QR 方法解非对称特征值问题

李元楷 21307140011

2022 年 12 月 22 日

1 简介

本文简要说明了如何利用隐式 QR 技术进行 QR 方法求解实非对称矩阵的特征值问题。大体上，首先对一个矩阵上 Hessenberg 化，其后利用隐式 Q 定理对其进行处理。首先我们给出不带位移的初始版本算法，其只能处理实特征值矩阵；而后我们再给出双步位移的算法，其能处理复特征值问题。最后我们研究这两个算法的收敛性能（主要是双步位移）。本文所使用的代码至少可以安全地计算出 500 阶矩阵的特征值问题（更高阶的没有做过实验）。

2 准备工作

2.1 上 Hessenberg 化

事实上，观察到任何一个实矩阵必定正交相似于一个上 Hessenberg 矩阵。只要不断地作用 Householder 变换即可。下面的代码即可保证将一个 n 阶矩阵 A 化为上 Hessenberg 阵。

```
function A = Upperhessenberg(A,n) % A the matrix, n the order of A
    for i = 1:n-1
        flag = 0; % 0 means all zeros, 1 means find non-zero element
        % In fact, we find the first non-zero element other than the diag
        for j = i+1:n
            if A(j,i) ~= 0
                flag = 1;
                temp = Householder(A(j:n,i),n-j+1,n);
                break;
            end
        end
        if flag == 1
            A = temp*A*temp';
        end
    end
end
```

我们顺便给出这里的 Householder 变换的代码和将要用到的 Givens 变换的代码。

2.2 Householder 变换

对应向量 v 的 Householder 变换，其中 l 为 v 的长度，M 为矩阵的大小。

```
function H = Householder(v,l,M)
    v = v / norm(v,"inf"); %normalize
    h = zeros(1,l)';
    h(2:l) = v(2:l);    %the first element to be zero,start from 2
    h(1) = v(1) + v(1) / abs(v(1)) * norm(v);
    h = h / norm(h);
    H = zeros(M,M);
    if M+1-l >= 1
        H(1:M-l,1:M-l) = eye(M-l);
    end
    H(M+1-l:M,M+1-l:M) = eye(l)- 2 * (h * h');
end
```

2.3 Givens 变换

对应向量 v 的 givens 变换，pos 为生成 0 的位置。

```
function G = Givens(v,pos,M)
     G = eye(M);
     G(pos+2,pos+2) = 0;
     G(pos+1,pos+1) = 0;
    % tan(theta) = v(2,1)/v(1,1)
    if v(1,1) == 0
        G(pos+1,pos+1) = 0;
        G(pos+1,pos+2) = 1;
        G(pos+2,pos+1) = -1;
        G(pos+2,pos+2) = 0;
    else
        t = v(2,1)/v(1,1);
        G(pos+2,pos+2)= 1/sqrt(1+t*t);
        G(pos+1,pos+1) = G(pos+2,pos+2);
        %Make sure tan(theta) = v(2,1)/v(1,1) instead of -v(2,1)/v(1,1)
        if sign(t) == 1
            G(pos+1,pos+2) = sqrt(1-1/(t*t+1));
            G(pos+2,pos+1) = -G(pos+1,pos+2);
        else
            G(pos+1,pos+2) = -sqrt(1-1/(t*t+1));
            G(pos+2,pos+1) = -G(pos+1,pos+2);
        end
```

```
    end
end
```

2.4 隐式 Q 定理

$Th : H = Q^T AQ \in \mathbf{R}^{n\times n}$是一个不可约上 Hessenberg 矩阵, 其中,$Q \in \mathrm{R}^{n\times n}$是正交矩阵,

则 Q 的第 2 至第 n 列由 Q 的第一列所唯一确定.

事实上这是被 Hessenberg 矩阵本身的性质保证的。

3 原始的隐式 QR 迭代

利用隐式 Q 定理，我们可以不作 $H_k = Q_k R_k, H_{K+1} = R_k Q_k$, 而是通过一系列其他的正交阵的乘积来充当 Q。首先引入一个 bugle，然后通过一系列 Givens 变换将其“赶出”矩阵；这样，这一系列酉阵的乘积就是 Q. 这里 A 是要处理的矩阵，n 是它的大小，m 是真正需要处理的矩阵大小（一般来说，左上角一旦出现了特征值，就不再考虑，转而考虑更小的矩阵）

```
function A = Singlestep_Iteration(A,n,m)
    %The first givens rotation, introducing the bugle
    G = eye(n);
    G(n-m+1,n-m+1) = 0;
    G(n-m+2,n-m+2) = 0;
    norm = sqrt(A(n-m+1,n-m+1) ^ 2 + A(n-m+2,n-m+1) ^ 2);
    G(n-m+1,n-m+1) = A(n-m+1,n-m+1) / norm;
    G(n-m+2,n-m+2) = G(n-m+1,n-m+1);
    G(n-m+1,n-m+2) = A(n-m+2,n-m+1) / norm;
    G(n-m+2,n-m+1) = -G(n-m+1,n-m+2);
    A = G*A*G';
    % Pop the bugle using Givens rotation
    for i=n-m+1:n-2
        temp = Givens(A(i+1:i+2,i),i,n);
        A = temp*A*temp';
    end
end
```

