

特殊矩阵奇异值分解的算法设计和 Jacobi 方法 中加速技术的功效分析

吴开亮

数学科学学院 1101110030

目录

1 反对称矩阵的 Schur 分解	2
1.1 问题的提出	2
1.2 三对角化	2
1.3 隐式 QR 迭代	4
1.4 基于隐式 QR 迭代的反对称矩阵 Schur 分解算法	5
1.5 数值特性分析	5
1.5.1 算法的复杂度分析	5
1.5.2 算法的误差分析	6
1.5.3 数值实验与结果分析	6
2 矩阵 $D + \rho zz^T$ 的特征值与特征向量的计算	14
2.1 问题的提出	14
2.2 基本理论的建立	14
2.3 基于二分法的特征值计算	17
2.4 特征向量的计算	20
2.5 一般情况的前期处理	20
2.6 数值特性分析	22
2.6.1 算法的复杂度分析	22
2.6.2 计算特征值的稳定性分析	23
2.6.3 数值实验与结果分析	24
3 单边 Jacobi 方法中的加速技术效果分析	31
3.1 问题的提出	31
3.2 计算奇异值分解的单边 Jacobi 方法	31
3.3 单边 Jacobi 方法的加速技术	33
3.3.1 de Rijk 加速技巧	33
3.3.2 预处理加速技术	33
3.4 数值实验与加速技术的加速效果分析	34
4 结语	40

1 反对称矩阵的 Schur 分解

1.1 问题的提出

设 $A \in R^{n \times n}$, 而且 $A^T = -A$ 。给出一种计算 A 的实 Schur 分解的数值方法。并用理论分析和数值实验的方法来说明你所给出的算法的数值特性。

我们知道, 对称 QR 方法是将 QR 方法应用到对称矩阵, 并充分利用其对称性而得。类比于对称 QR 方法的思想, 我们给出了反对称矩阵的 schur 分解算法。

1.2 三对角化

若 A 是 n 阶实反对称矩阵, 并假定 A 的上 Hessenberg 分解为

$$Q^T A Q = T$$

其中 Q 是正交矩阵, T 是上 Hessenberg 矩阵, 则容易验证 T 必是反对称三对角矩阵。因此, 对一个实反对称矩阵而言, 上 Hessenberg 化实质就是将其三对角化。如果在约化过程中充分利用其对称性, 还可使其约化的运算量大为减少。

将 A 作如下分块

$$A = \begin{pmatrix} \alpha_1 & -v_0^T \\ v_0^T & A_0 \\ 1 & n-1 \end{pmatrix}$$

从约化一个矩阵为上 Hessenberg 矩阵的 Householder 变换法不难推出, 利用 Householder 变换将 A 约化为反对称三对角矩阵的第 k 步为:

(1) 计算 Householder 变换 $\tilde{H}_k \in R^{(n-k) \times (n-k)}$ 使得

$$\tilde{H}_k v_{k-1} = \beta_k e_1, \beta_k \in R;$$

(2) 计算

$$\frac{1}{n-k-1} \begin{pmatrix} \alpha_{k+1} & -v_k^T \\ v_k^T & A_k \\ 1 & n-k-1 \end{pmatrix} = \tilde{H}_k A_{k-1} \tilde{H}_k.$$

如果用上述约化过程所产生的 α_k, β_k 和 \tilde{H}_k 定义

$$T = \begin{pmatrix} \alpha_1 & -\beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & -\beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix},$$

$$Q = H_1 H_2 \cdots H_{n-2}, H_k = \text{diag}\left(I_k, \tilde{H}_k\right)$$

其中

$$\begin{pmatrix} \alpha_{n-1} & -\beta_{n-1} \\ \beta_{n-1} & \alpha_n \end{pmatrix} = \tilde{H}_{n-2} A_{n-3} \tilde{H}_{n-2},$$

则有

$$Q^T A Q = T.$$

从上述约化过程容易看出，第 k 步约化的主要工作量是计算

$$\tilde{H}_k A_{k-1} \tilde{H}_k.$$

设

$$\tilde{H}_k = I - \beta v v^T, v \in R^{n-k},$$

则利用 A_{k-1} 的对称性，易得

$$\tilde{H}_k A_{k-1} \tilde{H}_k = A_{k-1} + v w^T - w v^T,$$

其中

$$w = u - \frac{1}{2} \beta (v^T u) v, u = \beta A_{k-1} v.$$

这样我们就得到了如下算法。

算法 1.1 计算反对称矩阵三对角分解：Householder 变换法

for $k = 1 : n - 2$

$$[v, \beta] = \text{house}\left(A(k+1:n, k)\right)$$

$$u = \beta A(k+1:n, k+1:n) v$$

$$w = u - (\beta u^T v / 2) v$$

$$A(k+1, k) = \|A(k+1:n, k)\|_2$$

$$A(k, k+1) = A(k+1, k)$$

$$A(k+1:n, k+1:n) = A(k+1:n, k+1:n) + v w^T - w v^T$$

end

1.3 隐式 QR 迭代

完成了把 A 约化为三对角矩阵 T 的任务之后, 我们的下一个任务就是选取适当的位移进行 QR 迭代。由于此时 A 实特征值均为 0, 因而对单位移的 QR 方法就不用使用位移了 (位移为 0)。

迭代的具体形式为

$$\begin{cases} T_k = Q_k R_k; \\ T_{k+1} = R_k Q_k. \end{cases}, k = 0, 1, 2, \dots$$

其中 $T_0 = T$ 是反对称三对角矩阵。由于 QR 迭代保持上 Hessenberg 形和反对称性的特点, 我们立即可知上述迭代产生的 T_k 都是反对称三对角矩阵。与一般的 QR 分解方法一样, 这里我们也假定迭代中所出现的 T_k 都是不可约的, 即次对角线元均不为零。

再考虑如何具体实现每一步的 QR 迭代:

$$T = QR, \tilde{T} = RQ. \quad (1.1)$$

当然我们可以利用 Givens 变化来直接实现 T 的 QR 分解, 进而完成一步迭代。但是, 我们有一个更漂亮的做法, 就是以隐式迭代实现 T 到 \tilde{T} 的变换。

我们知道, 迭代 (1.1) 的本质就是用正交相似变换将 T 变为 \tilde{T} , 即 $\tilde{T} = Q^T T Q$ 。因此, \tilde{T} 本质上是由 Q 的第一列完全确定的, 我们采用若干 Givens 变换来实现, 则第 k 变换可以表示为

$$T_{k+1} = G_k T_k G_k^T; G_k = G(k, k+1, \theta).$$

算法 1.2 隐式反对称 QR 迭代

```
x = T(1,1); z = T(2,1)
for k = 1 : n - 1
    [c,s] = givens(x,z)
    T = G_k T G_k^T; G_k = G(k,k+1,c,s)
    if k < n - 1
        x = T(k+1,k); z = T(k+2,k)
    end
end
```

1.4 基于隐式 QR 迭代的反对称矩阵 Schur 分解算法

类比如一般的 QR 算法，综合上面的讨论，可得如下算法。

算法 1.3 反对称矩阵 A 的 Schur 分解算法

Step1 三对角化：用算法 1.1 计算 A 的三对角分解，得

$$T = U_0^T A U_0; Q = U_0.$$

Step2 收敛性判断

Step2.1 把所有满足条件

$$|t_{i+1,i}| = |t_{i,i+1}| \leq (|t_{i,i}| + |t_{i+1,i+1}|) \varepsilon$$

的 $t_{i+1,i}$ 和 $t_{i,i+1}$ 置零。

Step2.2 确定最大的非负整数 m 和最小的非负整数 ℓ ，使得

$$T = \begin{bmatrix} T_{11} & 0 & 0 \\ 0 & T_{22} & 0 \\ 0 & 0 & T_{33} \end{bmatrix} \quad \begin{matrix} \ell \\ n - \ell - m \\ m \end{matrix},$$

其中 T_{33} 为对角块为零矩阵或者 2×2 分块上三角矩阵，而 T_{22} 为不可约的反对称三对角矩阵。

Step2.3 如果 $m = n$ ，则输出相关信息，结束；否则进行下一步。

Step3 QR 迭代：对 T_{22} 用算法 1.2 迭代一次得

$$T_{22} = GT_{22}G^T, G = G_1 G_2 \cdots G_{n-m-\ell-1}.$$

Step4 $Q = Q \text{diag}(I_\ell, G, I_m)$ ，然后转入 **Step2**。

1.5 数值特性分析

1.5.1 算法的复杂度分析

对于算法 1.1，计算 $\tilde{H}_k A_{k-1} \tilde{H}_k$ 为算法的主要工作量，经以利用反对称矩阵得性质进行优化，运算量仅为 $4(n-k)^2$ ，所以算法 1.1 的运算量大约为 $4 \sum_{k=1}^n (n-k)^2$ ，即 $4n^3 / 3 + O(n^2)$ 。而对于一般的矩阵，上 Hessenberg 化需要的运算量为 $10n^3 / 3 + O(n^2)$ 。如果需要将变换累积起来，则还需增加运算量 $4n^3 / 3 + O(n^2)$ 。

对于算法 1.2, 运算量为 $10n$; 如果要累积变换矩阵, 还需要增加运算量 $6n^2$ 。此外, 实际计算时, 三对角矩阵 T 是以一个 $n - 1$ 维向量存储的 (反对称矩阵对角线元素均为 0)。因此空间复杂度仅为 $O(n)$ 。

综上所述, 可分析出整个算法 1.3 的运算量平均约为 $4n^3 / 3$, 空间复杂度为 $O(n)$; 如果 Schur 分解的正交变换矩阵 Q 也需要计算, 则运算量平均约为 $9n^3$, 空间复杂度为 $O(n^2)$ 。

1.5.2 算法的误差分析

经误差分析, 设该算法计算得到分块对角矩阵满足

$$Q^T (A + E) Q = \begin{pmatrix} \hat{B} & \\ & O \end{pmatrix}$$

其中 Q 为正交矩阵, $\|E\|_2 \approx \|A\|_2 \mathbf{u}, i = 1, 2, \dots, n$. 这里

$$\hat{B} = \text{diag} \left(\begin{bmatrix} 0 & -\hat{t}_1 \\ \hat{t}_1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -\hat{t}_2 \\ \hat{t}_2 & 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 & -\hat{t}_s \\ \hat{t}_s & 0 \end{bmatrix} \right)$$

为计算得到的分块对角矩阵。

类比于 Weyl 定理的证明, 可以说明

$$\left\| \begin{bmatrix} 0 & -t_j \\ t_j & 0 \end{bmatrix} - \begin{bmatrix} 0 & -\hat{t}_j \\ \hat{t}_j & 0 \end{bmatrix} \right\|_2 \approx \|A\|_2 \varepsilon$$

这就是说, 我们设计的算法 1.3 计算得到的 Schur 分解精度是相当精确的。

1.5.3 数值实验与结果分析

在这一节, 我们选择若干算例, 通过 Matlab 编程实现反对称矩阵的 Schur 分解, 并分析我们设计的数值算法 (简称 ASM-Schur) 的数值特性 (计算精度、CPU 时间等)。最后我们将 ASM-Schur 算法与 matlab 中的 schur 函数进行对比分析。

为便于分析, 我们定义几种误差, 来刻画计算精度。

定义 1.1 矩阵相对误差 (m.b.e) : 其定义为

$$m.b.e = \frac{\|Q_a^T A_0 Q_a - B_f\|_\infty}{\|A_0\|_\infty}$$

其中 $A_0 = A$ 表示初始迭代矩阵, Q_a 为迭代过程中累计的相似变换, B_f 为迭代最后的分块对角阵。

定义 1.2 向前误差 (a.f.e) 与正交性误差 (o.e) 对于给定的形如 $diag(B, O)$ 的分块对角矩阵和正交矩阵 Q , 这里

$$B = \text{diag} \left(\begin{bmatrix} 0 & -t_1 \\ t_1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -t_2 \\ t_2 & 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 & -t_s \\ t_s & 0 \end{bmatrix} \right), t_i \in R, i = 1, 2, \dots, s.$$

将单位移反对称 Schur 分解算法 (或双位移反对称 Schur 分解算法) 用于矩阵

$Qdiag(B, O)Q^T$ 。计算出的近似 Schur 分解 $\hat{Q}diag(\hat{B}, O)\hat{Q}^T$ 与已知的分解

$Qdiag(B, O)Q^T$ 之间的绝对误差作为向前误差, 即定义两个误差

$$o.e = \|\hat{Q}^T \hat{Q} - I\|_2,$$

$$a.f.e = \frac{1}{2} \left\| \begin{bmatrix} B & \\ & O_{(n-s) \times (n-s)} \end{bmatrix} - \begin{bmatrix} \hat{B} & \\ & O_{(n-\hat{s}) \times (n-\hat{s})} \end{bmatrix} \right\|_\infty = \max_{1 \leq i \leq \max(s, \hat{s})} |t_i - \hat{t}_i|,$$

