矩阵分解及其应用

汕头大学数学系　04061012　邓宇善

摘　要

本文概述了三个重要的矩阵分解方法 —**LU** 分解，**QR** 分解和奇异值分解（SVD）的性质及相关定理，给出其计算方法及算法的 C 语言程序实现，并讨论矩阵分解在求解线性方程组 $\mathbf{Ax} = \mathbf{b}$ 和线性拟合问题 $\mathbf{Ax} \approx \mathbf{b}$ 的最小二乘解或极小范数解中的应用。从一些方面对比三个矩阵分解方法，得知要有一个解决问题的“完美算法”是困难的。此外，本文还给出矩阵分解的一些其他应用，目的在于说明矩阵分解在理论及工程中的重要性。

关键词: **LU** 分解；Cholesky 分解；**QR** 分解；奇异值分解；最小二乘问题

Matrix Decompositions and Their Applications

ABSTRACT

We summarize the properties and related theorems of three important matrix decompositions, which are $\mathbf{LU}$ *decomposition*, $\mathbf{QR}$ *decomposition* and *singular value decomposition*(SVD). We present their computational methods and their implementations in C programming language. Matrix decompositions' applications in solving systems of linear equations $\mathbf{Ax} = \mathbf{b}$ and finding the least squares solution or minimum norm solution of linear fitting problem $\mathbf{Ax} \approx \mathbf{b}$ will also be discussed here. By comparing three decomposition methods in some aspects, we know that there is hardly a "perfect algorithms" to solve the problem. In order to show the importance of matrix decompositions in science and engineering, we present some other applications of matrix decompositions.

Keywords: $\mathbf{LU}$ Decomposition; Cholesky Decomposition; $\mathbf{QR}$ Decomposition; Singular Value Decomposition; Least Squares Problem

目 录

1. 引言

在数值线性代数中，有两个很重要的问题：线性方程组的求解（包括矩阵求逆），和矩阵特征系统的计算。研究这两个课题的文献很多，但不难发现，矩阵分解在当中是一个十分重要的方法。

1.1 记号约定

本文使用的记号基本与 [3] 一致。矩阵用加粗大写字母（$\mathbf{A}$，$\mathbf{B}$）表示，向量用加粗小写字母（$\mathbf{a}, \mathbf{b}$）表示；常量用斜体大写字母（M，N）表示；其余变量用斜体小写字母表示。序数词从 0 开始，这点与一般的习惯有所不同，但与 C 语言是一致的，我们在正文中会经常提醒读者。

$\mathbb{C}$ 表示复数域，$\mathbb{R}$ 表示实数域，$\mathbb{R}^{M\times N}$ 表示 M 行 N 列矩阵组成的线性空间：

$$\mathbf{A}\in\mathbb{R}^{M\times N}\Leftrightarrow\mathbf{A}=[a_{ij}]=\begin{pmatrix}a_{00}&a_{01}&\cdots&a_{0,N-1}\\a_{10}&a_{11}&\cdots&a_{1,N-1}\\\vdots&\vdots&\ddots&\vdots\\a_{M-1,0}&a_{M-1,1}&\cdots&a_{M-1,N-1}\end{pmatrix}\qquad a_{ij}\in\mathbb{R}$$

若 $\mathbf{A}$ 为一矩阵，则 a_{ij} 表示其第 i 行第 j 列的元素。矩阵 $\mathbf{A}$ 的转置用 $\mathbf{A}^T$ 表示，其行列式为 $\det(\mathbf{A})$，$\mathrm{null}(\mathbf{A})$ 为 $\mathbf{A}$ 的零空间，$\mathrm{range}(\mathbf{A})$ 为 $\mathbf{A}$ 的值域，其余的矩阵运算和性质与通常一致。

向量 $\mathbf{x}$ 的 2– 范数为

$$\|\mathbf{x}\|_2=\left(\sum_{i=0}^{N-1}|x_i|^2\right)^{1/2}$$

矩阵 $\mathbf{A}$ 的 2– 范数为

$$\|\mathbf{A}\|_2=\max_{\|\mathbf{x}\|_2=1}\|\mathbf{A}\mathbf{x}\|_2$$

矩阵 $\mathbf{A}$ 的无穷范数为

$$\|\mathbf{A}\|_\infty=\max_{0\le i\le N-1}\sum_{j=0}^{N-1}|a_{ij}|$$

矩阵 $\mathbf{A}$ 的 Frobenius 范数为

$$\|\mathbf{A}\|_F=\left(\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|a_{ij}|^2\right)^{1/2}$$

本文将不加证明地使用上述范数的一些基本性质，其详细证明可参见 [4]。

本文的程序使用的是 C 语言，在 GCC4.1.2 下编译调试。为了使程序简洁易读，我们把一些常用的操作封装到文件“nr.h”中，其意思基本可由其函数名字看出，详情可参看附录 A；另外，程序的排版使用 Literate Programming 的风格，如赋值号“=”变成“←”，等于号“==”变成“≡”等，使程序更符合一般的阅读习惯。对于计算工作量的分析，本文给出的是渐近意义下所作的运算次数，用大 O 表示法表示。

1.2 求解线性方程组

所谓一般线性方程组是指形式为

$$\begin{aligned}
a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0,N-1}x_{N-1} &= b_0 \\
a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1,N-1}x_{N-1} &= b_1 \\
a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2,N-1}x_{N-1} &= b_2 \\
\cdots \qquad\qquad & \cdots \\
a_{M-1,0}x_0 + a_{M-1,1}x_1 + \cdots + a_{M-1,N-1}x_{N-1} &= b_{M-1}
\end{aligned} \tag{1}$$

的方程组，其中 x_j（$j = 0, 1, \ldots, N-1$）为未知数，M 是方程的个数，系数 a_{ij}（$i = 0, 1, \ldots, M-1$，$j = 0, 1, \ldots, N-1$）和右边项 b_j（$j = 0, 1, \ldots, N-1$）均为已知常数。在本文中，我们只讨论实数域上的线性方程组，故上述的 a_{ij} 与 b_j 都是实数。

方程组（1）可以写成矩阵形式

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2}$$

其中 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 是系数矩阵，$\mathbf{x} \in \mathbb{R}^N$ 和 $\mathbf{b} \in \mathbb{R}^M$ 是列向量

$$\mathbf{A} = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,N-1} \\ a_{10} & a_{11} & \cdots & a_{1,N-1} \\ & \cdots & & \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_{N-1} \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ \cdots \\ b_{M-1} \end{pmatrix}$$

对线性方程组（2），本文将讨论以下三种情况：

若 $\mathbf{M} = \mathbf{N}$，且 $\mathbf{A}$ 非奇异，则方程组有唯一解。§2.4 将讨论直接三角分解的求解方法，并给出 $\mathbf{A}$ 分别为严格对角占优的三对角阵、对称正定阵时的特殊三角分解方法。

若 $\mathbf{M} > \mathbf{N}$，方程组（2）称为是超定线性方程组，一般而言，超定线性方程组可能是无解的，但在科学与工程中经常要求出此类方程组的最近似解，即求解

$$\mathbf{A}\mathbf{x} \approx \mathbf{b} \quad \text{且} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\| = \min_{\mathbf{y}} \|\mathbf{A}\mathbf{y} - \mathbf{b}\| \tag{3}$$

在以 2– 范数作为近似程度的度量尺度时，该问题称为最小二乘问题，矩阵 $\mathbf{QR}$ 分解和奇异值分解均可用于求解最小二乘问题，我们将会在 §3.6 和 §4.4 中讨论这个问题。

若 $\mathbf{M} < \mathbf{N}$，或 $\mathbf{M} = \mathbf{N}$ 但 $\mathbf{A}$ 奇异，方程组（2）是退化的，或称为是欠定的，由线性代数可知，欠定方程组要么无解，要么有无穷多个解。对于后者，它的所有解构成一个集合，该集合由其一个特解 $\mathbf{x}_p$ 和 $\mathbf{A}$ 的零空间 null($\mathbf{A}$) 构成。矩阵的奇异值分解可以求出欠定方程组的一个特解及 null($\mathbf{A}$) 的一组基；或者，我们也可将该问题划分到最小二乘问题中，我们希望寻找一个有代表性的最近似解，使得该解的长度最短，称为极小范数解，矩阵的 $\mathbf{QR}$ 分解及奇异值分解都可求出该解，我们将会在 §3.6 和 §4.4 中讨论。

由此可见，矩阵分解的技巧，可用于求解方程组（2）的任何一种情形，其重要性可见一斑。

1.3 矩阵特征系统的计算

对于 $\mathbf{A} \in \mathbb{R}^{N\times N}$（这里我们也只在实数域上讨论），其特征值问题是：求 $\lambda \in \mathbb{C}$ 及非零向量 $\mathbf{x}$，使

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{4}$$

显然要使（4）成立，λ 必须满足：

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{5}$$

$p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$ 称为 $\mathbf{A}$ 的特征多项式。一般而言，我们不能通过有限次运算准确求解高次方程（5）的根，故矩阵特征系统的计算方法只能是迭代法。

矩阵特征值问题的计算并不是本文讨论的重点，只将其当作矩阵分解的应用来讨论，不对算法作详尽的分析（其详细讨论可参见 [1][5] 等）。因为矩阵分解在这些算法中都是很基础的组成部分，且算法又为迭代法，所以详细讨论这些矩阵分解是十分必要的。

反之，应用特征值问题的计算，我们将得到更高级矩阵分解 — 奇异值分解。在对矩阵 $\mathbf{A}$ 进行奇异值分解时，需要计算出 $\mathbf{A}^T\mathbf{A}$ 的特征值及对应的特征向量，奇异值分解的复杂性取决于特征值问题的计算，可见，矩阵分解与特征值问题有着紧密的联系。