上面的代码是一次迭代的代码，具体使用时，只要再判断左上方是否出现特征值来不断调整 m 的值；同时监视所有的下次对角元绝对值之和作为作为收敛与否的判据。

4 双步位移的隐式 QR 迭代

带位移的 QR 迭代无疑会快上许多；选取双位移是因为即使在出现了复特征值的情况下，两步位移也能保证只做实数运算。这里我们选取所谓的 Francis 位移，即 A 的右下角 2*2 矩阵的特

征值作为位移。同样，先引入一个 2*2 的 bugle，随后用一系列 Householder 变换将这个 bugle 向下移，最后用一个 givens 变换收尾。下列代码中，A 是要处理的矩阵，m 是其大小，r 是左上角已完成的大小，n 是从右下角开始收缩而未完成的大小，tol 是可容忍的误差（按单个元素计）。注意到，出现了变量 flag；这是为了处理当 bugle 提前消失的情况：这时我们从小矩阵开始重新引入 bugle，并消去。注意到这一段代码默认了矩阵 A 至少有三行三列，故整个函数的第一个 if 判断就来处理小矩阵的情况，使用内置的 schur 函数，运算量极小。

```
function A = Doublestep_Iteration(A,n,m,r,tol)
    if n-r <=2
        l = eye(m);
        [l(r:n,r:n) ,t] = schur(A(r:n,r:n));
        A = l'*A*l;

    else


    flag = 0 ;
    % Below we introduce the first bugle(Using Francis displacement):
    a = A(n,n);
    b = A(n-1,n-1);
    c = A(n-1,n);
    d = A(n,n-1);
    sigma = ((a+b)+sqrt((a+b)^2-4*(a*b-c*d)))/2;
    re = real(sigma);
    no = norm(sigma);

    H_begin = zeros(3,1);
    H_begin(1,1) = A(r,r)^2+A(r,r+1)*A(r+1,r)-2*re*A(r,r)+no^2;
    H_begin(2,1) = A(r+1,r)*(A(r,r)+A(r+1,r+1)-2*re);
    H_begin(3,1) = A(r+1,r) *A(r+2,r+1);



    Hnorm = norm(H_begin);

    H_begin(1,1) = H_begin(1,1)/Hnorm;
    H_begin(2,1) = H_begin(2,1)/Hnorm;
    H_begin(3,1) = H_begin(3,1)/Hnorm;

    H_rest = null(H_begin');
    H_small = [H_begin H_rest];

```

```
    H = zeros(m);
    H(r:r+2,r:r+2) = H_small;
    for i =r+3:m
        H(i,i)=1;
    end
    for i =1:r-1
        H(i,i)=1;
    end


    A = H'*A*H;

    %Next we chase the bugle

    for i = r+1:n-2
        if abs(norm(A(i+1:i+2,i-1))) >= tol
            h_small = Householder(A(i:i+2,i-1),3,3);
            house = eye(m);
            house(i:i+2,i:i+2) = h_small;
            A = house'*A*house;
        else
            flag = 1;
            break
        end
    end

    if flag == 0
        %Finally a givens rotation
        g_small = Givens(A(n-1:n,n-2),n-2,n);
        g = eye(m);
        g(1:n,1:n) = g_small;
        A = g*A*g';
    end

    if flag == 1
        A = Doublestep_Iteration(A,n,m,i,tol);
    end
    end
end
```