其中 t_i 或 \hat{t}_i 指标超过的部分表示零。 $o.e$ 反映了 Schur 分解算法给出的结果的正交性效果, $a.f.e$ 反映了计算结果的精度。

定义 1.3 元素相对向后误差 (e.b.e) : 设需要被 Schur 分解的矩阵为 $A_0 = (a_{ij}^0)$,

分解后的得到 Schur 分解为 $A_f = QB_f Q^T = (a_{ij}^f)$ 定义为

$$e.b.e = \max_{i,j} \frac{|a_{ij}^0 - a_{ij}^f|}{|a_{ij}^0|}, \quad \text{if } |a_{ij}^0| > \varepsilon.$$

(1) 算例 1.1

为体现一般性, 我们选择随机矩阵进行实验。在 $[-1, 1]$ 上随机产生一个 n 阶反对称矩阵 $A = (a_{ij})$, 其中 a_{ij} 为 $[-1, 1]$ 之间的随机数。具体产生的方法为: 先随机产生 A 的上三角部分, 即 $a_{ij} = 2 * rand - 1, (i < j)$, 再将其下三角部分赋值为 $a_{ij} = -a_{ji}, (i > j)$. 最后将对角线上的元素赋值为 0。

在数值实验过程中, 我们取 $\varepsilon = 1.0 \times 10^{-15}$ 作为计算精度。

表 1.1 给出的是我们设计的反对称矩阵的 Schur 分解算法 (简称 ASM-Schur 算法) 的数值结果, 为便于对比, 我们也列出了 matlab 中的 schur 函数的计算结

果。需要指出的是，我们的数值结果均为 8 次随机矩阵分解实验得到的结果的平均值。

表 1.1 ASM-Schur 算法求解随机反对称矩阵的结果分析

阶数 n		20	40	100	200	400	800	1500
ASM-Schur	m.b.e	6.89e-12	5.62e-11	7.29e-10	3.23e-9	9.04e-9	6.64e-8	9.49e-7
	e.b.e	1.59e-11	7.13e-11	1.77e-10	8.22e-9	4.26e-8	3.10e-7	7.11e-6
	CPU(s)	0.01152	0.03194	0.1223	0.4547	1.9281	7.2829	22.7532
matlab schur	m.b.e	2.51e-14	3.66e-13	1.05e-11	9.64e-11	4.36e-10	7.63e-9	1.18e-7
	e.b.e	1.26e-12	7.12e-12	3.89e-11	1.99e-10	6.43e-10	5.06e-8	8.41e-7
	CPU(s)	0.00051	0.00214	0.02568	0.13814	0.79543	4.43541	25.6159

我们绘制误差曲线（自然对数坐标系下）及计算时间曲线分别如图 1.1 和图 1.2 所示。从表 1.1 和图 1.1 我们可以分析得出，ASM-Schur 算法的计算精度不及 matlab 中的 schur 函数，但是计算结果还是比较满意的，对于高阶的问题，它也能给出较为精确的结果。

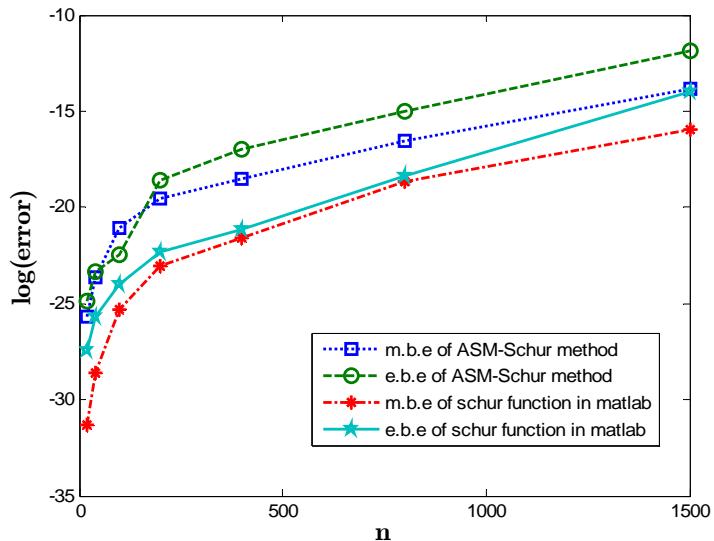


图 1.1 自然对数坐标系下：计算结果的误差关于矩阵的阶 n 的曲线

从表 1.1 和图 1.2 我们可以看到，在 n 比较小的情况下，我们实现的 ASM-Schur 算法速度没有 matlab 中的 schur 函数快；随着阶 n 的增加，ASM-Schur 算法的耗时增加比 schur 函数增加的要缓慢些；对于高阶问题，ASM-Schur 算法耗时要小一些。

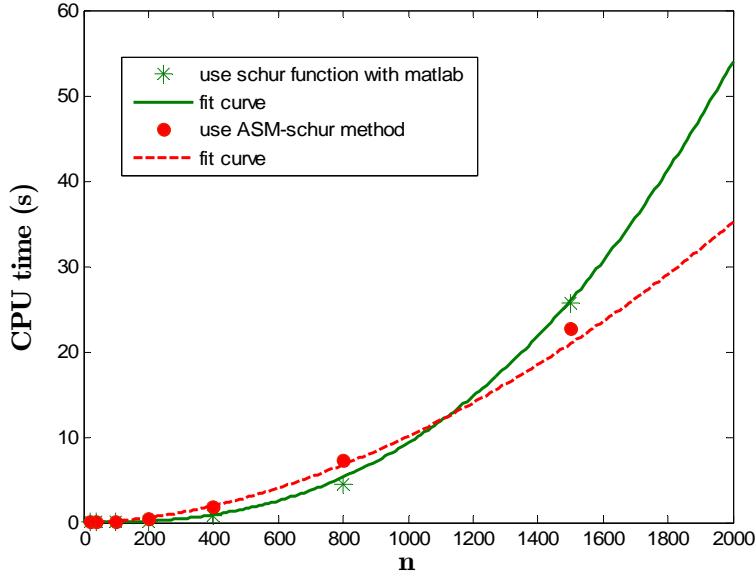


图 1.2 计算的 CPU 时间（秒）关于矩阵的阶 n 的曲线（实线为拟合曲线）

(2) 算例 1.2

对于给定的正偶数 n ，用命令 $t_i = 2 * rand - 1$ 在 $[-1,1]$ 上随机产生 $n / 2$ 个随机数 $t_i (i = 1, 2, \dots, n)$ ，并由此构造矩阵

$$B = \text{diag} \left(\begin{bmatrix} 0 & -t_1 \\ t_1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -t_2 \\ t_2 & 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 & -t_{n/2} \\ t_{n/2} & 0 \end{bmatrix} \right).$$

然后我们用 **matlab** 随机产生一个正交矩阵 Q （命令 `orth(randn(n,n))`）。这样我们就构造了一个随机的反对称矩阵 $A = QBQ^T$ ，并知道它的 `schur` 分解就是 QBQ^T 。

接下来，我们用 A 作为算例进行数值实验，并用向前误差（**a.f.e**）与正交性误差（**o.e**）进行计算结果的误差分析。在数值实验过程中，我们取 $\varepsilon = 2.2 \times 10^{-16}$ （机器精度）作为计算精度。

表 1.2 给出的是 **ASM-Schur** 算法的数值结果，为便于对比，我们也列出了 **matlab** 中的 `schur` 函数的计算结果。这里，我们的数值结果均为 8 次随机矩阵分解实验得到的结果的平均值。

表 1.2 ASM-Schur 算法求解随机反对称矩阵的结果分析

阶数 n		20	40	100	200	400	800	1500
ASM-Schur	a.f.e	4.28e-15	5.87e-15	9.53e-15	1.89e-14	2.76e-14	4.35e-14	5.77e-14
	o.e	8.54e-15	1.15e-14	1.79e-14	2.84e-14	6.17e-14	7.79e-14	9.95e-14
	CPU(s)	0.01068	0.02988	0.13125	0.42277	1.89632	7.46751	21.9695
matlab schur	a.f.e	1.47e-15	2.35e-15	3.43e-15	6.13e-15	9.27e-15	1.25e-14	1.41e-14
	o.e	3.73e-15	5.74e-15	8.10e-15	1.29e-14	2.01e-14	2.89e-14	3.17e-14
	CPU(s)	0.00052	0.00244	0.02714	0.14493	0.78233	4.69161	24.1361

我们绘制误差曲线（自然对数坐标系下）及计算时间曲线分别如图 1.3 和图 1.4 所示。从表 1.2 和图 1.3 我们可以分析得出，随着阶 n 的增加，计算结果的误差也增加；此外，结果显示 ASM-Schur 算法的计算精度可与 matlab 中的 schur 函数相媲美，但还是略有差距；计算结果还是比较满意的，对于高阶的问题，它也能给出较为精确的结果。

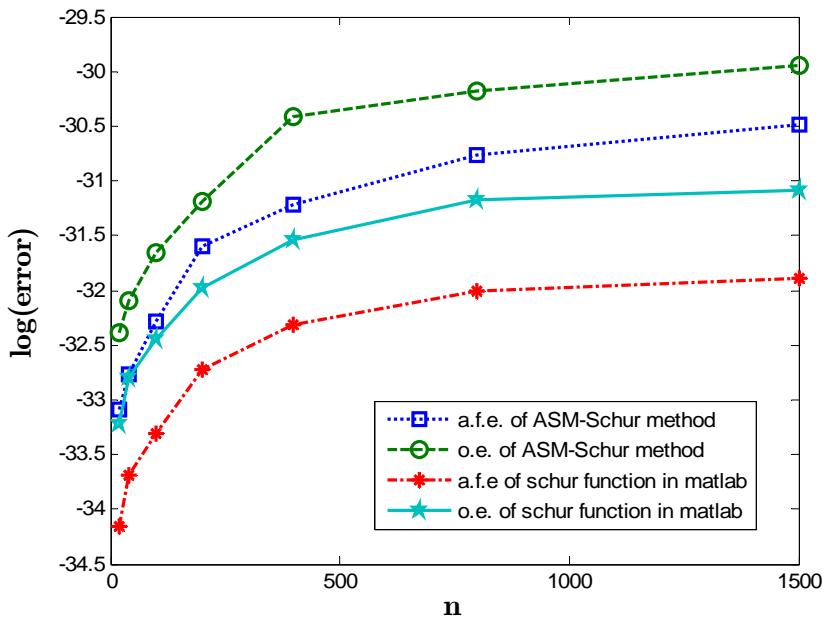


图 1.3 自然对数坐标系下：计算结果的误差关于矩阵的阶 n 的曲线

从表 1.2 和图 1.4 我们可以看到，在 n 比较小的情况下，我们实现的 ASM-Schur 算法速度没有 matlab 中的 schur 函数快；随着阶 n 的增加，ASM-Schur 算法的耗时增加比 schur 函数增加的要缓慢些；对于高阶问题，ASM-Schur 算法耗时要小一些。

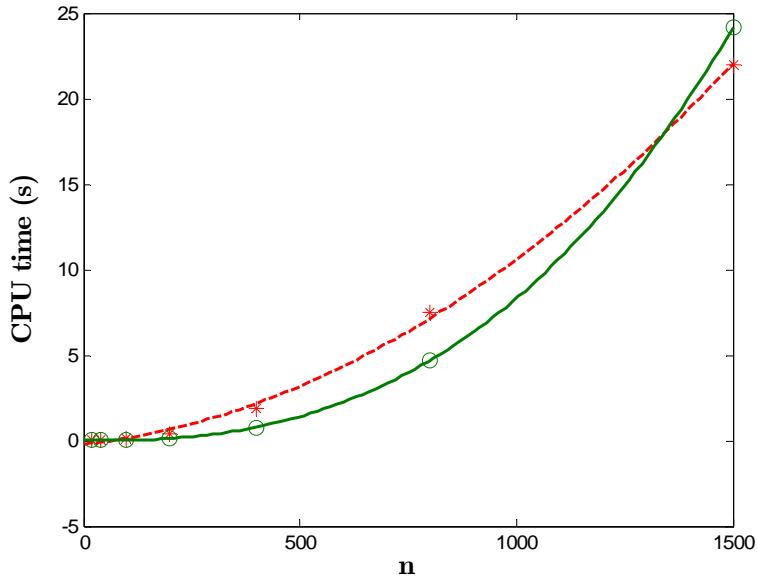


图 1.4 计算的 CPU 时间 (秒) 关于阶 n 的曲线 (实线为拟合曲线)

