

数值分析第二次大作业

张晋

学号：15091060

最后更新于：May 4, 2017

目录 | 0

1	题目	2
2	算法设计方案	3
	2.1 Householder 变换	3
	2.2 QR 分解	5
	2.3 带位移的 QR 迭代	6
3	问题求解	7
4	源程序	9
	4.1 C 语言版大作业	9
	4.2 MATLAB 版 QR 分解	17
5	计算结果	18
6	讨论	24

试求矩阵 $\mathbf{A} = [a_{ij}]_{10 \times 10}$ 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：

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

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 \mathbf{A} 经过拟上三角化后所得的拟上三角阵 $\mathbf{A}^{(n-1)}$ ；
 - (c) 对矩阵 $\mathbf{A}^{(n-1)}$ 使用双步位移 QR 法迭代结束后所得的矩阵 \mathbf{Q} 和 \mathbf{R} ，并把 $\mathbf{R} \times \mathbf{Q}$ 打印，看是不是拟上三角阵；
 - (d) 矩阵 \mathbf{A} 的全部特征值 $\lambda_i = (R_i, I_i) \quad (i = 1, 2, \dots, 10)$ ；
 - (e) 矩阵 \mathbf{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	3
2.2 QR 分解	5
2.3 带位移的 QR 迭代	6

2.1 Householder 变换

Theorem 2.1.1. 我们称矩阵

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

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 $n-1$.

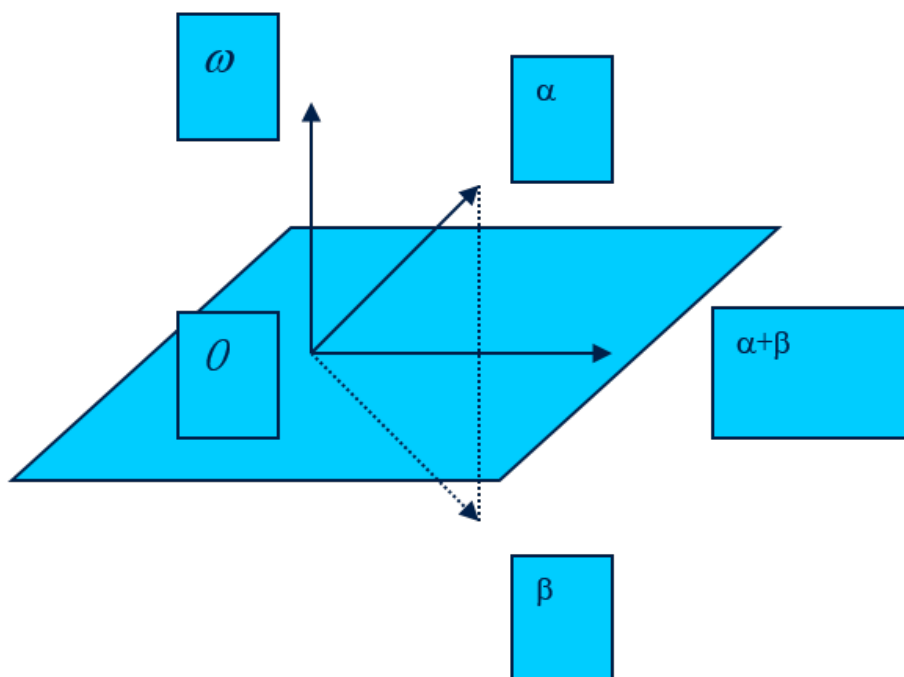


图 2.1: Householder

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

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

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

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

即 A_{k+1} 与 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 $J = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$, 且 $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$. 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  $\sigma_k$ 
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.