2. 矩阵的 LU 分解及其应用

对方程组（2），设 $M = N$，即 $\mathbf{A} \in \mathbb{R}^{N\times N}$，$\mathbf{b} \in \mathbb{R}^N$，且 $\mathbf{A}$ 非奇异，本节讨论满足上述条件的线性方程组的直接三角分解方法，并讨论一些特殊情况下的特殊方法。

2.1 Gauss 消去法与 LU 分解

Gauss 消去法是求解线性方程组最简单有效的直接法，首先（遵循我们的约定，从第 0 步开始），令 $\mathbf{A}^{(0)} = [a_{ij}^{(0)}] = \mathbf{A}$，设 $a_{00}^{(0)} \neq 0$，令 $l_{i0} = a_{i0}^{(0)}/a_{00}^{(0)}$，$i = 1, 2, \cdots, N-1$，对每个 i，将 $\mathbf{A}$ 的第 i 行减去第 0 行的 l_{i0} 倍，这样，$\mathbf{A}^{(0)}$ 变换成 $\mathbf{A}^{(1)} = [a_{ij}^{(1)}]$，$\mathbf{A}^{(1)}$ 的第 0 列对角元以下均为 0。

一般地，设经过第 k 步（$0 \leq k \leq N-2$）后，$\mathbf{A}$ 变成 $\mathbf{A}^{(k)} = [a_{ij}^{(k)}]$，在第 $k+1$ 步中，假设 $a_{k,k}^{(k)} \neq 0$，令 $l_{ik} = a_{ik}^{(k)}/a_{kk}^{(k)}$，$i = k+1, \cdots, N-1$，利用类似上述的初等行变换将 $\mathbf{A}^{(k)}$ 的第 k 列的对角元以下的元素变成 0。这相当于 $\mathbf{A}^{(k)}$ 左乘矩阵

$$\mathbf{L}_{k+1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{N-1,k} & & 1 \end{pmatrix} \tag{6}$$

如果依次有 $a_{kk}^{(k)} \neq 0$，$k = 0, \cdots, N-2$，则可进行 $N-1$ 步，得到与（2）等价的方程组

$$\mathbf{A}^{(N-1)}\mathbf{x} = \mathbf{b}^{(N-1)} \tag{7}$$

其中 $\mathbf{A}^{(N-1)}$ 是一个上三角阵，

$$\mathbf{A}^{(N-1)}=\begin{pmatrix} a_{00}^{(0)} & a_{01}^{(0)} & \cdots & a_{0,N-1}^{(0)} \\ & a_{11}^{(1)} & \cdots & a_{1,N-1}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{N-1,N-1}^{(N-1)} \end{pmatrix} \tag{8}$$

右边项 $\mathbf{b}$ 经相应的变换后成为 $\mathbf{b}^{(N-1)}$，这样就完成了消去过程。因为 $\mathbf{A}$ 非奇异，所以 $a_{N-1,N-1}^{(N-1)}\neq 0$。下一步解（7），因 $\mathbf{A}^{(N-1)}$ 是上三角阵，故只要逐次向后代入

$$\begin{aligned} x_{N-1}&=\frac{b_{N-1}^{(N-1)}}{a_{N-1,N-1}^{(N-1)}} \\ x_i&=\frac{1}{a_{ii}^{(i)}}\left(b_i^{(i)}-\sum_{j=i+1}^{N-1}a_{ij}^{(i)}x_j\right) \qquad i=N-2,N-3,\cdots,0 \end{aligned} \tag{9}$$

以上由消去过程和回代过程（9）求出（2）的解的过程就是 Gauss 消去法。不妨用（6）将 $\mathbf{A}$ 的消去过程写成矩阵乘积的形式：

$$\mathbf{L}_{N-1}\mathbf{L}_{N-2}\cdots\mathbf{L}_1\mathbf{A}^{(0)}=\mathbf{A}^{(N-1)}.$$

记 $\mathbf{L}=(\mathbf{L}_{N-1}\cdots\mathbf{L}_1)^{-1}$，易知

$$\mathbf{L}=\begin{pmatrix} 1 & & & & \\ l_{10} & 1 & & & \\ l_{20} & l_{21} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{N-1,0} & l_{N-1,1} & \cdots & l_{N-1,N-2} & 1 \end{pmatrix} \tag{10}$$

$\mathbf{L}$ 是一个单位下三角阵，$\mathbf{L}$ 对角线以下的元素就是各步消去的乘数。另外由（8）可知 $\mathbf{A}^{(N-1)}$ 为上三角阵，记 $\mathbf{U}=[u_{ij}]=\mathbf{A}^{(N-1)}$，于是我们得到

$$\mathbf{A}=\mathbf{LU} \tag{11}$$

这样，我们从 Gauss 消去法的进一步讨论中得到了矩阵的 $\mathbf{LU}$ 分解。以上将 $\mathbf{A}$ 分解成单位下三角与上三角乘积的分解称为 $\mathbf{A}$ 的 Doolittle 分解；类似地我们也可以将 $\mathbf{A}$ 分解为下三角与单位上三角的乘积，这种分解称为 Crout 分解。以下选择 Doolittle 分解作讨论，Crout 分解也可作类似讨论。

2.2 矩阵 LU 分解的计算

现讨论如何对给定非奇异矩阵 $\mathbf{A}$ 作 $\mathbf{LU}$ 分解。由上一小节，我们可由 Gauss 消去法得到 $\mathbf{A}$ 的 $\mathbf{LU}$ 分解，现从另一个角度考虑 $\mathbf{LU}$ 分解的计算公式，假设 $\mathbf{A}$ 可作 $\mathbf{LU}$ 分解，由矩阵乘法得

$$a_{ij}=\sum_{k=0}^{\min\{i,j\}}l_{ik}u_{kj} \qquad i,j=0,1,\cdots,N-1 \tag{12}$$

因为 $l_{ii}=1$，故（12）的 N^2 个方程可解出 N^2 个未知数。用以下算法可得到 **L** 和 **U** 的全部元素：

对 $j=0,1,\cdots,N-1$.

首先，对 $i=0,1,\cdots,j$，用以下公式计算 u_{ij}

$$u_{ij}=a_{ij}-\sum_{k=0}^{i-1}l_{ik}u_{kj} \tag{13}$$

再对 $i=j+1,j+2,\cdots,N-1$，用以下公式计算 l_{ij}

$$l_{ij}=\frac{1}{u_{ij}}\left(a_{ij}-\sum_{k=0}^{j-1}l_{ik}u_{kj}\right) \tag{14}$$

注意，必须完成上述两步后才能对 j 的值加 1；另外，我们约定和式 $\sum_{k=0}^{-1}=0$。这种算法称为 Crout 算法。

依次执行以上的迭代过程时，方程（13）和（14）右边需要用到的 l_{ij} 和 u_{ij} 在之前已被算出，可见 Crout 算法是可行的。另外，每个 a_{ij} 只被用过一次，于是，在编制程序的时候，如果 **A** 不再用到，我们可以把 l_{ij} 和 u_{ij} 储存在对应的 a_{ij} 的位置上，这种储存方式称为紧凑形式。Crout 算法自上到下、自左到右地逐列计算出以下矩阵

$$\begin{pmatrix} u_{00} & u_{01} & \cdots & u_{0,N-1} \\ l_{10} & u_{11} & \cdots & u_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ l_{N-1,0} & l_{N-1,1} & \cdots & u_{N-1,N-1} \end{pmatrix}$$

2.3 选主元的 LU 分解

注意到上述算法能顺利进行的条件是 $u_{ii}\neq 0$（$i=0,1,\cdots,N-2$），由 Gauss 消去法与 **LU** 分解的关系可知，这与之前假设的 $a_{ii}^{(i)}\neq 0$（$i=0,1,\cdots,N-2$）是一致的。事实上，在数值计算中，即使 $u_{ii}\neq 0$，但 $|u_{ii}|$ 很小，由于舍入误差的影响，也会导致结果不可靠。在 [4] 中有这样一个例子

$$\begin{pmatrix} 1.00\times 10^{-5} & 1.00 \\ 1.00 & 1.00 \end{pmatrix} \tag{15}$$

如果我们只用三位十进制浮点运算按（13）（14）式求解，因为 $u_{00}=a_{00}$ 很小，致使 $l_{10}=\frac{a_{10}}{u_{00}}$ 是 10^5 的数量级，于是，当我们计算

$$u_{11}=a_{11}-l_{10}a_{01}=1.00-1.00\times 10^5$$

时，a_{11} 的值就会被掩盖了。这个现象对于线性方程组求解是十分有害的，为此，我们使用选主元的方法解决这个问题。

在 Gauss 消去过程的第 k 步（$k\geq 0$）中，我们在 $\mathbf{A}^{(k)}$ 右下角的 $N-k$ 阶子阵中选取绝对值最大的元素 $a_{i_kj_k}^{(k)}$，即

$$|a_{i_kj_k}^{(k)}|=\max\{|a_{ij}|\,\big|\,k\leq i\leq N-1,k\leq j\leq N-1\} \tag{16}$$

因 $\mathbf{A}^{(k)}$ 非奇异，故有 $a_{i_kj_k}^{(k)} \neq 0$，这时选出了主元 $a_{i_kj_k}^{(k)}$；然后将 $\mathbf{A}^{(k)}$ 的第 i_k 行与第 k 行，第 j_k 列与第 k 列交换，使（14）可顺利进行，并尽量减少舍入误差的影响。这种选主元的方法称为完全主元法。

