SVD（奇异值分解）算法及其评估

本文第一部分对 SVD 进行了简单的介绍，给出了定义和奇异值分解定理；第二部分简要地列举了 SVD 的应用；第三部分则构造和分析了各种求解 SVD 的算法，特别对传统 QR 迭代算法和零位移 QR 迭代算法进行了详细完整的分析；第四部分给出了复矩阵时的处理办法；第五部分是对各种算法的一个简要的总结。

一、 SVD 简介

定义 1.1 设 $A\in R^{m\times n}$， A^TA 的特征值的非负平方根称作 A 的奇异值； A 的奇异值的全体记作 $\sigma(A)$ [1]。

当 A 为复矩阵 $C^{m\times n}$ 时，只需将 A^TA 改为 A^HA，定义 1.1 仍然成立。

定理 1.1（奇异值分解定理） 设 $A\in R^{m\times n}$，则必存在正交矩阵

$$U=[u_1,...,u_m]\in R^{m\times m} \text{ 和 } V=[v_1,...,v_n]\in R^{n\times n}$$

使得

$$U^TAV=\begin{bmatrix}\Sigma_r & 0\\ 0 & 0\end{bmatrix}\begin{matrix}r\\ m-r\end{matrix}\quad\text{，}\quad\ldots(1.1)$$

$$\qquad\quad r\quad n-r$$

其中 $\Sigma_r=diag(\sigma_1,...,\sigma_r),\sigma_1\geq...\geq\sigma_r>0$ [2]。

当 A 为复矩阵 $C^{m\times n}$ 时，只需将定理中 U,V 改为酉矩阵（Unitary Matrix），其它不变，定理 1.2 仍然成立[1]。

称分解式(1.1)为矩阵 A 的奇异值分解，通常简称为 SVD。σ_i 是 A 的奇异值，向量 u_i 和 v_i 分别是第 i 个左奇异向量和第 i 个右奇异向量。

从 A 的奇异值分解，我们可以得到 A 的一些非常有用的信息，下面的推论就列举其中几条最基本的结论[1]：

推论 1.2 设 $A\in C_r^{m\times n}$，则

(1) A 的非零奇异值的个数就等于 $r=rank(A)$ ；

(2) $v_{r+1},...,v_n$ 是 $\square(A)$ 的一组标准正交基；

(3) $u_1,...,v_r$ 是 $\square(A)$ 的一组标准正交基；

(4) $A=\sum_{i=1}^{r}\sigma_iu_iv_i^H$ 称为 A 的满秩奇异值分解；

其中 $\square(A)$，$\square(A)$ 分别指得是 A 的零空间和值域。

为了方便，我们采用如下表示奇异值的记号：

$\sigma_i(A)$=A 的第 i 大奇异值；

$\sigma_{\max}(A)$=A 的最大奇异值；

$\sigma_{\min}(A)$=A 的最小奇异值。

现在来考察矩阵奇异值的几何意义[1,2]，不妨设 $m=n$，此时有

$$E_n=\{y\in C^n:y=Ax,x\in C^n,\|x\|_2=1\}$$

是一个超椭球面，它的 n 个半轴长正好是 A 的 n 个奇异值 $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$，这些轴所在的直线正好是 A 的左奇异向量所在的直线，它们分别是对应的右奇异向量所在直线的象。

一般地我们假设 $m \geq n$，（对于 $m < n$ 的情况，我们可以先对 A 转置，然后进行奇异值分解，最后对所求得的 SVD 分解式进行转置就可以得到原式 SVD 分解式），此时我们对(1.1)进行化简将 U 表示为：

$$U = (U_1, U_2), \ldots (1.2)$$

则可以得到更加细腻的 SVD 分解式[2,3]：

$$A = U_1 \Sigma V^T \ldots (1.3)$$

其中 U_1 具有 n 列 m 维正交向量， V 和(1.1)式中的定义相同；$\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$，并且 $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$ 为矩阵 A 的奇异值。

二、 SVD 应用

在现代科学计算中 SVD 具有广泛的应用，在已经比较成熟的软件包 LINPACK 中列举的应用有以下几点[3]:

(1) 确定矩阵的秩(rank)

假设矩阵 A 的秩为 r，那么 A 的奇异值满足如下式子

$$\sigma_1 \geq ... \geq \sigma_r > \sigma_{r+1} = ... = \sigma_n = 0;$$

反之，如果 $\sigma_r \neq 0$，且 $\sigma_{r+1} = ... = \sigma_n = 0$，那么矩阵 A 的秩为 r，这样奇异值分解就可以被用来确定矩阵的秩了。

事实的计算中我们几乎不可能计算得到奇异值正好等于 0，所以我们还要确定什么时候计算得到的奇异值足够接近于 0，以致可以忽略而近似为 0。关于这个问题，不同的算法有不同的判断标准，将在给出各种算法的时候详细说明。

(2) 确定投影算子

假设矩阵 A 的秩为 r,那么我们可以将(1.3)式中的 U_1 划分为以下的形式

$$U_1 = \left(U_1^{(1)}, U_2^{(1)}\right)$$

其中 $U_1^{(1)}$ 为 $m \times r$ 的矩阵，并且 $U_1^{(1)}$ 的列向量构成了矩阵 A 的列空间的正交向量基。

容易得到 $P_A = U_1^{(1)} U_1^{(1)T}$ 为投影到矩阵 A 的列空间上的正交投影算子；而 $P_A^{\perp} = (U_2^{(1)}, U_2)(U_2^{(1)}, U_2)^T$ 则是到矩阵 A 列空间的正交补空间上的投影。

同理如果将 V 划分为以下的形式

$$V = \left(V_1, V_2\right)$$

其中 V_1 为 $n \times r$ 的矩阵，则 V_1 的列向量构成矩阵 A 的行空间的正交向量基。且 $R_A = V_1 V_1^T$ 为投影到矩阵 A 的行空间上的正交投影算子；而 $R_A^{\perp} = V_2 V_2^T$ 则是到矩阵 A 行空间的正交补空间上的投影。

(3) 最小二乘法问题(LS 问题)

促进人们研究 SVD 并且应用 SVD 比较早的应该是最小二乘法问题了，在前一份关于 QR 分解的报告已经对 LS 问题进行了一些介绍，这里继续讨论 SVD 在 LS 问题中的应用。

LS 问题即相当于，设 $A \in R^{m\times n}\ (m>n), b \in R^m$，求 $x \in R^n$ 使得

$$\|Ax-b\|_2 = \min\{\|Av-b\|_2 : v \in R^n\} \ldots\ldots (2.1)$$

假设已知矩阵 A 有式子(1.1)得到的SVD分解式为 $U\Sigma V^T$，U 和 V 分别为 m,n 阶正交方阵，而 $\sum$ 为和 A 具有相同维数的对角矩阵，那么我们可以得到[4]：

$$\begin{aligned} Ax-b &= U\Sigma V^T x - b \\ &= U(\Sigma V^T x) - U(U^T b) \ldots\ldots (2.2) \\ &= U(\Sigma y - c) \end{aligned}$$

其中 $y = V^T x$， $c = U^T b$。

因为 U 是一正交矩阵，所以 $\|Ax-b\|_2 = \|U(\Sigma y - c)\|_2 = \|\Sigma y - c\|_2$，从而把原最小二乘法问题化为求使 $\|\Sigma y - c\|_2$ 最小的 y 这一最小二乘法问题，因为 Σ 为对角矩阵，所以使得新的这一最小二乘法问题简单的多，接着将对此仔细分析[4]。

假设矩阵 A 的秩为 r，则有：

$$\Sigma y = \begin{bmatrix} \sigma_1 y_1 \\ \vdots \\ \sigma_r y_r \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad \Sigma y - c = \begin{bmatrix} \sigma_1 y_1 - c_1 \\ \vdots \\ \sigma_r y_r - c_r \\ -c_{r+1} \\ -c_{r+2} \\ \vdots \\ -c_m \end{bmatrix}$$

可知 $y_i = c_i / \sigma_i, (i = 1,2,...,r)$ 使得 $\Sigma y - c$ 达到它的最小长度 $\left[\sum_{i=r+1}^{m} c_i^2\right]^{1/2}$，并且可见当 $r = m$ 时，上面的这一长度为 0，也就是当矩阵 A 的列张成 $\square^m$ 空间时最小二乘法问题可以无误差地求解。而当 $r < n$ 时， $y_{k+1},...,y_n$ 可以任意取，而不影响 $\Sigma y - c$ 的长度。

我们将对 $\sum$ 转置并且对非零的对角元素求逆所得到的矩阵定义为 Σ^+，那么 $y = \Sigma^+ c$ 的前 r 个元素将等于 $c_i / \sigma_i, (i = 1,2,...,r)$，并且其余的元素为 0。并且由 $y = V^T x$， $c = U^T b$，容易得到：

$$x = V \Sigma^+ U^T b \ldots\ldots (2.3)$$

由此得到的是 LS 问题的最小范数解。

而文献[3]中还给出了一般通解的形式如下：

$$x = V \Sigma^+ U^T b + V_2 w \ldots\ldots (2.4)$$

其中 V_2 如前定义，而 w 是任意的 $n-r$ 维向量。

(4) 广义逆问题(pseudo-inverse)

记 $A^+ = V\sum^+ U^T$，从(2.3)式我们可以看出，最小二乘法的解为 $x = A^+ b$，和一般的线性方程组 $Ax = b$ 的解为 $x = A^{-1}b$ 相类似，所以我们当我们已知矩阵 A 的奇异值分解 $A = U\Sigma V^T$ 后可以定义 A 的广义逆为 $A^+ = V\sum^+ U^T$。

(5) 条件数

如果已知矩阵 A 的秩为 r，那么在式子(2.3)中，解 x 随着矩阵 A 的扰动而改变的剧烈程度有多大呢？这可以用矩阵的条件数来衡量，条件数的定义如下：

$$\kappa_r(A) = \sigma_1 / \sigma_r \text{.....(2.5)}$$

以上只是针对 SVD 的应用，而简单地介绍了 LS 问题，广义逆；而在文献[5,6]中则对这两个问题有详细的说明，在以后的报告中也将进行更进一步的分析，并且综合其它方法来全面分析和处理这些问题。

除了这些传统的应用以外，在图像压缩和大型数据库的数据恢复中，SVD 也具有广泛的应用[7]。

三、 各种 SVD 算法及其特点

首先来对 SVD 算法的发展来做简单的回顾[11,12]：关于 SVD 算法的研究最早可以追溯到 1873 年 Beltrami 所做的工作，这中间在理论方面进行了大量的工作，这个历史过程可以参考 Stewart 的文献[8]。但是直到 1965 年 Golub 和 Kahan 才在 SVD 的数值计算领域取得突破性进展[9]，并且于 1969 给出了比较稳定的算法[10]（以下简称传统 QR 迭代算法），这也是后来在 LINPACK 中所采用的方法[3]。它的中心思想是用正交变换将原矩阵化为双对角线矩阵，然后再对双对角线矩阵迭代进行 QR 分解。

六十年代一份没有出版的技术报告中，Kahan 证明了双对角线矩阵的奇异值可以精确地计算，具有和原矩阵元素的相对的精确度；进一步，1990 年 Demmel 和 Kahan 给出了一种零位移的 QR 算法(zero-shift QR algorithm)，这种算法计算双对角矩阵的奇异值具有很高的相对精度[13]，并且由此得到的奇异向量也具有很高的精度[14]。

Fernando 和 Parlett 在 1994 年将 qd 算法应用到奇异值的计算上，从而得到了一种全新的比 zero-shift QR algorithm 更加精确和快速的计算奇异值的算法[15,16]。

而 Demmel 和 Veselic 在文献[17]中则说明了用 Jacobi 方法与其它方法相比计算所得到的奇异值和奇异向量具有更高的精度，可惜 Jacobi 算法比 DK 算法速度要慢的多；在文献[18]中对 Jacobi 方法进行了改进使得其在速度上几乎和 DK 算法相当。

和 Strassen 算法类似，SVD 问题也有快速的分而制之算法，在文献[19,20]对其有详细的介绍，由此得到的算法在总计算量上比现有的 LAPACK 软件包中所采用的方法要少的多。

在文献[21,22]中给出的 bisection 算法也可以对双对角线矩阵计算得到具有相对精度的全部奇异值。

以下就开始对各种算法原理进行详细说明，并分析它们的计算量，计算的精确度，以及所占得内存。

3.1：传统 QR 迭代算法[1,2,3]

设 $A \in R^{m\times n}(m \geq n)$，可知奇异值分解可从实对称矩阵 $C = A^T A$ 的 Schur 分解导出[1]，因此我们自然想到先利用对称 QR 方法来实现 C 的 Schur 分解，然后借助 C 的 Schur 分解来实现 A 的奇异值分解，然而这样做有两个缺点：一是计算 $C = A^T A$ 要很大的计算量；二是计算 $C = A^T A$ 会引入较大的误差。因此 Golub 和 Kahan 在 1965 年提出了另一种十分稳定的方法，其基本思想就是隐含地应用对称 QR 算法于 $A^T A$ 上，而并不需要将 $C = A^T A$ 计算出来。

方法第一步是：将 A 二对角化，即求正交矩阵 U_1 和 V_1，使得

$$U_1^T A V_1 = \begin{bmatrix} B \\ 0 \end{bmatrix}_{m-n}^{n} \quad \ldots\ldots(3.1.1)$$

其中

$$B = \begin{bmatrix} \delta_1 & \gamma_2 & 0 & 0 & 0 \\ 0 & \delta_2 & \gamma_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \gamma_n \\ 0 & 0 & 0 & 0 & \delta_n \end{bmatrix} \ldots\ldots(3.1.2)$$

分解式(3.1.1)可以用 Householder 变换来实现，将 A 分块为

$$A = \begin{bmatrix} v_1 & A_1 \\ 1 & n-1 \end{bmatrix}$$

先计算 m 阶 Householder 变换 P_1 使得

$$P_1 v_1 = \delta_1 e_1 (\delta_1 \in R, e_1 \in R^m)$$

并且形成：

$$P_1 A_1 = \begin{bmatrix} u_1^T \\ \tilde{A}_1 \end{bmatrix} \begin{matrix} 1 \\ m-1 \end{matrix}$$

再计算 n－1 阶 Householder 变换 $\tilde{H}_1$ 使得

$$\tilde{H}_1 u_1 = \gamma_2 e_1 (\gamma_2 \in R, e_1 \in R^{n-1})$$

并形成：

$$\tilde{A}_1 \tilde{H}_1 = \begin{bmatrix} v_2 & A_2 \\ 1 & n-2 \end{bmatrix}$$

然后对 $k = 2,3,...,n-2$ 依次进行：

(a) 计算 m-k+1 阶 Householder 变换 $\tilde{P}_k$ 使得

$$\tilde{P}_k v_k = \delta_k e_1 (\delta_k \in R, e_1 \in R^{m-k+1})$$

并且形成：

$$\tilde{P}_k A_k = \begin{bmatrix} u_k^T \\ \tilde{A}_k \end{bmatrix} \begin{matrix} 1 \\ m-1 \end{matrix}$$