(3) 单位移与双位移的对比分析

前面我们提到, 反对称矩阵的 Schur 分解算法 (简称 ASM-Schur 算法) 除了采用已经介绍的单位移 (对反对称矩阵即 0 位移) 隐式 QR 分解来进行, 也可采用双位移的隐式 QR 分解来进行。

下面我们用算例 1.1 来测试两种位移计算的效果。在数值实验过程中, 我们取 $\varepsilon = 1.0 \times 10^{-15}$ 作为计算精度。

表 1.3 给出的是单位移的 ASM-Schur 算法(直接用前面算例 1.1 计算的结果) 和双位移 ASM-Schur 算法的数值结果对比。我们这里列出的数值结果均为 8 次随机矩阵分解实验得到的结果的平均值。

表 1.3 单位移与双位移的 ASM-Schur 算法求解随机反对称矩阵的结果对比

阶数 n		20	40	100	200	400	800	1500
单位移	m.b.e	6.89e-12	5.62e-11	7.29e-10	3.23e-9	9.04e-9	6.64e-8	9.49e-7
	e.b.e	1.59e-11	7.13e-11	1.77e-10	8.22e-9	4.26e-8	3.10e-7	7.11e-6
	CPU(s)	0.01152	0.03194	0.1223	0.4547	1.9281	7.2829	22.7532
双位移	m.b.e	4.97e-12	3.27e-11	4.90e-10	1.33e-9	7.36e-9	4.11e-8	6.78e-7
	e.b.e	1.09e-11	5.75e-11	9.56e-11	3.89e-9	2.41e-8	1.32e-7	2.41e-6
	CPU(s)	0.02625	0.04886	0.1527	0.4588	2.0292	7.5785	20.9631

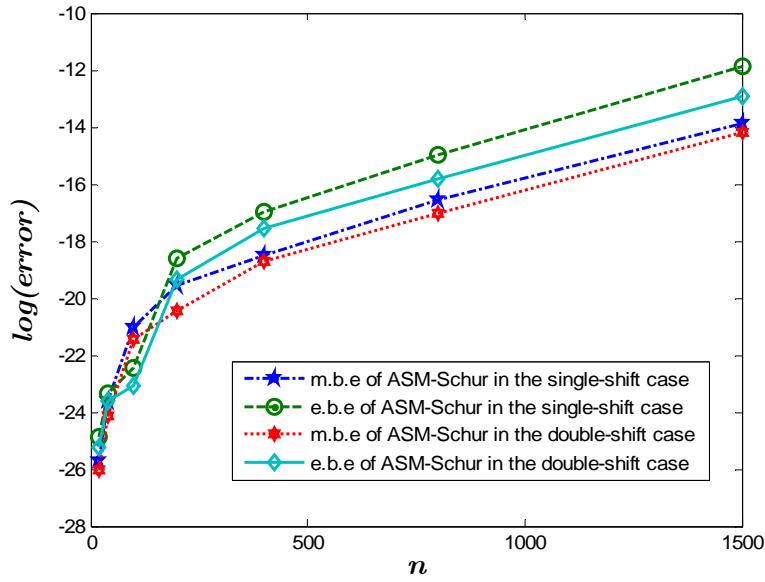


图 1.5 自然对数坐标系下：单位移与双位移的误差关于矩阵的阶 n 的曲线

我们绘制误差曲线（自然对数坐标系下）及计算时间曲线分别如图 1.5 和图 1.6 所示。从表 1.3 和图 1.5 我们可以分析得出，双位移 ASM-Schur 算法的计算精度的计算精度比单位移 ASM-Schur 算法的计算精度要高，高将近 1.5-2 倍。

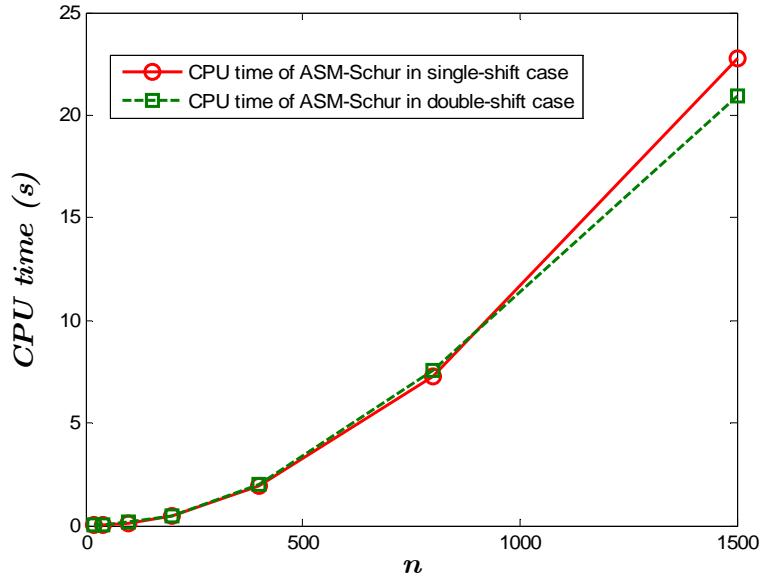


图 1.2 计算的 CPU 时间（秒）关于矩阵的阶 n 的曲线

从表 1.3 和图 1.6 我们可以看到，计算的 CPU 时间关于矩阵的阶 n 呈现近似多项式增长关系。在 n 比较小的情况下，双位移的 ASM-Schur 算法速度没有单

位移的 **ASM-Schur** 算法快；随着阶 n 的增加，双位移的 **ASM-Schur** 算法的耗时增加比单位移的 **ASM-Schur** 算法增加的要缓慢些；对于高阶问题，双位移的 **ASM-Schur** 算法耗时要小一些。

因此，对于要求计算结果高精度的反对称矩阵 **Schur** 分解的实际问题，采用位移的 **ASM-Schur** 算法效果比单位移的 **ASM-Schur** 算法好；对于低阶的要求快速计算的问题，采用单位移 **ASM-Schur** 算法快些，而对于高阶问题，采用双位移的 **ASM-Schur** 算法快些。

2 矩阵 $D + \rho z z^T$ 的特征值与特征向量的计算

2.1 问题的提出

设 $D = \text{diag}(d_1, d_2, \dots, d_n) \in R^{n \times n}$, $\rho \in R$, $z \in R^n$ 。给出一种计算 $D + \rho z z^T$ 的特征值与特征向量的数值方法，并用数值实验的方法来说明你所出的算法的数值特性。

为方便表示，我们记 $H = D + \rho z z^T$ 。事实上，求矩阵 H 的特征值和特征向量的问题，等价为求矩阵 H 的谱分解。设矩阵 H 的谱分解为

$$V^T H V = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n),$$

其中 $V = [v_1, v_2, \dots, v_n]$ 为正交矩阵， $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ 为 H 的特征向量。

因此，下面我们讨论如何求 H 的谱分解。

2.2 基本理论的建立

以下只考虑 $\rho \neq 0$ 的情况，对于 $\rho = 0$ 的情况，无需计算，结果是显然的。

在特殊情况下，关于矩阵 H 的特征值和特征向量，我们有如下的结果。

定理 2.1 设构成矩阵 H 的实数 ρ, z_j, d_j 满足

$$\rho \neq 0; z_j \neq 0, j = 1, 2, \dots, n; d_1 > d_2 > \dots > d_n.$$

则有

(1) $\lambda_1, \lambda_2, \dots, \lambda_n$ 正好是函数

$$f(\lambda) = 1 + \rho z^T (D - \lambda I)^{-1} z \quad (2.1)$$

的 n 个零点；

(2) 当 $\rho > 0$ 时，有

$$\lambda_1 > d_1 > \lambda_2 > d_2 > \dots > \lambda_n > d_n;$$

而当 $\rho < 0$ 时，有

$$d_1 > \lambda_1 > d_2 > \lambda_2 > \dots > d_n > \lambda_n;$$

(3) 对应于 λ_j 的单位特征向量为

$$v_j = \frac{(D - \lambda_j I)^{-1} z}{\|(D - \lambda_j I)^{-1} z\|_2}, i = 1, 2, \dots, n. \quad (2.2)$$

证明 由假设知 $(D + \rho z z^T) v_i = \lambda_i v_i$, $\|v_i\|_2 = 1$. 可以证明 $D - \lambda_i I$ 非奇异。若不然, 则存在 i 使得 $e_i^T (D - \lambda_i I) = 0$, 从而 $-\rho z^T v_i e_i^T z = e_i^T (D - \lambda_i I) v_i = 0$, 注意到又有 $z^T v_i \neq 0$, 故 $e_i^T z = 0$, 矛盾于 $z_j \neq 0, j = 1, 2, \dots, n$ 。利用 $D - \lambda_i I$ 非奇异和 $(D + \rho z z^T) v_i = \lambda_i v_i$, 我们有

$$v_i = -\rho z^T v_i (D - \lambda_i I)^{-1} z, i = 1, 2, \dots, n. \quad (2.3)$$

由于 $-\rho z^T v_i \in R$, 所以结论 (3) 得证。

由 (2.3) 式两边左乘 z^T , 我们有

$$1 = -\rho z^T (D - \lambda_i I)^{-1} z$$

此即 $f(\lambda_i) = 0$, 故 λ_i 是 $f(\lambda)$ 的零点。下证 $f(\lambda)$ 正好有 n 个零点。

设 z_j 表示 z 的第 j 个分量, 则

$$f(\lambda) = 1 + \rho \sum_{j=1}^n \frac{z_j^2}{d_j - \lambda}.$$

于是

$$f'(\lambda) = \rho \sum_{j=1}^n \frac{z_j^2}{(d_j - \lambda)^2}.$$

这表明 $f(\lambda)$ 在任意区间 (d_{i+1}, d_i) 之间是严格单调的: 当 $\rho > 0$ 时, 单调增加; 当 $\rho < 0$ 时, 单调减少。故 $f(\lambda)$ 正好有 n 个零点。结论 (1) 得证。

此外, 以上分析表明, 当 $\rho > 0$ 时, $f(\lambda)$ 的 n 个零点正好分别位于 n 个区间

$$(d_n, d_{n-1}), \dots, (d_2, d_1), (d_1, +\infty);$$

当 $\rho < 0$ 时, $f(\lambda)$ 的 n 个零点正好分别位于 n 个区间

$$(-\infty, d_n) (d_n, d_{n-1}), \dots, (d_2, d_1).$$

从而结论 (2) 得证。

由上述定理的证明，我们可以画出当 $\rho > 0$ 时由 (2.1) 所定义的函数 $f(\lambda)$ 的示意图如图 2.1 所示（当 $\rho < 0$ 时类似，在此省略）。

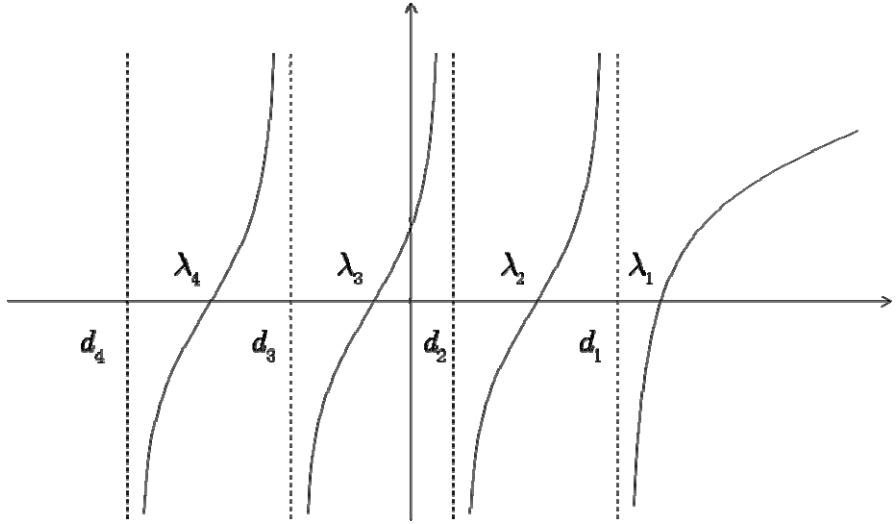


图 2.1 当 $\rho > 0$ 时函数 $f(\lambda)$ 的示意图

若我们知道实数 ρ 以及一组实数 $\{\hat{\lambda}_j\}$ 和对角矩阵 $D = \text{diag}(d_1, d_2, \dots, d_n)$ 满足分隔性条件，则如下定理说明，我们可以构造一个形如 $\hat{H} = D + \rho z z^T$ 的矩阵 \hat{H} 使得 $\{\hat{\lambda}_j\}$ 正好是 \hat{H} 的所有特征值。

定理 2.2 设给定的实数 ρ 以及 n 个实数 $\{\hat{\lambda}_j\}$ 和对角矩阵 D 满足分隔性条件，即当 $\rho > 0$ 时，有

$$\hat{\lambda}_1 > d_1 > \hat{\lambda}_2 > d_2 > \dots > \hat{\lambda}_n > d_n;$$