在实际当中我们更多使用的是只在 $\mathbf{A}$ 的第 k 列对角线以下选取主元的办法，称为列主元法或部分主元法，即将（16）换成

$$|a_{i_kk}^{(k)}| = \max\{|a_{ij}| \,\big|\, k \leq i \leq N-1\} \tag{17}$$

然后我们只需交换 $\mathbf{A}^{(k)}$ 的第 i_k 行与第 k 行，运算量比完全主元法要少。可以证明列主元法的舍入误差一般已较小，效果与完全主元法相差不大（更多的讨论可见 [1]），故我们给出的程序使用列主元法。

列主元法有换行的步骤，我们写出其对应的矩阵表示。在线性代数中，我们知道如果交换矩阵 $\mathbf{A}$ 的第 i 行与第 j 行，等价于 $\mathbf{A}$ 左乘一个初等排列阵 $\mathbf{I}_{ij}$，即 $\mathbf{I}_{ij}\mathbf{A}$，记

$$\mathbf{e}_k = (0, \cdots, 0, 1, 0, \cdots, 0)^T$$

是第 k 个（$0 \leq k \leq N-1$）分量为 1 的单位向量，所以

$$\mathbf{I}_{ij} = \big(\mathbf{e}_0, \ \cdots, \ \overset{i}{\mathbf{e}_j}, \ \cdots, \ \overset{j}{\mathbf{e}_i}, \ \cdots, \ \mathbf{e}_{N-1}\big)$$

$\mathbf{I}_{ij}$ 是单位阵 $\mathbf{I}$ 交换第 i，j 列（行）而得的，若 $i = j$，我们约定 $\mathbf{I}_{ii} = \mathbf{I}$。同理，若用 $\mathbf{I}_{ij}$ 右乘 $\mathbf{A}$，则相当于交换 $\mathbf{A}$ 的第 i，j 列。易知，初等排列阵是对合矩阵，即 $\mathbf{I}_{ij} = \mathbf{I}_{ij}^{-1}$。

在编写程序时，我们并不用初等排列阵左乘矩阵来实现行交换，而是只要用一个数组记录矩阵的行交换情况，以避免工作量较大的矩阵乘法。开始时，将一 N 维数组 pos 初始化为 $\{0, 1, \cdots, N-1\}$，当矩阵交换第 i 和第 j 行时，就交换数组 pos 中 $pos[i]$ 与 $pos[j]$ 的值，以后我们使用 $A[pos[i]][j]$ 引用交换行后的矩阵元素 a'_{ij}（其值不一定等于原来的 a_{ij}）。与排列矩阵类似，我们称 pos 为排列数组，在后面的程序实现中，我们将用到该技巧。

列主元消去法的每一步，先换行后消去，所以有 $\mathbf{A}^{(k+1)} = \mathbf{L}_{k+1}\mathbf{I}_{k,i_k}\mathbf{A}^{(k)}$，于是，列主元消去过程可写成矩阵形式

$$\mathbf{U} = \mathbf{A}^{(N-1)} = \mathbf{L}_{N-1}\mathbf{I}_{N-2,i_{N-2}}\mathbf{L}_{N-2}\mathbf{I}_{N-3,i_{N-3}}\cdots\mathbf{L}_1\mathbf{I}_{0,i_0}\mathbf{A} \tag{18}$$

其中 $i_k > k$（$k = 0, 1, \cdots, N-2$）。

由（18）和初等排列阵的对合性可得

$$\mathbf{A} = \mathbf{I}_{0,i_0}\mathbf{L}_1^{-1}\mathbf{I}_{1,i_1}\mathbf{L}_2^{-1}\cdots\mathbf{I}_{N-2,i_{N-2}}\mathbf{L}_{N-1}^{-1}\mathbf{U}$$

令排列阵

$$\mathbf{P} = \mathbf{I}_{N-2,i_{N-2}}\mathbf{I}_{N-3,i_{N-3}}\cdots\mathbf{I}_{0,i_0}$$

于是有

$$\begin{aligned}\mathbf{PA} = {} & (\mathbf{I}_{N-2,i_{N-2}}\mathbf{I}_{N-3,i_{N-3}}\cdots\mathbf{I}_{1,i_1}\mathbf{L}_1^{-1}\mathbf{I}_{1,i_1}\cdots\mathbf{I}_{N-3,i_{N-3}}\mathbf{I}_{N-2,i_{N-2}})\cdot \\ & (\mathbf{I}_{N-2,i_{N-2}}\cdots\mathbf{I}_{2,i_2}\mathbf{L}_2^{-1}\mathbf{I}_{2,i_2}\cdots\mathbf{I}_{N-2,i_{N-2}}) \\ & \cdots(\mathbf{I}_{N-2,i_{N-2}}\mathbf{L}_{N-2}^{-1}\mathbf{I}_{N-2,i_{N-2}})\mathbf{L}_{N-1}^{-1}\mathbf{U}\end{aligned}$$

记

$$\mathbf{L}'_k = \mathbf{I}_{N-2,i_{N-2}} \cdots \mathbf{I}_{k,i_k} \mathbf{L}_k^{-1} \mathbf{I}_{k,i_k} \cdots \mathbf{I}_{N-2,i_{N-2}} \qquad k = 1, \cdots, N-2$$
$$\mathbf{L}'_{N-1} = \mathbf{L}_{N-1}^{-1}$$

可以验证，$\mathbf{L}'_k$ 为一单位下三角阵，令 $\mathbf{L} = \mathbf{L}'_1 \mathbf{L}'_2 \cdots \mathbf{L}'_{N-1}$，显然，$\mathbf{L}$ 也是一个单位下三角阵，这样我们有以下定理。

定理 2.1　$\mathbf{A}$ 为非奇异阵，则存在排列矩阵 $\mathbf{P}$，单位下三角阵 $\mathbf{L}$ 和上三角阵 $\mathbf{U}$，使

$$\mathbf{PA} = \mathbf{LU} \tag{19}$$

有了以上的理论，我们可以讨论带列主元法的 $\mathbf{LU}$ 分解了。在 §2.2 中，我们用 Crout 算法依次计算 $\mathbf{L}$ 与 $\mathbf{U}$ 的元素，那时并未考虑选主元，现考虑带列主元法的 Crout 算法。以下可以看到（13）与（14）的写法对选主元是有利的。

首先（即第 0 步），我们对 $\mathbf{A}$ 的第 0 列施行列主元法，得到 $\mathbf{A}^{(0)} = [a_{ij}^{(0)}]$，然后按（13）（14）式进行第 0 步计算。设带列主元法的 Crout 算法进行到第 j 步（$j > 0$），由于前面可能实施了行交换，故在（13）（14）式中，a_{ij} 应记为 $a_{ij}^{(j-1)}$，l_{ij} 应记为 $l_{ij}^{(j-1)}$，u_{ij} 应记为 $u_{ij}^{(j-1)}$。另外，观察到（14）式括号的部分与（13）式在形式上是相同的，特别当 $i = j$ 时两者相等，并等于（14）式中作除数的对角元，选主元的办法是为了使这个对角元的绝对值尽量大，因此我们令

$$s_i = a_{ij}^{(j-1)} - \sum_{k=0}^{j-1} l_{ik}^{(j-1)} u_{kj}^{(j-1)} \qquad i = j, j+1, \cdots, N-1 \tag{20}$$

选取 $s = s_{i_k}$ 满足

$$|s_{i_k}| = \max\{|s_i| \,\big|\, j \le i \le N-1\} \tag{21}$$

然后交换当前矩阵（我们使用紧凑形式）的第 j 与第 i_k 行，并交换 s_j 与 s_{i_k} 的值，于是就可用 Crout 算法进行第 j 步计算，利用（20）式，（14）式可写成 $l_{ij}^{(j)} = s_i/s$，$i = j+1, j+2, \cdots, N-1$。

经过以上的 $N-1$ 步后，就得到

$$\mathbf{PA} = \mathbf{LU}$$

下面是带列主元法的 $\mathbf{LU}$ 分解的程序实现，对于矩阵的换行操作，我们用到了在上面提及的排列数组（pos）的技巧，这个技巧使程序更快捷有效。

```
#include "nr.h"
int ludcmp(int n, double A[][n], int pos[])
{
    int i, j, k, i_k;
    double sum, s[n];
    for (j ← 0; j < n; j++) {
        for (i ← 0; i < j; i++) {//用公式(13)计算u_ij。
            for (k ← 0, sum ← 0.0; k < i; k++)
                sum += A[pos[i]][k] * A[pos[k]][j];
```

```
            A[pos[i]][j] -= sum;
            s[i] ← 0;
        }
        for (i ← j; i < n; i++) {//用公式(20)计算s_i。
            for (k ← 0, sum ← 0.0; k < j; k++)
                sum += A[pos[i]][k] * A[pos[k]][j];
            s[i] ← A[pos[i]][j] - sum;
        }
        i_k ← fabs_max(n, s); //选取i_k，使|s_{i_k}| = max{|s_i| | j ≤ i ≤ N-1}。
        if (s[i_k] == 0.0) {//若A奇异。
            printf("Singular Matrix in LUdcmp\n");
            return -1;
        }
        if (i_k ≠ j) {
            swap_doub(&s[j], &s[i_k]); //交换s_i与s_{i_k}的值。
            swap_int(&pos[j], &pos[i_k]); //更新排列数组。
        }
        A[pos[j]][j] ← s[j]; //令a_jj^(j) = s_{i_k}。
        for (i ← j + 1; i < n; i++)
            A[pos[i]][j] ← s[i] / s[j]; //求得l_ij = s_i/s_j。
    }
    return 0;
}
```