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

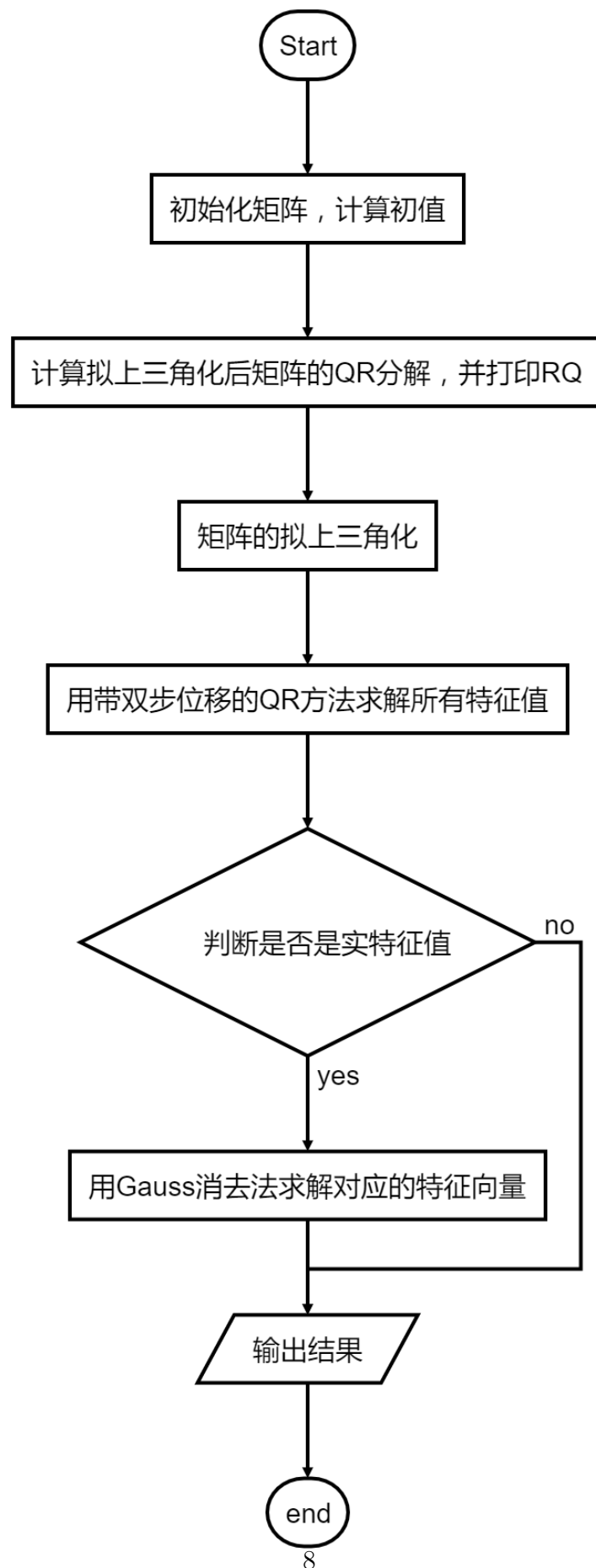


图 3.1: 流程图

4.1 C 语言版大作业	9
4.2 MATLAB 版 QR 分解	17

4.1 C 语言版大作业

```
1 #include<stdio.h>
2 #include<math.h>
3 #include<string.h>
4 #define N 20
5 const double eps=1e-12;
6 double a[N][N],B[N][N],C[N][N];
7 int n=10;
8
9 typedef struct{
10 //定义复数结构体
11     double Re;
12     double Im;
13 }ComplexNumber;
14
15 void zeroMat(double a[N][N]){
16     for(int i= 1; i<=10; i++) {
17         for(int j = 1; j<=10; j++) {
18             if (fabs(a[i][j])<eps)
19                 a[i][j] = 0;
20         }
21     }
22 }
23
24 void def(){
25     //初始化A数组
26     for(int i=1;i<=n;i++){
27         for(int j=1;j<=n;j++){
28             if(i==j)
```

```

29         a[i][j]=1.52*cos(i+1.2*j);
30     else if (i!=j)
31         a[i][j]=sin(0.5*i+0.2*j);
32     }
33 }
34
35 void eye(double Q[N][N]){
36     //定义单位阵
37     for (int i=1;i<=n;i++)
38         for (int j=1;j<=n;j++)
39             Q[i][j]=(i==j);
40 }
41
42 int Is_zero(double Q[N][N],int r,int k){
43     //K>0表示在QR中调用, K<0表示在DQR中调用
44     int kk=n;//更改数组大小
45     if (k<0){kk=-k;k=1;}
46     for (int i=r+k;i<=kk;i++)
47         if (fabs(Q[i][r])>eps)
48             return 0;
49     return 1;
50 }
51
52
53 int sgn(double x){
54     if (x>0)
55         return 1;
56     return -1;
57 }
58
59 void Matirix_M(double A[N][N],double x[N],double y[N],double h){
60     for (int i=1;i<=n;i++)y[i]=0;
61     for (int i=1;i<=n;i++)
62         for (int j=1;j<=n;j++)
63             y[i]+=A[i][j]*x[j]/h;
64 }
65
66 double Vector_M(double x[N],double y[N]){
67     double sum=0;
68     for (int i=1;i<=n;i++)
69         sum+=x[i]*y[i];
70     return sum;
71 }
72
73 void MatirixM_M(double X[N][N],double Y[N][N]){
74     for (int i=1;i<=n;i++)
75         for (int j=1;j<=n;j++)
76             for (int k=1;k<=n;k++)
77                 B[i][j]+=X[i][k]*Y[k][j];