(b) 计算 n－k 阶 Householder 变换 $\tilde{H}_k$ 使得

$$\tilde{H}_k u_k = \gamma_{k+1} e_1 (\gamma_{k+1} \in R, e_1 \in R^{n-k})$$

并形成：

$$\tilde{A}_k \tilde{H}_k = \begin{bmatrix} v_{k+1} & A_{k+1} \\ 1 & n-k-1 \end{bmatrix}$$

进行到$k = n-2$之后，再计算$m-n+2$阶 Householder 变换矩阵$\tilde{P}_{n-1}$使得

$$\tilde{P}_{n-1} v_{n-1} = \delta_{n-1} e_1 (\delta_{n-1} \in R, e_1 \in R^{m-n+2})$$

并形成：

$$\tilde{P}_{n-1} A_{n-1} = \begin{bmatrix} \gamma_n & 1 \\ v_n & \end{bmatrix} m-n+1$$

然后计算$m-n+1$阶 Householder 变换矩阵$\tilde{P}_n$使得

$$\tilde{P}_n v_n = \delta_n e_1 (\delta_n \in R, e_1 \in R^{m-n+1})。$$

现令

$$P_k = diag\left(I_{k-1}, \tilde{P}_k\right), k = 2, \ldots n$$

$$H_k = diag\left(I_k, \tilde{H}_k\right), k = 1, 2, \ldots n-2$$

$$U_1 = P_1 P_2 \ldots P_n，\quad V_1 = H_1 H_2 \ldots H_{n-2}$$

$$B = \begin{bmatrix} \delta_1 & \gamma_2 & 0 & 0 & 0 \\ 0 & \delta_2 & \gamma_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \gamma_n \\ 0 & 0 & 0 & 0 & \delta_n \end{bmatrix}$$

则有：

$$U_1^T A V_1 = \begin{bmatrix} B \\ 0 \end{bmatrix}_{m-n}^{n}$$

即实现了分解(3.1.1)。

将 A 二对角化以后，下一步任务就是对三对角矩阵$T = B^T B$进行带 Wilkinson 位移的对称 QR 迭代，这一步也可以不通过明确地将 T 计算出来而进行。

进行 QR 迭代的第一步是取矩阵$T = B^T B$的右下角2×2主子阵：

$$\begin{bmatrix} \delta_{n-1}^2 + \gamma_{n-1}^2 & \delta_{n-1}\gamma_n \\ \delta_{n-1}\gamma_n & \delta_n^2 + \gamma_n^2 \end{bmatrix}$$

靠近$\delta_n^2 + \gamma_n^2$最近的特征值作为位移μ，这一步不需将$T = B^T B$计算出来。

第二步就是，确定 Givens 变换$G_1 = G(1, 2, \theta)$，其中$c = \cos(\theta), s = \sin(\theta)$满足

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} \delta_1^2 - \mu \\ \delta_1 \gamma_2 \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \delta_1^2 - \mu \\ \delta_1 \gamma_2 \end{bmatrix} = \begin{bmatrix} \sigma \\ 0 \end{bmatrix}$$

这里$\delta_1^2 - \mu$和$\delta_1 \gamma_2$是$T - \mu I$的第一列位于(1,1)和(1,2)位置的仅有的两个非零元素，这一步也不需先将$T = B^T B$计算出来。

迭代的第三步就是确定正交矩阵 Q 使得 $Q^T(G_1^T TG_1)Q$ 为对称三对角阵，这相当于将 BG_1 二对角化，可以用“驱逐出境”法如下进行，取 $n=3$ 为例：

可知 BG_1 在(2,1)位置上出现了一个我们不希望有的非零元素，于是我们可以左乘一个(1,2)坐标平面的 Givens 变换 J_1 消去这一非零元素；但是这样又在(1,3)位置上出现了一个非零元素；因此，我们又需右乘一个(2,3)坐标平面的 Givens 变换 G_2 消去这一非零元素；这又会在(3,2)位置上出现了一个非零元素，再左乘一个(2,3)坐标平面的 Givens 变换 J_2 消去这一非零元素。最终完成了 $n=3$ 时的 BG_1 二对角化任务。

对于一般的 n，用完全类似的方法可确定 2n－3 个 Givens 变换 $J_1, G_2, J_2, G_3, \ldots, G_{n-1}, J_{n-1}$ 将 BG_1 中不受欢迎的元素都驱逐出境， 即使：

$$J_{n-1}J_{n-2}\ldots J_1(BG_1)G_2\ldots G_{n-1}$$

为二对角矩阵，而且这样得到的 $G_2\ldots G_{n-1}$ 满足

$$(G_2\ldots G_{n-1})e_1 = e_1$$

这样我们就得到了计算二对角阵奇异值的最基本的 QR 迭代算法了。

为了方便，我们在《QR 分解算法及其评估》中的**算法 2.3.1** 的基础上构造以下算法；构造函数 $Givens(x,y,c,s,r)$，当已知 x,y 的值时，计算出满足

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix},$$

的 c,s,r；算法如下：

算法 3.1.1：

给定数值 x,y，本函数计算 $c=\cos(\theta), s=\sin(\theta)$, 使得 $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$，

$function: \begin{bmatrix} c & s & r \end{bmatrix} = Givens(x,y)$

if $y=0$

$c=1, s=0;$

else

if $\left(|y|>|x|\right)$

$\tau = -x/y; s=\sqrt{1+\tau^2}, r=-y*s; s=1/s; c=s\tau;$

else

$\tau = -y/x; c=\sqrt{1+\tau^2}, r=x*c; c=1/c; s=c\tau;$

end

end

并且可知每次对奇异向量的更新都相当于对奇异矩阵右乘上一个相应的 Givens 矩阵 $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$，这只改变了矩阵的两列，具体操作可如下进行：

算法 3.1.2：

$Update(c,s,v_1,v_2)$: replace n-vectors v_1 and v_2 by $c*v_1-s*v_2$ and $s*v_1+c*v_2$

for i = 1 to n　　//这里的n为向量v_1， v_2的维数

$t=v_1(i)$

$v_1(i)=c*t-s*v_2(i)$

$v_2(i)=s*t+c*v_2(i)$

endfor

由此可知传统 SVD 算法中完成一次 QR 迭代可如下进行：

算法 3.1.3：

（1） 输入二对角矩阵 B 的对角元素$\delta_{\underline{i}}...\delta_{\overline{i}}$和次对角元素$\gamma_{\underline{i}+1}...\gamma_{\overline{i}}$；

//其中$\underline{i},\overline{i}$分别为子矩阵 B 在总矩阵中的上下标

（2） $d=\left[\left(\delta_{\overline{i}-1}^2+\gamma_{\overline{i}-1}^2\right)-\left(\delta_{\overline{i}}^2+\gamma_{\overline{i}}^2\right)\right]/2$,

$\mu=(\delta_{\overline{i}}^2+\gamma_{\overline{i}}^2)+d-sign(d)\sqrt{d^2+\delta_{\overline{i}-1}^2\gamma_{\overline{i}}^2}$,

$x=\delta_{\underline{i}}^2-\mu, y=\delta_{\underline{i}}\gamma_{\underline{i}+1}, k=\underline{i}$,

$Q=I, P=I$;

（3） 计算$c=\cos(\theta), s=\sin(\theta)$和$r$使得

$$[x,y]\begin{bmatrix} c & s \\ -s & c \end{bmatrix}=[r,0],$$

//可直接输入 x，y 调用**算法 3.1.1** 得到c,s和r；

并且更新

$$\begin{bmatrix} x & \gamma_{k+1} \\ y & \delta_{k+1} \end{bmatrix}=\begin{bmatrix} \delta_k & \gamma_{k+1} \\ 0 & \delta_{k+1} \end{bmatrix}\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

$Update(c,s,q_k,q_{k+1})$ //利用**算法 3.1.2**

//其中q_k,q_{k+1}分别为矩阵Q的第 k 和 k+1 列

（4） 如果$k>\underline{i}$，则$\gamma_k=r$；否则进行下一步

（5） 计算$c=\cos(\theta), s=\sin(\theta)$和$r$使得

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} r \\ 0 \end{bmatrix},$$

//可直接输入 x，y 调用**算法 3.1.1** 得到c,s和r；

$\delta_k=r$

$Update(c,s,p_k,p_{k+1})$ //利用**算法 3.1.2**

//其中p_k,p_{k+1}分别为矩阵P的第 k 和 k+1 列

（6） 如果$k<\overline{i}-1$，则

$$\begin{bmatrix} x & y \\ \delta_{k+1} & \gamma_{k+2} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} \gamma_{k+1} & 0 \\ \delta_{k+1} & \gamma_{k+2} \end{bmatrix}$$

$k = k+1$，转步（3）；

否则，

$$\begin{bmatrix} \gamma_{\bar{i}} \\ \delta_{\bar{i}} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} \gamma_{\bar{i}} \\ \delta_{\bar{i}} \end{bmatrix}$$

迭代结束。

上述算法的导出是在 $T = B^T B$ 不可约的条件下进行的。从 $T = B^T B$ 容易推出，T 不可约的充分必要条件是 δ_i 和 γ_i（除 δ_n 外）都不为零，而当某个 $\gamma_i = 0$ 时，B 具有形状

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}$$

因此，可将 B 的奇异值分解问题分解为两个低阶二对角阵的奇异值分解问题；而当某个 $\delta_i = 0$ 时，我们可以给 B 依次左乘 $(i,i+1),(i,i+2),\cdots,(i,n)$ 坐标平面内适当选取的 Givens 变换使 B 变为第 i 行全为零的二对角阵. 因此，此种情形亦可约化为两个低阶二对角阵的奇异值分解问题.

在实际计算时，当 δ_i 或 γ_j 很小时，就可将 B 分解为两个低阶二对角阵的奇异值分解问题.通常使用的准则是：当

$$|\delta_i| \le \varepsilon \|B\|_\infty \text{ 或 } |\gamma_j| \le \varepsilon \left(|\delta_j| + |\delta_{j-1}| \right)$$

时，就将 δ_i 或 γ_j 视作零，这里 ε 是一个略大于机器精度的正数.

综述上面的讨论，就可得到传统的计算奇异值分解的算法如下：

算法 3.1.4 (传统的 SVD 算法)

（1）输入 $A \in R^{m\times n}(m \ge n)$ 及允许误差 ε.

（2）二对角化：计算 Householder 变换 $P_1,\cdots,P_n$， $H_1,\cdots,H_{n-2}$ 使得

$$(P_1 \cdots P_n)^T A (H_1 \cdots H_{n-2}) = \begin{bmatrix} B \\ 0 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix},$$

其中 $B = \begin{bmatrix} \delta_1 & \gamma_2 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & \gamma_n \\ 0 & & & \delta_n \end{bmatrix}$；

$$U := P_1 P_2 \cdots P_n, \qquad V := H_1 H_2 \cdots H_{n-2}.$$

（3）收敛性检验：

（i）将所有满足

$$|\gamma_j| \le \varepsilon \left(|\delta_j| + |\delta_{j-1}| \right)$$

的 j 置零；

（ii）如果 $\gamma_j = 0, j = 2,\cdots,n$，则输出有关信息结束；否则，$\gamma_1 := 0$，确定正整数 $p < q$，使得

$\gamma_p = \gamma_{q+1} = \cdots = \gamma_n = 0$，　　$\gamma_j \neq 0$，　$p < j \le q$；

（iii）如果存在 i 满足 $p \le i \le q-1$ 使得

$$|\delta_i| \le \varepsilon \|B\|_\infty,$$

则 $\delta_i := 0, x := \gamma_{i+1}, y := \delta_{i+1}, \gamma_{i+1} := 0, l := 1$，转步（iv），否则转步（4）.

（iv）确定 $c = \cos\theta, s = \sin\theta$ 和 σ 使

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma \end{bmatrix},$$

//这也相对于 $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} \sigma \\ 0 \end{bmatrix}$, 所以可以直接调用**算法 3.1.1** 得到

$\delta_{i+l} := \sigma$,　$U := UG(i, i+l, \theta)^T$;

//这相当于 $U(1:n; i, i+l) = U(1:n; i, i+l)\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T = U(1:n; i, i+l)\begin{bmatrix} c & -s \\ s & c \end{bmatrix}$;

（v）如果 $l < q - i$，则

$x := s\gamma_{i+l+1}$，$\gamma_{i+l+1} := c\gamma_{i+l+1}$，$y := \delta_{i+l+1}$，$l := l+1$，

转步（iv），否则转步（i）.

（4）SVD 迭代：应用**算法 3.1.3** 于二对角阵

$$B_1 = \begin{bmatrix} \delta_p & \gamma_{p+1} & & & 0 \\ & \delta_{p+1} & \gamma_{p+2} & & \\ & & \ddots & \ddots & \\ & & & \ddots & \gamma_q \\ 0 & & & & \delta_q \end{bmatrix},$$

得

$$B_1 := P^T B_1 Q,$$

$$U := U\,diag(I_p, P, I_{n-p-q}),\ V := V\,diag(I_p, Q, I_{n-p-q})$$

然后转步（3）。

这一算法可计算任意一个 $m \times n$ 实矩阵 A 的奇异值分解：$A = U\Sigma V^T$。如果用 $\hat{U}$，$\hat{V}$ 和 $\hat{\Sigma}$ 分别表示 U，V 和 Σ 的计算值，则误差分析的结果表明：

$\hat{U} = W + \Delta U$，其中 $W^T W = I_m, \|\Delta U\|_2 \le \varepsilon$；

$\hat{V} = Z + \Delta V$，其中 $Z^T Z = I_n, \|\Delta V\|_2 \le \varepsilon$；

$\hat{\Sigma} = W^T(A + \Delta A)Z$，其中 $\|\Delta A\|_2 \le \|A\|_2 \varepsilon$。

这里 ε 为略大于机器精度的一个数。由此可见，这一算法有相当好的数值稳定性;再加上奇异值对扰动的不敏感性，即知利用这一算法可求得相当精确的奇异值，

其计算值和真实值之间的误差不大于 $\phi(n,p)\|A\|_2\varepsilon$，而因为 $\|A\|_2=\sigma_1$，所以较大的奇异值具有较高的相对精度，而越小的奇异值所具有的相对精度也越低，当奇异值大小和 $\sigma_1\varepsilon$ 接近时几乎完全失去精确度。

因为算法中涉及到迭代，所以去求算法所需要的准确的运算量是不可能，但是根据每一次循环所需要的计算量以及实际的试验测试还是可以得到运算复杂度的。虽然算法中出现了迭代，但是由于它是呈 3 次方快速收敛的[2]，所以本算法的复杂度并不高只有 $O(n^3)$ [13]，文献[23]当中给出了前面系数的大概估计为 20，也就是算法的复杂度是 $20n^3$。

关于**算法 3.1.3**，**算法 3.1.4**，在文献[2]中也给出了另外一种风格的伪代码，文献[3,24,25]都有完整的原代码。

3.2：传统 QR 迭代算法和零位移的 QR 迭代算法组合成的混合算法[13]