用上述程序对（15）进行 **LU** 分解，有

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0.00001 & 1.00000 \\ 1.00000 & 1.00000 \end{pmatrix} = \begin{pmatrix} 1.00000 & 0.00000 \\ 0.00001 & 1.00000 \end{pmatrix}\begin{pmatrix} 1.00000 & 1.00000 \\ 0.00000 & 0.99999 \end{pmatrix}$$

LU 分解的稳定性取决于消去过程的中 $\mathbf{A}^{(k)}$ 的元素大小，若消去过程中出现较大的元素，则 **LU** 分解可能会不稳定。大量的实践证明，在带选列主元的 **LU** 分解中元素迅速增长的情况是非常罕见的，所以该方法可放心使用 [5]。更多更详尽的误差分析可参看 [1][2] 等的著作。

2.4 线性方程组的直接三角法

LU 分解的主要作用是求解线性方程组。由定理 2.1，解方程组 $\mathbf{Ax} = \mathbf{b}$ 就化为求解 $\mathbf{LUx} = \mathbf{Pb}$，这时可分两步求解。设 $\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})^T$，$\mathbf{Pb} = (b_0^{(0)}, b_1^{(1)}, \cdots, b_{N-1}^{(N-1)})^T$，先解 $\mathbf{Ly} = \mathbf{Pb}$，因为 $\mathbf{L}$ 是下三角阵，故用向前代入的方法求解：

$$\mathbf{y}_i = b_i^{(i)} - \sum_{k=0}^{i-1} l_{ik}u_{kj} \qquad i = 0, 1, \cdots, N-1 \tag{22}$$

然后解 $\mathbf{Ux} = \mathbf{y}$。因为 $\mathbf{U}$ 是上三角阵，故可用类似（9）的回代的方法求解：

$$\mathbf{x}_i = \frac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{N-1} u_{ij}x_j\right) \qquad i = N-1, N-2, \cdots, 0 \tag{23}$$

当中我们约定 $\sum_N^{N-1} = \sum_0^{-1} = 0$。上述算法称为线性方程组的 **LU** 分解方法。

我们将以上代入过程的封装到函数 *lubksb* 中，之前用 *ludcmp* 得到的 *pos* 现作为参数用于计算 **Pb**。

```
void lubksb(int n, double A[][n], double x[], double b[], int pos[])
{
    int i, k;
    double sum;
    for (i ← 0; i < n; i++) {//向前代入。
        for (k ← 0, sum ← 0.0; k < i; k++)
            sum += A[pos[i]][k] * x[k];
        x[i] ← b[pos[i]] - sum;
    }
    for (i ← n - 1; i ≥ 0; i--) {//向后回代。
        for (k ← i + 1, sum ← 0.0; k < n; k++)
            sum += A[pos[i]][k] * x[k];
        x[i] ← (x[i] - sum) / A[pos[i]][i];
    }
}
```

因为 **LU** 分解与 Gauss 消去法本质上是等价的，所以使用它们求解线性方程组的工作量均为 $O(N^3)$ 级。从这点看，**LU** 分解似乎意义不大，但不妨考虑以下情况：我们需要求解多个方程组，它们有相同的系数矩阵 **A**，不同的只是右边项。例如，在使用逆幂法求解绝对值最小的特征值时（逆幂法的讨论可参见数值分析教材），就会遇到这种情况。这时，**LU** 分解就可以使我们的工作“一劳永逸”，我们先对 **A** 进行 **LU** 分解，然后使用向前代入（22）和向后回代（23）便可解出每个方程组，显然（22）（23）的工作量只有 $O(N^2)$ 级。这就是 **LU** 分解比 Gauss 消去法更精妙之处。

另外，对于 **LU** 分解方法，我们要补充两点，首先，在数值上，能确保 **LU** 分解方法有效的条件除了系数矩阵为非奇异外，该矩阵还应该是良态的，即并不接近奇异。假设方程组的系数矩阵 **A** 有 $\delta\mathbf{A}$ 的变化（或称为扰动），则解 **x** 将产生 $\delta\mathbf{x}$ 的变化，即

$$(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}$$

在假设扰动 $\delta\mathbf{A}$ 较小的情况下，则有

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}\big(1 + O(\|\delta\mathbf{A}\|)\big) \tag{24}$$

其中

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$$

称为非奇异方阵的条件数。上述等式中范数可以是任意的范数，其证明可参看 [4]。由（24）可知，系数矩阵的条件数是相对误差的放大倍数，故当 $\kappa(\mathbf{A})$ 很大时，即使系数矩阵只有微小的变化，但也会对解产生巨大影响，这时我们称方程组是“病态的”。矩阵接近奇异的程

度可通过条件数来度量，我们将在 §4.2 中详细讨论这些问题，矩阵的奇异值分解技巧可帮助我们求解病态方程组。

其次，矩阵的 **LU** 分解会破坏矩阵的某些特性，如稀疏性，故对于稀疏的线性系统，我们经常会使用别的办法（如迭代法等）求解，还有，若矩阵有一点的改变，**LU** 分解不易于更新，而下一节讨论的 **QR** 分解却在较多情况下能实现工作量为 $O(N^2)$ 级的更新。

2.5 三对角方程组的追赶法

A 称为三对角阵，如果

$$\mathbf{A}=\begin{pmatrix} b_0 & c_0 & & & \\ a_1 & b_1 & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & a_{N-1} & b_{N-1} \end{pmatrix} \tag{25}$$

假设 **A** 不需经过换行就可以进行 **LU** 分解，可以验证 **L** 和 **U** 有以下形式：

$$\mathbf{L}=\begin{pmatrix} 1 & & & & \\ l_1 & 1 & & & \\ & l_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & l_{N-1} & 1 \end{pmatrix} \qquad \mathbf{U}=\begin{pmatrix} u_0 & c_0 & & & \\ & u_1 & c_1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & c_{N-2} \\ & & & & u_{N-1} \end{pmatrix} \tag{26}$$

利用（25）和（26）及矩阵乘法规则，可求得（26）中的 l_i 与 u_i：

$$\begin{aligned} u_0 &= b_0 \\ l_i &= \frac{a_i}{u_{i-1}} \qquad i=1,2,\cdots,N-1 \\ u_i &= b_i - l_i c_{i-1} \qquad i=1,2\cdots,N-1 \end{aligned} \tag{27}$$

我们把这个特殊的 **LU** 分解写成函数 *ludcmp_tridiag*，显然，只需使用一个 $3\times N$ 的二维数组储存三对角阵 **A**，以节省内存。其中，A[0]、A[1] 和 A[2] 分别用于存放 $\{c_0,c_1,\cdots,c_{N-2}\}$、$\{b_0,b_1,\cdots,b_{N-1}\}$ 和 $\{l_1,l_2,\cdots,l_{N-1}\}$。

```
void ludcmp_tridiag(int n, double A[][n])
{
    int i;
    for (i ← 1; i < n; i++) {
        A[2][i] ← A[2][i] / A[1][i − 1];
        A[1][i] ← A[1][i] − A[2][i] ∗ A[0][i − 1];
    }
}
```

按照直接三角分解方法，分两步求解三对角方程组 $\mathbf{Ax}=\mathbf{d}$，即解 $\mathbf{Ly}=\mathbf{d}$ 和 $\mathbf{Ux}=\mathbf{y}$，计算公式为：

$$\begin{aligned} y_0&=d_0 \\ y_i&=d_i-l_iy_{i-1} \qquad i=1,2,\cdots,N-1 \end{aligned} \tag{28}$$

$$\begin{aligned} x_{N-1}&=\frac{y_{N-1}}{u_{N-1}} \\ x_i&=\frac{y_i-c_ix_{i+1}}{u_i} \qquad i=N-2,N-3,\cdots,0 \end{aligned} \tag{29}$$

使用（27）–（29）求解三对角方程组的算法称为追赶法，或称为 Thomas 算法。

```
void thomas(int n, double A[][n], double x[], double d[])
{
    int i;
    ludcmp_tridiag(n, A);
    x[0] ← d[0];
    for (i ← 1; i < n; i++)
        x[i] ← d[i] - A[2][i] * x[i - 1];
    x[n - 1] ← x[n - 1] / A[1][n - 1];
    for (i ← n - 2; i ≥ 0; i--)
        x[i] ← (x[i] - A[0][i] * x[i + 1]) / A[1][i];
}
```

显然，追赶法能实现的条件是 $u_i\neq 0$，$i=0,1,\cdots,N-1$。以下定理给出能使追赶法进行的一个充分条件。

定理 2.2　$\mathbf{A}$ 为三对角阵，且严格对角占优，即满足

$$\begin{aligned} &|b_0|>|c_0| \qquad |b_{N-1}|>|a_{N-1}| \\ &|b_i|>|a_i|+|c_i| \qquad i=1,2,\cdots,N-2 \end{aligned} \tag{30}$$

则

$$u_i\neq 0 \quad \text{且} \quad \frac{|c_i|}{|u_i|}<1 \qquad i=0,1,\cdots,N-1 \tag{31}$$

$$|b_i|-|a_i|<|u_i|<|b_i|+|a_i| \qquad i=1,2,\cdots,N-1 \tag{32}$$

证明　先用归纳法证明（31）。对 $i=0$ 有 $|u_0|=|b_0|>|c_0|$，所以 $|u_0|\neq 0$，且 $\dfrac{|c_0|}{|u_0|}<1$。