同样，这里只是一步迭代的展开；具体使用时，还要仔细设置收敛条件，来让 A 收敛到实 Schur 形状，代码就放在源码文件中，不单独给出了。

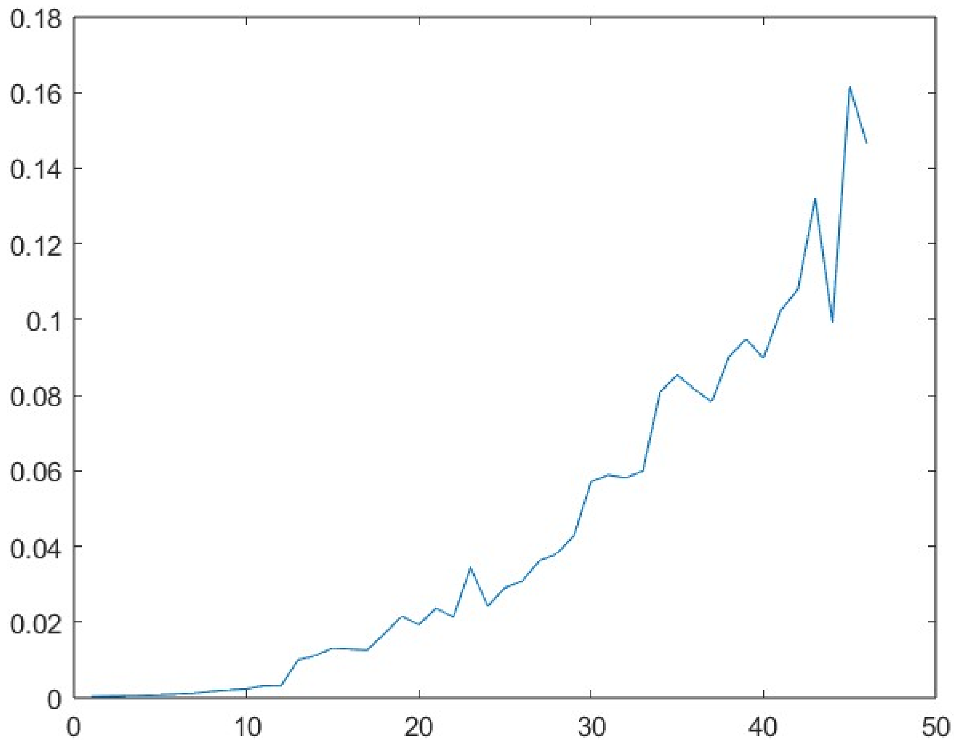

图 1: 50 阶（20）

5 数值实验

5.1 迭代速度（双步位移）

使用 tic-toc 函数来计时，并画出其与矩阵阶数的关系图。到 50 阶的每一阶矩阵随机取了 20 个，取平均值。到 100 阶的每一阶矩阵随机取了 10 个，取平均值。

5.2 迭代历史-2*2 块的收敛历史

由于代码中，循环元 i 即代表了迭代的进程；故只要在代码中添加 display(i) 即可用肉眼观察到收敛的进程。事实上，我们发现绝大多数 2*2 块在两到三次迭代后就能收敛，但少数 2*2 块要运算多次（有的甚至上百）才能收敛。给出的源码中保留了 display(i)，稍加运行便可知。

5.3 Tolerance 的选取

事实上 tolerance 的选取很有讲究，合适的 tolerance 能大大加快运行时间。这里以 100 阶矩阵为例，展示 tolerance 的作用。可见，tolerance 也并非越大越好：算不准后反而会让计算变慢。

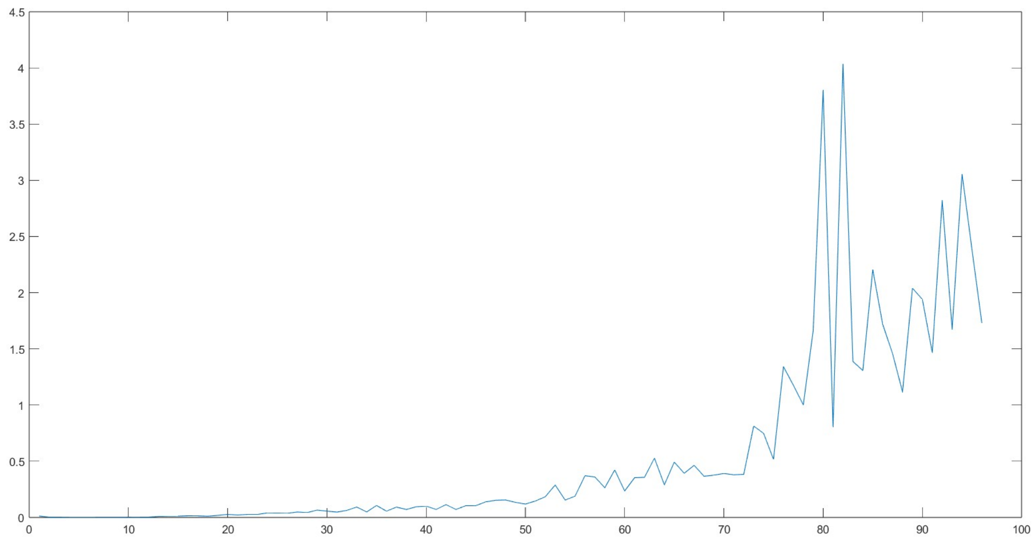

图 2: 100 阶（10）

Tolerance	Time
0.0001	2.0122s
0.00001	1.1961s
0.000001	2.6913s
0.0000001	2.5761s