当 $\rho < 0$ 时，有

$$d_1 > \hat{\lambda}_1 > d_2 > \hat{\lambda}_2 > \dots > d_n > \hat{\lambda}_n.$$

则必存在一个形如 $\hat{H} = D + \rho z z^T$ 的矩阵 \hat{H} 使得 $\{\hat{\lambda}_j\}$ 正好是 \hat{H} 谱集，而且 \hat{z} 由下面的公式给出

$$\hat{z}_i = \pm \left(\frac{\hat{\lambda}_i - d_i}{\rho} \prod_{j=1}^{i-1} \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \prod_{j=i+1}^n \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \right)^{\frac{1}{2}}. \quad (2.4)$$

\hat{z}_i 的符号可以任取。

证明：若 \widehat{H} 存在，则必有

$$\text{tr} \widehat{H} = \hat{\lambda}_1 + \hat{\lambda}_2 + \cdots + \hat{\lambda}_n,$$

$$P_{\widehat{H}}(\lambda) = \det(\lambda I - \widehat{H}) = \prod_{j=1}^n (\lambda_j - \hat{\lambda}_j) = P(\lambda) \quad (2.5)$$

直接计算有

$$\begin{aligned} P_{\widehat{H}}(\lambda) &= \det(\lambda I - \widehat{H}) = \det\left(\lambda I - D - \rho z z^T\right) \\ &= \det(\lambda I - D) \det\left(I - \rho(\lambda I - D)^{-1} z z^T\right) \\ &= \prod_{j=1}^n (\lambda - d_j) \left(1 - \rho z (\lambda I - D)^{-1} z^T\right) \\ &= \left(1 - \rho \sum_{j=1}^n \frac{z_j}{\lambda - d_j}\right) \prod_{j=1}^n (\lambda - d_j). \end{aligned}$$

取 $\lambda = d_i$ ，整理即得证 (2.4) 式。分割性可以保证 $\frac{\hat{\lambda}_i - d_i}{\rho} \prod_{j=1}^{i-1} \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \prod_{j=i+1}^n \frac{d_i - \hat{\lambda}_j}{d_i - d_j}$

是正数。

下面证明 $\{\hat{\lambda}_j\}$ 正好是 \widehat{H} 谱集。这只需证明对一切的 λ 恒成立 (2.5) 即可。

注意到

$$P(\lambda) = \prod_{j=1}^n (\lambda - \hat{\lambda}_j) = \lambda^n - \left(\sum_{j=1}^n \hat{\lambda}_j \right) \lambda^{n-1} + \cdots,$$

$$P_{\widehat{H}}(\lambda) = \prod_{j=1}^n (\lambda - \hat{\lambda}_j) = \lambda^n - \left(\text{tr} \widehat{H} \right) \lambda^{n-1} + \cdots,$$

这两个多项式前面两项相等，而且他们有 $n-1$ 个互补相同的数 d_2, \dots, d_n 上的取值

也相同，所以，这两个多项式恒等，得证。

2.3 基于二分法的特征值计算

下面我们来考虑形如 $D + \rho z z^T$ 的矩阵 H 的特征值计算问题。这里我们假设构成 H 的 D 和 z 满足

$$d_j - d_{j+1} \geq \varepsilon, j = 1, 3, \dots, n-1, \quad |z_i| \geq \varepsilon, i = 1, 2, \dots, n$$

其中 $\varepsilon = (\|D\|_2 + |\rho| \|z\|_2) \mathbf{u}$, 这里 \mathbf{u} 表示机器精度。

理论上讲, 我们可以根据定理 2.1 的结论 (1), 直接求 $f(\lambda)$ 的根得到特征值。但是这样做的结果是, 我们并没有充分利用该问题的特点。

从数值计算的角度, 根据 $f(\lambda)$ 的特点, 我们需要对特征方程做一点小小的变形, 来减小计算误差, 增加精度。下面我们将对 $\rho > 0$ 的情形分三种情况来讨论, $\rho < 0$ 的情形类似。

情形一 $2 \leq i \leq n$ 且 $f\left(\frac{d_i + d_{i-1}}{2}\right) > 0$ 。此时特征方程的根 λ_i 位于 $\left(d_i, \frac{d_i + d_{i-1}}{2}\right)$ 内。这种情况我们对自变量作平移, 使坐标原点平移至 d_i , 即作变换 $\lambda = \mu + d_i$ 。令 $g_i(\mu) = f(\mu + d_i)$, 则求 λ_i 的问题转化为求方程

$$g_i(\mu) = 0$$

位于 $\left(0, \frac{d_{i-1} - d_i}{2}\right)$ 内之根 μ_i 的问题。一旦 μ_i 已经求得, 则所需的 λ_i 即为 $\mu_i + d_i$ 。

直接计算有

$$g_i(\mu) = f(\mu + d_i) = 1 + \sum_{j=1}^{i-1} \frac{z_j^2}{\delta_j - \mu} + \sum_{j=i}^n \frac{z_j^2}{\delta_j - \mu}$$

其中 $\delta_j = d_j - d_i$ 。

这样作的好处是, 对每个 $\mu \in \left(0, \frac{\delta_{i-1}}{2}\right)$, $g_i(\mu)$ 的值都可以计算得相对精度很高。

注意到, 若 λ 与 d_i 十分接近, 计算 $d_i - \lambda$ 会发生相消的问题, 进而产生较大的相对误差, 而这一误差就会导致 $\frac{z_i^2}{d_i - \lambda}$ 的计算值与真值相差甚远。但是平移之后就能很好的避免这种误差的引起。

情形二 $2 \leq i \leq n$ 且 $f\left(\frac{d_i + d_{i-1}}{2}\right) < 0$ 。此时特征方程的根 λ_i 位于 $\left(\frac{d_i + d_{i-1}}{2}, d_{i-1}\right)$ 内。这种情况我们对自变量作平移, 使坐标原点平移至 d_{i-1} , 即作变换

$\lambda = \mu + d_{i-1}$ 。令 $g_i(\mu) = f(\mu + d_{i-1})$, 则求 λ_i 的问题转化为求方程 $g_i(\mu) = 0$ 位于

$$\left(\frac{d_i - d_{i-1}}{2}, 0 \right)$$

内之根 μ_i 的问题。

直接计算有

$$g_i(\mu) = f(\mu + d_i) = 1 + \sum_{j=1}^{i-1} \frac{z_j^2}{\delta_j - \mu} + \sum_{j=i}^n \frac{z_j^2}{\delta_j - \mu}$$

其中 $\delta_j = d_j - d_{i-1}$ 。

只要求得 $g_i(\mu)$ 位于 $\left(\frac{\delta_i}{2}, 0 \right)$ 内的根 μ_i , 则所需的 λ_i 即为 $\mu_i + d_{i-1}$ 。

情形三 $i = 1$ 。此时作变换 $\lambda = \mu + d_1$, 则有

$$g_1(\mu) = f(\mu + d_1) = 1 + \sum_{j=1}^n \frac{z_j^2}{\delta_j - \mu}$$

其中, $\delta_j = d_j - d_1$ 。

定义 $\delta_0 = |\rho| \|z\|_2^2$, 则可以证明

$$0 < \lambda_1 - d_1 < \delta_0.$$

事实上, 对 H 和 D 用 Weyl 定理, 我们有

$$|\lambda_1 - d_1| \leq \|H - D\|_2 = \|\rho z z^T\| = |\rho| \|z\|_2^2.$$

注意到 $\lambda_1 - d_1 > 0$, 这样我们就证明了

$$0 < \lambda_1 - d_1 < \delta_0.$$

因此在这种情况下, 我们只要求出 $g_1(\mu)$ 在 $(0, \delta_0)$ 内的根 μ_1 , 则 $\lambda_1 = d_1 + \mu_1$ 就是所要求的最大特征值。

注记 2.1 如果 $|\rho| \|z\|_2^2$ 比较大, 则区间 $(0, \delta_0)$ 也比较大, 因而在其中用二分法

精确地寻找 μ_1 是需要很多步得, 为了快速寻找, 我们可以缩短以上搜索区间。

注意到, 由矩阵的迹与特征值之间地关系, 我们有

$$\lambda_1 + \lambda_2 + \cdots + \lambda_n = \text{tr} H = d_1 + d_2 + \cdots + d_n + |\rho| \|z\|_2^2.$$

理论上可以证明, $\lambda_j, j \geq 2$ 的计算误差不超过 $8\varepsilon + \varepsilon_0$, 这里 $\varepsilon = (\|D\|_2 + |\rho| \|z\|_2) \mathbf{u}$, ε_0 为二分法最后终止的区间长度。则我们可以取上面的 δ_0 为

$$\delta_0 = \min \left(\|\rho\| \|z\|_2^2, \|\rho\| \|z\|_2^2 + \sum_{j=2}^n (d_j - \lambda_j + 8\varepsilon + \varepsilon_0) \right).$$

2.4 特征向量的计算

假定我们已经用上面所介绍的方法求得 H 的 n 个近似特征值 $\hat{\lambda}_1, \hat{\lambda}_2, \dots, \hat{\lambda}_n$, 下面我们来计算 $\hat{\lambda}_j$ 对应的近似特征向量 \hat{v}_j 。

根据定理 2.2, 一个自然的想法就是利用 (2.2) 式来计算 v_j , 但是这样做会使计算的 \hat{v}_j 与真实的 v_j 相差非常大, 特别是属于两个不同的特征值所对应的特征向量的计算结果可以是几乎平行的。事实上, 当 d_j 与 λ_j 很靠近时, 尽管 $\hat{\lambda}_j$ 与 λ_j 很靠近, 但是 $\frac{z_j^2}{d_j - \lambda_j}$ 与 $\frac{z_j^2}{d_j - \hat{\lambda}_j}$ 就可能相差很大。

解决这一问题的方法是, 由计算出的特征值, 利用定理 2.2, 构造一个形如 $D + \rho z z^T$ 的矩阵 \hat{H} , 然后用矩阵 \hat{H} 的特征向量来作为 H 的近似, 具体计算步骤如下:

第一步 计算

$$\hat{z}_i = \text{sign}(z_i) \left(\frac{\hat{\lambda}_i - d_i}{\rho} \prod_{j=1}^{i-1} \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \prod_{j=i+1}^n \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \right)^{\frac{1}{2}}, k = 1, 2, \dots, n.$$

第二步 计算

$$\hat{v}_j = \frac{(D - \lambda_j I)^{-1} \hat{z}}{\|(D - \lambda_j I)^{-1} \hat{z}\|_2}, i = 1, 2, \dots, n.$$

2.5 一般情况的前期处理

对于一般的形如 $D + \rho z z^T$ 的矩阵 H , 不一定满足

$$\rho \neq 0; z_j \neq 0, j = 1, 2, \dots, n; d_1 > d_2 > \dots > d_n.$$

此时，我们要采取一定的预处理技术。

在实际计算时，需事先给定一个准则，来判断何时两个数视为相等，何时一个数视作零。我们取

$$\varepsilon = (\|D\|_2 + |\rho| \|z\|_2) \mathbf{u}$$

来作为误差限，其中 \mathbf{u} 表示机器精度。当 $|d_j - d_i| < \varepsilon$ 时，就视为它们相等；而当 $|z_i| < \varepsilon$ 时就视为它等于零。

下面我们来阐述如何对于一般的形如 $D + \rho z z^T$ 的矩阵 H 进行预处理使其满足以上特殊情形的计算条件。

(1) 首先，查找如果有两个指标 $i < j$ 使得 $d_i = d_j$ ，则我们可以取 (i, j) 坐标平面的平面旋转变换 $P_{ij} = G(i, j, \theta)$ 使 $P_{ij}^T z$ 的第 j 个分量为零，而且易证，此时有 $P_{ij}^T D P_{ij} = D$ ，这样进行若干步，我们可以找到一个由一些平面旋转变换的乘积构成的正交矩阵 V_1 使得 $V_1^T D V_1 = D$ ，而且 $V_1^T z = (\xi_1, \xi_2, \dots, \xi_n)^T$ 满足：若 $\xi_i \xi_j \neq 0, i \neq j$ ，则必有 $d_i \neq d_j$ 。

(2) 然后，对 $V_1^T z$ 的分量进行若干次两两对换，可使其所有不为零的分量都位于它的前面，即可以找到一个排列方阵 P_1 ，使

$$P_1 V_1^T z = (\xi_{\pi(1)}, \xi_{\pi(2)}, \dots, \xi_{\pi(n)})^T,$$

$$\xi_{\pi(i)} \neq 0, i = 1, 2, \dots, r,$$

$$\xi_{\pi(i)} = 0, i = r + 1, \dots, n.$$

其中 π 是 $\{1, 2, \dots, n\}$ 的一个排列。再由 P_1 的取法我们知道，矩阵

$$P_1^T V_1^T D V_1 P_1 = P_1^T D P_1 = \text{diag}(d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(n)})$$