设 $u_{i-1}\neq 0$（$i>0$），且 $\dfrac{|c_{i-1}|}{|u_{i-1}|}<1$，由（27）有

$$|u_i|=|b_i-l_ic_{i-1}|\geq |b_i|-\frac{|a_i||c_{i-1}|}{|u_{i-1}|}>|b_i|-|a_i| \tag{33}$$

再由条件（30）可得 $|u_i|>|c_i|$，故 $|u_i|\neq 0$，且有 $\dfrac{|c_i|}{|u_i|}<1$。因此（31）成立；

又因为

$$|u_i|\leq |b_i|+|l_i||c_{i-1}|=|b_i|+\frac{|a_i||c_{i-1}|}{|u_{i-1}|}<|b_i|+|a_i| \tag{34}$$

由（33）（34）可得（32）。 □

若线性方程组的系数矩阵为满足定理 2.2 条件的三对角阵时，可以使用追赶法求解，且中间变量 u_i 有（31）和（32）的估计，可以稳定有效地算出结果。在三次样条插值中，我们需要求解 M 关系式或 m 关系式，从而得到三次样条插值函数。所谓的 M（或 m）关系式其实是一线性方程组，其系数矩阵为严格对角占优的三对角阵，即满足定理 2.2 条件，故可用追赶法求解。在求解周期样条插值时，我们会遇到循环三对角方程组，这时可使用修正的追赶法（详见 [4]）。

由追赶法的计算公式（27）–（29）可知，计算的工作量是 $O(N)$ 级的，比一般的 $\mathbf{LU}$ 分解法的运算次数要少得多。

2.6 Cholesky 分解

$\mathbf{LU}$ 分解讨论的是所有的非奇异阵，上述追赶法的讨论提醒我们，具体问题应具体分析。同样，进一步讨论对称正定矩阵的 $\mathbf{LU}$ 分解，我们可得到 Cholesky 分解。

我们曾在 §2.3 中指出，对非奇异阵 $\mathbf{A}$，要使 $\mathbf{A}=\mathbf{LU}$ 能成立的的充分条件是 $a_{ii}^{(i)}\neq 0$（$i=0,1,\cdots,N-2$），容易证明，这个条件等价于 $\mathbf{A}$ 的顺序主子式 $\Delta_i\neq 0$（$i=0,1,\cdots,N-1$）。为得到对称正定阵的 Cholesky 分解，先证明几个简单的定理。

定理 2.3　非奇异矩阵 $\mathbf{A}\in\mathbb{R}^{N\times N}$，若其顺序主子式 $\Delta_i\neq 0$（$i=0,1,\cdots,N-1$），则存在唯一的 $\mathbf{LU}$ 分解，其中 $\mathbf{L}$ 为单位下三角阵，$\mathbf{U}$ 为上三角阵。

证明　由以上讨论可知存在性，下证唯一性。假设有两个分解式

$$\mathbf{A}=\mathbf{L}_1\mathbf{U}_1=\mathbf{L}_2\mathbf{U}_2$$

其中 $\mathbf{L}_1$，$\mathbf{L}_2$ 是单位下三角阵，$\mathbf{U}_1$，$\mathbf{U}_2$ 是上三角阵。因 $\mathbf{A}$ 非奇异，所以 $\mathbf{L}_1$，$\mathbf{L}_2$，$\mathbf{U}_1$，$\mathbf{U}_2$ 均可逆。故有

$$\mathbf{U}_1\mathbf{U}_2^{-1}=\mathbf{L}_1^{-1}\mathbf{L}_2$$

因为 $\mathbf{U}_2^{-1}$ 为上三角阵，所以 $\mathbf{U}_1\mathbf{U}_2^{-1}$ 也是上三角阵，同理，$\mathbf{L}_1^{-1}\mathbf{L}_2$ 是单位下三角阵。因此，只能是 $\mathbf{U}_1\mathbf{U}_2^{-1}=\mathbf{L}_1^{-1}\mathbf{L}_2=\mathbf{I}$，即 $\mathbf{L}_1=\mathbf{L}_2$，$\mathbf{U}_1=\mathbf{U}_2$，唯一性得证。 □

若我们将 $\mathbf{U}$ 进一步分解成 $\mathbf{U}=\mathbf{D}\widetilde{\mathbf{U}}$，其中 $\mathbf{D}$ 为对角阵，$\widetilde{\mathbf{U}}$ 是单位上三角阵，且 $\mathbf{D}=\mathrm{diag}(u_{00},u_{11},\cdots,u_{N-1,N-1})$，则定理 2.3 可写成。

定理 2.4　若非奇异矩阵 $\mathbf{A}\in\mathbb{R}^{N\times N}$ 的所有顺序主子式均不等于 0，则 $\mathbf{A}$ 有唯一的 $\mathbf{LDU}$ 分解，其中 $\mathbf{L}$ 为单位下三角阵，$\mathbf{D}$ 为对角阵，$\mathbf{U}$ 为单位上三角阵。

若 $\mathbf{A}$ 对称正定，则其顺序主子式均大于 0，由定理 2.4 及 $\mathbf{A}$ 的对称性可知，$\mathbf{A}=\mathbf{L}_1\mathbf{D}\mathbf{L}_1^T$；再由 $\mathbf{A}$ 的正定性可知，$\mathbf{D}=\mathrm{diag}(d_0,d_1,\cdots,d_{N-1})$ 也是正定的，记 $\mathbf{D}^{1/2}=\mathrm{diag}(\sqrt{d_0},\sqrt{d_1},\cdots\sqrt{d_{N-1}})$，则有

$$\mathbf{A}=\mathbf{L}_1\mathbf{D}^{1/2}\mathbf{D}^{1/2}\mathbf{L}_1^T$$

令 $\mathbf{L}=\mathbf{L}_1\mathbf{D}^{1/2}$，我们得到 Cholesky 分解定理：

定理 2.5　若 $\mathbf{A}\in\mathbb{R}^{N\times N}$，$\mathbf{A}$ 对称正定，则存在唯一的对角元素为正的下三角阵 $\mathbf{L}$，使

$$\mathbf{A}=\mathbf{L}\mathbf{L}^T \tag{35}$$

这种对称分解归功于 Cholesky，故称为 $\mathbf{A}$ 的 Cholesky 分解。设 $\mathbf{A}=[a_{ij}]$，$\mathbf{L}=[l_{ij}]$，类似 Crout 算法，我们可逐列计算出 $\mathbf{L}$ 的元素，设第 0 列到第 $j-1$ 列已算出，则第 j 列可如下计算

$$\begin{aligned} l_{jj} &= \left(a_{jj}-\sum_{k=0}^{j-1} l_{jk}^2\right)^{1/2} \\ l_{ij} &= \frac{1}{l_{jj}}\left(a_{ij}-\sum_{k=0}^{j-1} l_{ik}l_{jk}\right) \qquad i=j+1,j+2,\cdots,N-1 \end{aligned} \tag{36}$$

于是，与 $\mathbf{LU}$ 分解法一样，可利用 $\mathbf{A}$ 的 Cholesky 分解式分两步求解 $\mathbf{Ax}=\mathbf{b}$，这种方法称为 Cholesky 方法或平方根法。这里我们只给出 Cholesky 分解的实现。

```
#include "nr.h"
int Chodcmp(int n, double A[][n])
{
    int i, j, k;
    double sum;
    for (j ← 0; j < n; j++) {
        for (k ← 0, sum ← 0.0; k < j; k++)
            sum += A[j][k] * A[j][k];
        A[j][j] -= sum;
        if (A[j][j] ≡ 0.0) {//分解中断，A不正定。
            printf("A Is Not Positive Definite!\n");
            return -1;
        }
        A[j][j] ← sqrt(A[j][j]);
        for (i ← j + 1; i < n; i++) {
            for (k ← 0, sum ← 0.0; k < j; k++)
                sum += A[i][k] * A[j][k];
            A[i][j] ← (A[i][j] - sum) / A[j][j];
            A[j][i] ← A[i][j];
        }
    }
    return 0;
}
```

由上面的推导可知，Cholesky 方法是基于 $\mathbf{LU}$ 分解法的，故它也是 Gauss 消去法的变形，可以验证，它们的工作量都是 $O(N^3)$ 级的，但由更精细的分析（参见 [4]）可知，Cholesky 方法比 $\mathbf{LU}$ 分解法的运算次数要小一个常数因子 2，Cholesky 方法的是 $O(\frac{1}{6}N^3)$，而 $\mathbf{LU}$ 分解法的是 $O(\frac{1}{3}N^3)$，这是由于利用了矩阵对称正定的性质。

由（35）式和矩阵乘法规则可知

$$a_{ii}=\sum_{k=0}^{i} l_{ik}^2 \qquad i=0,1,\cdots,N-1$$

由此推出 $|l_{ik}|\le\sqrt{a_{ii}}$，所以 Cholesky 分解方法的中间变量有界，算法是稳定可靠的。

最后，我们以一个例子结束本节讨论。首先介绍一个著名的矩阵：

$$\mathbf{H}_N = \begin{pmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{N} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{N+1} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{N} & \frac{1}{N+1} & \cdots & \frac{1}{2N-1} \end{pmatrix}$$

该矩阵称为 N 阶 Hilbert 矩阵，是一个对称矩阵，并且可以证明它是正定的，满足的定理 2.5 的条件，所以可作 Cholesky 分解 $\mathbf{H}_N = \mathbf{L}\mathbf{L}^T$，我们用上述程序计算 $N = 5$ 时的分解，保留小数点后六位数字有

$$\mathbf{L} = \begin{pmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.500000 & 0.288675 & 0.000000 & 0.000000 & 0.000000 \\ 0.333333 & 0.288675 & 0.074536 & 0.000000 & 0.000000 \\ 0.250000 & 0.259808 & 0.111803 & 0.018898 & 0.000000 \\ 0.200000 & 0.230940 & 0.127775 & 0.037796 & 0.004762 \end{pmatrix}$$

