if (L[r].Im == 0) {
                printf("lambda[%d] = %.12e \n", r , L[r].Re);
                gauss(L[r].Re);
            }
            else {
                printf("lambda[%d] = %.12e + i*%.12e\n", r , L[r].Re, L[r].Im);
            }
        }
}

```

```
int main(){
    double Q[N][N];
    freopen("Works.in","r",stdin);
    freopen("Works.out","w",stdout);
    def();
    Householder_Triangularization(a);
    printf("A_n-1:\n");
    output(a);
    QR(a,Q);
    printf("\nR:\n");
    output(a);
    printf("\nQ:\n");
    output(Q);
    memset(B,0,sizeof(B));
    MatirixM_M(a,Q);
    printf("\nR*Q:\n");
    output(B);
    def();
    DQR(a);
    return 0;
}
```

4.2 MATLAB 版 QR 分解

```
%fid=fopen('A_out.txt','w');
A=[
-0.894522   -0.099331   -1.099832   -0.766504   0.170760    -1.934883
     -0.083902   0.913257    -0.640798   0.194673;
-2.347878   2.372058    1.827999    0.326656    0.208236    2.088987
     0.184786    -1.263015   0.679069    -0.467215;
0.000000    1.735954    -1.165023   -1.246744   -0.629823   -1.984820
     0.297575    0.633930    -0.130852   0.304030;
0.000000    0.000000    -1.292938   -1.126239   1.190783    -1.308773
     0.186015    0.423673    -0.101960   0.194366;
0.000000    0.000000    0.000000    1.577711    0.816936    0.446153
     -0.043651   -0.466598   0.294123    -0.103442;
0.000000    0.000000    -0.000000   0.000000    -0.772898   -1.601028
     -0.291269   -0.243434   0.673629    0.262477;
0.000000    0.000000    0.000000    0.000000    -0.000000   -0.729677
     -0.007965   0.971074    -0.129897   0.027802;
0.000000    0.000000    -0.000000   0.000000    -0.000000   0.000000
     0.794554    -0.452514   0.504890    -0.121121;
0.000000    0.000000    0.000000    0.000000    -0.000000   0.000000
     -0.000000   0.703991    0.126754    -0.371470;
0.000000    0.000000    0.000000    0.000000    -0.000000   0.000000
     0.000000    0.000000    -0.491959   0.408151;
];
for  i=1:5
[q,r]=qr(A);
r
A=r*q;
end
A
r
q
e=diag(A)
eig(A)
%fclose(fid);
```

5 计算结果

后面为输出结果:

A_{n-1} 1 ~5 列:

$-8.945216982281e-001$	$-9.933136491826e-002$	$-1.099831758877e+000$	$-7.665038709077e-001$	$1.707601141456e-001$
$-2.347878362416e+000$	$2.372057921598e+000$	$1.827998552316e+000$	$3.266556884714e-001$	$2.082360583635e-001$
$0.000000000000e+000$	$1.735954469946e+000$	$-1.165023367477e+000$	$-1.246744443518e+000$	$-6.298225489084e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-1.292937563924e+000$	$-1.126239225902e+000$	$1.190782911924e+000$
$0.000000000000e+000$	$0.000000000000e+000$	$8.357856452328e-017$	$1.577711153032e+000$	$8.169358328160e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-9.764615357502e-017$	$0.000000000000e+000$	$-7.728975134989e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$5.527597418716e-017$	$0.000000000000e+000$	$-6.046925797698e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$-1.453366930807e-016$	$0.000000000000e+000$	$-2.216961051353e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$7.284428787973e-017$	$0.000000000000e+000$	$-8.264895477591e-018$
$0.000000000000e+000$	$0.000000000000e+000$	$1.475442153596e-017$	$0.000000000000e+000$	$-1.196693128035e-016$

A_{n-1} 6 ~10 列:

$-1.934882558889e+000$	$-8.390208705246e-002$	$9.132565113143e-001$	$-6.407977009188e-001$	$1.946733678685e-001$