的前 r 个对角元互不相同。

(3) 最后再对 $d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(r)}$ 进行若干次对换，使他们按从大到小的次序排列，即可以找到一个 r 阶排列方阵 P_2 使得

$$P_2^T \text{diag}(d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(r)}) P_2 = \text{diag}(\gamma_1, \gamma_2, \dots, \gamma_r),$$

其中 $\gamma_1, \gamma_2, \dots, \gamma_r$ 是由 $d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(r)}$ 从大到小排序得到。

综上所述，我们得到，对于任意的形如 $D + \rho zz^T$ 的矩阵 H ，可以构造一个正交矩阵 V ，使

$$V^T HV = \begin{pmatrix} H_1 & 0 \\ 0 & 0 \\ r & n-r \end{pmatrix}$$

其中 $H_1 = D_1 + \rho ww^T$ ，这里

$$\begin{aligned} D_1 &= \text{diag}(\gamma_1, \gamma_2, \dots, \gamma_r); \\ D_2 &= \text{diag}(d_{\pi(r+1)}, d_{\pi(r+2)}, \dots, d_{\pi(n)}); \\ w &= (\xi_1, \xi_2, \dots, \xi_r), \xi_i \neq 0, i = 1, 2, \dots, r. \end{aligned}$$

因此，要求 H 的谱分解，我们只需求 H_1 的谱分解即可，而后者是我们已经讨论过的特殊情况，可快速稳定地求解。

2.6 数值特性分析

2.6.1 算法的复杂度分析

(1) 满足定理 2.1 的特殊情形

对于区间 $[d_i, d_{i-1}]$ 里的特征值 λ_i ，计算它的运算量为

$$\left\lfloor \log_2 \left((d_{i-1} - d_i) / \varepsilon \right) \right\rfloor \cdot O(n)$$

因此，计算所有特征值的运算量不超过

$$\max_i \left\lfloor \log_2 \left((d_{i-1} - d_i) / \varepsilon \right) \right\rfloor \cdot O(n^2) \leq \left\lfloor \log_2 \left(2 \|d\|_\infty / \varepsilon \right) \right\rfloor \cdot O(n^2)$$

其中 ε 表示对特征值的精度要求。

根据 2.3 节我们可以看出，根据特征值计算特征向量的第一步需要 $4n - 3$ 次加减运算和 $3n - 2$ 次乘除运算和一次开方运算，第二步需要 $2n - 1$ 次加减运算和 $2n$ 次乘除运算和一次开方运算。所以计算一个特征值的运算量为

$$4n - 3 + 3n - 2 + 1 + 2n - 1 + 2n + 1 = 9n - 4$$

因此，计算所有的特征值的运算量为 $9n^2 - 4n = O(n^2)$ 。

(2) 一般的形如 $D + \rho zz^T$ 的矩阵 H

对于一般的情形，预处理过程还需要一些运算量。预处理的第一步运算量不超过 $O(n - r)$ ；第二步用冒泡法，无需加减运算，只需交换赋值，大约需要交换 $O(n - r)$ 次；最后第三步排序运算量为 $O(r^2)$ 次。所以总的运算量不超过 $O(n^2)$ 。但是综合前面的考虑，此时计算特征值和特征向量的运算量均为 $O(r^2)$ 。所以总的运算量 $\zeta(n)$ 不超过 $O(n^2)$ 。

2.6.2 计算特征值的稳定性分析

前面我们提到，当 d_j 与 λ_j 很靠近时，直接计算矩阵 H 的特征值会带来很大的误差，而我们提出的算法精度很高，是数值稳定的。下面进行分析。

通过分析方程求解过程中的停止准则，可以证明

$$\|zz^T - \hat{z}\hat{z}^T\|_2 \leq O(\varepsilon) (\|D\|_2 + \|z\|_2^2).$$

这表明 $D + \rho zz^T$ 和 $D + \rho \hat{z}\hat{z}^T$ 非常靠近，因此可以将 $D + \rho \hat{z}\hat{z}^T$ 的特征值和特征向量稳定的近似。

接着我们注意到定理 2.2 中 \hat{z} 的公式只需要浮点数 $d_j - d_i$ 和 $\hat{\lambda}_j - d_i$ 的差、这些差的乘积和商、以及一个平方根。我们假设浮点算术运算是足够准确的，以至于对于一切的加减乘除运算 $\odot \in \{+, -, \times, /\}$ 有

$$\text{float}(a \odot b) = (a \odot b)(1 + \delta),$$

以及 $\text{sqrt}(a) = \sqrt{a}(1 + \delta)$ ，这里 $|\delta| = O(\varepsilon)$ 。那么这个公式可以计算到很高的相对精度，特别的，不计上溢和下溢可以证明

$$\text{float} \left(\left(\prod_{j=1}^{i-1} \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \prod_{j=i+1}^n \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \right)^{\frac{1}{2}} \right) = (1 + (4n - 2)\delta) \left(\prod_{j=1}^{i-1} \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \prod_{j=i+1}^n \frac{d_i - \hat{\lambda}_j}{d_i - d_j} \right)^{\frac{1}{2}}.$$

类似地，可以分析得出特征向量也可以计算到高的相对精度。

总之，假若浮点算术运算是足够精确的话，我们的算法计算矩阵 $D + \rho \hat{z}\hat{z}^T$ 的特征值和特征向量很精确，而它们与原矩阵的特征值和特征向量只有一点误差。这意味着我们的算法是数值稳定的。

2.6.3 数值实验与结果分析

在这一小节，我们选择若干算例，通过编写计算机程序实现矩阵 $D + \rho zz^T$ 的特征值和特征向量的计算，分析我们设计的算法的数值特性（计算精度、CPU 时间等），并将我们的数值结果与著名的计算特征值和特征向量的 QR 方法进行对比。

为便于分析，我们定义几种误差，来刻画我们的算法计算特征值和特征向量的精度。

首先，针对特征值的误差分析，我们引入绝对误差、相对误差和向后误差的概念。

定义 2.1 绝对误差 (a.e.) 定义矩阵 H 近似特征值的绝对误差为：所有的准确特征值 $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_n)$ 与所有近似的特征值 $\hat{\lambda} = (\hat{\lambda}_1, \hat{\lambda}_2, \dots, \hat{\lambda}_n)$ 之差的范数 $\|\lambda - \hat{\lambda}\|$ 。这里的范数可以取 2 范数 $\|\bullet\|_2$ 或无穷范数 $\|\bullet\|_\infty$ ，我们的讨论中均取 2 范数 $\|\bullet\|_2$ 分析。

用绝对误差来刻画近似特征值的精确程度是有局限性的，因为它没有反映出它在精确特征值中所占的比例。因此，我们通常考虑用相对误差来刻画：

定义 2.2 相对误差 (r.e.) 定义为矩阵 H 近似特征值的相对误差为

$$\text{r.e.}(H, \lambda) = \frac{\|\lambda - \hat{\lambda}\|}{\|\lambda\|},$$

其中，范数 $\|\bullet\|$ 可以取 2 范数 $\|\bullet\|_2$ 或无穷范数 $\|\bullet\|_\infty$ ，我们的讨论中均取 2 范数 $\|\bullet\|_2$ 分析。

我们知道，矩阵 H 的特征值是函数

$$f(\lambda) = 1 + \rho \sum_{j=1}^n \frac{z_j^2}{d_j - \lambda}.$$

的零点。据此，我们定义向后误差。

定义 2.3 向后误差 (b.e.) 定义为矩阵 H 近似特征值的向后误差为

$$\text{b.e.}(H, \lambda) = \|f(\hat{\lambda})\|_\infty = \max_{1 \leq i \leq n} |f(\hat{\lambda}_i)|,$$

其中 $f(\hat{\lambda}) = (f(\hat{\lambda}_1), f(\hat{\lambda}_2), \dots, f(\hat{\lambda}_n))$ 。

接下来，我们针对近似特征向量的误差引入绝对误差、相对误差和正交性的概念。

设 $v_i, \hat{v}_i, i = 1, 2, \dots, n$ 分别表示矩阵 H 的 n 个准确单位特征向量与近似单位特征向量，并记

$$V = [v_1, v_2, \dots, v_n], \hat{V} = [\hat{v}_1, \hat{v}_2, \dots, \hat{v}_n].$$

此外，为保证 v_i 的唯一性，我们将这里所说的单位特征向量统一规定向量的第一个不为零的分量为正。

定义 2.4 绝对误差 (a.e.) 定义矩阵 H 的近似特征向量的绝对误差为

$$\text{a.e.}(H, V) = \|V - \hat{V}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |v_j^i - \hat{v}_j^i|.$$

定义 2.5 相对误差 (r.e.) 定义为矩阵 H 近似特征值的相对误差为

$$\text{r.e.}(H, V) = \max_{1 \leq j \leq n} \sum_{i=1}^n \left| \frac{v_j^i - \hat{v}_j^i}{v_j^i} \right|.$$

(1) 算例 2.1

为体现一般性，我们选择随机矩阵进行数值实验。在 $[-1, 1]$ 上随机产生 ρ 和

$$d_j, z_j, j = 1, 2, \dots, n.$$

由此构造出随机矩阵 $H = D + \rho zz^T$ 。在数值实验过程中，我们取 $\varepsilon = 1.0 \times 10^{-14}$ 作为计算精度。

为便于对比，我们编写了著名的 QR 方法的程序进行特征值和特征向量来计算。此外，用于对比的 QR 方法采用的是单步位移，这是因为我们讨论的矩阵 $H = D + \rho zz^T$ 是对称矩阵，特征值均为实数，无需采用双重步位移。这里也取 $\varepsilon = 1.0 \times 10^{-15}$ 作为计算精度。

需要指出的是，我们的数值结果均为 8 次随机矩阵分解实验得到的结果的平均值。

表 2.1 给出的是我们设计的算法和单步位移的 QR 方法针对随机选取的矩阵 $H = D + \rho zz^T$ 的特征值和特征向量的计算的数值结果。

表 2.1 关于随机矩阵 $H = D + \rho z z^T$ 特征值与特征向量的计算结果

阶数 n		20	50	100	200	400	800	
our method	λ	a.e	4.15e-14	1.08e-13	7.93e-13	3.01e-12	2.76e-11	9.11e-10
		r.e	3.69e-14	1.62e-13	6.97e-13	4.94e-12	3.63e-11	1.096e-9
		b.e	2.46e-15	2.23e-14	7.47e-14	1.82e-13	7.31e-13	7.24e-12
	v	a.e	5.15e-12	9.02e-12	2.68e-11	1.78e-10	2.320e-9	1.68e-8
		r.e	5.76e-11	9.48e-11	5.53e-10	8.597e-9	9.04e-8	2.67e-7
	CPU(s)	0.005274	0.02661	0.191764	1.09872	4.75902	28.8515	
QR method	λ	a.e	3.22e-15	1.04e-14	1.37e-13	9.19e-13	4.66e-12	8.17e-11
		r.e	5.55e-15	2.44e-14	1.19e-13	8.64e-13	4.32e-12	9.28e-11
		b.e	8.24e-15	3.17e-14	5.78e-14	5.99e-13	4.04e-12	2.93e-11
	v	a.e	1.45e-13	7.12e-13	6.86e-12	5.22e-11	7.98e-10	9.83e-9
		r.e	1.63e-12	7.97e-12	8.75e-11	6.54e-10	9.46e-9	1.045e-7
	CPU(s)	0.004253	0.01968	0.121140	1.16523	5.3560	31.6284	

我们绘制特征值的误差曲线（对数坐标系下）和特征向量误差曲线（对数坐标系下）及 CPU 时间曲线分别如图 2.2 和图 2.3 及图 2.4 所示。

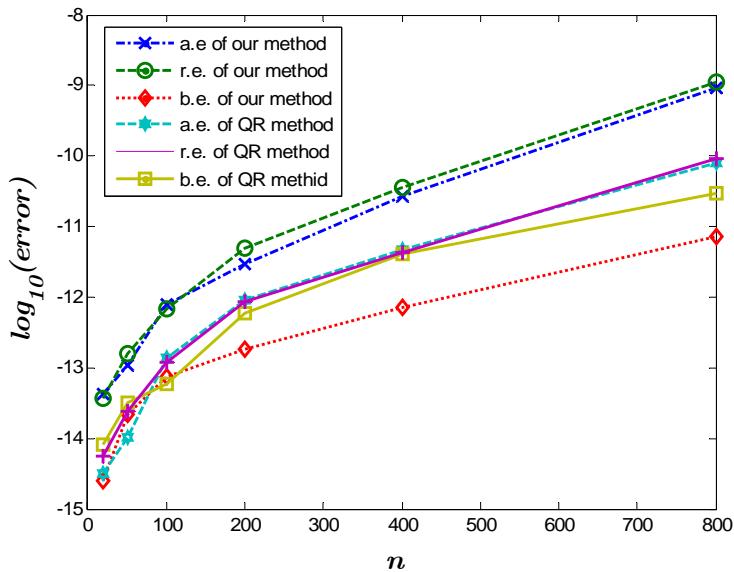


图 2.2 对数坐标系下：计算的特征值的误差关于矩阵的阶 n 的曲线

从表 2.1 和图 2.2 我们可以分析得出，随着阶 n 的增加，计算的特征值的误差也跟着增加；此外，结果显示我们所设计的算法的计算精度与 QR 方法的计算结果相差将近一个数量级的差距，但是对于规模较小的问题，计算精度差距很小；计算结果还是比较满意的。此外，注意到我们算法求得的特征值的向后误差精度