观察到 $\mathbf{L}$ 的对角元是依次递减，若以 Hilbert 作为线性方程组的系数矩阵，我们使用直接三角分解方法求解时会遇到很小的对角元，而在回代过程中，对角元是充当除数的，这必然会使解的分量变得巨大，对稳定性造成很坏的影响，可见 $\mathbf{H}_N$ 是“病态”矩阵，条件数很大，我们会在后面继续讨论其病态性。

3. 矩阵的 QR 分解及其应用

上节讨论的 $\mathbf{LU}$ 分解只适用于非奇异的方阵，而本节讨论的 $\mathbf{QR}$ 分解则适用于更多的矩阵，我们有以下定理

定理 3.1　设 $\mathbf{A} \in \mathbb{R}^{M\times N}$，则存在正交矩阵 $\mathbf{Q} \in \mathbb{R}^{M\times M}$ 和上三角矩阵 $\mathbf{R} \in \mathbb{R}^{M\times N}$，使

$$\mathbf{A} = \mathbf{QR} \tag{37}$$

所谓正交矩阵，即 $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$，$\mathbf{I}$ 为单位矩阵。在给出 $\mathbf{QR}$ 分解定理的证明之前，我们先引入两个正交变换：Householder 变换和 Givens 变换，这两个变换是我们构造出上述正交矩阵的关键。

3.1 正交变换

3.1.1 Householder 变换

定义 3.1　设 $\mathbf{w} \in \mathbb{R}^N$，且 $\|\mathbf{w}\|_2 = 1$，$\mathbf{I}$ 为 N 阶单位矩阵，则

$$\mathbf{P} = \mathbf{I} - 2\mathbf{w}\mathbf{w}^T \tag{38}$$

称为 Householder 矩阵，或 Householder 变换，或 Householder 反射变换。

易知，Householder 矩阵是对称阵，且是正交阵。它的一个重要应用是对非零向量 $\mathbf{x}=(x_0,x_1,\cdots,x_{N-1})^T$，求 Householder 矩阵 $\mathbf{P}$，使得 $\mathbf{Px}=l_0\mathbf{e}_0$，其中 $\mathbf{e}_0=(1,0,\cdots,0)^T\in\mathbb{R}^N$。因为 $\mathbf{P}$ 为正交矩阵，所以

$$l_0^2=l_0^2\mathbf{e}_0^T\mathbf{e}_0=\mathbf{x}^T\mathbf{P}^T\mathbf{Px}=\mathbf{x}^T\mathbf{x}=\|\mathbf{x}\|_2^2$$

令 $\mathbf{w}=\frac{\mathbf{x}-l_0\mathbf{e}_0}{\|\mathbf{x}-l_0\mathbf{e}_0\|_2}$，则由（38）得

$$\begin{aligned}\mathbf{Px}&=\mathbf{x}-\frac{2(\mathbf{x}-l_0\mathbf{e}_0)(\mathbf{x}^T\mathbf{x}-l_0\mathbf{e}_0^T\mathbf{x})}{\|\mathbf{x}-l_0\mathbf{e}_0\|_2^2}\\&=\mathbf{x}-\frac{(\mathbf{x}-l_0\mathbf{e}_0)(\mathbf{x}^T\mathbf{x}-l_0\mathbf{e}_0^T\mathbf{x}+l_0^2\mathbf{e}_0^T\mathbf{e}_0-l_0\mathbf{x}^T\mathbf{e}_0)}{\|\mathbf{x}-l_0\mathbf{e}_0\|_2^2}\\&=\mathbf{x}-(\mathbf{x}-l_0\mathbf{e}_0)=l_0\mathbf{e}_0\end{aligned}$$

上述构造的 Householder 变换将向量第 0 个分量以下的分量都变为 0。以下是 $\mathbf{P}$ 的计算公式。为避免计算时损失有效数位，取

$$l_0=-\text{sgn}(x_0)\|\mathbf{x}\|_2\qquad\text{sgn}(x_0)=\begin{cases}1, & x_0\geq 0\\-1, & x_0<0\end{cases}\tag{39}$$

$$\mathbf{u}=\mathbf{x}-l_0\mathbf{e}_0=\big(x_0+\text{sgn}(x_0)\|\mathbf{x}\|_2,x_1,\cdots,x_{N-1}\big)^T\tag{40}$$

$$\beta=\big(\|\mathbf{x}\|_2(\|\mathbf{x}\|_2+|x_0|)\big)^{-1}\tag{41}$$

$$\mathbf{P}=\mathbf{I}-\beta\mathbf{u}\mathbf{u}^T\tag{42}$$

按（39）—（42）计算 $\mathbf{P}$，可使 $\mathbf{Px}=l\mathbf{e}_0$ 对 $\mathbf{x}\neq\mathbf{0}$ 成立。

若我们要使 $\mathbf{x}$ 的第 k 个（$k>0$）分量以下的分量变为 0，则我们可先用（39）—（42）构造 $\bar{\mathbf{P}}\in\mathbb{R}^{(N-k)\times(N-k)}$，使得向量 $\bar{\mathbf{x}}=(x_k,x_{k+1},\cdots,x_{N-1})^T$ 化为 $l_k\bar{\mathbf{e}}_0=(l_k,0,\cdots,0)^T\in\mathbb{R}^{N-k}$ 的形式，再令

$$\mathbf{P}=\begin{pmatrix}\mathbf{I}_k & \mathbf{0}\\ \mathbf{0} & \bar{\mathbf{P}}\end{pmatrix}$$

其中 $\mathbf{I}_k$ 为 k 阶单位矩阵。这样构造的变换矩阵 $\mathbf{P}$ 仍是 Householder 矩阵，它使得

$$\mathbf{Px}=(x_0,x_1,\cdots,x_{k-1},l_k,0,\cdots,0)^T$$

其中 $l_k=-\text{sgn}(x_k)\|\bar{\mathbf{x}}\|_2$，这时 $\mathbf{P}$ 的计算公式与之前的类似，但（40）和（41）应改为

$$\mathbf{u}=\big(0,\cdots,0,x_k+\text{sgn}(x_k)\|\bar{\mathbf{x}}\|_2,x_{k+1},\cdots,x_{N-1}\big)^T\tag{43}$$

$$\beta=\big(\|\bar{\mathbf{x}}\|_2(\|\bar{\mathbf{x}}\|_2+|x_k|)\big)^{-1}\tag{44}$$

以下是 Householder 变换的实现，函数 *householder* 将数组 $x[0:N-1]$ 的第 k 个分量以下分量化成 0，变换结果保存在原来的 $x[0:N-1]$ 上。由（42）可知，Householder 阵是被 β 和 $\mathbf{u}$ 唯一确定的，故我们只需保存 β 和 $\mathbf{u}$，而不必保存整个 Householder 阵，这样做除了可减少储存空间外，还可以减少以后的运算量，对 $\forall\mathbf{x}\in\mathbb{R}^N$，考虑下面方程

$$\mathbf{Px}=\mathbf{x}-\beta\mathbf{u}\mathbf{u}^T\mathbf{x}\tag{45}$$

计算左边项的工作量是 $O(N^2)$ 的，而计算右边项时，可先计算 $\beta\mathbf{u}^T\mathbf{x}$，然后再将上述因子乘以 $\mathbf{u}$ 的每个分量，其计算的工作量是 $O(2N)$ 的，大大节省运算，这个技巧将在后面的 **QR** 分解中派上用场。易知，Householder 变换的计算量是 $O(N)$ 级的。

```
#include "nr.h"
void householder(int n, int k, double u[], double *beta, double x[])
{
    int i, j;
    double length;
    length ← norm2(n − k, &x[k]);
    if (length ≡ 0 ∨ length ≡ fabs(x[k]))
        goto end; //若‖x̄‖2 = 0或xk以下元素已为0，则不需做任何事。
    u[k] ← (x[k] ≥ 0) ? (x[k] + length) : (x[k] − length);
    for (i ← k + 1; i < n; i++)
        u[i] ← x[i];
    *beta ← 1 / (length * (length + fabs(x[k])));
    x[k] ← x[k] − u[k];
    for (i ← k + 1; i < n; x[i] ← 0, i++);
  end:;
}
```

3.1.2 Givens 变换

Householder 可将向量的接连几个分量变为 0，而以下构造的 Givens 变换可将向量中指定的分量变为 0。如下矩阵

$$\mathbf{G}(i,k,\theta)=\begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & 1 & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \tag{46}$$

称为 Givens 矩阵，或 Givens 变换（或旋转变换）。显然，$\mathbf{G}(i,k,\theta)$ 是正交矩阵。对向量 $\mathbf{x}=(x_0,\cdots,x_i,\cdots,x_k,\cdots,x_{N-1})^T$，$x_k\neq 0$，令

$$c=\cos(\theta)=\frac{x_i}{\sqrt{x_i^2+x_k^2}} \qquad s=\sin(\theta)=\frac{x_k}{\sqrt{x_i^2+x_k^2}} \tag{47}$$