$2.088987009941e+000$	$1.847861910289e-001$	$-1.263015266080e+000$	$6.790694668499e-001$	$-4.672150886500e-001$
$-1.984820180992e+000$	$2.975750060800e-001$	$6.339300596595e-001$	$-1.308518928772e-001$	$3.040301036095e-001$
$-1.308772983895e+000$	$1.860151662666e-001$	$4.236733936881e-001$	$-1.019600826545e-001$	$1.943660914505e-001$
$4.461531723828e-001$	$-4.365092541609e-002$	$-4.665979167188e-001$	$2.941231566184e-001$	$-1.034421113665e-001$
$-1.601028244046e+000$	$-2.912685474827e-001$	$-2.434337858321e-001$	$6.736286084510e-001$	$2.624772904937e-001$
$-7.296773946362e-001$	$-7.965456279819e-003$	$9.710739102007e-001$	$-1.298967368574e-001$	$2.780242081241e-002$
$0.000000000000e+000$	$7.945539612976e-001$	$-4.525143454606e-001$	$5.048901527575e-001$	$-1.211210193512e-001$
$0.000000000000e+000$	$-1.819189274915e-016$	$7.039911373514e-001$	$1.267535523498e-001$	$-3.714696735513e-001$
$0.000000000000e+000$	$1.280699160267e-016$	$0.000000000000e+000$	$-4.919586872214e-001$	$4.081509766399e-001$

R 1 ∼ 5 列:

$2.512509079248e+000$	$-2.181265513645e+000$	$-1.316649918648e+000$	$-3.235549645996e-002$	$-2.553867638933e-001$
$0.000000000000e+000$	$-1.972851789714e+000$	$2.276016623827e-001$	$7.014634531795e-001$	$5.947854062317e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$2.407240343439e+000$	$1.722528346094e+000$	$-4.505704077291e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$1.595615664669e+000$	$6.397406649436e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$-1.456242593458e+000$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$8.975288153029e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$-7.745003985679e-018$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$5.471400724120e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$2.321397713515e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$0.000000000000e+000$	$-2.603604216329e-017$

R 6 ∼10 列:

$-1.263236417243e+000$	$-1.428067524844e-001$	$8.551127106196e-001$	$-4.064323860885e-001$	$3.672920640297e-001$
$5.340587633903e-001$	$-3.303516655914e-001$	$6.131164883828e-002$	$-2.842350072136e-001$	$-1.020581006877e-001$
$3.392449531044e+000$	$-1.121444617851e-001$	$-1.448802456850e+000$	$7.311045593936e-001$	$-4.847285804893e-001$
$3.502375215214e-001$	$-6.543701621428e-002$	$-3.985833699409e-001$	$2.393366716784e-001$	$-9.059576583919e-002$
$-1.416208323337e+000$	$-2.740258546573e-001$	$2.820474174287e-001$	$3.146375562716e-002$	$2.179593898341e-001$
$1.239676225669e+000$	$1.437893689789e-001$	$-1.965876544047e-001$	$-5.501588373900e-001$	$-1.563867947060e-001$
$0.000000000000e+000$	$-8.001939216933e-001$	$3.239240677887e-001$	$-4.348133701956e-001$	$1.296863513999e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$1.309558711582e+000$	$-4.522300337931e-001$	$-2.541297741346e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$7.388504903301e-017$	$-6.591084569144e-001$	$4.900007103780e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-8.286706929455e-017$	$0.000000000000e+000$	$6.373330490059e-002$

Q 1 ~ 5 列:

$-3.560272500571e-001$	$4.439873952785e-001$	$-6.935939248834e-001$	$6.597513287486e-002$	$3.701042887335e-001$
$-9.344755733655e-001$	$-1.691554235406e-001$	$2.642533895704e-001$	$-2.513596481179e-002$	$-1.410065879813e-001$