比较高，比 QR 方法要高两个数量级，这主要是因为我们的特征值是针对特征方程用二分法求根直接得到的，而 QR 方法是从特征值的角度得到的。

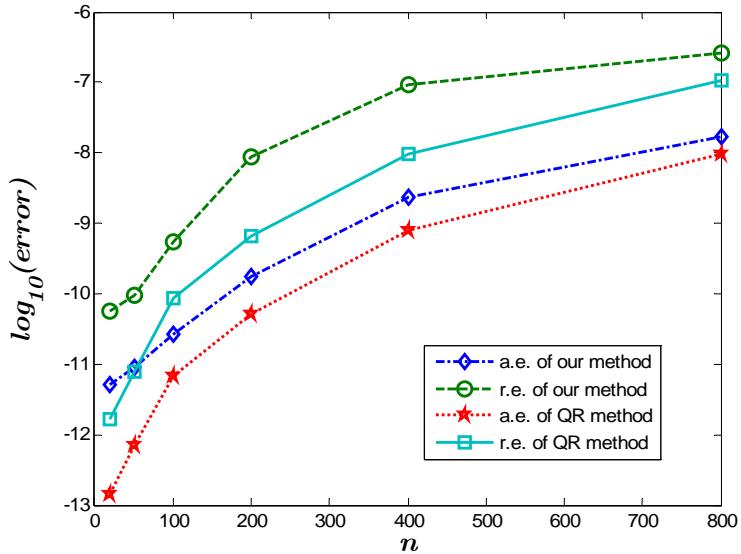


图 2.3 对数坐标系下：计算的特征向量的误差关于矩阵的阶 n 的曲线

从表 2.1 和图 2.3 我们可以分析得出，随着阶 n 的增加，计算的特征向量的误差也跟着增加；此外，结果显示 QR 方法的计算特征向量的精度很高，我们所设计的算法的计算精度与之相差将近两个数量级的差距；总体来说，我们的算法计算结果还是比较满意的。

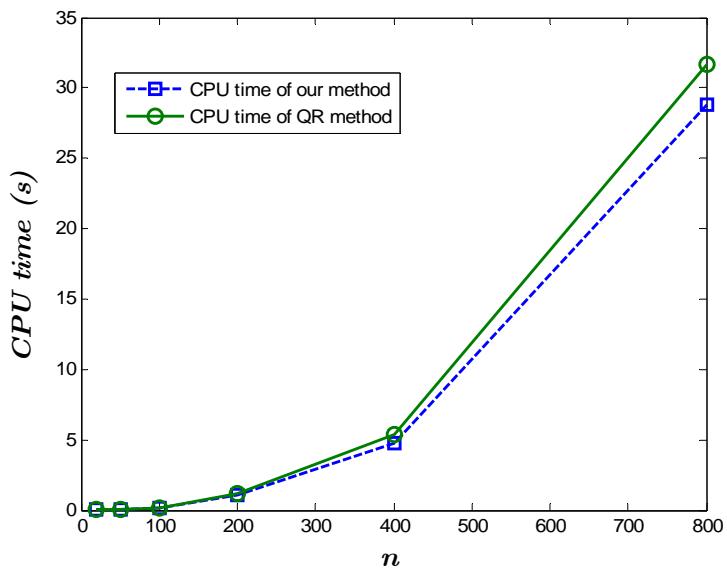


图 2.4 CPU 时间（秒）关于矩阵的阶 n 的曲线

从表 2.1 和图 2.4 我们可以看到，在 n 比较小的情况下，我们实现的算法速度没有著名的 QR 方法快；随着阶 n 的增加，我们的算法的耗时增加比 QR 方法增加的要缓慢些；对于高阶问题，我们的算法对于这种特定问题耗时要小一些。

(2) 算例 2.2

这一小节，我们随机选择一种特殊情况的矩阵 $H = D + \rho z z^T$ 进行实验：即在 $[-1, 1]$ 上随机产生 ρ 和

$$z_j, j = 1, 2, \dots, n.$$

然后在 $[-1, 1]$ 上随机产生单调递减的 d_j ，由此构造出随机矩阵 $H = D + \rho z z^T$ 。在数值实验过程中，我们取 $\varepsilon = 1.0 \times 10^{-15}$ 作为计算精度。

为便于对比，我们编写了 QR 方法的程序进行特征值和特征向量来计算。此外，用于对比的 QR 方法采用的是单步位移，这是因为我们讨论的矩阵 H 是对称矩阵，特征值均为实数，无需采用双重步位移。这里也取 $\varepsilon = 1.0 \times 10^{-15}$ 作为计算精度。

表 2.2 给出的是我们设计的算法和单步位移的 QR 方法针对随机选取的矩阵 $H = D + \rho z z^T$ 的特征值和特征向量的计算的数值结果。需要指出的是，我们的数值结果均为 8 次随机矩阵分解实验得到的结果的平均值。

表 2.2 关于随机矩阵 $H = D + \rho z z^T$ 特征值与特征向量的计算结果

阶数 n		20	50	100	200	400	800	
our method	λ	a.e	7.86e-15	1.64e-14	2.82e-14	4.41e-14	5.71e-14	1.25e-13
	λ	r.e	1.52e-15	2.12e-15	3.23e-15	4.07e-15	4.92e-15	5.79e-15
	λ	b.e	8.25e-15	1.14e-14	3.26e-14	5.94e-14	8.77e-14	1.02e-13
	v	a.e	8.62e-15	1.32e-14	4.90e-14	7.88e-14	2.72e-13	5.65e-13
	v	r.e	9.03e-14	1.48e-13	5.35e-13	9.64e-13	3.81e-12	6.48e-12
	CPU(s)	0.004892	0.021554	0.16997	0.73386	1.65438	3.86120	
QR method	λ	a.e	2.41e-15	2.10e-14	1.31e-13	8.65e-13	4.11e-12	7.16e-11
	λ	r.e	3.69e-15	3.55e-14	1.88e-13	9.97e-13	5.87e-12	8.54e-11
	λ	b.e	7.22e-15	1.98e-14	3.65e-14	9.31e-14	2.83e-13	6.49e-13
	v	a.e	1.32e-13	6.57e-13	5.75e-12	6.33e-11	7.69e-10	8.32e-9
	v	r.e	2.01e-12	8.12e-12	8.21e-11	7.17e-10	8.91e-9	1.293e-7
	CPU(s)	0.004300	0.018536	0.13685	1.18902	5.3447	29.8176	

我们绘制特征值的误差曲线（对数坐标系下）和特征向量误差曲线（对数坐标系下）及 CPU 时间曲线分别如图 2.5 和图 2.6 及图 2.7 所示。

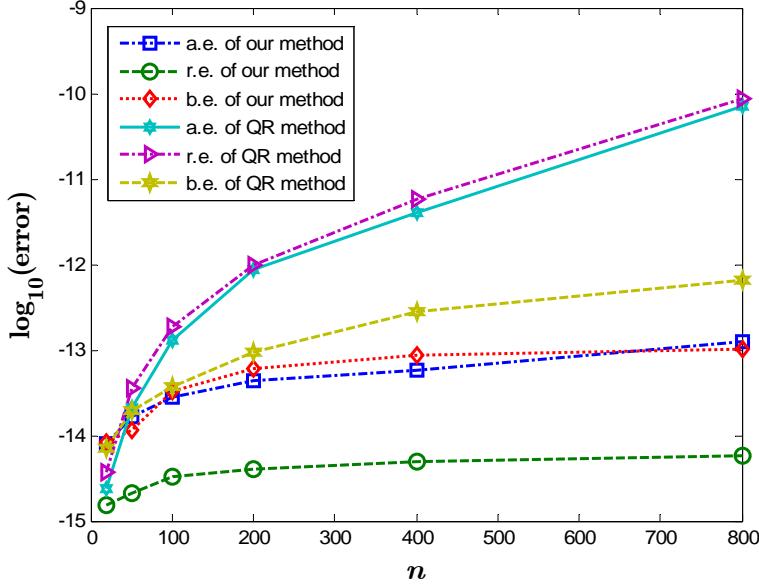


图 2.5 对数坐标系下：计算的特征值的误差关于矩阵的阶 n 的曲线

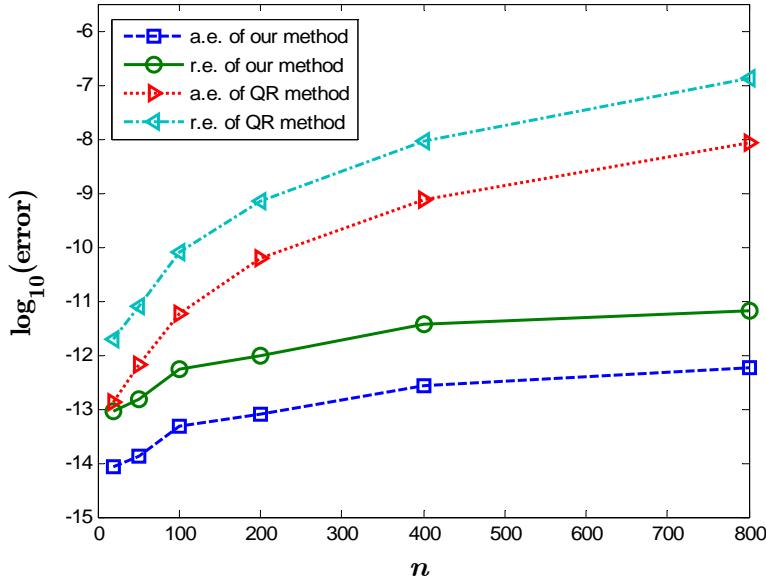


图 2.6 对数坐标系下：计算的特征向量的误差关于矩阵的阶 n 的曲线

从表 2.2 和图 2.5 我们可以分析得出，随着阶 n 的增加，计算的特征值的误差也跟着增加；此外，结果显示我们所设计的算法的计算精度非常高，比 QR 方

法的计算结果将高出近 2 个数量级以上；对于这种特殊情况，计算不需要预处理，因而计算结果还是比较满意的。此外，注意到我们算法求得的特征值的向后误差精度比较高，比 QR 方法也要高两个数量级，这主要是因为我们的特征值是针对特征方程用二分法求根直接得到的，而 QR 方法是从矩阵的角度得到的。

从表 2.2 和图 2.6 我们可以分析得出，随着阶 n 的增加，计算的特征向量的误差也跟着增加；此外，结果显示我们的算法计算特征向量的精度很高，我们所设计的算法的计算精度与比 QR 方法精度高出两个数量级以上；总体来说，我们的算法计算结果还是很令人满意的。

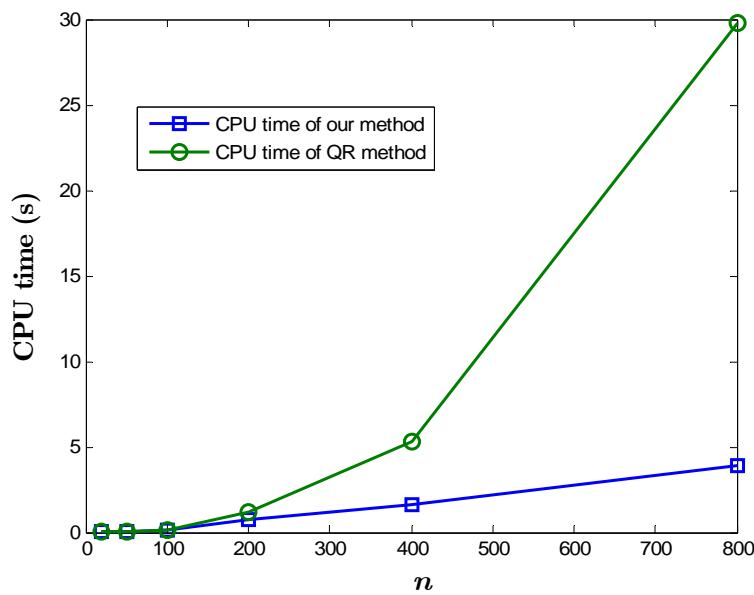


图 2.7 CPU 时间（秒）关于矩阵的阶 n 的曲线

从表 2.2 和图 2.7 我们可以看到，在 n 比很小的情况下，我们实现的算法速度没有著名的 QR 方法快；随着阶 n 的增加，我们的算法的耗时增加比 QR 方法增加的要缓慢些；对于 $n > 100$ 的高阶问题，我们的算法对于这种特定问题耗时要比 QR 方法小很多，即速度明显要快很多，这充分体现了我们的算法在求这类特殊问题上的优势。