可使 $\mathbf{G}(i,k,\theta)\mathbf{x}$ 的第 k 个分量为零。为方便计算，把（47）写成

$$\begin{aligned} &\text{若 } |x_k|\geq|x_i|,\ t=\frac{x_i}{x_k},\ s=\frac{\text{sgn}(x_k)}{\sqrt{1+t^2}},\ c=\text{sgn}(x_k)st \\ &\text{若 } |x_k|<|x_i|,\ t=\frac{x_k}{x_i},\ c=\frac{\text{sgn}(x_i)}{\sqrt{1+t^2}},\ s=\text{sgn}(x_i)ct \end{aligned} \tag{48}$$

当 $x_k=0$ 时，不需要作变换，令 $c=1$，$s=0$ 即可。

Givens 矩阵的计算很简单，只要输入 x_i 和 x_k 就可按照（48）式计算 c 和 s，下面程序把 c 和 s 分别输出到 cs[0] 和 cs[1]。

```
#include "nr.h"
void givens(double x_i, double x_k, double cs[])
{
    if (x_k == 0) {
        cs[0] = 1;
        cs[1] = 0;
    } else {
        double t;
        if (fabs(x_k) >= fabs(x_i)) {
            t = x_i / x_k;
            cs[1] = 1 / sqrt(1 + t * t);
            cs[0] = cs[1] * t;
            if (x_k < 0) {
                cs[0] = -cs[0];
                cs[1] = -cs[1];
            }
        } else {
            t = x_k / x_i;
            cs[0] = 1 / sqrt(1 + t * t);
            cs[1] = cs[0] * t;
            if (x_i < 0) {
                cs[0] = -cs[0];
                cs[1] = -cs[1];
            }
        }
    }
}
```

可以验证，$\mathbf{G}(i,k,\theta)\mathbf{x}$ 只改变 $\mathbf{x}$ 的第 i，k 个分量，$\mathbf{G}(i,k,\theta)\mathbf{A}$ 只改变 $\mathbf{A}$ 的第 i，k 行，$\mathbf{AG}(i,k,\theta)$ 只改变 $\mathbf{A}$ 的第 i，k 列。故我们在计算与 Givens 矩阵的乘积时，只需计算相应的两行或两列即可。因此，对一个 N 阶方阵作一次 Givens 变换的工作量是 $O(2N)$ 级的。

3.2 矩阵的 QR 分解

定理 3.2　设 $\mathbf{A}\in\mathbb{R}^{M\times N}$，则存在正交阵 $\mathbf{P}\in\mathbb{R}^{M\times M}$，使 $\mathbf{PA}=\mathbf{R}$，其中 $\mathbf{R}\in\mathbb{R}^{M\times N}$ 为上三角阵。

证明　用 Householder 变换构造 $\mathbf{P}$，不妨设 $M\geq N$。记 $\mathbf{A}$ 的第 0 列记为 $\mathbf{a}_0$，按公式（39）—（42），可得 Householder 矩阵 $\mathbf{P}_0$，使

$$\mathbf{P}_0\mathbf{a}_0=l_0\mathbf{e}_0$$

令 $\mathbf{A}^{(0)}=\mathbf{P}_0\mathbf{A}$，其第 0 列除对角元外均为 0。假设对某个 $j>0$，有 Householder 矩阵

$\mathbf{P}_{j-1}\cdots\mathbf{P}_1\mathbf{P}_0$ 使得

$$\mathbf{A}^{(j-1)}=\mathbf{P}_{j-1}\cdots\mathbf{P}_1\mathbf{P}_0\mathbf{A}=\begin{pmatrix}\mathbf{R}^{(j-1)} & \mathbf{B}^{(j-1)}\\ \mathbf{0} & \bar{\mathbf{A}}^{(j-1)}\end{pmatrix}$$

其中 $\mathbf{R}^{(j-1)}$ 为 j 阶上三角方阵，$\bar{\mathbf{A}}^{(j-1)}\in\mathbb{R}^{(M-j)\times(N-j)}$。这时，我们用（43）（44）（42）构造的变换矩阵

$$\mathbf{P}_j=\begin{pmatrix}\mathbf{I}_j & \mathbf{0}\\ \mathbf{0} & \bar{\mathbf{P}}_j\end{pmatrix} \tag{49}$$

其中 $\mathbf{I}_j$ 是 j 阶单位阵。记 $\bar{\mathbf{A}}^{(j-1)}$ 的第 0 列为 $\bar{\mathbf{a}}_0^{(j-1)}$，则矩阵 $\bar{\mathbf{P}}_j\in\mathbb{R}^{M-j}$ 使得

$$\bar{\mathbf{P}}_j\bar{\mathbf{a}}_0^{(j-1)}=l_j(1,0,\cdots,0)^T$$

故用 $\mathbf{P}_j$ 对 $\mathbf{A}^{j-1}$ 作变换有

$$\mathbf{A}^{(j)}=\mathbf{P}_j\mathbf{A}^{(j-1)}=\begin{pmatrix}\mathbf{R}^{(j)} & \mathbf{B}^{(j)}\\ \mathbf{0} & \bar{\mathbf{A}}^{(j)}\end{pmatrix} \tag{50}$$

其中 $\mathbf{R}^{(j)}$ 为 $j+1$ 阶上三角方阵；若遇到 $\mathbf{A}^{(j-1)}$ 的第 j 列的对角元以下的分量均为 0 时，则令 $\mathbf{P}_j=\mathbf{I}$。这样经 $N-1$ 步运算后得到，

$$\mathbf{P}_{N-2}\mathbf{P}_{N-3}\cdots\mathbf{P}_0\mathbf{A}=\mathbf{R}$$

其中 $\mathbf{R}$ 为上三角阵，$\mathbf{P}=\mathbf{P}_{N-2}\mathbf{P}_{N-3}\cdots\mathbf{P}_0$ 为正交矩阵。 □

类似地，我们也可使用 Givens 变换将 $\mathbf{A}$ 对角线下的元素逐个变为 0，从而得到定理 3.2 的结论。因为 $\mathbf{P}$ 为正交阵，只要令 $\mathbf{Q}=\mathbf{P}^T$，就有 $\mathbf{A}=\mathbf{QR}$，即可证明定理 3.1。

对一般矩阵而言，利用 Givens 变换作 $\mathbf{QR}$ 分解的工作量大约是用 Householder 变换的两倍，即使是用快速 Givens 变换，其计算 $\mathbf{QR}$ 分解的效果仍然慢于 Householder 变换 [6]，故我们给出的程序使用 Householder 变换对矩阵作 $\mathbf{QR}$ 分解，但在下面的讨论中，我们可以看到，对某些特殊矩阵作 $\mathbf{QR}$ 时，Givens 变换还是很有用的。

以下的程序是使用 Householder 变换进行的 $\mathbf{QR}$ 分解，我们把分解结果的上三角阵 $\mathbf{R}$ 保存在 $\mathbf{A}$ 的位置上，正交阵保存在 $Q[0:M-1][0:M-1]$ 上。值得注意的是，在 C 语言中，由于二维数组是逐行储存的，其列向量并不能按一维数组引用，为节省代码，我们使用了一个临时的数组保存其转置。函数 $transpose$ 是用于获得矩阵转置的，具体定义请见附录。

另外，因 $\mathbf{Q}^T$ 是由 $\mathbf{P}_{N-2}\mathbf{P}_{N-3}\cdots\mathbf{P}_0$ 累积得到，以及 $\mathbf{A}^{(j)}$ 由 $\mathbf{P}_j$ 与 $\mathbf{A}^{(j-1)}$ 相乘得到，可见矩阵乘法频繁使用，故有必要作优化。观察到（49），矩阵左乘 $\mathbf{P}_j$ 时，只改变其后 $M-j$ 行，所以只需用 $\bar{\mathbf{P}}_j$ 左乘其后 $M-j$ 行构成的子矩阵即可，特别地，计算 $\mathbf{A}^{(j)}$ 时，只需计算 $\bar{\mathbf{P}}_j\bar{\mathbf{A}}^{(j-1)}$。另外，使用（45）式右边的计算，上述的矩阵乘法会进一步减低运算量。设 $M\geq N$，则使用 Householder 变换的 $\mathbf{QR}$ 分解的工作量是 $O(M^2N)$ 的。

```
#include "nr.h"
void qrdcmp(int m, int n, double A[m][n], double Q[m][m])
```

```
{
    int i, j, k, min;
    double At[n][m], Qt[m][m], u[m], beta, tmp;
    transpose(m, n, A, At); //将A^T保存在At上。
    min ← (m ≥ n) ? n : m;
    identy_matrix(m, Qt);
    for (k ← 0; k < min − 1; k++) {
        householder(m, k, u, &beta, At[k]); //计算β和u。
        for (j ← 0; j < m; j++) {//计算P_k P_{k−1} ··· P_0。
            for (i ← k, tmp ← 0.0; i < m; i++)
                tmp += u[i] * Qt[i][j];
            tmp *= beta;
            for (i ← k; i < m; i++)
                Qt[i][j] −= tmp * u[i];
        }
        for (j ← k + 1; j < n; j++) {//计算A^(k) = P_k A^(k−1)。
            for (i ← k, tmp ← 0.0; i < m; i++)
                tmp += u[i] * At[j][i];
            tmp *= beta;
            for (i ← k; i < m; i++)
                At[j][i] −= tmp * u[i];
        }
    }
    transpose(n, m, At, A);
    transpose(m, m, Qt, Q);
}
```