$0.000000000000e+000$	$-8.799213803066e-001$	$-4.007708665994e-001$	$3.812160145536e-002$	$2.138528196492e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-5.371036454452e-001$	$-1.260096502461e-001$	$-7.068831837949e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$3.471965927751e-017$	$9.887789321494e-001$	$-1.266092216426e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-4.056352488489e-017$	$4.378988184867e-017$	$5.307477730505e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$2.296238277072e-017$	$-2.478877344479e-017$	$2.352952836071e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$-6.037481611537e-017$	$6.517693104408e-017$	$6.253700423368e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$3.026049645531e-017$	$-3.266736725224e-017$	$-1.803836264252e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$6.129185054653e-018$	$-6.616690490620e-018$	$7.737359003671e-017$

Q 6 ~ 10 列:

$1.873680253022e-001$	$-1.616846253861e-002$	$1.142209065421e-001$	$4.846147534278e-002$	$-5.435281546003e-002$
$-7.138562494120e-002$	$6.160046789174e-003$	$-4.351719447171e-002$	$-1.846341016476e-002$	$2.070796067082e-002$
$1.082645668876e-001$	$-9.342424307229e-003$	$6.599886483483e-002$	$2.800189963180e-002$	$-3.140602039975e-002$
$-3.578648243181e-001$	$3.088106413322e-002$	$-2.181569912327e-001$	$-9.255932185744e-002$	$1.038115266702e-001$
$-6.409685206678e-002$	$5.531080075230e-003$	$-3.907390568777e-002$	$-1.657821824708e-002$	$1.859359069582e-002$
$-6.851618290904e-001$	$5.912435352117e-002$	$-4.176796180699e-001$	$-1.772124834679e-001$	$1.987557610042e-001$
$-5.886032010032e-001$	$-9.581355780502e-002$	$6.768677853804e-001$	$2.871804513252e-001$	$-3.220922591440e-001$
$2.182477409330e-016$	$-9.929517580142e-001$	$-9.993700299902e-002$	$-4.240112211755e-002$	$4.755572027991e-002$
$-9.418745488859e-017$	$2.150264502325e-016$	$5.375789043480e-001$	$-5.611563234389e-001$	$6.293746914713e-001$
$7.348820816588e-017$	$-1.736576984577e-016$	$4.208932092508e-017$	$7.464002047925e-001$	$6.654973585866e-001$

$R * Q\, 1 \sim 5$ 列:

$1.143817643298e+000$	$2.643043667321e+000$	$-1.774014655111e+000$	$-7.804541680122e-002$	$3.406398561353e-001$
$1.843581807358e+000$	$1.334470111482e-001$	$-9.893074658739e-001$	$5.579861874654e-001$	$3.915081963850e-002$
$0.000000000000e+000$	$-2.118182245729e+000$	$-1.889928052623e+000$	$-5.708018640629e-001$	$1.154750215959e+000$
$0.000000000000e+000$	$0.000000000000e+000$	$-8.570109902230e-001$	$4.314991197035e-001$	$-1.023023164209e+000$
$0.000000000000e+000$	$0.000000000000e+000$	$-1.414667033083e-017$	$-1.439901996509e+000$	$-5.672756725063e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-5.272155056859e-017$	$1.456606955283e-016$	$6.579553960773e-001$
$0.000000000000e+000$	$0.000000000000e+000$	$-5.029401194954e-017$	$4.663621929838e-017$	$2.028726399036e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$-9.430668015218e-017$	$1.559077381528e-016$	$6.346313720807e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$-1.694164409331e-017$	$4.124264650296e-017$	$4.686324787275e-017$
$0.000000000000e+000$	$0.000000000000e+000$	$3.906332198803e-019$	$-2.616559352009e-017$	$8.227677638012e-018$

$R * Q\, 6 \sim 10$ 列:

$1.461454600932e+000$	$-9.542620239375e-001$	$4.390636920106e-001$	$7.812046500198e-001$	$-3.242676232229e-001$
$-2.951491456489e-001$	$1.302107099262e-002$	$-6.809912257186e-001$	$-1.407766316366e-001$	$4.534362093239e-003$
$-2.585301662617e+000$	$1.678124198417e+000$	$-1.154349560089e+000$	$-1.428583810585e+000$	$8.738827597003e-001$