3 单边 Jacobi 方法中的加速技术效果分析

3.1 问题的提出

Jacobi 方法是求解对称矩阵谱分解的一种古老的方法，由 Jacobi 在 1845 年首先提出。在历史的发展过程中，由于 Jacobi 方法与 QR 方法以及后来发展起来的分而治之法相比速度太慢，大约要慢 10 到 15 倍，因此很少被使用。后来 Demmel 和 Veselic 于 1992 年通过仔细的理论分析和大量的数值实验，揭示了 Jacobi 方法在小特征的计算方面远胜于 QR 方法和分而治之法。最近 Drmac 和 Veselic 通过适当的预处理和细致的安排 Jacobi 迭代过程的扫描次序，使得单边的 Jacobi 方法的速度大为提高，约比 Golub-Kahan SVD 算法快 2 倍。这样，就既保证了 Jacobi 方法在计算精度上的优越性，又使其在计算速度上可与现有的方法媲美。

我们在此主要通过 matlab 编程实现 Jacobi 方法，通过数值例子来展示“de Rijk 加速技巧”和“预处理加速技术”的功效，并将 Jacobi 方法与 Golub-Kahan SVD 算法和快速 QR 方法进行对比。

3.2 计算奇异值分解的单边 Jacobi 方法

设 $A = [\alpha_{ij}] \in R^{m \times n}$. 这一小节，我们简要介绍求 A 的奇异值的单边 Jacobi 方法。

单边的 Jacobi 方法是形式地应用求解对称矩阵谱分解的 Jacobi 方法到对称矩阵上而得到的。令

$$A = [a_1, a_2, \dots, a_n], B = [\beta_{ij}] = [a_i^T a_j] = A^T A,$$

则应用 Jacobi 方法于 $B = A^T A$ 上，就是先要对选定的下标 p 和 q ($p < q$)，确定 $c = \cos \theta, s = \sin \theta$ ，使得

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} \beta_{pp} & \beta_{pq} \\ \beta_{qp} & \beta_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \sigma_p & 0 \\ 0 & \sigma_q \end{bmatrix},$$

然后作 Jacobi 变换

$$B_1 = G(p, q; c, s)^T B G(p, q; c, s) = G(p, q; c, s)^T A^T A G(p, q; c, s),$$

由 $\beta_{ij} = a_i^T a_j$ 可知，可不将 B 明确算出即可进行，只需计算 B 的三个元。而上式可通过变换

$$A_1 = A G(p, q; c, s)$$

隐含的完成。注意到

$$\begin{bmatrix} \beta_{pp} & \beta_{pq} \\ \beta_{qp} & \beta_{qq} \end{bmatrix} = \begin{bmatrix} a_p & a_q \end{bmatrix}^T \begin{bmatrix} a_p & a_q \end{bmatrix},$$

如果我们令

$$\begin{bmatrix} \tilde{a}_p & \tilde{a}_q \end{bmatrix} = \begin{bmatrix} a_p & a_q \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

则可改写上式为

$$\begin{bmatrix} \tilde{a}_p & \tilde{a}_q \end{bmatrix}^T \begin{bmatrix} \tilde{a}_p & \tilde{a}_q \end{bmatrix} = \begin{bmatrix} \sigma_p & 0 \\ 0 & \sigma_q \end{bmatrix}.$$

这表明，选择 c, s 转化为选择 c, s 满足 $\tilde{a}_p^T \tilde{a}_q = 0$ ，这样的 c, s 可由 Jacobi 变换给出。

需要注意的是，上面计算过程中，每进行一次 Jacobi 变换就需要计算三个向量内积。实际上，我们有

$$\begin{aligned} \sigma_p &= \tilde{a}_p^T \tilde{a}_p = \beta_{pp} - t\beta_{pq}, \\ \sigma_q &= \tilde{a}_q^T \tilde{a}_q = \beta_{qq} + t\beta_{pq}, \end{aligned}$$

我们只需计算一个向量内积即可。事实上，如果我们事先将 $\sigma_i = a_i^T a_i$ 算好，在后面的迭代过程中就只需利用这一公式修改 σ_p 和 σ_q 即可。因此，我们只需计算一个向量内积 $\beta_{pq} = a_p^T a_q$ 。

总之，单边 Jacobi 方法就是对 A 进行一系列的旋转变换，使其最终收敛到一个列正交的矩阵 $C = [c_1, c_2, \dots, c_n]$ 。一旦 C 已经得到，则所求的奇异值和左奇异向量即为

$$\sigma_i = \|c_i\|_2, u_i = c_i / \sigma_i, i = 1, 2, \dots, n.$$

3.3 单边 Jacobi 方法的加速技术

在这一小节，我们介绍人们提出的若干加速技术中的两个“de Rijk 加速技巧”和“预处理加速技术”。

3.3.1 de Rijk 加速技巧

一种既简单又有效的加速方法是 de Rijk 的加速方法。即在每一行扫描前加入以下过程：在对第 p 行进行扫描时，先将 2 范数最大的列移至第 p 列。这样做了之后，在对第 p 列和第 q 列作 Jacobi 变化之后，可以保证第 p 列的 2 范数增加，而第 q 列的 2 范数减少，从而加速收敛。

3.3.2 预处理加速技术

最有效的加速技巧是对 A 先做列主元的 QR 分解

$$Q^T AP = \begin{bmatrix} R \\ 0 \end{bmatrix},$$

其中 Q 是 m 阶正交矩阵， P 是排列方阵，而矩阵 R 具有以下形式

$$R = \begin{bmatrix} r_{11} & \cdots & \cdots & \cdots & r_{1n} \\ \ddots & & & & \\ & r_{kk} & \cdots & r_{kn} & \\ 0 & & & & 0 \end{bmatrix} \in R^{n \times n},$$

并且其对角元满足 $|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{kk}| \geq 0$.

若 $k = n$ ，则可对 R^T 应用单边的 Jacobi 方法，计算其奇异值分解

$$R^T = U_1 \Sigma V_1^T.$$

从而有

$$Q^T AP = \begin{bmatrix} U_1 \Sigma V_1^T \\ 0 \end{bmatrix}$$

由此可得 A 的奇异值分解为

$$U = Q \begin{bmatrix} V_1 & 0 \\ 0 & I \end{bmatrix}, V = P U_1$$

这样做的原因是，当 $m \gg n$ 时，对 R^T 应用单边的 Jacobi 方法就比直接应用在 A 上的运算量小得多。

若 $k < n$ ，则计算 $R(1:k, 1:n)^T$ 的 QR 分解

$$Q_1 R(1:k, 1:n)^T = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

其中 Q_1 是 n 阶正交矩阵， R_1 是 k 阶上三角矩阵，然后利用算法来计算 R_1^T 的奇异值分解

$$R_1^T = U_1 \Sigma_1 V_1^T.$$

于是就有

$$Q_1^T R(1:k, 1:n)^T = \begin{bmatrix} V_1 \Sigma_1 U_1^T \\ 0 \end{bmatrix},$$

$$R(1:k, 1:n)^T = [U_1 \Sigma_1 V_1^T, 0] Q_1^T,$$

$$Q^T A P = \begin{bmatrix} [U_1 \Sigma_1 V_1^T, 0] Q_1^T \\ 0 \end{bmatrix},$$

由此即得 A 的奇异值分解为

$$A = U \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} V^T,$$

其中

$$U = Q \begin{bmatrix} U_1 & 0 \\ 0 & I_{m-k} \end{bmatrix}, V = P Q_1 \begin{bmatrix} V_1 & 0 \\ 0 & I_{n-k} \end{bmatrix}.$$

3.4 数值实验与加速技术的加速效果分析

为分析以上两种加速技术的功效，我们考虑如下四种算法进行数值实验，并对实验结果作对比分析。

- Jacobi-0: 单边的 Jacobi 方法；
- Jacobi-R: 使用 de Rijk 加速技巧的单边的 Jacobi 方法；
- Jacobi-P: 使用预处理加速技术的单边的 Jacobi 方法；
- Jacobi-PR: 使用 de Rijk 加速技巧和预处理加速技术的单边的 Jacobi 方法。

我们分析多次随机重复实验得到的：计算所用的平均 CPU 时间 t 、平均扫描次数 n_s 和平均使用旋转变换的次数 n_t ，以此来衡量加速技术的加速效果。为便于分析，我们定义平均加速度率 η_a 如下

$$\eta_a = \left(\frac{n_s}{\hat{n}_s} \cdot \frac{n_t}{\hat{n}_t} \cdot \frac{\hat{t}}{t} \right)^{\frac{1}{3}},$$

其中 $\hat{t}, \hat{n}_s, \hat{n}_t$ 加速之后计算所用的平均 CPU 时间、平均扫描次数和平均使用旋转变换的次数，平均加速度率衡量的是整体加速了多少倍。此外还定义时间加速度率

$$\eta_a = \frac{t}{\hat{t}},$$

来衡量速度提升多少倍。

为体现数值结果的一般性，我们选择随机矩阵进行实验。在 $[-1, 1]$ 上随机产生 $2n \times n$ 的矩阵 $A = (a_{ij})$ ，其中 a_{ij} 为 $[-1, 1]$ 之间的随机数，即 $a_{ij} = 2 * rand - 1$ 。

在数值实验过程中，我们取 $\varepsilon = \sqrt{n}\mathbf{u}$ 作为计算精度。为提高分析结果的准确度，我们将每次实验重复随机选取 100 次，取平均值作为我们的实验结果。

表 3.1 给出的是我们用以上四种算法计算 A 奇异值分解的数值结果：计算所用的平均 CPU 时间、平均扫描次数和平均使用旋转变换的次数。其中 \mathbf{u} 为机器精度。

表 3.1 计算结果与加速效果分析

阶数 n		20	50	100	150	200	250
Jacobi-0	CPU(s)	0.05118	0.42567	2.32844	6.21449	13.1539	19.9757
	扫描次数	9	13	18	19	20	22
	变换次数	1216	8973	48365	104325	178825	260134
Jacobi-R	CPU(s)	0.04259	0.35532	1.93484	4.60212	8.72217	13.53604
	扫描次数	7	9	12	13	14	16
	变换次数	1015	7473	32041	74884	139392	188117
Jacobi-P	CPU(s)	0.03195	0.25921	1.23455	3.41928	6.52376	9.51226
	扫描次数	5	7	9	10	11	12
	变换次数	796	4939	27395	57898	95087	145137
Jacobi-PR	CPU(s)	0.028342	0.24521	1.02109	2.71735	6.10138	8.54103
	扫描次数	5	6	8	9	10	11
	变换次数	709	4435	24565	49121	72245	111633

绘制四种算法计算所用的平均 CPU 时间 t 、平均扫描次数 n_s 和平均使用旋转变换的次数 n_t 与问题的阶 n 的关系曲线分别如图 3.1、图 3.2 和图 3.3 所示。