从传统方法的数值稳定性的分析中，可以看出，有必要引进具有更高精度的数值方法，所以在文献[13]中引进了一种求二对角矩阵的奇异值的方法，此方法对每一个奇异值都具有较高的相对精度，具体如以下各节所述。

假定我们已经求出上述传统方法中的二对角矩阵 B，于一般的带位移的 QR 迭代法不同的是我们这里选取 $\mu=0$；所不同还有，刚开始我们选取一个 Givens 变换矩阵将矩阵 B 的(1,2)位置上的元素 b_{12} 消零，而不是象传统方法引入一个非零元素 b_{21}；

以 n=4 时的例子具体如下：

先右乘 Givens 变换矩阵 J_1 将矩阵 B 的(1,2)位置上的元素 b_{12} 消零；

$$B^{(1)}=BJ_1=\begin{bmatrix} b_{11}^{(1)} & 0 & 0 & 0 \\ b_{21}^{(1)} & b_{22}^{(1)} & b_{23} & 0 \\ 0 & 0 & b_{33} & b_{34} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

然后左乘 Givens 变换矩阵 J_2 将矩阵元素 b_{21} 消零；

$$B^{(2)}=J_2BJ_1=\begin{bmatrix} b_{11}^{(2)} & b_{12}^{(2)} & b_{13}^{(2)} & 0 \\ 0 & b_{22}^{(2)} & b_{23}^{(2)} & 0 \\ 0 & 0 & b_{33} & b_{34} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

注意到：

$$\begin{bmatrix} b_{12}^{(2)} & b_{13}^{(2)} \\ b_{22}^{(2)} & b_{23}^{(2)} \end{bmatrix}=\begin{bmatrix} \sin\theta_2 b_{22}^{(1)} & \sin\theta_2 b_{23} \\ \cos\theta_2 b_{22}^{(1)} & \cos\theta_2 b_{23} \end{bmatrix}$$

是秩为 1 的矩阵，所以右乘 Givens 变换矩阵 J_3 将矩阵 B 的(1,3)位置上的元素 $b_{13}^{(2)}$ 消零时(2,3)位置上的元素 $b_{23}^{(2)}$ 也被消零了：

$$B^{(3)} = J_2 B J_1 J_3 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(3)} & 0 & 0 \\ 0 & b_{22}^{(3)} & 0 & 0 \\ 0 & b_{32}^{(3)} & b_{33}^{(3)} & b_{34} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

然后继续左乘 Givens 变换矩阵 J_4 将矩阵元素 $b_{32}^{(3)}$ 消零；

$$B^{(4)} = J_4 J_2 B J_1 J_3 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(3)} & 0 & 0 \\ 0 & b_{22}^{(4)} & b_{23}^{(4)} & b_{24}^{(4)} \\ 0 & 0 & b_{33}^{(4)} & b_{34}^{(4)} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

按同样的道理构造 J_5，J_6，最终可以将重新变换回二对角矩阵，从而完成一次迭代。

由上面的例子，并且利用**算法 3.1.1**，可知：假如已知 B 为 n 阶的二对角矩阵，对角元素为 $\delta_1,...,\delta_n$，非对角元素为 $\gamma_2,...,\gamma_n$；则经过一次零位移的 QR 迭代以后的 $\delta_1,...,\delta_n$ 和 $\gamma_2,...,\gamma_n$ 可由如下算法给出：

算法 3.2.1：

$oldc = 1$
$x = \delta_{\underline{i}}$
$y = \gamma_{\underline{i}+1}$
for $i = \underline{i}, \overline{i} - 1$　　//这里 $\underline{i}$，$\overline{i}$ 为子矩阵在总矩阵中的上下标
　call $[c,s,r] = Givens(x,y)$　　//调用**算法 3.1.1**
　$Update(c,s,v_i,v_{i+1})$　//利用**算法 3.1.2**
　　//其中 v_i, v_{i+1} 分别为矩阵 V 的第 i 和 i+1 列
　$if\ (i \neq \underline{i})\ \gamma_i = olds * r$
　$x = oldc * r$
　$y = -\delta_{i+1} * s$
　$h = \delta_{i+1} * c$
　call $[c,s,r] = Givens(x,y)$　　//调用**算法 3.1.1**
　$Update(c,s,u_i,u_{i+1})$　//利用**算法 3.1.2**
　　//其中 u_i, u_{i+1} 分别为矩阵 U 的第 i 和 i+1 列
　$\delta_i = r$
　$x = h$
　$if\ (i \neq \overline{i} - 1)\ y = \gamma_{i+2}$
　$oldc = c$
　$olds = s$
endfor
$\gamma_{\overline{i}} = -h * s$

$\delta_{\bar{i}} = h * c$

文献[13]中还给出了随着迭代的进行而最终累积的误差所满足的两个定理：

定理 3.2.1：

假设 B 是 n 阶的二对角矩阵，B'是对 B 进行一次零位移的 QR 迭代后得到的矩阵，$\sigma_1 \geq ... \geq \sigma_n$是原矩阵的准确的奇异值，而$\sigma'_1 \geq ... \geq \sigma'_n$是矩阵$B'$的奇异值；则当以下不等式成立时

$$\omega \equiv 69n^2\varepsilon < 1$$

则有以下式子满足：

$$|\sigma_i - \sigma'_i| \leq \frac{\omega}{1-\omega}\sigma_i$$

由此可知用零位移 QR 迭代 k 次以后得到的矩阵B_k，其奇异值$\sigma_{k1} \geq ... \geq \sigma_{kn}$，有以下式子满足：

$$|\sigma_i - \sigma_{ki}| \leq \left(\frac{1}{(1-\omega)^k} - 1\right)\sigma_i \approx 69kn^2\varepsilon\sigma_i$$，（当$k\omega$□ 1时，可取近似式）

定理 3.2.2：

假设 B 是 n 阶的二对角矩阵，B'是对 B 进行一次零位移的 QR 迭代后得到的矩阵，$\sigma_1 \geq ... \geq \sigma_n$是原矩阵的准确的奇异值，而$\sigma'_1 \geq ... \geq \sigma'_n$是矩阵$B'$的奇异值；并且 Givens 变换时的旋转角$\theta$都满足$\sin^2\theta \leq \tau < 1$，则当以下不等式成立时

$$\omega \equiv \frac{88n\varepsilon}{(1-\tau)^2} < 1$$

则有以下式子满足：

$$|\sigma_i - \sigma'_i| \leq \frac{\omega}{1-\omega}\sigma_i$$

由此可知用零位移 QR 迭代 k 次以后得到的矩阵B_k，其奇异值$\sigma_{k1} \geq ... \geq \sigma_{kn}$，有以下式子满足：

$$|\sigma_i - \sigma_{ki}| \leq \left(\frac{1}{(1-\omega)^k} - 1\right)\sigma_i \approx \frac{88kn\varepsilon}{(1-\tau)^2}\sigma_i$$，（当$k\omega$□ 1时，可取近似式）

而可知τ的值在算法计算的过程中可以很容易地检测。

文献中[13]还分析了传统 QR 迭代算法不能得到满足精度的三个原因；首要的一条是判断收敛的依据。

假设已知 B 为 n 阶的二对角矩阵，对角元素为$\delta_1,...,\delta_n$，非对角元素为$\gamma_2,...,\gamma_n$；在软件包 LINPACK 中用到的判断对角元素为零的依据是：

if $(|\gamma_{i+1}| + |\gamma_i| + |\delta_i| == |\gamma_{i+1}| + |\gamma_i|)$,

then ……(3.2.1)

$\delta_i = 0$

这相对于当$|\delta_i| < 0.5\varepsilon\left(|\gamma_{i+1}| + |\gamma_i|\right)$，就设$\delta_i = 0$，好处是不用知道机器精度$\varepsilon$。

另外一个依据是：

$if\ (|\delta_i| + |\delta_{i-1}| + |\gamma_i| == |\delta_i| + |\delta_{i-1}|),$

then ……(3.2.2)

$\gamma_i = 0$

分析知判断依据(3.2.1)可能会在没有 0 奇异值的情况下引入一个 0 奇异值，所以显然不可取；而再来看判断依据(3.2.2)，不妨取如下例子，η足够小，以致浮点运算中$1+\eta=1$；考虑矩阵

$$B(x) = \begin{bmatrix} \eta^2 & 1 & & \\ & 1 & x & \\ & & 1 & 1 \\ & & & \eta^2 \end{bmatrix}$$

当$x = \eta$时，容易验证最小的奇异值约为η^3；但是用判断依据(3.2.2)，则会将x设为 0，而$B(0)$的最小奇异值约为$\eta^2/\sqrt{2}$。

所以新的收敛判断依据必须要确保，当把γ_i设为 0 时，不会引起奇异值的大波动；同时严格来说我们不能把一个非 0 的δ_i设为 0。我们令$\underline{\sigma}$ 表示最小奇异值的可靠的下界估计，其值可如下迭代计算：

算法 3.2.2：

$\lambda_n = |\delta_n|$

for $j = n-1$ to 1 step -1 do

$$\lambda_j = |\delta_j|\left(\frac{\lambda_{j+1}}{\lambda_{j+1} + |\gamma_{j+1}|}\right)$$

$\mu_1 = |\delta_1|$

for $j = 1$ to n-1 step 1 do

$$\mu_{j+1} = |\delta_{j+1}|\left(\frac{\mu_j}{\mu_j + |\gamma_{j+1}|}\right)$$

Then $B^{-1-1}_{\infty} = \min_j \lambda_j ; B^{-1-1}_{1} = \min_j \mu_j$

$\underline{\sigma} = \min(B^{-1-1}_{\infty}, B^{-1-1}_{1})$

文献[13]证明了$\underline{\sigma} \le \sigma_{\min}(B) \le n^{1/2}\underline{\sigma}$；从中可以看出最简单可取的收敛判据是当$\gamma_i$小于$tol * \underline{\sigma}$时（$tol$是相对误差容许限），将它设为 0。但是这是一个过于保守的估计，耗时太久，所以一般取以下更实用的收敛判断依据：

Conergence Criterion 1*a*

假定μ_j是由**算法 3.2.2** 计算得到，则当$|\gamma_j / \mu_{j-1}| \le tol$时，把$\gamma_j$设置为 0 时。

Conergence Criterion 1*b*

假定 λ_j 是由**算法 3.2.2** 计算得到，则当 $\left|\gamma_j / \lambda_j\right| \le tol$ 时，把 γ_j 设置为 0 时。

Conergence Criterion 2*a*

假定 μ_j 是由**算法 3.2.2** 计算得到，当不需要求奇异向量时，则当

$\gamma_n^2 \le 0.5 * tol * \left[\left(\min_{j<n} \mu_j /(n-1)^{1/2} \right)^2 - \left|\delta_n\right|^2 \right]$ 时，把 γ_n 设置为 0 时。

Conergence Criterion 2*b*

假定 λ_j 是由**算法 3.2.2** 计算得到，当不需要求奇异向量时，则当

$\gamma_2^2 \le 0.5 * tol * \left[\left(\min_{j>1} \lambda_j /(n-1)^{1/2} \right)^2 - \left|\delta_1\right|^2 \right]$ 时，把 γ_2 设置为 0 时。

既然是混合算法，那么就要有一个依据来判断什么时候采用什么算法比较有优势，令 $\bar{\sigma} = \max_i(\left|\delta_i\right|, \left|\gamma_i\right|)$，现在就来给出这个选择算法的判断依据：

if (*fudge* * *tol* * ($\underline{\sigma} / \bar{\sigma}$) $\le \varepsilon$)

　　采用零位移 QR 迭代法

else

　　采用传统的位移 QR 迭代法

endif

这个判据的依据是：判断传统的位移 QR 迭代法所引起的误差上限 $\varepsilon\bar{\sigma}$ 是否超过了最大所能容许的误差 $tol * \underline{\sigma}$；而一般取因子 $fudge > 1$，这是为了在奇异值过于集中时，而少采用零位移的 QR 迭代法，一般来说取 $fudge = \min(n, m)$。

实际的应用中，当上面判据已经决定采用传统的位移 QR 迭代法，还要经过一步判断以确认是否回到采用零位移 QR 迭代法的路上。具体原因如下：当计算得到所要采取的位移平方 σ^2，知道 Givens 变换的旋转角的 tangent 值为

$$\tan\theta = \frac{\delta_1 * \gamma_2}{\sigma^2 - \delta_1^2} = \frac{\gamma_2}{\delta_1}\left(\frac{\sigma^2}{\delta_1^2} - 1 \right)^{-1}$$

所以当 $\frac{\sigma^2}{\delta_1^2} - 1 \approx -1$ 时，这个旋转和零位移 QR 迭代法中的 Givens 旋转变换很接近，而因为此时零位移 QR 迭代法更快更精确，所以我们将采用零位移 QR 迭代法。

tol 是一个用户可选择的 $\varepsilon \le tol \le 1$ 的数，当接近 1 的时候，几乎都采用传统的位移 QR 迭代法，这时计算得到的奇异值只有绝对精度；而当接近 ε 时，则几乎都采用零位移 QR 迭代法，这将使算法失去立方收敛速度，而降低效率。

接着要考虑的是向下越界(Underflow)，当从 γ_n 中减去一个数以使得它逼近 0 时，有可能造成向下越界(Underflow)。为防止向下越界，可以把 $\left|\gamma_j\right|$ 和 $maxit * \lambda$ 相比较，

当$|\gamma_j|$小于 *maxit*λ* 时就认为收敛；其中 *maxit* 是内部 QR 法最大可能出现的迭代次数，而 λ 是向下越界限（即机器可识别的最小正数）；这也就是说当矩阵具有接近或小于 λ 的奇异值时，算法将失效。

总以上我们已经完成了构造新算法的所有理论准备，但是具体的程序中还有很多的细节值得研究和考虑，以下一一说明，更详细的说明可参考[13]：

1）“从上往下”还是“从下往上”开始进行“驱逐出境”运算：

因为零位移的 QR 迭代法是按小的奇异值到大的奇异值的次序逐个收敛的，所以矩阵的元素如果是左上角大于右下角时，按前面所将的“从上往下”开始进行“驱逐出境”时，收敛将会是快速的；而反之矩阵的元素如果是左上角小于右下角时，选择“从下往上”开始进行“驱逐出境”更快。而为了简单起见算法中只是按着比较$|s_1|$和$|s_n|$的大小来判断方向的；而当矩阵分成很多子矩阵时，每一个子矩阵都有自己的方向。

首先来给出传统方法的 Upward 算法如下：

算法 3.1.3b：

（1） 输入二对角矩阵 B 的对角元素 $\delta_{\underline{i}}...\delta_{\overline{i}}$ 和次对角元素 $\gamma_{\underline{i}+1}...\gamma_{\overline{i}}$；

//其中 $\underline{i}$，$\overline{i}$ 分别为子矩阵 B 左上角元素和右下角元素在总矩阵中的标号

（2） $d=\left[\left(\delta_{\underline{i}+1}^2+\gamma_{\underline{i}+2}^2\right)-\left(\delta_{\underline{i}}^2+\gamma_{\underline{i}+1}^2\right)\right]/2$,