```

```

78 }
79
80 void copy(double X[N][N],double Y[N][N]){
81     for(int i=1;i<=n;i++)
82         for(int j=1;j<=n;j++)
83             X[i][j]=Y[i][j];
84 }
85
86 void output(double A[N][N]){
87     for(int i=1;i<=n;i++){
88         for(int j=1;j<=n;j++)
89             printf("%.12e ",A[i][j]);
90         printf("\n");
91     }
92 }
93
94 void QR(double A[N][N],double Q[N][N]){
95     double u[N],w[N],p[N],d,c,h;
96     eye(Q);
97     for(int r=1;r<n;r++){
98         d=0;
99         if(Is_zero(A,r,1))continue;
100         for(int i=r;i<=n;i++)
101             d+=A[i][r]*A[i][r];
102         d=sqrt(d);
103         c=-sgn(A[r][r])*d;
104         h=c*c-c*A[r][r];
105         for(int i=1;i<=n;i++){
106             if(i<r)
107                 u[i]=0;
108             else if(i==r)
109                 u[i]=A[r][r]-c;
110             else if(i>r)
111                 u[i]=A[i][r];
112         }
113         memset(w,0,sizeof(w));
114         Matirix_M(Q,u,w,1);
115         for(int i=1;i<=n;i++)
116             for(int j=1;j<=n;j++)
117                 Q[i][j]-=(w[i]*u[j]/h);
118         memset(p,0,sizeof(p));
119         for(int i=1;i<=n;i++)
120             for(int j=1;j<=n;j++)
121                 p[i]+=(A[j][i]*u[j]/h);
122         for(int i=1;i<=n;i++)
123             for(int j=1;j<=n;j++)
124                 A[i][j]-=(u[i]*p[j]);
125     }
126 }

```

```

127
128 void Householder_Triangularization(double A[N][N]){
129     double u[N],p[N],q[N],d,c,h;
130     for(int r=1;r<n-1;r++){
131         d=0;
132         if(Is_zero(A,r,2))continue;
133         for(int i=r+1;i<=n;i++)
134             d+=A[i][r]*A[i][r];
135         d=sqrt(d);
136         c=-sgn(A[r+1][r])*d;
137         h=c*c-c*A[r+1][r];
138         for(int i=1;i<=n;i++){
139             if(i<=r)
140                 u[i]=0;
141             else if(i==(r+1))
142                 u[i]=A[i][r]-c;
143             else if(i>r)
144                 u[i]=A[i][r];
145         }
146         memset(p,0,sizeof(p));
147         for(int i=1;i<=n;i++)
148             for(int j=1;j<=n;j++)
149                 p[i]+=(A[j][i]*u[j]/h);
150         Matirix_M(A,u,q,h);
151         c=Vector_M(p,u)/h;
152         for(int i=1;i<=n;i++)
153             q[i]=-u[i]*c;
154         for(int i=1;i<=n;i++)
155             for(int j=1;j<=n;j++)
156                 A[i][j]=(q[i]*u[j]+u[i]*p[j]);
157     }
158 }
159
160
161 void select(int k,double b[N][N]){
162     double max,c;
163     int m=k;
164     max=b[k][k];
165     for(int i=k+1;i<=n;i++)
166         if(fabs(b[i][k])>fabs(max)){
167             max=b[i][k];
168             m=i;
169         }
170     if(m!=k)
171         for(int i=k;i<=n+1;i++){
172             c=b[m][i];
173             b[m][i]=b[k][i];
174             b[k][i]=c;
175         }