$-8.134730259318e-001$	$4.755641447902e-001$	$-3.951755882759e-001$	$-4.241791598500e-001$	$3.396131374910e-001$
$1.224964946486e+000$	$-3.455910828759e-001$	$4.516704416505e-001$	$3.294865537961e-001$	$-4.202787739855e-002$
$-9.340137131103e-001$	$2.547201414468e-001$	$-6.965685047393e-001$	$2.194090527527e-002$	$-2.596005659042e-001$
$4.709967037320e-001$	$-2.449715460025e-001$	$-8.077439833310e-001$	$9.726139591002e-002$	$8.578610393813e-002$
$3.062200889581e-016$	$-1.300328624888e+000$	$-3.739826989665e-001$	$8.562468804235e-003$	$-3.914678236392e-001$
$9.660107939969e-017$	$-3.000540383773e-016$	$-3.543228021145e-001$	$7.355995090042e-001$	$-8.873200325455e-002$
$6.352474720579e-018$	$7.107121565932e-017$	$1.198130793216e-017$	$4.757055182990e-002$	$4.241434606534e-002$

特征值与向量如下:

$\lambda_1 = 9.432879572769e-001$

Eigenvector=(0.079620, 0.045421, -0.018272, -0.047961, -0.349567, 0.207215, -0.152312, 0.820634, -0.355466, 0.028866)

$\lambda_2 = 6.489488202110e-001$

Eigenvector=(0.108435, 0.071344, 0.382502, -0.047100, -0.717804, 0.181519, -0.226006, 0.388381, 0.289696, 0.024333)

$\lambda_3 = -9.891143464725e-001$+ i*1.084758631513e-001

$\lambda_4 = -9.891143464725e-001$- i*1.084758631513e-001

$\lambda_5 = 4.954990923624e-002$

Eigenvector=(-0.213768, -0.206774, 0.386829, -0.031112, -0.380939, -0.125174, 0.644716, -0.308201, -0.295977, 0.043723)

$\lambda_6 = -1.493147080915e+000$

Eigenvector=(-0.561341, 0.778192, 0.014364, -0.277602, 0.003568, -0.002548, -0.022061, -0.011758, -0.013173, 0.035016)

$\lambda_7 = 1.590313458807e+000$

Eigenvector=(0.062377, -0.011231, -0.252846, -0.130988, -0.381985, 0.815575, -0.123377, -0.067721, 0.271945, 0.100282)

$\lambda_8 = -2.336865932238e+000$+ i*8.934379210213e-001

$\lambda_9 = -2.336865932238e+000$- i*8.934379210213e-001

$\lambda_{10} = 3.389613438816e+000$

Eigenvector=(-0.104872, -0.217677, -0.474694, -0.259384, -0.304665, -0.259452, 0.086866, 0.405258, 0.509628, 0.239515)

讨论 6

1. 在对 $n \times n$ 实矩阵 $\boldsymbol{A}$ 作 QR 分解或双步位移分解时不需要形成具体的矩阵 $\boldsymbol{H}_i$，这样可以避免矩阵与矩阵相乘，大大减少了计算量。

2. 在 QR 分解中，R 矩阵可以直接储存在 A 的上三角部分，而在 M_k 的 QR 分解中，不需要再生成 $\boldsymbol{B}, \boldsymbol{C}$ 矩阵，直接在 $\boldsymbol{M}, \boldsymbol{A}$ 矩阵中迭代即可，以节约储存空间。

3.QR 方法适用于计算一般实矩阵的全部特征值，但对于大型实矩阵，则收敛速度不够用了，这时我们为了加速收敛，可以引入位移量, 通常, 位移越离特征值越近, 收敛速度就越快，如果位移 σ 与某个特征值非常接近, 则 $\boldsymbol{A}_{n,n}^{(k)} - \sigma$ 就非常接近于 0. 这说明 $\boldsymbol{A}_{n,n}^{(k)}$ 通常会首先收敛到 $\boldsymbol{A}$ 的一个特征值. 所以令 $\sigma = \boldsymbol{A}_{n,n}^{(k)}$ 是一个不错的选择. 但是, 如果这个特征值是复数, 这种位移选取方法就可能失效.

假设 $\sigma \in \mathbb{C}$ 是 $\boldsymbol{A}$ 的某个复特征值 λ 的一个很好的近似, 则其共轭 $\overline{\sigma}$ 也应该是 $\overline{\lambda}$ 的一个很好的近似. 因此我们可以考虑**双位移策略**, 即先以 σ 为位移迭代一次, 然后再以 $\overline{\sigma}$ 为位移迭代一次, 如此不断交替进行迭代. 这样就有

$$\begin{aligned} A_1 - \sigma I &= Q_1 R_1, \\ A_2 &= R_1 Q_1 + \sigma I, \\ A_2 - \overline{\sigma} I &= Q_2 R_2, \\ A_3 &= R_2 Q_2 + \overline{\sigma} I. \end{aligned}$$

容易验证

$$A_3 = Q_2^T A_2 Q_2 = Q_2^* Q_1^* A_1 Q_1 Q_2 = Q^* A_1 Q,$$

其中 $Q = Q_1 Q_2$.

我们注意到 σ 可能是复的, 所以 Q_1 和 Q_2 都可能是复矩阵. 但我们却可以选取适当的 Q_1 和 Q_2, 使的 $Q = Q_1 Q_2$ 是实正交矩阵从而 $A_3 = Q^T A_1 Q$ 也是实矩阵. 因此我们无需计算 A_2, 而是直接由 A_1 计算出 A_3.

4. 最后，在用 Gauss 消去法求解 $(A - \lambda I)X = 0$, 求得到对应的特征向量时, 需要将 $X[n]$ 预设为 1，因为由于线性方程组 $AX = b$ 中的 b 向量全为 0，所以解的自由度为 1，需要自己先将 $X[n]$ 的值定下来，才能迭代出剩下的解，不然求出的的解全为 0.