$\mu=(\delta_{\underline{i}}^2+\gamma_{\underline{i}+1}^2)+d-sign(d)\sqrt{d^2+\delta_{\underline{i}+1}^2\gamma_{\underline{i}+1}^2}$,

$x=\delta_{\overline{i}}^2-\mu, y=\delta_{\overline{i}-1}\gamma_{\overline{i}}, k=\overline{i}$,

$Q=I, P=I$;

（3） 计算 $c=\cos(\theta), s=\sin(\theta)$ 和 σ 使得

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} \sigma \\ 0 \end{bmatrix},$$

如果 $k<\overline{i}$ ，则 $\gamma_{k+1}=\sigma$；

更新:

$$\begin{bmatrix} x & y \\ \gamma_k & \delta_{k-1} \end{bmatrix}=\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} \delta_k & 0 \\ \gamma_k & \delta_{k-1} \end{bmatrix}$$

$Update(c, s, p_k, p_{k-1})$ //利用**算法 3.1.2**

//其中 p_k, p_{k-1} 分别为矩阵 P 的第 k 和 k－1 列

（4） 计算 $c=\cos(\theta), s=\sin(\theta)$ 和 σ 使得

$$\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} c & s \\ -s & c \end{bmatrix}=\begin{bmatrix} \sigma & 0 \end{bmatrix},$$

$\delta_k=\sigma$

$Update(c, s, q_k, q_{k-1})$ //利用**算法 3.1.2**

//其中 q_k, q_{k-1} 分别为矩阵 Q 的第 k 和 k－1 列

（5） 如果 $k > \underline{i}+1$，则

$$\begin{bmatrix} x & \delta_{k-1} \\ y & \gamma_{k-1} \end{bmatrix} = \begin{bmatrix} \gamma_k & \delta_{k-1} \\ 0 & \gamma_{k-1} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

$k = k-1$，转步（3）；

否则，

$$\begin{bmatrix} \gamma_{\underline{i}+1} & \delta_{\underline{i}} \end{bmatrix} = \begin{bmatrix} \gamma_{\underline{i}+1} & \delta_{\underline{i}} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

迭代结束。

接着再给出零位移 QR 迭代算法的 Upward 算法如下：

算法 3.2.1b:

$oldc = 1$
$x = \delta_{\bar{i}}$
$y = \gamma_{\bar{i}}$
for $i = \bar{i} : -1 : (\underline{i}+1)$
  call $[c, s, r] = Givens(x, y)$ //调用**算法 3.1.1**
  $Update(c, s, u_i, u_{i-1})$ //利用**算法 3.1.2**
    //其中 u_i, u_{i-1} 分别为矩阵 U 的第 i 和 i－1 列
  $if\ (i \neq \bar{i})\quad \gamma_{i+1} = -olds * r$
  $x = oldc * r$
  $y = -\delta_{i-1} * s$
  $h = \delta_{i-1} * c$
  $oldc = c$
  $olds = s$
  call $[c, s, r] = Givens(x, y)$
  $Update(c, s, v_i, v_{i-1})$ //利用**算法 3.1.2**
    //其中 v_i, v_{i-1} 分别为矩阵 V 的第 i 和 i－1 列
  $\delta_i = r$
  $x = h$
  $if\ (i \neq \underline{i}+1)\quad y = \gamma_{i-1}$
endfor
$\gamma_{\underline{i}+1} = -h * s$
$\delta_{\underline{i}} = h * c$

2） 应用收敛判据

在前面已经给出了四个收敛判据，因为应用收敛判据需要很多额外的运算量，所以尽量只当可能收敛时才利用收敛判据；从经验观察可知，当“从上往下”开始进行“驱逐出境”时，一般最底层的元素 δ_n 趋向于收敛到最小的奇异值，而且 γ_n 比

其它非对角元素更快地趋于 0；所以当应用“从上往下”“驱逐出境”时，一般采用 *Criterion* 1*a*， *Criterion* 2*a*，以及对最下角元素 γ_n 用 *Criterion* 1*b*；反之当应用“从下往上”“驱逐出境”时，一般采用 *Criterion* 1*b*， *Criterion* 2*b*，以及对最上角元素 γ_2 用 *Criterion* 1*a*。

3） 当碰到 δ_i 或 γ_i 为零时的处理方法与 **3.1** 中的传统 QR 迭代的 SVD 算法中的处理方法相同。

4） 当二对角矩阵收敛出现 2×2 的子矩阵时，直接计算，具体最作法如下：

算法 3.2.3：

令 G 是上对角的 2×2 矩阵，元素为 $G=\begin{bmatrix} g_{11} & g_{12} \\ 0 & g_{22} \end{bmatrix}$，在大矩阵 B 中的位置是 i，$i+1$

//以下进行右奇异向量的计算

/* compute $\begin{bmatrix} a & c \\ c & b \end{bmatrix} = G^T G$ */

$a = g_{11}^2$

$b = g_{12}^2 + g_{22}^2$

$c = g_{11} * g_{12}$

/* compute the Jacobi rotation which diagonalizes $\begin{bmatrix} a & c \\ c & b \end{bmatrix}$ */

$\zeta = (b-a)/(2c)$

$t = sign(\zeta)/\left(|\zeta| + \sqrt{1+\zeta^2}\right))$

$cs = 1/\sqrt{1+t^2}$

$sn = cs * t$

/* update columns i and $i+1$ of B */

for $k = i$ to $i+1$

$tmp = B_{ki}$

$B_{ki} = cs * tmp - sn * B_{k,i+1}$

$B_{k,i+1} = sn * tmp + cs * G_{k,i+1}$

endfor

/* update the matrix V of right singular vectors */

for $k = 1$ to n

$tmp = V_{ki}$

$V_{ki} = cs * tmp - sn * V_{k,i+1}$

$V_{k,i+1} = sn * tmp + cs * V_{k,i+1}$

Endfor

//以下进行左奇异向量的计算

$$\alpha = \sqrt{B_{ii}^2 + B_{i+1,i}^2}$$

$$\beta = \sqrt{B_{i,i+1}^2 + B_{i+1,i+1}^2}$$

$$c1 = B_{ii} / \alpha; c2 = B_{i+1,i+1} / \beta$$

$$s1 = B_{i+1,i} / \alpha; s2 = B_{i,i+1} / \beta$$

/* update rows i and $i+1$ of B */

for $k = i$ to $i+1$

$tmp = B_{ik}$

$B_{ik} = c1 * tmp + s1 * B_{i+1,k}$

$B_{i+1,k} = s2 * tmp + c2 * G_{i+1,k}$

endfor

/* update the matrix U of right singular vectors */

for $k = 1$ to n

$tmp = U_{ki}$

$U_{ki} = c1 * tmp + s1 * U_{k,i+1}$

$U_{k,i+1} = s2 * tmp + c2 * U_{k,i+1}$

endfor

至此我们完成了所有的准备工作，在文献[13]中给出了完整的算法描述，在它的基础上略作修改说明，给出完整的算法描述如下：

算法 3.2.4：

ε = machine precision

λ = underflow threshold (smallest positive normalized number)

n = dimension of the matrix

tol = relative error tolerance (currently 100ε)

$maxit$ = maximum number of QR inner loops (currently $3n^2$)

初始化 U，V

//当直接对二对角矩阵应用此算法则直接设 U＝I，V＝I

//当对一般矩阵二对角化以后才利用此算法则设，

//$U := P_1 P_2 \cdots P_n$，$V := H_1 H_2 \cdots H_{n-2}$.

Bidiagonal Singular Value Decomposition

Compute $\underline{\sigma} \le \sigma_{\min}(B)$ //利用**算法 3.2.2**

$\bar{\sigma} = \max(|\delta_i|, |\gamma_i|)$

$thresh = \max(tol \cdot \underline{\sigma}, \max it \cdot \lambda)$

/* any γ_i less than $thresh$ in magnitude may be set to zero */

Loop:

/* Find bottommost nonscalar unreduced block diagonal submatrix of B */
let $\bar{i}$ be the smallest i such that $|\gamma_{i+1}|$ through $|\gamma_n|$ are at most $thresh$, or n if no such i exists
if $\bar{i}=1$, goto Done
let i' be the largest i less than $\bar{i}$ such that $|\gamma_{i+1}| \le thresh$, or 0 if no such i exists
$\underline{i} = i' + 1$
/* Apply algorithm to unreduced block diagonal submatrix from $\underline{i}$ to $\bar{i}$ */
if $\bar{i} = \underline{i} + 1$, then
/* 2 by 2 submatrix, handle specially */
Compute SVD of 2 by 2 submatrix, setting $\gamma_{\underline{i}+1}$ to 0
//利用**算法 3.2.3**
goto Loop
endif

if submatrix from $\underline{i}$ to $\bar{i}$ disjoint from submatrix of last past through Loop, then
/* Choose bulge chasing direction */
if $|\delta_{\underline{i}}| \ge |\delta_{\bar{i}}|$, then
$direction = "down"$
else
$direction = "up"$
endif
endif

/* Apply convergence criteria */
if $direction = "down"$, then
Apply convergence criterion 1*b* to $\gamma_{\bar{i}}$
Apply convergence criterion 1*a*
else
Apply convergence criterion 1*a* to $\gamma_{\underline{i}+1}$
Apply convergence criterion 1*b*
endif

/* Compute shift */
if $fudge * tol * \underline{\sigma} / \bar{\sigma} \le \varepsilon$, then
/* Use zero shift because tiny singular values present */
$shift = 0$
else
if $direction = "down"$, then
$s = \delta_{\bar{i}}$
$shift =$ smallest singular value of bottom 2 by 2 corner
else

$s = \delta_{\underline{i}}$

$shift$ = smallest singular value of bottom 2 by 2 corner

endif

if $(shift/s)^2 \le eps$, then

/* Use zero shift, since shift rounds to 0 */

$shift = 0$

endif

endif

/* Perform QR iteration */

if $shift = 0$, then

if $direction = "down"$, then

do implicit zero-shift QR downward

//上述语句传入参数 $\underline{i}$， $\overline{i}$ 调用**算法 3.2.1**，

if $|\gamma_{\overline{i}}| \le thresh$, set $\gamma_{\overline{i}} = 0$

else

do implicit zero-shift QR upward

//上述语句传入参数 $\underline{i}$， $\overline{i}$ 调用**算法 3.2.1b**，

if $|\gamma_{\underline{i}+1}| \le thresh$, set $\gamma_{\underline{i}+1} = 0$

endif

else

if $direction = "down"$, then

do standard shifted QR downward

//上述语句传入参数 $\underline{i}$， $\overline{i}$ 调用**算法 3.1.3**，

if $|\gamma_{\overline{i}}| \le thresh$, set $\gamma_{\overline{i}} = 0$

else

do standard shifted QR upward

//上述语句传入参数 $\underline{i}$， $\overline{i}$ 调用**算法 3.1.3b**，

if $|\gamma_{\underline{i}+1}| \le thresh$, set $\gamma_{\underline{i}+1} = 0$

endif

endif

goto Loop

Done: sort singular values　　//对特征值和特征向量进行排序

在文献[14]中对这一算法进行了比较形象的分析，简单地说这一算法是一种传统带位移的 QR 迭代算法和零位移 QR 迭代算法的混合算法。当矩阵“条件良好”时（σ_n 和 σ_1 相差不多时）采用传统带位移的 QR 迭代算法，反之采用零位移 QR 迭代算法；零位移 QR 迭代算法非常的精确，但是当矩阵“条件良好”时，收敛却非常的慢，幸运的是传统带位移的 QR 迭代算法在“条件良好”时精确度很高；所以我们只需在矩阵“条件不良好”时引进高精度的零位移 QR 迭代算法，这样就在整体

上保证了算法的精确度同时也保证了收敛速度。文献[14]也给出了一种表述比较简洁的整体算法如下：

算法 3.2.5： Bidiagonal SVD Algorithm (simplified)

Loop:

Find the bottommost unreduced submatrix of B; call it $\hat{B}$.

(Let s and e be the starting and ending indices of $\hat{B}$ within B.

Then $b_e = 0$ if e < n, $b_{s-1} = 0$ if s > 1 and $b_i \neq 0$ for $s \le i \le e$.)

If $\hat{B}$ is 1 by 1 (s = e), we are done.

Apply the stopping criterion to $\hat{B}$; if any b_i are set to 0, return to Loop

Estimate the smallest singular value $\underline{\sigma}$ and the largest singular value $\bar{\sigma}$ of $\hat{B}$.

if n* $\underline{\sigma} / \bar{\sigma} < \max(\varepsilon / tol, 0.01)$ then

Use implicit zero-shift QR on $\hat{B}$

else

Use standard shifted QR on $\hat{B}$

endif

Goto Loop

由**算法 3.2.4，3.2.5** 求得的奇异向量在误差估计方面满足如下的定理：

定理 3.2.3：

由**算法 3.2.5** 计算所得到的第 i 个左奇异向量以及右奇异向量与准确值之间的误差不超过：

$$p(n,m) * tol / relgap_i$$

其中 $p(n,m)$ 是矩阵维数 n 和迭代次数 m 的一个低阶函数，tol 是程序中控制相对精度的参量，与 σ_i 对应的 $relgap_i$ 定义如下：

$$relgap_i = \min_{j \neq i} \left| \sigma_i - \sigma_j \right| / \left| \sigma_i + \sigma_j \right|$$

文献[13]中以准确向量和计算所得向量之间的角度形式给出了误差为：

$$\max\left(\theta(\hat{u}_i, u_i), \theta(\hat{v}_i, v_i)\right) \le p(n) * \varepsilon / relgap_i \equiv p(n) * \varepsilon / \min(\left|\sigma_i - \sigma_{i\pm1}\right| / \sigma_i)$$

试验检验得到的结果，**混合算法 3.2.4，3.2.5** 在性能上和**传统算法 3.1.4** 相比，如果考虑二对角化过程，那么**算法 3.2.4，3.2.5** 最佳的情况下可比**传统算法 3.1.4** 快 2.7 倍，而最坏的情况下会比**传统算法 3.1.4** 慢 1.6 倍;而当不考虑二对角，而仅仅从二对角矩阵出发求 SVD，那么**算法 3.2.4，3.2.5** 最佳的情况下可比**传统算法 3.1.4** 快 7.7 倍，而最坏的情况下会比**传统算法 3.1.4** 慢 3.4 倍[13]。

3.3：quotient difference(qd) 算法

在文献[15]中，将零位移 QR 迭代算法中的一步零位移的 QR 分解用两步零位移的 LR 分解来代替，从而得到求解 SVD 的 qd 算法。

在下面的讨论中，将经常遇到 Cholesky 分解中的概念和结论，这里简便起见我们把它们当成已知，而关于 Cholesky 分解的详细介绍在下一篇报告中将详细介绍，部分概念在前面的矩阵求逆的报告中也有介绍。

已知对于一个正定的厄密矩阵 A，对它可以进行 Cholesky 分解 $A = LL^*$，其中 L 为下三角矩阵。

现在定义一个对称正定矩阵 $A = LL^*$ 的 Cholesky LR 变换（transform）为

$$\hat{A} = L^* L$$

Cholesky LR 算法包含一系列这样的 Cholesky 变换，是一种特殊的 Rutishauser's LR 算法。以下的定理给出了构造 Cholesky LR 算法的理论依据：