```

```

176 }
177
178 //Gauss消元法
179 void gauss(double lambda)
180 {
181     double X[N];
182     double m,sum,t;
183     def();
184     memset(X,0,sizeof(X));
185     for (int i=1;i<=n;i++){
186         a[i][i]-=lambda;
187         a[i][n+1]=0;
188     }
189     for (int k=1;k<n;k++){
190         select (k,a);
191         for (int i=k+1;i<=n;i++){
192             m=a[i][k]/a[k][k];
193             for (int j=k;j<=n+1;j++)
194                 a[i][j]-=m*a[k][j];
195         }
196     }
197     X[n]=1;
198     for (int i=n-1;i>=1;i--){
199         sum=0;
200         for (int j=i+1;j<=n;j++)
201             sum+=a[i][j]*X[j];
202         if (fabs(a[i][i])>eps)
203             X[i]=(a[i][n+1]-sum)/a[i][i];
204         else
205             X[i]=0;
206     }
207     t=0;
208     for (int i=1;i<=n;i++)
209         t+=X[i]*X[i];
210     t=sqrt(t);
211     printf("Eigenvector=");
212     for (int i=1;i<=n;i++)
213         printf("%lf ", X[i]/t);
214     printf("\n");
215 }
216
217 void DQR(double A[N][N]){
218     double Q[N][N],M[N][N],s,t,re,im,a,b,c,d;
219     double u[N],v[N],p[N],q[N],h,det;
220     ComplexNumber L[N];
221     int m=n,LL=1000,r=1;
222     for (int k=1;k<=100;k++){
223         if (m == 1) {
224             L[r].Re = A[m][m];

```

```

225         L[r].Im = 0; break;
226     }
227     else if (m<=0) break;
228     if (fabs(A[m][m-1])<=eps){
229         L[r].Re = A[m][m];
230         L[r].Im = 0;
231         m--;r++;continue;
232     }
233     a=A[m-1][m-1];
234     b=A[m-1][m];
235     c=A[m][m-1];
236     d=A[m][m];
237     re=(a+d)/2;
238     det=(a-d)*(a-d)+4*b*c;
239
240     if ((m==2)|| (fabs(A[m-1][m-2])<=eps)){
241         if (det>0){
242             L[r].Re = re+sqrt(det)/2;
243             L[r].Im = 0;
244             L[r+1].Re = re-sqrt(det)/2;
245             L[r+1].Im = 0;
246         }
247         else if (det<0){
248             L[r].Re = re;
249             L[r].Im=sqrt(fabs(det))/2;
250             L[r+1].Re = re;
251             L[r+1].Im = -L[r].Im;
252         }
253         m-=2;
254         r+=2;
255         continue;
256     }
257     if (k==LL) break;
258     s=a+d;
259     t=a*d-c*b;
260     memset(M,0,sizeof(M));
261     for (int i=1;i<=m;i++){
262         for (int j=1;j<=m;j++){
263             for (int l=1;l<=m;l++){
264                 M[i][j]+=A[i][l]*A[l][j];
265                 M[i][j] -= s*A[i][j];
266                 M[i][j] += t*(i==j);
267             }
268         }
269         for (int r=1;r<=m;r++){
270             d=0;
271             if (Is_zero(M,r,-m)) continue;
272             for (int i=r;i<=m;i++){
273                 d+=M[i][r]*M[i][r];

```

```

274         d=sqrt(d);
275         c=-sgn(M[r][r])*d;
276         h=c*c-c*M[r][r];
277         for (int i=1;i<=m;i++){
278             if (i<r)
279                 u[i]=0;
280             else if (i==r)
281                 u[i]=M[r][r]-c;
282             else if (i>r)
283                 u[i]=M[i][r];
284         }
285         memset(p,0,sizeof(p));
286         memset(q,0,sizeof(q));
287         memset(v,0,sizeof(v));
288         for (int i=1;i<=m;i++)
289             for (int j=1;j<=m;j++)
290                 v[i]+=(M[j][i]*u[j]/h);
291         for (int i=1;i<=m;i++)
292             for (int j=1;j<=m;j++){
293                 M[i][j]-=(u[i]*v[j]);
294                 p[i]+=(A[j][i]*u[j]/h);
295                 q[i]+=(A[i][j]*u[j]/h);
296             }
297
298         c=0;
299         for (int i=1;i<=m;i++)
300             c+=p[i]*u[i]/h;
301         for (int i=1;i<=m;i++)
302             q[i]-=u[i]*c;
303         for (int i=1;i<=m;i++)
304             for (int j=1;j<=m;j++)
305                 A[i][j]-=(q[i]*u[j]+u[i]*p[j]);
306         zeroMat(A);
307     }
308
309 }
310 zeroMat(A);
311 for (int r = 1; r<=10; r++){
312     printf("\n");
313     if (L[r].Im == 0) {
314         printf("lambda[%d] = %.12e \n", r , L[r].Re);
315         gauss(L[r]. Re);
316     }
317     else {
318         printf("lambda[%d] = %.12e + i*%.12e\n", r , L[r].Re, L[r].Im);
319     }
320 }
321 }
322