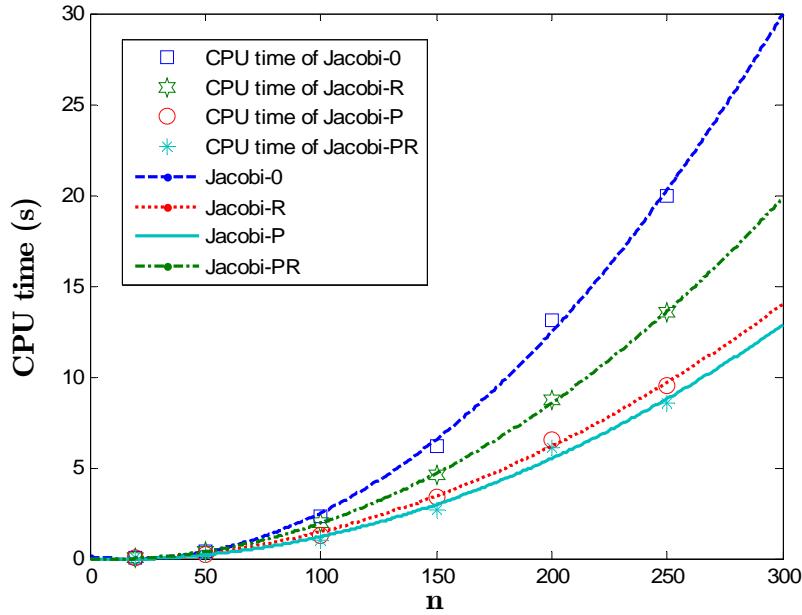


图 3.1 四种算法计算所用的平均 CPU 时间 t 关于阶 n 的曲线

从表 3.1 和图 3.1 我们可以分析得出：随着 n 的增加，CPU 时间呈现多项式增长；三种加速算法 Jacobi-R、Jacobi-P 和 Jacobi-PR 的加速效果很明显；对比 Jacobi-0 和 Jacobi-R 的 CPU 时间曲线，可知 de Rijk 加速技巧具有很好的加速效果；对比 Jacobi-0 和 Jacobi-P 的 CPU 时间曲线，可知预处理技术具有很好的加速效果，且效果优于 de Rijk 加速技巧；对比 Jacobi-0 和 Jacobi-PR 的 CPU 时间曲线，可知同时使用两种加速技术能获得更好的加速效果。此外，我们还可以观察出：当 n 较小时，加速效果不是非常明显，但 n 越大，加速效果越明显。

从表 3.1 和图 3.2 我们可以分析得出：随着 n 的增加，平均扫描次数 n_s 呈现近似对数增长；三种加速算法 Jacobi-R、Jacobi-P 和 Jacobi-PR 的加速效果很明显；对比 Jacobi-0 和 Jacobi-R 的平均扫描次数曲线，可知 de Rijk 加速技巧具有很好的加速效果；对比 Jacobi-0 和 Jacobi-P 的平均扫描次数曲线，可知预处理技术具有很好的加速效果，且效果优于 de Rijk 加速技巧；对比 Jacobi-0 和 Jacobi-PR 的平均扫描次数曲线，可知同时使用两种加速技术能获得更好的加速效果。

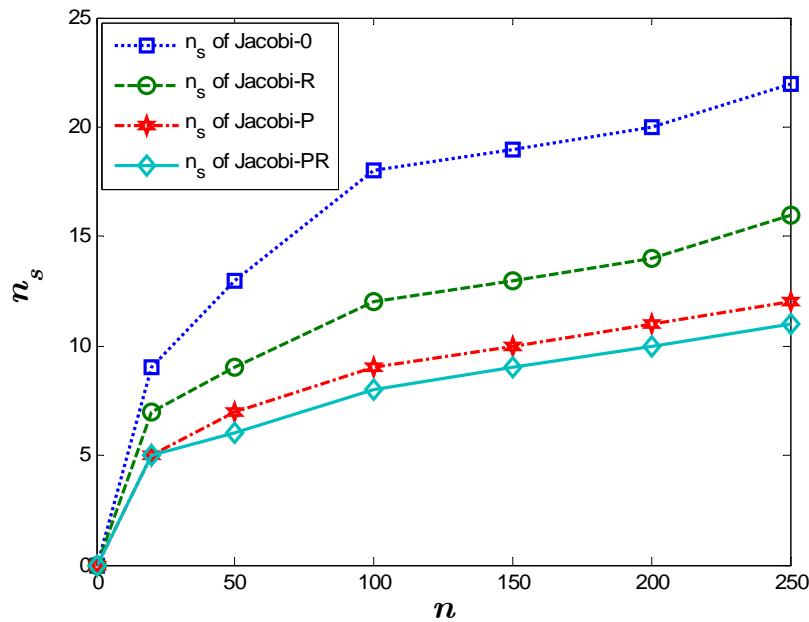


图 3.2 四种算法计算所用的平均扫描次数 n_s 关于阶 n 的曲线

此外，从表 3.1 和图 3.2 中我们还可以观察出：当 n 较小时，加速效果不是很明显，但 n 越大，加速效果越明显。从表 3.1 可以看出，当 $n \geq 100$ 时，Jacobi-R 算法相比于 Jacobi-0 算法而言平均扫描次数降低 6 次以上，Jacobi-P 算法相比于 Jacobi-0 算法而言平均扫描次数降低 9 次以上，Jacobi-PR 算法相比于 Jacobi-0 算法而言平均扫描次数降低 11 次以上。这就是说：de Rijk 加速技巧平均可降低扫描次数降低 6 次以上；预处理加速技术平均可降低扫描次数降低 9 次以上；同时使用两种技术平均可降低扫描次数降低 11 次以上，而且随着 n 增加，降低的次数越多。因此，这些加速技术的使用，极大地减少了扫描次数，从而起到了加速作用。

从表 3.1 和图 3.3 我们可以分析得出：随着 n 的增加，平均变换次数 n_t 呈现多项式增长；三种加速算法 Jacobi-R、Jacobi-P 和 Jacobi-PR 的加速效果很明显；对比 Jacobi-0 和 Jacobi-R 的平均变换次数曲线，可知 de Rijk 加速技巧具有很好的加速效果；对比 Jacobi-0 和 Jacobi-P 的平均变换次数曲线，可知预处理技术具有很好的加速效果，且效果优于 de Rijk 加速技巧；对比 Jacobi-0 和 Jacobi-PR 的平均变换次数 n_t 曲线，可知同时使用两种加速技术能获得更好的加速效果。

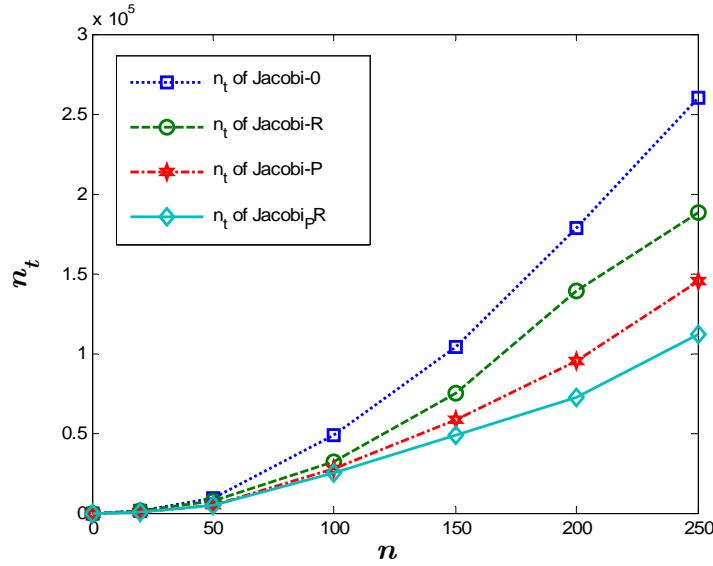


图 3.3 四种算法计算所用的平均变换次数 n_t 关于阶 n 的曲线

此外，从表 3.1 和图 3.3 中，我们还可以观察出：当 n 较小时，加速效果不是非常明显，但 n 越大，加速效果越明显。两种加速技术的使用，减少了扫描次数，因而变换次数也降低了。

综上所述，de Rijk 加速技巧和预处理加速技术在单边的 Jacobi 方法中，扮演了很好的加速角色。具体表现为：极大地降低了扫描次数、减少了变换次数、减小了 CPU 时间，从而起到了加速作用。

下面我们分析三种加速算法的加速率 η ，并用加速率 η 来进一步分析加速效果与阶 n 的关系。根据加速率的定义，我们用表 3.1 中的数据求得平均加速率 η_a 和时间加速率 η_t 如表 3.2 所示。

表 3.2 三种加速算法的平均加速率和时间加速率

阶数 n		20	50	100	150	200	250	平均值
Jacobi-R	η_a	1.2278	1.2760	1.3967	1.4009	1.4034	1.4105	1.3526
	η_t	1.1917	1.1980	1.2034	1.3504	1.5081	1.5757	1.3379
Jacobi-P	η_a	1.6392	1.7695	1.8814	1.8837	1.8901	1.9038	1.8280
	η_t	1.6019	1.6422	1.8861	1.8975	2.0163	2.1000	1.8573
Jacobi-PR	η_a	1.7731	1.9669	2.1617	2.1725	2.2017	2.2172	2.0822
	η_t	1.8058	1.9359	2.2803	2.2870	2.3059	2.3388	2.1590

从表 3.2 可以看出: Jacobi-R 算法的平均加速率和时间加速率大约分别为 1.35 和 1.34 左右, Jacobi-P 算法的平均加速率和时间加速率大约分别为 1.83 和 1.86 左右 Jacobi-R 算法的平均加速率和时间加速率大约分别为 2.08 和 2.16 左右。

为分析加速率关于 n 的变化趋势。通过表 3.2 可绘制三种加速算法的平均加速率 η_a 和时间加速率 η_t 关于 n 的关系曲线如图 3.4 所示。

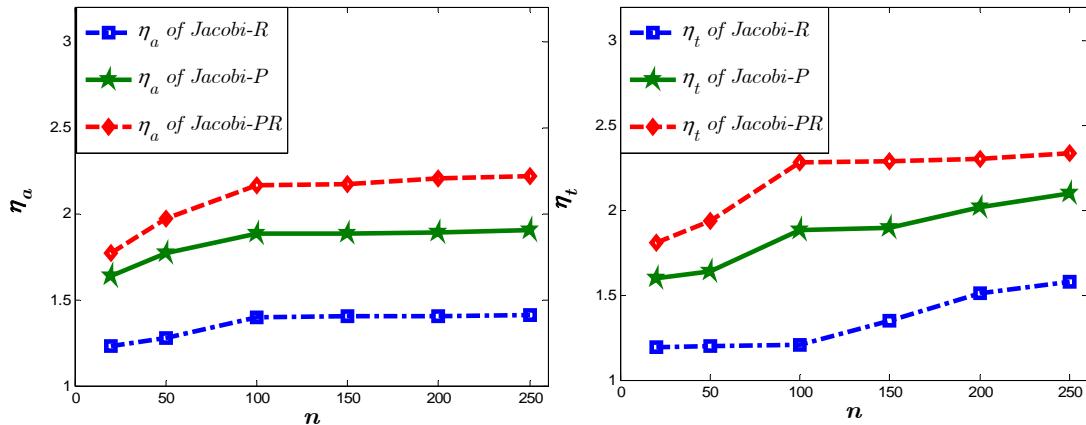


图 3.4 三种算法平均加速率 η_a 和时间加速率 η_t 关于 n 的关系曲线

通过图 3.4 我们可以看出: 当 n 较小时, 加速技术的效果不是非常明显, 当 n 达到 100 时, 出现了明显的加速作用。从平均加速率的走势上看, 当 $n \geq 100$ 时, 平均加速率虽缓慢增加, 但基本持平, 这就是说算法的平均加速率近似认为是一定的。从时间加速率的走势上看, de Rijk 加速技巧的加速效果随着 n 增加越来越显著, 但当 n 到 250 左右时也有持平的趋势; 预处理加速技术的加速效果随着 n 增加也越来越显著; 而两种加速技术同时使用时, 加速率当然增加了, 但是当 $n \geq 100$ 时, 随着 n 增加加速率保持在一个很小的范围内缓慢增加, 这就是说 Jacobi-PR 算法的时间加速率可近似认为是一定的。

4 结语

至此，我们完成了以下工作：

- 针对反对称矩阵的特殊结构，我们通过三对角化和隐式 QR 迭代，设计了反对称矩阵 Schur 分解的算法，并对算法的时间复杂度、空间复杂度和理论误差进行了理论分析。最后通过数值实验对算法的数值特性进行了分析，并与 matlab 自带的 eig 函数进行了对比。数值实验的具体程序实现还是有待改进，比如程序中可能存在许多冗余和存储不合理的地方，如果仔细分析加以改进，应该可以在一定程度上加快我们的计算速度。
- 对矩阵 $H = D + \rho zz^T$ ，我们设计了计算其特征值与特征向量的算法，并对算法的时间复杂度和稳定性进行了理论分析。最后通过数值实验对算法的数值特性进行了分析，并与著名的计算一般矩阵特征值的 QR 方法进行了对比。对于一般形式 $H = D + \rho zz^T$ ，我们的算法速度还有待改善。
- 用 matlab 编写程序实验了单边的 Jacobi 方法，最后通过数值实验对算法的数值特性进行了分析，选择了若干算例来检测算法中的两种加速技术：“de Rijk 加速技巧”和“预处理加速技术”，进行了加速效果的分析。通过数值实验，我基本掌握了单边的 Jacobi 方法的思想和加速技术的思想。

总之，通过本次数值实验，我更加认识到了算法设计中的关键思想，即如何算得快，如何算得准。比如，一些加速技术和合理安排存储有利于加快算法的速度；通过平移计算可以减少计算误差等等，这将会对我们在后面的学习和研究过程中帮助很大。