定理 3.3.1

令 $\hat{A}$ 是对称正定矩阵 $A = LL^*$ 的 Cholesky 变换后的矩阵，并且它的 Cholesky 分解为 $\hat{A} = \hat{L}\hat{L}^*$；那么 L 存在如下的 QR 分解式：

$$L = Q\hat{L}^*;$$

从**定理 3.3.1** 可以得知一步 QR 分解算法等价于两步 Cholesky 算法[15]。而且式子 $\hat{L}\hat{L}^T = L^T L$ 保证了 $\hat{L}$ 和 L 的带宽一致，特别地当 L 是二对角矩阵 B 的时候，$A = B^T B$ 为三对角矩阵，此时 $\hat{L}$ 也为二对角矩阵，记为 $\hat{B}$；以下我们来说明如何从 B 推导得到 $\hat{B}$；为了方便我们采用和文献[15]中一致的表示如下：

$$B = bidiag\begin{Bmatrix} b_1 & b_2 & . & b_{n-2} & b_{n-1} \\ a_1 & a_2 & . & . & a_{n-1} & a_n \end{Bmatrix}$$

和

$$\hat{B} = bidiag\begin{Bmatrix} \hat{b}_1 & \hat{b}_2 & . & \hat{b}_{n-2} & \hat{b}_{n-1} \\ \hat{a}_1 & \hat{a}_2 & . & . & \hat{a}_{n-1} & \hat{a}_n \end{Bmatrix}$$

由**定理 3.3.1** 可以得知

$B^T = Q\hat{B}$； $\hat{B} = Q^T B^T$

其中 Q 是 n－1 个旋转矩阵的乘积：

$$Q = G_1 G_2 ... G_{n-1}$$

可知在消去 B^T 中的下次对角线元素 b_k 时，所需要作修改的矩阵部分为：

$$\begin{matrix} 0 & \hat{a}_{k-1} & \hat{b}_{k-1} & & \\ & 0 & \tilde{a}_k & 0 & \\ & & b_k & a_{k+1} & 0 \\ & & & b_{k+1} & a_{k+2} \end{matrix} \qquad \ldots(3.3.1)$$

在平面旋转 G_k^T 之后此部分变为

$$\begin{matrix} 0 & \hat{a}_{k-1} & \hat{b}_{k-1} & & \\ & 0 & \hat{a}_k & \hat{b}_k & \\ & & 0 & \tilde{a}_{k+1} & 0 \\ & & & b_{k+1} & a_{k+2} \end{matrix} \quad \ldots(3.3.2)$$

取 $B^{(0)} = B^T$，则对 $k = 1,...,n-1$，求

$$B^{(k)} = G_k^T B^{(k-1)}; \quad \ldots(3.3.3)$$

最终可以得到 $\hat{B} = B^{(n-1)}$。

由(3.3.1)和(3.3.2)，并取 $\tilde{a}_1 = a_1$ 和 $c_k^2 + s_k^2 = 1$，可得到计算公式为：

$$\hat{a}_k = \sqrt{\tilde{a}_k^2 + b_k^2} \quad \ldots(3.3.4)$$

$$s_k = b_k / \hat{a}_k$$

$$c_k = \tilde{a}_k / \hat{a}_k$$

$$\hat{b}_k = s_k a_{k+1} = b_k a_{k+1} / \hat{a}_k \quad \ldots(3.3.5)$$

$$\tilde{a}_{k+1} = c_k a_{k+1} = \tilde{a}_k a_{k+1} / \hat{a}_k \quad \ldots(3.3.6)$$

定义：

$$cabs(x, y) = \sqrt{x^2 + y^2} \quad \ldots(3.3.7)$$

可以得到如下的算法来实现一次 Cholesky 变换：

算法 3.3.1(oqd):

$\tilde{a} = a_1$

for $k = 1, n-1$

$\hat{a}_k = cabs(\tilde{a}, b_k)$

$\hat{b}_k = b_k * (a_{k+1} / \hat{a}_k)$

$\tilde{a} = \tilde{a} * (a_{k+1} / \hat{a}_k)$

end for

$\hat{a}_n = \tilde{a}$

这个算法离最终的实用算法还要经过很多改进，但是从这里我们已经可以看出它和 DK Zero Shift QR 算法相比所具有的优越性了，以下图表给出了它们每进行一次循环时在复杂度上以及额外变量上的比较。

	DK	oqd
cabs	2	1*2
Divisions	2	1*2
Multiplication	6	2*2
Conditionals	1	0
Assignments	7	3*2
Auxiliary variables	6	1

从中可以看出 DK 方法用了 6 个辅助变量，而 oqd 只用了 1 个；从而 oqd 在读写操作上节省了大量的时间。

我们很容易避免出现在 oqd 算法中的开平方运算，为了方便，我们设 $b_n = 0$；并定义 $q_k = a_k^2, e_k = b_k^2, k = 1, 2, ..., n$；从而得到 oqd 算法的变形算法：

算法 3.3.2(dqd)：

$d = q_1$
for $k = 1, n-1$
$\hat{q}_k = d + e_k$
$\hat{e}_k = e_k * (q_{k+1} / \hat{q}_k)$
$d = d * (q_{k+1} / \hat{q}_k)$
end for
$\hat{q}_n = d$

算法 3.3.2(dqd)中的中间变量 d 还可以去掉，从而得到 Rutishauser 最早提出的 qd 算法如下：

算法 3.3.3(qd)：

$\hat{e}_0 = 0$
for $k = 1, n-1$
$\hat{q}_k = (q_k - \hat{e}_{k-1}) + e_k$
$\hat{e}_k = e_k * q_{k+1} / \hat{q}_k$
end for
$\hat{q}_n = q_n - \hat{e}_{n-1}$

关于以上三种 qd 型的算法在复杂度上的比较如下：

	oqd	dqd	qd
cabs	1	0	0
Divisions	2	1	1
Multiplication	4	2	1
Additions	1	1	1
Subtractions	0	0	1
Assignments	3	3	2
Auxiliary variables	1	1	0

对于这三种算法在数值上都是稳定的，因为所有的中间变量都以 $\|B\|^2$ 为界限，这保证了求得的误差和 σ_1^2 相比都很小；但是对于最小的奇异值，却不一定具有很高相对精度。

试验证明，**算法 3.3.1(oqd)，算法 3.3.2(dqd)**所求得的结果和 DK 算法一样具有很高的相对精度，而**算法 3.3.3(qd)**却不具有相对精度。

算法 3.3.2(dqd)虽然比**算法 3.3.1(oqd)**快，但是在适用范围上，却没有**算法 3.3.1**

(oqd)广，比如在可表示数值从 2^{-1022} 到 2^{1023} 的双精度机器上，dqd 算法可以将条件数不超过 2^{1022} 的二对角矩阵进行对角化，而 oqd 算法却可以做到将条件数不超过 2^{2045} 的二对角矩阵进行对角化。当然一般的时候 $2^{1022} \approx 10^{308}$ 已经够实际应用了。

在精确度方面的分析表明，dqd 算法每进行一次循环，对任何奇异值的改变不超过 3(n-1) *ulps* (units in the last place)。

在收敛速度方面，qd 系列算法具有线性收敛速度，且其非对角线元素 $\left\{b_i^{(l)}\right\}_{l=1}^{\infty}$ 收敛于 0 的速度正比于 σ_{i+1}/σ_i [15]。

上面只给出了算法的核心部分，关于当碰到 s_i 或 e_i 为零时的处理方法，以及收敛判据等问题可以采用和**算法 3.6** 完全相同的处理方法，所以在此不进行重复。

最后给出求解奇异向量的方法，假定 B 的奇异值分解式为 $B = U\Sigma V^T$；并且取 $B_1 = B$。根据**定理 3.4** 可以得知：

$$B_2 = Q_1^T B_1^T = Q_1^T V \Sigma U^T$$
$$B_3 = Q_2^T B_2^T = Q_2^T U \Sigma V^T Q_1$$
……
$$B_{2k+1} = Q_{2k}^T \ldots Q_4^T Q_2^T U \Sigma V^T Q_1 Q_3 \ldots Q_{2k-1}$$

当 $k \to \infty$ 时，$B_{2k+1} \to \Sigma$，所以：

$$U = \lim_{k\to\infty} Q_2 Q_4 \ldots Q_{2k}$$
$$V = \lim_{k\to\infty} Q_1 Q_3 \ldots Q_{2k-1}$$

从而得到了构造 U 和 V 的方法：对矩阵 B 应用 oqd 或 dqd 算法，分别累积奇数次和偶数次的平面旋转变换，最终得到奇异向量构成的矩阵 V 和 U。

和 QR 迭代算法类似的是，qd 系列算法也可以构造带位移的算法形式(当然在处理方法上是有很大不同的)：

算法 3.3.4(oqds)：

$\tilde{a} = a_1$
for $k = 1, n-1$
$\quad \hat{a}_k = \sqrt{\tilde{a}^2 + b_k^2 - \tau^2}$
$\quad \hat{b}_k = b_k * (a_{k+1} / \hat{a}_k)$
$\quad \tilde{a} = \sqrt{\tilde{a}^2 - \tau^2} * (a_{k+1} / \hat{a}_k)$
end for
$\hat{a}_n = \sqrt{\tilde{a}^2 - \tau^2}$

可以验证 $\hat{B}^T\hat{B} = BB^T - \tau^2 I$ ……(3.3.8)

为了保证 $\hat{B}$ 为实矩阵，那么位移 τ 必须满足：

$$\tau \le \sigma_n[B] \ldots\ldots (3.3.9)$$

这一限制条件在带位移的 dqd 算法中可不必满足：

算法 3.3.5(dqds)：

$d = q_1 - \tau^2$

for $k = 1, n-1$

$\hat{q}_k = d + e_k$

$\hat{e}_k = e_k * (q_{k+1} / \hat{q}_k)$

$d = d * (q_{k+1} / \hat{q}_k) - \tau^2$

end for

$\hat{q}_n = d$

算法 3.3.3(qd)也有带位移的形式，但是考虑到没有实用性，这里就不列出了。

由式子(3.3.8)可以得知，带位移的 qd 系列算法不具有象 qd 系列算法那样由**定理 3.4** 保证的正交联系性(orthogonal connection):

$B^T = Q\hat{B}$

从而 $B^T, \hat{B}$ 的奇异值也不是相等的，而是相差一个位移量 τ，这点要特别注意；关于位移 τ 的选取应该使得 $\hat{q}_n$ 尽可能地小，所以一般取

$\tau^2 \approx d_n = q_n(1 - e_{n-1} / \hat{q}_{n-1})$

带位移的 qd 系列算法具有的是平方次收敛速度，所以在速度上比零位移的 qd 系列算法要快的多。

但是在求奇异向量时却也要付出较大的代价，因为位移是不可修复的，并且是非正交变换，这就造成了奇异向量的丢失。此时一个办法就是间隔地采用零位移，这样就可以保留其中的一个奇异向量集合。

对于一个二对角矩阵 B，如果它的最后一个元素 $a_n = 0$，那么它的 oqd 变换得到的矩阵 $\hat{B}$ 的元素 $\hat{a}_n = 0, \hat{b}_{n-1} = 0$，其中 $B^T = Q\hat{B}$；所以 Q 的最后一列就是和奇异值 0 相对应的右奇异向量了。

所以当奇异值已知时，可以将它们排序，作为 oqds 算法中的位移，然后依次来得到所需要的奇异向量。具体算法如下：

算法 3.3.6：

1), 初始化： $B_n = B; U = I, j = n, \sigma_{n+1}^2 = 0$

2), 对 B_j 应用 dqds 算法，其中 $\tau^2 = \sigma_j^2 - \sigma_{j+1}^2$；得到 $\hat{B}_j$

3), 对 $\hat{B}_j$ 应用 dqd 算法得到 B_{j-1}，并且保留 $\{c_i^2; s_i^2, i = 1, ..., j-1\}$，

其中 $c_i^2 = \dfrac{d_i}{d_i + e_i}; s_i^2 = \dfrac{e_i}{d_i + e_i}$

4), 取它们的正的平方根 $\{c_i; s_i, i = 1, ..., j-1\}$

5), 将相应的平面旋转变换累积到矩阵 U 中， $U = UG_i; i = 1, ..., j-1$，

6), $j = j - 1$, 当 j＝1 时停止，否则从第 2）步开始循环。

类似的算法也用来求 V，当对 B 应用一次 dqd 算法可以得到矩阵 $\hat{B}$（其中 $B^T = Q\hat{B}$），现在假定 $B = U\Sigma V^T$， $\hat{B} = \hat{U}\hat{\Sigma}\hat{V}^T$；由 $B^T = Q\hat{B}$ 可以得知：

$$B = U\Sigma V^T = \hat{B}^T Q^T = \hat{V}\hat{\Sigma}^T\hat{U}^T Q^T;$$

$$V = Q\hat{U};$$

对于矩阵 $\hat{B}$ 可以用**算法 3.3.6** 求得 $\hat{U}$，而 Q 也容易求出，从而可以求出原矩阵奇异值分解中的 V。

3.4：Jacobi 方法

虽然以上 3.2 和 3.3 中的方法都可以对二对角矩阵求出具有很高的相对精度的奇异值和奇异值分解式；但是将一个稠密矩阵二对角化这个过程却可能会造成很大的相对误差[17]。所以所有基于先将矩阵二对角化的 SVD 方法（前面的方法；以及最后要提到的分而治之算法）；都不可能具有可靠的良好的相对精确度。以下所要介绍的 Jacobi 方法和这些方法完全不同，它通过一系列平面旋转（一般文献中在介绍 Jacobi 方法时为了方便也将平面旋转称为 Jacobi 变换）；最终使得矩阵收敛于一个对角矩阵[18]：

$$G^{(k+1)} = G^{(k)} J_k, k = 0,1,2,...(G^{(0)} \equiv G)$$

其中每一个 J_k 设计成，使得对于一对选定的指标 (p_k, q_k)，有如下式子成立：

$$((G^{(k+1)})^T G^{(k+1)})_{p_k, q_k} = 0$$

最终收敛时，令 $G_\infty = \lim_{k\to\infty} G^{(k)}, V = \Pi_{k=0}^{\infty} J_k$，有 $G_\infty = GV$，且 G_∞ 具有互相正交的列向量，可以将 G_∞ 写成 $G_\infty = \hat{U}\Sigma$，其中 Σ 为对角矩阵，$\hat{U}$ 为正交矩阵；从而得到原矩阵的 SVD 分解式。

在文献[17]中对基本的 Jacobi 算法的推导过程进行了详细的说明。首先我们来给出一个对选择收敛判据和误差分析很有用的定理：

定理 3.4.1

令 $G = BD$ 是一个任意的满秩矩阵，其中 D 为对角矩阵，B 是列向量范数全为 1 的矩阵。并且 $\delta G = \delta BD$ 是 G 的一个扰动（误差），令 σ_i, σ_i' 分别是 G 和 $G + \delta G$ 的第 i 个奇异值，那么当 $\|\delta B\|_2 \equiv \eta < \sigma_{\min}(B)$ 时，σ_i, σ_i' 满足如下式子：

$$\frac{|\sigma_i - \sigma_i'|}{\sigma_i} \le \frac{\eta}{\sigma_{\min}(B)} \le \kappa(B) * \eta \quad \ldots\ldots(3.4.1)$$