```



```

323 int main(){
324     double Q[N][N];
325     freopen("Works.in","r",stdin);
326     freopen("Works.out","w",stdout);
327     def();
328     Householder_Triangularization(a);
329     printf("A_n-1:\n");
330     output(a);
331     QR(a,Q);
332     printf("\nR:\n");
333     output(a);
334     printf("\nQ:\n");
335     output(Q);
336     memset(B,0,sizeof(B));
337     MatirixM__M(a,Q);
338     printf("\nR*Q:\n");
339     output(B);
340     def();
341     DQR(a);
342     return 0;
343 }

```

4.2 MATLAB 版 QR 分解

```
1 %fid=fopen('A_out.txt','w');
2 A=[
3 -0.894522 -0.099331 -1.099832 -0.766504 0.170760 -1.934883
   -0.083902 0.913257 -0.640798 0.194673;
4 -2.347878 2.372058 1.827999 0.326656 0.208236 2.088987
   0.184786 -1.263015 0.679069 -0.467215;
5 0.000000 1.735954 -1.165023 -1.246744 -0.629823 -1.984820
   0.297575 0.633930 -0.130852 0.304030;
6 0.000000 0.000000 -1.292938 -1.126239 1.190783 -1.308773
   0.186015 0.423673 -0.101960 0.194366;
7 0.000000 0.000000 0.000000 1.577711 0.816936 0.446153
   -0.043651 -0.466598 0.294123 -0.103442;
8 0.000000 0.000000 -0.000000 0.000000 -0.772898 -1.601028
   -0.291269 -0.243434 0.673629 0.262477;
9 0.000000 0.000000 0.000000 0.000000 -0.000000 -0.729677
   -0.007965 0.971074 -0.129897 0.027802;
10 0.000000 0.000000 -0.000000 0.000000 -0.000000 0.000000
   0.794554 -0.452514 0.504890 -0.121121;
11 0.000000 0.000000 0.000000 0.000000 -0.000000 0.000000
   -0.000000 0.703991 0.126754 -0.371470;
12 0.000000 0.000000 0.000000 0.000000 -0.000000 0.000000
   0.000000 0.000000 -0.491959 0.408151;
13 ];
14 for i=1:5
15 [q,r]=qr(A);
16 r
17 A=r*q;
18 end
19 A
20 r
21 q
22 e=diag(A)
23 eig(A)
24 %fclose(fid);
```

计算结果 | 5

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	1.57771153032e + 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

$R\ 1 \sim 5$ 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

$R\ 6 \sim 10$ 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

$Q \sim 5$ 列:

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

$Q \sim 10$ 列:

$1.873680253022e - 001$	$-1.616846253861e - 002$	$1.1422090065421e - 001$	$4.846147534278e - 002$	$-5.435281546003e - 002$
$-7.138562494120e - 002$	$6.160046789174e - 003$	$-4.351719447171e - 002$	$-1.846341016476e - 002$	$2.070796067082e - 002$
$1.082645668876e - 001$	$-9.342424307229e - 003$	$6.599886483483e - 002$	$2.800189963180e - 002$	$-3.140602039975e - 002$
$-3.578648243181e - 001$	$3.088106413322e - 002$	$-2.181569912327e - 001$	$-9.255932185744e - 002$	$1.038115266702e - 001$
$-6.409685206678e - 002$	$5.531080075230e - 003$	$-3.907390568777e - 002$	$-1.657821824708e - 002$	$1.859359069582e - 002$
$-6.851618290904e - 001$	$5.912435352117e - 002$	$-4.176796180699e - 001$	$-1.772124834679e - 001$	$1.987557610042e - 001$
$-5.886032010032e - 001$	$-9.581355780502e - 002$	$6.768677853804e - 001$	$2.871804513252e - 001$	$-3.220922591440e - 001$
$2.182477409330e - 016$	$-9.929517580142e - 001$	$-9.993700299902e - 002$	$-4.240112211755e - 002$	$4.755572027991e - 002$
$-9.418745488859e - 017$	$2.150264502325e - 016$	$5.375789043480e - 001$	$-5.611563234389e - 001$	$6.293746914713e - 001$
$7.348820816588e - 017$	$-1.736576984577e - 016$	$4.208932092508e - 017$	$7.464002047925e - 001$	$6.654973585866e - 001$

$R * Q$ 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:

$$\begin{aligned}
\lambda_1 &= 9.432879572769e - 001 \\
\text{Eigenvector} &= (0.079620, \quad 0.045421, \quad -0.018272, \quad -0.047961, \quad -0.349567, \quad 0.207215, \quad -0.152312, \quad 0.820634, \quad -0.355466, \quad 0.028866) \\
\\
\lambda_2 &= 6.489488202110e - 001 \\
\text{Eigenvector} &= (0.108435, \quad 0.071344, \quad 0.382502, \quad -0.047100, \quad -0.717804, \quad 0.181519, \quad -0.226006, \quad 0.388381, \quad 0.289696, \quad 0.024333) \\
\\
\lambda_3 &= -9.891143464725e - 001 + i*1.084758631513e-001 \\
\lambda_4 &= -9.891143464725e - 001 - i*1.084758631513e-001 \\
\\
\lambda_5 &= 4.954990923624e - 002 \\
\text{Eigenvector} &= (-0.213768, \quad -0.206774, \quad 0.386829, \quad -0.031112, \quad -0.380939, \quad -0.125174, \quad 0.644716, \quad -0.308201, \quad -0.295977, \quad 0.043723) \\
\\
\lambda_6 &= -1.493147080915e + 000 \\
\text{Eigenvector} &= (-0.561341, \quad 0.778192, \quad 0.014364, \quad -0.277602, \quad 0.003568, \quad -0.002548, \quad -0.022061, \quad -0.011758, \quad -0.013173, \quad 0.035016) \\
\\
\lambda_7 &= 1.590313458807e + 000 \\
\text{Eigenvector} &= (0.062377, \quad -0.011231, \quad -0.252846, \quad -0.130988, \quad -0.381985, \quad 0.815575, \quad -0.123377, \quad -0.067721, \quad 0.271945, \quad 0.100282) \\
\\
\lambda_8 &= -2.336865932238e + 000 + i*8.934379210213e-001 \\
\lambda_9 &= -2.336865932238e + 000 - i*8.934379210213e-001 \\
\\
\lambda_{10} &= 3.389613438816e + 000 \\
\text{Eigenvector} &= (-0.104872, \quad -0.217677, \quad -0.474694, \quad -0.259384, \quad -0.304665, \quad -0.259452, \quad 0.086866, \quad 0.405258, \quad 0.509628, \quad 0.239515)
\end{aligned}$$

1. 在对 $n \times n$ 实矩阵 \mathbf{A} 作 QR 分解或双步位移分解时不需要形成具体的矩阵 \mathbf{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 \mathbf{B}, \mathbf{C} 矩阵, 直接在 \mathbf{M}, \mathbf{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 $\mathbf{A}_{n,n}^{(k)} - \sigma$ 就非常接近于 0. 这说明 $\mathbf{A}_{n,n}^{(k)}$ 通常会首先收敛到 \mathbf{A} 的一个特征值. 所以令 $\sigma = \mathbf{A}_{n,n}^{(k)}$ 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 \mathbf{A} 的某个复特征值 λ 的一个很好的近似, 则其共轭 $\bar{\sigma}$ 也应该是 $\bar{\lambda}$ 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 $\bar{\sigma}$ 为位移迭代一次, 如此不断交替进行迭代. 这样就有

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

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. 但我们却可以选择适当的 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, 不然求出的解全为 0.