其中 $\kappa(B) = \dfrac{\sigma_{\max}(B)}{\sigma_{\min}(B)} \le \dfrac{n^{1/2}}{\sigma_{\min}(B)}$，$n$ 为矩阵 G 的列数。

通常应用中，如果 $|\delta G_{ij} / G_{ij}| \le \eta / n$，那么就有 $\|\delta B\|_2 \le \eta$，此时奇异值的相对误差满足式子(3.4.1)。

现在来构造算法，令 $G_0 = B_0 D_0$ 是最初的矩阵，$G_m = B_m D_m$ 是 G_{m-1} 经过一次 Jacobi 旋转过得到的矩阵，其中 D_m 为对角矩阵，B_m 是列向量范数全为 1 的矩阵。可以知

道对 $G = BD$ 进行单边的 Jacobi 变换相当于对矩阵 $H = G^T G = DB^T BD = DAD$ 进行双边的 Jacobi 变换[17]。和一般的单边 Jacobi 算法不一样的地方是：根据**定理 3.4.1** 收敛判据应该为，所有的 $H_{ij}/(H_{ii}H_{jj})^{1/2}$ 足够小，而不仅仅是 $H_{ij}/\max_{kl}|H_{kl}|$ 足够小。从而得到如下完整的最基本的 Jacobi 算法：

算法 3.4.1

repeat

for all pairs $i < j$

/* compute $\begin{bmatrix} a & c \\ c & b \end{bmatrix} \equiv$ the (i, j) submatrix of $G^T G$ */

$a = \sum_{k=1}^{n} G_{ki}^2$

$b = \sum_{k=1}^{n} G_{kj}^2$

$c = \sum_{k=1}^{n} G_{ki} * G_{kj}$

/* compute the Jacobi rotation which diagonalizes $\begin{bmatrix} a & c \\ c & b \end{bmatrix}$ */

$\zeta = (b - a)/(2c)$

$t = sign(\zeta)/\left(|\zeta| + \sqrt{1 + \zeta^2}\right)$

$cs = 1/\sqrt{1 + t^2}$

$sn = cs * t$

/* update columns i and j of G */

for $k = 1$ to n

$tmp = G_{ki}$

$G_{ki} = cs * tmp - sn * G_{kj}$

$G_{kj} = sn * tmp + cs * G_{kj}$

endfor

/* update the matrix V of right singular vectors */

for $k = 1$ to n

$tmp = V_{ki}$

$V_{ki} = cs * tmp - sn * V_{kj}$

$V_{kj} = sn * tmp + cs * V_{kj}$

endfor

endfor

until convergence $\left(all\,|c|/\sqrt{ab} \le tol\right)$

最终得到的矩阵 G 的列向量的范数就是所要求的奇异值，并且最终归一化以后的矩阵 G 的列向量就是左奇异向量。

关于奇异值的误差有如下的定理：

定理 3.4.2

令 G_m 是有限精度 ε 下单边 Jacobi 算法中所得到的矩阵列，G_{m+1} 由 G_m 经过一次单边 Jacobi 变换得到；那么如下图所示它们满足如下关系：

$$\begin{array}{ccc} G_m & \rightarrow & G_{m+1} \\ +\delta G_m & \downarrow & \square \\ & G'_m & \end{array}$$

顶上的水平箭头表示 G_{m+1} 由 G_m 经过一次浮点数精度下的单边 Jacobi 变换得到；斜对角箭头表示 G_{m+1} 由 G'_m 经过精确的单边 Jacobi 变换得到，所以 G_{m+1} 和 G'_m 具有完全相同的奇异值和奇异向量；垂直箭头表示 $G'_m = G_m + \delta G_m$，记 $\delta G_m = \delta B_m D_m$，其中 D_m 为使 $G_m = B_m D_m$ 中的 B_m 具有单位列向量形式的对角矩阵；那么有如下的误差限成立：

$$\left\| \delta B_m \right\|_2 \le 72\varepsilon$$

这就为估计一次 Jacobi 变换所带来的误差提供了方便。

结合**定理 3.4.1**，可以得到如下的误差估计式子：

推理 3.4.3：

假定**算法 3.4.1** 收敛，并且 G_M 为最终满足收敛判据的矩阵；对 $0 \le m \le M$ 记 $G_m = B_m D_m$；记 σ_j 是原矩阵 G_0 的第 j 个精确的奇异值，σ'_j 为第 j 个计算得到的奇异值，那么有如下的误差估计式成立：

$$\frac{\left| \sigma_j - \sigma'_j \right|}{\sigma_j} \le (72\varepsilon * M + n^2 * \varepsilon + n * tol) * \max_{0 \le k \le M} \kappa(B_k) + n\varepsilon ;$$

从而得知 Jacobi 算法具有很良好的相对精度。

对于奇异向量也有很良好的相对精度的误差估计如下：

定理 3.4.4：

假定 $V = \left[v_1, ..., v_n \right]$，$U = \left[u_1, ..., u_n \right]$ 是由**算法 3.4.1** 计算得到的左奇异向量和右奇异向量；而 $V_T = \left[v_{T1}, ..., v_{Tn} \right]$，$U_T = \left[u_{T1}, ..., u_{Tn} \right]$ 是矩阵真实的左奇异向量和右奇异向量；并且简记 $\bar{\kappa} \equiv \max_{0 \le k \le M} \kappa(B_k)$；那么计算得到的奇异向量和真实向量之差满足如下式子：

$$\max(\left\| u_{Ti} - u_i \right\|_2, \left\| v_{Ti} - v_i \right\|_2) \le \frac{(n-0.5)^{1/2} * \bar{\kappa} * (72\varepsilon * M + n^2 * \varepsilon + n * tol)}{relgap_{\sigma_i}} + (9M + n + 1)\varepsilon$$

其中 $rel\,gap_{\sigma_i}$ 为第 i 个奇异值的相对间隔为 $rel\,gap_{\sigma_i} \equiv \min_{k \ne i} \frac{\left| \sigma_i - \sigma_k \right|}{\sigma_i + \sigma_k}$。

虽然 Jacobi 算法具有很良好的相对精度，但是它的收敛速度却很慢，所以在文献[18]中进行了很多努力来提高它的性能，下面开始将一一介绍。

第一，假定 $G \in R^{m \times n}$，当 $m \square n$ 时，我们可以先计算 G 的 QR 分解式：

$$G = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \quad \ldots\ldots (3.4.2)$$

然后再对 $n\times n$ 的上三角矩阵 R 进行 SVD 分解，这时我们所进行的 Jacobi 旋转变换就是再 n 维空间中进行的了，而不是在 m 维空间中，这样可以大大减少运算量。

第二，对 G 先进行 QR 分解的另一个更重要的优势是：达到对 G 进行预处理的作用，从而提高了收敛速度。假定 $G\in R^{m\times n}$ 为列满秩矩阵，对它进行列主元 QR 分解，可以得到如下式子：

$$GP_1 = Q_G\begin{pmatrix} R_G \\ 0 \end{pmatrix} \text{……(3.4.3)}$$

如果令 $R_G^T = U_{R_G}\Sigma V_{R_G}^T$ 为 R_G^T 的奇异值分解；其中 $V_{R_G}^T$ 为 Jacobi 旋转变换累积所得到的矩阵，$U_{R_G}\Sigma$ 为最终的极限矩阵，那么此时有：

$$G = Q_G\begin{bmatrix} V_{R_G} & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} \Sigma \\ 0 \end{bmatrix}\left(P_1U_{R_G}\right)^T \text{……(3.4.4)}$$

为 G 的 SVD；当左奇异向量不需要时，我们就没有必要累积 Jacobi 旋转变换，而右奇异向量总是很容易求得的。

假如再令：

$$R_G^TP_2 = Q_{R_G}R_{R_G} \text{……(3.4.5)}$$

为 R_G^T 带列主元的 QR 分解式，并且令 $R_{R_G}^T = U_{R_{R_G}^T}\Sigma V_{R_{R_G}^T}^T$ 为 $R_{R_G}^T$ 的 SVD，此时有

$$G = Q_G\begin{bmatrix} P_2U_{R_{R_G}^T} & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} \Sigma \\ 0 \end{bmatrix}\left(P_1Q_{R_G}V_{R_{R_G}^T}\right)^T \text{……(3.4.6)}$$

此时和(3.4.4)相对应的是仅当右奇异向量需要时，才有必要累积 Jacobi 旋转变换，而左奇异向量很容易求得的。

而如果我们先求 R_{R_G} 的 SVD 式子：$R_{R_G} = U_{R_{R_G}}\Sigma V_{R_{R_G}}^T$，此时有

$$G = Q_G\begin{bmatrix} P_2V_{R_{R_G}} & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} \Sigma \\ 0 \end{bmatrix}\left(P_1Q_{R_G}U_{R_{R_G}}\right)^T \text{……(3.4.7)}$$

R_{R_G} 的右奇异向量矩阵 $V_{R_{R_G}}$ 变成了 G 的左奇异向量矩阵中的一个因子。

类似于(3.4.3)，(3.4.5)这样的 QR 分解重复进行可以加快 Jacobi SVD 算法的收敛速度。在文献[18]中所进行的数值试验采用了 2 次这样的 QR 分解。

第三，充分利用上三角形式（或下三角形式）

对于三角形式的矩阵利用 Jacobi 算法每进行一次迭代都包含 $n(n-1)/2$ 次旋转变换，可惜的是如果没有特殊处理，经过第一次旋转以后，矩阵的上三角形式（或下三角形式）就遭到了破坏。现在我们来构造合适的策略来充分地利用这些三角结构使得旋转变换更快速。现在以下三角矩阵为例，比如取 $L = R_G^T$ 或 $L = R_{R_G}^T$ 令 $n = n_1 + n_2$，考虑矩阵 L 的如下分块结构：

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, L_{11}\in R^{n_1\times n_1}, L_{22}\in R^{n_2\times n_2}.$$

可以看出对矩阵 L 的最后 n_2 列上的 Jacobi 变换最作用在 $n_2\times n_2$ 的矩阵 L_{22} 上，由

此考虑，我们应该从小矩阵 L_{22} 开始进行 Jacobi 变换，这样相当于我们先改进了矩阵 L 的最后 n_2 列的正交性，同时避免了和 $n_1 \times n_2$ 的零矩阵上的计算。可知在 n 维空间上进行 $n_2(n_2-1)/2$ 次 Jacobi 旋转，需要的运算量为 $3nn_2^2+O(n_2^2)$ *flops*；而在 n_2 维空间上进行 $n_2(n_2-1)/2$ 次 Jacobi 旋转，需要的运算量为 $3n_2^3+O(n_2^2)$ *flops*；当取 $n_2 \approx n/2$ 时，运算量减少为一半。

接着在矩阵 L 的剩下的前 n_1 列中进行 $n_1(n_1-1)/2$ 次 Jacobi 旋转变换，最后在下标为 $(i,j), 1 \le i \le n_1, n_1+1 \le j \le n$ 的元素上进行旋转变换，从而完成全部的 $n(n-1)/2$ 次旋转变换。

这种策略也称为行循环策略(row-cyclic strategy)，事实验证它不仅节省了在 0 块矩阵上的不必要的计算，而且对算法的快速收敛也有很大的帮助。事实上[18]，在对称 Jacobi 算法中非对角元素中离对角线近的收敛到 0 的速度慢于离对角线远的元素的收敛速度。对于 SVD 算法来说，也就是在 Jacobi 迭代过程中，前 n_1 列与后 n_2 列之间的相互正交性比它们各自子空间上的正交性更快地得到满足；所以在这些子空间内应该进行更多次的旋转变换，这中技术也叫 quasi-cyclic pivoting。并且因为 L_{11} 和 L_{22} 仍然是下三角矩阵，所以可以递归地应用 quasi-cyclic pivoting 技术。

在文献[18]中还说明了其中用到的带列主元 QR 分解算法的稳定性，以及带列主元 QR 分解得到的上三角矩阵 R 所具有的 rank revealing 性质：

$$R_{ii}^2 \ge \sum_{k=i}^{j} R_{kj}^2, 1 \le i \le j \le n$$

这些都为算法良好的相对精度提供了保证。

第四，将右奇异向量置后计算

考虑一个在 $n \times n$ 的矩阵 X 的 SVD 如下：

$$X_\infty \equiv XV_X = U_X \Sigma_X, \ldots\ldots (3.4.8)$$

其中 V_X 是累积的 Jacobi 旋转矩阵，由(3.4.8)我们可以得到，

$$V_X = X^T U_X \Sigma_X^{-1}, \text{或} V_X = X^{-1} U_X \Sigma_X,$$

很容易看出由 $V_X = X^T U_X \Sigma_X^{-1}$, 很容易求出右奇异向量 V_X，但是遗憾的是由此公式得到的解具有很差的精确度[26]。而可喜的是 $V_X = X^{-1} U_X \Sigma_X$, 可以得到令人满意的精确度。一般的处理方法是先求出最终的收敛时的极限矩阵 $\tilde{X}_\infty$，然后再解方程：

$$XV_X = \tilde{X}_\infty \ldots\ldots (3.4.9)$$

在文献[18]中也给出了当 X 为上三角矩阵或下三角矩阵时由(3.4.9)解出的 $\tilde{V}_X$ 和精确的解 $V_X' = X^{-1}\tilde{X}_\infty$ 满足如下关系：

$$\left|\tilde{V}_X - V_X'\right| \le \frac{\varepsilon \cdot \left|X^{-1}\right|\left|X\right| \cdot \left|V_X'\right|}{1-\varepsilon\left|X^{-1}\right|\left|X\right|} = \frac{\varepsilon \cdot \chi_2(X)}{1-\varepsilon \cdot \chi_2(X)}\left|V_X'\right|, 0 \le \varepsilon \le \frac{nu}{1-nu}$$

而一般经过列主元 QR 迭代得到的上三角矩阵 X 都具有较小的 $\chi_2(X)$，从中可以看出由(3.4.9)解出的 $\tilde{V}_X$ 具有良好的相对精度。

由(3.4.3)式，一般矩阵的 SVD 问题可以通过 QR 分解化为一个上三角矩阵的 SVD

问题，所以任何矩阵的右奇异向量都可以后置地精确求解，并且对于方程(3.4.9)，当 X 为上三角矩阵或下三角矩阵时，都可以快速地精确求解。

在试验上，也验证经过这些预处理，一般经过 4－6 次对矩阵进行的 Jacobi 旋转变换后，算法将得到收敛[18]。从而在速度上和 QR 系列的 SVD 方法相当。

3.5：分而治之方法[20]

分而治之算法的第一步也和 QR 迭代算法一样，先将矩阵二对角化得到二对角矩阵 B；然后将求 B 的 SVD 问题分为两个子问题。

先将 B 写成如下的形式：

$$B=\begin{pmatrix} B_1 & 0 \\ \alpha_k e_k & \beta_k e_1 \\ 0 & B_2 \end{pmatrix} \quad \text{……(3.5.1)}$$

其中 $B_1 \in R^{(k-1)\times k}$， $B_2 \in R^{(n-k)\times(n-k)}$ 为上二对角矩阵， e_j 是相应维数的向量中的第 j 个单位向量，并且一般我们取 $k=\lfloor n/2 \rfloor$。

现在假如我们已经求得了 B_1， B_2 的 SVD 如下：

$$B_1 = Q_1 \begin{pmatrix} D_1 & 0 \end{pmatrix} W_1^T, \quad B_2 = Q_2 D_2 W_2^T$$

并且令 $\begin{pmatrix} l_1^T & \lambda_1 \end{pmatrix}$ 为 W_1 的最后一行， f_2^T 为 W_2 的第一行。那么将这些带入(3.5.1)，我们可以得到：

$$B=\begin{pmatrix} Q_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & Q_2 \end{pmatrix} \begin{pmatrix} D_1 & 0 & 0 \\ \alpha_k l_1^T & \alpha_k \lambda_1 & \beta_k f_2^T \\ 0 & 0 & D_2 \end{pmatrix} \begin{pmatrix} W_1 & 0 \\ 0 & W_2 \end{pmatrix}^T \quad \text{……(3.5.2)}$$

可以看出中间的矩阵形式上很简单，除了对角元素和第 k 行上的元素，其它元素都为 0；这里先把它的 SVD 的计算问题放到最后，而假定已知其 SVD 为： $S\Sigma G^T$，将它带入(3.5.2)式，就可以得到 B 的 SVD 分解式：

$$B = Q\Sigma W^T$$

其中

$$Q=\begin{pmatrix} Q_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & Q_2 \end{pmatrix} S; \quad W=\begin{pmatrix} W_1 & 0 \\ 0 & W_2 \end{pmatrix} G$$

而计算 B_1， B_2 的 SVD 时也可递归地应用这种办法，至到子问题足够得小。

现在来讨论求解(3.5.2)式中间矩阵的 SVD 问题，我们注意到通过排序，将第 k 行和第 k 列移到第 1 行第 1 列，得到如下形式的矩阵：

$$M=\begin{pmatrix} z_1 & z_2 & \cdots & z_n \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} \quad \text{……(3.5.3)}$$

其中 $d_i, i=2,...,n$ 是矩阵 D_1， D_2 的对角元素；而 $z_i, i=1,...,n$ 是中间矩阵的第 k 行元素，其中 z_1 为原(k,k)位置上的元素；我们继续对矩阵 M 进行排序，并且为了方便定义： $d_1 \equiv 0$，使得排序后 $D=diag(d_1,d_2,...,d_n), 0=d_1 \le d_2 \le ... \le d_n$；并且假定：

$$d_{j+1}-d_j \ge \tau \|M\|_2, \quad |z_j| \ge \tau \|M\|_2 \quad \text{.....(3.5.4)}$$

其中 τ 是机器精度 ε 的一个很小的倍数，具体可见文献[27]，文献[27]中还说明了如何用 deflation 方法使得 M 都具有(3.5.4)的形式。

最后给出关于求解奇异值和奇异向量的定理如下：

定理 3.5.1：

假定 $S\Sigma G^T$ 为 M 的 SVD，记个矩阵为： $S=(s_1,...,s_n)$， $\Sigma = diag(\sigma_1,...,\sigma_n)$，$G=(g_1,...,g_n)$，其中 $0<\sigma_1<...<\sigma_n$；那么此时奇异值满足如下的间隔性质：

$$0=d_1<\sigma_1<d_2<\sigma_2<...<d_n<\sigma_n<d_n+\|z\|_2$$

并且满足如下久期方程(secular equation):

$$f(\sigma)=1+\sum_{k=1}^{n}\frac{z_k^2}{d_k^2-\sigma^2}=0 \quad \text{.....(3.5.5)}$$

奇异向量可如下求解：

$$s_i=\left(-1,\frac{d_2z_2}{d_2^2-\sigma_i^2},...,\frac{d_nz_n}{d_n^2-\sigma_i^2}\right)^T \Big/ \sqrt{1+\sum_{k=2}^{n}\frac{(d_kz_k)^2}{(d_k^2-\sigma_i^2)^2}} \quad \text{.....(3.5.6)}$$

$$g_i=\left(\frac{z_1}{d_1^2-\sigma_i^2},\frac{z_2}{d_2^2-\sigma_i^2},...,\frac{z_n}{d_n^2-\sigma_i^2}\right)^T \Big/ \sqrt{\sum_{k=1}^{n}\frac{z_k^{\,2}}{(d_k^2-\sigma_i^2)^2}} \quad \text{.....(3.5.7)}$$

在文献[20]提到了用 root-finder 方法来求解(3.5.5)以得到近似的奇异值，但是却没有公开这种方法这种方法，而只是提到此方法最初来自于和 R.-C. Li 的个人交往。而假如已经计算得到奇异值，那么就可以利用(3.5.6)，(3.5.7)来计算左奇异向量和右奇异向量了。

在性能方面分而治之算法无疑是很优秀的，当 $n=400$ 时，对二对角矩阵求解 SVD 比基于传统 QR 的算法快 9－10 倍，但是至今还没有证明它具有很良好的相对精度（也许并不具有）。

四、复矩阵的处理方法

4.1：简介

和实矩阵一样，也可以对复矩阵进行奇异值分解；和**定理 1.1.1** 相对，复矩阵也存在类似的奇异值分解定理：

定理 4.1.1（复奇异值分解定理） 设 $A \in C^{m\times n}$，则必存在酉矩阵

$U=[u_1,...,u_m] \in C^{m\times m}$ 和 $V=[v_1,...,v_n] \in C^{n\times n}$

使得

$$U^H AV = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} r \\ m-r \end{matrix}, \quad \ldots(1.1)$$
$$\begin{matrix} \\ r \end{matrix} \quad \begin{matrix} \\ n-r \end{matrix}$$

其中 $\Sigma_r = diag(\sigma_1,...,\sigma_r), \sigma_1 \geq ... \geq \sigma_r > 0$ [2]。

4.2：复矩阵 SVD 算法

处理复矩阵 SVD 问题的比较简单的方法是：先把矩阵二对角化，然后对二对角矩阵应用前面所列举的各种基于二对角矩阵的算法（如 QR 迭代算法），最终使之收敛到对角矩阵；从而完成奇异值分解。而很幸运的是，在二对角化过程中，可以利用报告《QR 分解算法及其评估》中的**算法 3.2.2** 来进行 Householder 变换；最终把原来的复矩阵化为一个实的二对角矩阵，而避免了在下一步的计算中出现复数运算（当然需要求奇异向量时，对奇异向量的更新过程中还是会遇到大量的复数和实数相乘的运算的）。

为了方便，以下将《QR 分解算法及其评估》中的**算法 3.2.2** 简称 **QR 算法 3.2.2**；为了理解方便，将其功能复述如下：

已知 $\begin{pmatrix} \xi_1 \\ \vdots \\ \xi_n \end{pmatrix}$ 为一列复向量，**QR 算法 3.2.2** 可以求得复数 σ，复向量 v，和实数 α；使得

$$\left(I - \sigma v v^H\right) \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_n \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \end{pmatrix}$$

记 $U^H = \left(I - \sigma v v^H\right)$ 为酉矩阵，并且一般 U^H 总是以因子形式 σ 和 v 保存。

而现在假如已知一行向量 $(\xi_1,...,\xi_n)$，求一酉矩阵 V，使得：

$$(\xi_1,...,\xi_n)V = (\alpha,0,...,0)$$

可如下进行，由 $(\xi_1,...,\xi_n)V = (\alpha,0,...,0)$ 可知：

$$V^H \begin{pmatrix} \overline{\xi_1} \\ \vdots \\ \overline{\xi_n} \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \end{pmatrix}$$

所以，只要把 $\begin{pmatrix} \overline{\xi_1} \\ \vdots \\ \overline{\xi_n} \end{pmatrix}$ 当作 **QR 算法 3.2.2** 的输入向量，而直接调用最终可以得到以因子形式 σ 和 v 表示的酉矩阵 V^H。

二对角化可以仿造实矩阵时的过程如下进行，关键的区别是要注意这里的 U_1^H，V_1^H 不再具有 Hermite 性（实数时是具有对称性的），并且仍然假设 $m \geq n$：

首先将 A 分块为

$$A=\begin{bmatrix} a_1 & A_1 \\ 1 & n-1 \end{bmatrix}$$

先计算 m 阶复 Householder 变换矩阵 $U_1^H=\left(I-\sigma u_1 u_1^H\right)$ 使得

$$U_1^H a_1=\delta_1 e_1(\delta_1\in R, e_1\in R^m)$$

并且形成：

$$U_1^H A_1=\begin{bmatrix} b_1^H \\ \tilde{A}_1 \end{bmatrix}\begin{matrix} 1 \\ m-1 \end{matrix}$$

取出行向量 b_1^H，（先化成它的转置共轭的列向量 b_1），然后计算 n－1 阶复 Householder 变换矩阵 $V_1^H=\left(I-\rho v_1 v_1^H\right)$ 使得

$$V_1^H b_1=\gamma_2 e_1(\gamma_2\in R, e_1\in R^{n-1})$$

并形成：

$$A_1 V_1=\begin{bmatrix} a_2 & A_2 \\ 1 & n-2 \end{bmatrix}$$

然后对 $k=2,3,...,n-1$（和实数情况不一样的是这里 k 要取到 n－1；这样做的必要性见稍后分析）依次进行：

(a) 计算 m-k+1 阶 Householder 变换 $\tilde{U}_k^H=\left(I-\sigma\tilde{u}_k\tilde{u}_k^H\right)$ 使得

$$\tilde{U}_k^H a_k=\delta_k e_1(\delta_k\in R, e_1\in R^{m-k+1})$$

并且形成：$\tilde{U}_k^H A_k=\begin{bmatrix} b_k^H \\ \tilde{A}_k \end{bmatrix}\begin{matrix} 1 \\ m-1 \end{matrix}$

(b) 计算 n－k 阶 Householder 变换 $\tilde{V}_k^H=\left(I-\rho\tilde{v}_k\tilde{v}_k^H\right)$ 使得

$$\tilde{V}_k^H b_k=\gamma_{k+1} e_1(\gamma_{k+1}\in R, e_1\in R^{n-k})$$

并形成：

$$\tilde{A}_k\tilde{V}_k=\begin{bmatrix} a_{k+1} & A_{k+1} \\ 1 & n-k-1 \end{bmatrix}$$

接着对 k 要取到 $n-1$，这一与实数不同的地方进行说明：可以知道当进行完 $k=2,3,...,n-2$ 步以后，得到的矩阵为：

$$A=\begin{bmatrix} \delta_1 & \gamma_2 & 0 & \cdots & 0 \\ 0 & \delta_2 & \gamma_3 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & a_{n,n-1} & a_{nn} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & a_{m,n-1} & a_{m,n} \end{bmatrix}$$

如果是在实数域，可知右上角已经化为满足二对角矩阵的形式了，所以只需要左乘

上两个 Householder 矩阵，分别将 $a_{n-1}=\begin{bmatrix}a_{n-1,n-1}\\ \vdots \\ a_{m,n-1}\end{bmatrix}, a_n=\begin{bmatrix}a_{nn}\\ \vdots \\ a_{m,n}\end{bmatrix}$ 化为 $\begin{bmatrix}\delta_{n-1}\\ 0\\ \vdots\end{bmatrix}, \begin{bmatrix}\delta_n\\ 0\\ \vdots\end{bmatrix}$ 就可以了，但是在复数域里，我们希望最后得到的是一个实的二对角矩阵，所以当我们对 $\begin{bmatrix}a_{n-1,n-1}\\ \vdots \\ a_{m,n-1}\end{bmatrix}$ 进行一次 Householder 变换将它化为 $\begin{bmatrix}\delta_{n-1}\\ 0\\ \vdots\end{bmatrix}$，还应该右乘上一个一阶的 Householder 矩阵（其实就是单个元素，并且这个元素的值等于 $e^{-i\theta}$，其中 $a_{n-1,n}=re^{-i\theta};r,\theta\in R$ ）把 $a_{n-1,n}$ 这个元素化为一个实数（并且易知这个实数就是 r ）；而很显然这两个就相对于上面的取 k 到 $n-1$ 时完成的工作。

最后，当进行到 $k=n-1$ 之后，再计算 $m-n+1$ 阶 Householder 变换矩阵 $\tilde{U}_n^H=\left(I-\sigma\tilde{u}_n\tilde{u}_n^H\right)$ 使得

$$\tilde{U}_n^H a_n=\delta_n e_1(\delta_n\in R, e_1\in R^{m-n+1})$$

现令

$$U_k=diag\left(I_{k-1},\tilde{U}_k\right),k=2,...n$$

$$V_k=diag\left(I_k,\tilde{V}_k\right),k=2,...n-1$$

$$U=U_1U_2...U_n,\quad V=V_1V_2...V_{n-1}$$

$$B=\begin{bmatrix}\delta_1 & \gamma_2 & 0 & 0 & 0\\ 0 & \delta_2 & \gamma_3 & 0 & 0\\ 0 & 0 & \ddots & \ddots & 0\\ 0 & 0 & 0 & \ddots & \gamma_n\\ 0 & 0 & 0 & 0 & \delta_n\end{bmatrix}$$

则有：

$$U^HAV=\begin{bmatrix}B\\ 0\end{bmatrix}_{m-n}^{n}$$

即实现 A 的二对角化。

综上所述，并且参考文献[2]的形式，可以得到如下的二对角化算法：

算法 4.2.1：

U＝I；V＝I；

for $j=1:n$

利用 **QR 算法 3.2.2** 求出与向量 $A(j:m,j)$ 相对应的 u_j,σ_j,δ_j

//σ_j,u_j,δ_j 分别对应返回参数中的 σ， v， α

//使得 $U_j^HA(j:m,j)=\left(\delta_j,0,...,0\right)^T$，其中 $U_j^H=\left(I-\sigma_ju_ju_j^H\right)$

更新： $A(j:m,j+1:n)=A(j:m,j+1:n)-(\sigma_ju_j)(u_j^HA(j:m,j+1:n))$

更新 $A(j,j)=\delta_j$；

$A(j+1:m,j)=0$

//因为本程序已经把 u_j 累积到 U 中，并且以后不会再用到 u_j，

//所以不必把 $u_j(2:m-j+1)$ 保存到 $A(j+1:m,j)$ 中，

//甚至 $A(j+1:m,j)=0$ 这一步也可以不做，知道它为 0 就可以了

//而当没有累积的话，除了要保存 $u_j(2:m-j+1)$

//还需要单独存储 $u_j(1),\sigma_j$

$U(1:m,j:m)=U(1:m,j:m)-(\bar{\sigma}_j\left(U(1:m,j:m)u_j\right))(u_j^H)$

//完成对 U 的更新 $U=U_1U_2...U_n$

if $j<n$

利用 **QR 算法 3.2.2** 求出与向量 $A(j,j+1:n)^H$ 相对应的 v_j,ρ_j,γ_{j+1}

// ρ_j,v_j,γ_{j+1} 分别对应返回参数中的 σ， v， α

//使得 $V_j^HA(j,j+1:n)^H=\left(\gamma_{j+1},0,...,0\right)^T$，其中 $V_j^H=\left(I-\rho_jv_jv_j^H\right)$

//这样就有 $A(j,j+1:n)V_j=\left(\gamma_{j+1},0,...,0\right)$

更新：

$A(j+1:m,j+1:n)=A(j+1:m,j+1:n)-(\bar{\rho}_jA(j+1:m,j+1:n)v_j)(v_j^H)$

更新 $A(j,j+1)=\gamma_{j+1}$；

$A(j,j+2:n)=0$；

//因为本程序已经把 v_j 累积到 V 中，并且以后不会再用到 v_j，

//所以不必把 $v_j(2:n-j)$ 保存到 $A(j,j+2:n)$ 中，

//甚至 $A(j,j+2:n)=0$ 这一步也可以不做，知道它为 0 就可以了

//当然如果没有累积的话，除了要保存 $v_j(2:n-j)$

//还需要单独存储 $v_j(1),\rho_j$

$V(1:n,j:n)=V(1:n,j:n)-(\bar{\rho}_j\left(V(1:n,j:n)v_j\right))(v_j^H)$

//完成对 V 的更新 $V=V_1V_2...V_{n-1}$

endif

end

当得到实的二对角矩阵后，如果不需要求奇异向量，那么可说所有的工作都是一样的了，对实的二对角矩阵，直接应用实矩阵情况下的算法就可以得到它的奇异值了，而此实二对角矩阵的奇异值就是原来复矩阵的奇异值。

当需要求奇异向量，即 U，V 时；还有一些变动和需要讨论的地方，以下将简单地展开分析。

首先，假设由上面所述的二对角化过程得到的矩阵为 B，并且由实矩阵情况下的算法计算得到 B 的 SVD 分解式为：$B=P\Sigma Q^T$；而有前面二对角化的过程可知：

$$U^H AV = \begin{bmatrix} B \\ 0 \end{bmatrix}_{m-n}^{n}$$，所以 $A = U\begin{bmatrix} P\Sigma Q^T \\ 0 \end{bmatrix} V^H = U\begin{bmatrix} P & 0 \\ 0 & I \end{bmatrix}_{m-n}^{n} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} Q^T V^H$

令：

$$S = U\begin{bmatrix} P & 0 \\ 0 & I \end{bmatrix} = \underset{n}{(U_1}, \underset{m-n}{U_2})\begin{bmatrix} P & 0 \\ 0 & I \end{bmatrix} = (U_1 P, U_2)；\ T = \left(Q^T V^H\right)^H = VQ$$

那么就由：

$$A = S\begin{bmatrix} \Sigma \\ 0 \end{bmatrix} T^H$$

为 A 的 SVD 分解。

而计算 S 和 T 有两种方法：

方法一：就是先求出 P，Q；然后把他们右乘到 U，V 上就得到 S，T 了，这里只是简单的矩阵相乘，就不加说明了。

方法二：由实矩阵的情况可以得知，P 和 Q 是由一系列 Givens 旋转矩阵相乘累积得到的，即 $P_i = P_{i-1} G_i\left(k, k+1, \theta\right), i = 1, 2, ...;$ 其中 $P_0 = I;\ G\left(k, k+1, \theta\right)$ 为 $\left(k, k+1\right)$ 平面的 Givens 旋转矩阵；类似的 $Q_i = Q_{i-1} G_i\left(k, k+1, \theta\right), i = 1, 2, ...; Q_0 = I$。所以我们也可以直接把 $G_i\left(k, k+1, \theta\right)$ 右乘累积到 U 和 V，而这样每次只改变 U，V 的两列。从而避免了两个矩阵的直接相乘。具体的操作为：将原来对 P 元素进行的操作全部转化为对 U 的元素进行的操作，原来对 Q 元素进行的操作全部转化为对 V 元素进行的操作，只是原来是实数和实数相乘，现在为一个复数和一个实数（用复数的实部乘以实数作为积的实部，复数的虚部乘以实数作为积的虚部）。

至此，可说完成了求解复数 SVD 的算法。

当然这两种方法的计算量哪一者大，还是值得讨论的，这里简单地分析如下：（注我们只考虑乘除运算量）。

先看方法一：$S = (U_1 P, U_2)$；求 $U_1 P$ 相当于求一个 $m \times n$ 的复矩阵和一个 $n \times n$ 的实矩阵相乘；而易知一个 $m \times n$ 的实矩阵和一个 $n \times n$ 的实矩阵相乘计算量为 mn^2；而一个复矩阵和一个实矩阵相乘，其计算量增倍，所以为 $2mn^2$；而现在假设 P 由 l 个 Givens 矩阵累积得到，每一次累积改变两列，：

$$\left(p_{ik}; p_{i,k+1}\right) = \left(p_{ik}; p_{i,k+1}\right) G\left(k, k+1, \theta\right) = \left(p_{ik}; p_{i,k+1}\right)\begin{bmatrix} c & s \\ -s & c \end{bmatrix}; i = 1, .., n$$

可知这个累积过程的计算量为 $4n$；所以累积 P 用的计算量为 $4nl$；所以求 S 总的运算量为：$2mn^2 + 4nl$；再看 $T = VQ$，可知 V 和 Q 相乘的运算量为 $2n^3$，并且易知 Q 和 P 是由数量相对的 Givens 矩阵累积得到的，所以累积 Q 的过程花费也同样为 $4nl$，总运算量为 $2n^3 + 4nl$，计算 S 和 T 的总过程的运算量为 $2n^2\left(n+m\right) + 8nl$。

接着再看方法二，对 U 的一次累积过程可如下进行：

$$\left(u_{ik}; u_{i,k+1}\right) = \left(u_{ik}; u_{i,k+1}\right) G\left(k, k+1, \theta\right) = \left(u_{ik}; u_{i,k+1}\right)\begin{bmatrix} c & s \\ -s & c \end{bmatrix}; i = 1, .., m$$

因为 U 为复矩阵，所以对每一个 i，运算量为 8 次，一次累积运算量为 $8m$，而总共有 l 次累积，所以运算量为 $8ml$；对 V 的累积可以类似地分析，得到对 V 的累积运

算量为$8nl$；所以总的运算量为$8(m+n)l$。

剩下的任务就是估计l了，首先看$2n^2(n+m)+8nl$和$8(m+n)l$相等时的l的值，可得$l=\frac{n^2(n+m)}{4m}$，且当l大于这个值时方法二的运算量就会大于方法一；因为对一个完整的$n\times n$矩阵应用一次 QR 迭代，左右各需要乘上n个 Givens 矩阵，现在假设每迭代一次产生一个奇异值（这是一个很理想的值了），接着在剩下的$(n-i)\times(n-i)$；$i=0,1,...n-2$矩阵进行 QR 迭代，那么可以得知总共需要$\frac{(n-1)\times(n+2)}{2}\approx\frac{n^2}{2}$；而当$m\geq n$时，很容易证明$\frac{n^2}{2}\geq\frac{n^2(n+m)}{4m}$；所以由此可知方法二的运算量大于或等于方法一的运算量，当$m=n$时取等号。当然这建立在一个很理想的假设上：每迭代一次产生一个奇异值；而实际往往要迭代很多次才产生一个奇异值，当然也存在特殊情况迭代一次就产生很多个奇异值，可是这个几率很小的，所以综合来说方法一的运算量比方法二要小的多。

从文献[30]中，可以看出 LAPACK 采用的是方法一；而从在文献[30]，可知当原矩阵为实矩阵时，它采用的也是方法一，在此作为补充，对实数时的情况也作一简单的分析，易知此时的运算量只需把原来设计到复数运算部分的运算量减半就可以得到了，分别为：$n^2(n+m)+8nl$和$4(m+n)l$；从而要使方法一比方法二优秀，必须满足$n^2(n+m)+8nl<4(m+n)l$，可得$l>\frac{n^2(n+m)}{4(m-n)}$；一般$l$是一个和$n^2$同复杂度的数，并且在文献[23]中，有算法总复杂度约为$20n^3$，从中反推，可以得知$l\approx2.5n^2$，这时要满足$l>\frac{n^2(n+m)}{4(m-n)}$，可知维数 m,n 要满足如下的式子：$9m>11n$；此时方法一才比方法二优秀，而反之方法二比方法一优秀。

五、小结

从上可以看出求解 SVD 有很多的方法，并且综合来说没有那一种方法具有绝对的优势，这大概也是近年来在求解 SVD 的各种方法都有人进行研究的原因吧！而总体来说，它们所具有的特点如下：

传统 QR 迭代算法：比较成熟，但是在性能和精确度上都不够理想，现在已经基本上被融入到传统 QR 迭代和零位移 QR 迭代的混合算法之中了。

传统 QR 迭代和零位移 QR 迭代的混合算法：从传统 QR 迭代算法发展而来，从中继承了很多成熟的技巧，所以也相对的完善，并且在具有较高的效率和良好的精确度，所以是一种比较适中的方法。

qd 算法：方法很新颖，发展的较晚；现在在理论已经比较完善，并且也有很多相关的数值试验；但是还没有发现一个比较正式的采用它的软件包，并且研究工作大多也只在德国开展，甚至连较完整的基本算法也没有给出，但是按文献[15]中所说它具有比传统 QR 迭代和零位移 QR 迭代的混合算法更高的效率，精确度也较好。

Jacobi 方法：这是唯一一种不需要先二对角化的方法，这保证了它的精确度是所有算法中最好的，但是它的效率不高（主要原因是收敛速度慢，并且每一步迭代的计算量大），在文献[18]中对它进行了各种改进，使得它的效率可以和其它方法相当，从而使得它也成为一种值得选用的方法，特别是要求很高精度的解时。

分而制之算法：这是一种快速算法，但相对精确度不一定可靠，并且其中的好多具体细节都因为技术没公开而缺少文献资料，如果要采用此方法则还要作更进一步的分析和研究。

六、参考文献

[1] 徐树方，矩阵计算的理论与方法，北京大学出版社，1994.

[2] G. H. Golub and C. F. von Loan (1996). Matrix Computation. The Johns Hopkins University Press.

[3] J. Dongarra, C. Moler, J. Bunch, G. W. Stewart, LINPACK User's Guide, SIAM, Philadelphia. 1979.

[4] Dan Kalman, A Singularly Valuable Decomposition: The SVD of a Matrix, The College Mahtematics Journal. Vol.27 No.1 Jan 1998, 2-23.

[5] C. L. Lawson and R. J. Hanson, Solving Least Squares Problems, Englewood Cliffs. New Jersey: Prentice-Hall, 1974.

[6] G. W. Stewart and Ji-guang Sun, Matrix Perturbation Theory, Harcourt Brace Jovanovich, 1990.

[7] David Richards and Adam Abrahamsen, Image Compression using Singular Value Decomposition, Applications of Linear Algebra, http://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall2001/AdamDave/slideshow.pdf.

[8] G. W. Stewart, On the early history of the singular value decomposition, SIAM Rev., 35:551-566,1993.

[9] G. H. Golub and W. Kahan, Caculating the singular values and pseudo-inverse of a matrix, SIAM J. Numer. Anal., 2:205-224, 1965

[10] P. A. Businger and G. H. Golub, Algorithm 358: Singular value decomposition of a complex matrix, Assoc. Comp. Math, 12:564-565, 1969.

[11] Gene H. Golub, Henk A. van der Vorst, Numerical Progress in Eigenvalue Computation in the 20th Century Working Document (1999), http://citeseer.ist.psu.edu/cache/papers/cs/10239/http:zSzzSzwww.math.ruu.nlzSzpeoplezSzvorstzSzeighistory.pdf/golub99numerical.pdf

[12] Nicholas J. Higham, Recent Developments in Dense Numerical Linear Algebra (2000), http://citeseer.ist.psu.edu/higham00recent.html

[13] James W. Demmel and W. Kahan, Accurate singular values of bidiagoanl matrices, SIAM J. Sci. Stat. Comput, 11(5):873-912, 1990.

[14] Percy Deift, James W. Demmel, Luen-Chau Li, and Carlos Tomei, The bidiagonal singular value decomposition and Hamiltonian mechanics, SIAM J. Numer. Anal, 28:1463-1516, 1991.

[15] K. Vince Fernando and Beresford N. Parlett, Accurate singular values and differential qd algorithms, Numer. Math, 67:191-229, 1994.

[16] Beresford N. Parlett, The new qd algorithms, In Acta Numerica, Cambridge University Press, 1995, pages 459-491.

[17] Demmel James and Veselic Kresimir, Jacobi's method is more accurate than QR, SIAM J. Matrix Anal. Appl. 13 (1992), no. 4, 1204--1245.

[18] Z.DRMAC, A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm, IMA J. Numer. Anal., 19(1999), pp191-213.

[19] Ming Gu and Stanley C. Eisenstat, A divide-and-conquer algorithm for the bidiagonal SVD, SIAM Journal on Matrix Analysis and Applications Volume 16, Number 1 pp. 79-92, 1995.

[20] I. Dhillon, J. Demmel, M. Gu, Efficient Computation of the Singular Value Decomposition with Applications to Least Squares Problems, LBL Report #36201, 1994, TECHNICAL REPORT.

[21] J. Barlow and J. Demmel, Computing accurate eigensystems of scaled diagonally dominant matrices, SIAM J. Numer. Anal., 27(1990), pp762-791

[22] K. V. Fernando and B. N. Parlett, Accurately counting singular values of bidiagonal matrices and eigenvalues of skew-symmetric tridiagonal matrices, SIAM J. Matrix Anal. Appl., 20(1998),pp373-399

[23] Jesse L. Barlow, More Accurate Bidiagonal Reduction for Computing the Singular Value Decomposition, SIAM Journal on Matrix Analysis and Applications, Volume 23, Number 3 pp. 761-798, 2002.

[24] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. Numerical Recipes in C : The Art of Scientific Computing. Cambridge University Press.

[25] J. H. Wilkinson and C. Reinsch, Linear Algebra: vol II of Handbook for Automatic Computation, New York: Springer-Verlag, 1971.

[26] Jessup. E. R and Sorensen. D. C, A parallel algorithm for computing the singular value decomposition of a matrix, SIAM J. Matrix Anal. Appl, 15,530-548,1994

[27] M. Gu and S. C. Eisenstat, A divide-and-conquer algotithm for the bidiagonal SVD, SIAM J. Matrix Anal. Appl., 1995

[28] RB Lehoucq. The computation of elementary unitary matrices. ACM Transactions on Mathematical Software, Volume 22,Issue 4(December 1996) P: 393 - 400, 1996

[29] RB Lehoucq. The computation of elementary unitary matrices. Draft,. Department of Computer Science, Rice University, July 1994.

[30] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen, Lapack Users' Guide, 3rd edn. SIAM Press, Philadelphia PA, 1999.