3.3 更新 QR 分解

与矩阵 **LU** 分解类似，矩阵的 **QR** 也可用于求解线性方程组。对方程组

$$\mathbf{Ax} = \mathbf{b} \tag{2}$$

若 $M = N$，且 $\mathbf{A}$ 非奇异，则可利用 **QR** 分解将方程组化成

$$\mathbf{Rx} = \mathbf{Q}^T\mathbf{b} \tag{51}$$

然后用向后回代求解（51）即可。但事实上，我们一般不会用这个技巧求解方程组，因为 **QR** 分解的工作量差不多是 **LU** 分解的两倍 [3]。

但有的时候，我们需要求解一连串的线性系统，而每个系统与前一个仅有很小的差别，这时，由于选主元的原因，**LU** 分解的更新并不容易。由上面的程序可以知道，一次完整 **QR** 分解的工作量是 $O(N^3)$（这里假设矩阵是方阵），然而，对于一种常见的情况，**QR** 分解可以在 $O(N^2)$ 的工作量下更新分解因子。

假设我们已有分解 $\mathbf{QR} = \mathbf{A} \in \mathbb{R}^{N\times N}$，现希望用较少的工作量更新 $\mathbf{A} + \mathbf{st}^T$ 的 **QR** 分解，其中 $\mathbf{s}, \mathbf{t} \in \mathbb{R}^N$ 为已知向量，从而有新的分解

$$\mathbf{A} + \mathbf{st}^T = \mathbf{Q}'\mathbf{R}' = \mathbf{Q}(\mathbf{R} + \mathbf{ut}^T) \tag{52}$$

其中 $\mathbf{u}=\mathbf{Q}^T\mathbf{s}$。假定已算出 Givens 旋转矩阵 $\mathbf{J}_0$，$\mathbf{J}_1$，$\cdots$，$\mathbf{J}_{N-2}$ 使

$$\mathbf{J}_0\mathbf{J}_1\cdots\mathbf{J}_{N-2}\mathbf{u}=\pm\|\mathbf{u}\|_2\mathbf{e}_0$$

其中 $\mathbf{J}_i=\mathbf{G}(i,i+1,\theta_i)$。当这些 Givens 矩阵作用于 $\mathbf{R}$ 后，可证

$$\mathbf{H}=\mathbf{J}_0\mathbf{J}_1\cdots\mathbf{J}_{N-2}\mathbf{R}$$

为上 Hessenberg 阵，所谓上 Hessenberg 阵是这样的矩阵：

定义 3.2　若 $\mathbf{H}=[h_{ij}]\in\mathbb{R}^{N\times N}$，当 $i>j+1$ 时有 $h_{ij}=0$（即次对角线以下的元素为零），则称 $\mathbf{H}$ 为上 Hessenberg 矩阵。

例如，一个 4×4 的上 Hessenberg 阵是具有以下形式的矩阵

$$\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix}$$

回到我们的讨论，易知

$$(\mathbf{J}_0\mathbf{J}_1\cdots\mathbf{J}_{N-2})(\mathbf{R}+\mathbf{u}\mathbf{t}^T)=\mathbf{H}\pm\|\mathbf{u}\|_2\mathbf{e}_0\mathbf{t}^T=\mathbf{H}'$$

仍为上 Hessenberg 阵。

因为上 Hessenberg 阵 $\mathbf{H}'$ 的次对角线以下元素均为 0，所以只需用 $N-1$ 次 Givens 变换就可完成其 $\mathbf{QR}$ 分解，即

$$\mathbf{K}_{N-2}\cdots\mathbf{K}_1\mathbf{K}_0\mathbf{H}'=\mathbf{R}'$$

其中，$\mathbf{K}_i=\mathbf{G}(i,i+1,\theta_i)$，$\mathbf{R}'$ 为上三角阵，令

$$\mathbf{Q}'=\mathbf{Q}\mathbf{J}_{N-2}^T\cdots\mathbf{J}_0^T\mathbf{K}_0^T\cdots\mathbf{K}_{N-2}^T$$

为正交阵，则得到修改后的矩阵的 $\mathbf{QR}$ 分解 $\mathbf{A}+\mathbf{s}\mathbf{t}^T=\mathbf{Q}'\mathbf{R}'$。由上推导可见，我们共用了 $2(N-1)$ 次 Givens 变换，故其工作量是 $O(N^2)$ 级的。这种更新的技巧对长方矩阵的情形也是适用的；或者我们也可以推广到 $\mathbf{A}+\mathbf{S}\mathbf{T}^T$ 的 $\mathbf{QR}$ 分解，其中，$\mathbf{S},\mathbf{T}\in\mathbb{R}^{N\times N}$，$\mathrm{rank}(\mathbf{S}\mathbf{T}^T)=p>1$。上述讨论的情况因为 $\mathrm{rank}(\mathbf{s}\mathbf{t}^T)=1$，所以也称为秩 1 更新。更多的更新方法可参看 [5]。

以下是秩 1 更新的程序实现。

```
#include "nr.h"
void qrupdt(int m, int n, double Q[][m], double R[m][n], double s[],
            double t[])
{
    int i, j, k, l, pos[3];
    double u[m], cs[2], tmp;
    for (j ← 0; j < m; j++) {//计算u = Q^T s。
```

```
        for (i ← 0, u[j] ← 0.0; i < m; i++)
            u[j] += Q[i][j] * s[i];
    }
    for (k ← m − 1; k ≥ 0; k−−) {//求使得u_k ≠ 0的最大k。
        if (u[k] ≠ 0.0)
            break;
    }
    if (k < 0)
        k ← 0;
    for (i ← k − 1; i ≥ 0; i−−) {
        givens(u[i], u[i + 1], cs);
        u[i] ← cs[0] * u[i] + cs[1] * u[i + 1];
        u[i + 1] ← 0.0;
        for (j ← i; j < n; j++) {//求H = J_0 ··· J_{N−2}R。
            tmp ← cs[0] * R[i][j] + cs[1] * R[i + 1][j];
            R[i + 1][j] ← cs[0] * R[i + 1][j] − cs[1] * R[i][j];
            R[i][j] ← tmp;
        }
        for (j ← 0; j < m; j++) {//求QJ^T_{N−2} ··· J^T_0。
            tmp ← cs[0] * Q[j][i] + cs[1] * Q[j][i + 1];
            Q[j][i + 1] ← cs[0] * Q[j][i + 1] − cs[1] * Q[j][i];
            Q[j][i] ← tmp;
        }
        if (u[i] ≡ 0.0)
            i −= 1;
    }
    for (i ← 0; i < m; i++) {//求H' = H ± ‖u‖_2 e_0 t^T。
        for (j ← 0; j < n; j++)
            R[i][j] += u[i] * t[j];
    }
    for (i ← 0; i < n − 1; i++) {
        if (R[i + 1][i] ≡ 0.0)
            continue;
        givens(R[i][i], R[i + 1][i], cs);
        for (j ← i; j < n; j++) {//求K_{N−2} ··· K_0 H'。
            tmp ← cs[0] * R[i][j] + cs[1] * R[i + 1][j];
            R[i + 1][j] ← cs[0] * R[i + 1][j] − cs[1] * R[i][j];
            R[i][j] ← tmp;
        }
        for (j ← 0; j < m; j++) {//求QJ^T_{N−2} ··· J^T_0 K^T_0 ··· K^T_{N−2}。
            tmp ← cs[0] * Q[j][i] + cs[1] * Q[j][i + 1];
            Q[j][i + 1] ← cs[0] * Q[j][i + 1] − cs[1] * Q[j][i];
            Q[j][i] ← tmp;
        }
    }
```

}

3.4 QR 分解的性质

上面我们证明了 $\mathbf{QR}$ 分解的存在性。现在我们讨论正交阵 $\mathbf{Q}$ 的列向量与 $\mathbf{A}$ 的值域 range($\mathbf{A}$) 和其正交补 range($\mathbf{A}$)$^\perp$ 的关系以及 $\mathbf{QR}$ 分解的唯一性问题。

定理 3.3　若 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 列满秩，且有 $\mathbf{QR}$ 分解式 $\mathbf{A} = \mathbf{QR}$，设 $\mathbf{A} = (\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_{N-1})$，$\mathbf{Q} = (\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{M-1})$ 是按列的划分，则

$$\text{span}\{\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_k\} = \text{span}\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k\} \qquad k = 0, 1, \cdots, N-1 \tag{53}$$

证明　比较 $\mathbf{A} = \mathbf{QR}$ 的第 k 列，我们有

$$\mathbf{a}_k = \sum_{i=0}^{k} r_{ik}\mathbf{q}_i \in \text{span}\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k\}$$

所以，span$\{\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_k\} \subseteq$ span$\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k\}$。又因为 $\mathbf{A}$ 列满秩，所以 span$\{\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_k\}$ 的维数是 k，而 span$\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_k\}$ 的维数显然也是 k，故两个空间相等，即（53）成立。 □

记 $\mathbf{Q}_0 = (\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{N-1})$，$\mathbf{Q}_1 = (\mathbf{q}_N, \mathbf{q}_{N+1}, \cdots, \mathbf{q}_{M-1})$，则由定理 3.3 可知，range($\mathbf{A}$) = range($\mathbf{Q}_0$)，range($\mathbf{A}$)$^\perp$ = range($\mathbf{Q}_1$)。如果我们把 $\mathbf{R}$ 作分块

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_0 \\ \mathbf{0} \end{pmatrix} \qquad \mathbf{R}_0 \in \mathbb{R}^{N\times N} \tag{54}$$

那么 $\mathbf{A}$ 的 $\mathbf{QR}$ 分解可写成 $\mathbf{A} = \mathbf{Q}_0\mathbf{R}_0$ 的形式，称为瘦形 $\mathbf{QR}$ 分解。若 $\mathbf{A}$ 列满秩，显然 $\mathbf{R}_0$ 是非奇异的，令

$$\mathbf{D} = \text{diag}\left(\frac{r_{00}}{|r_{00}|}, \frac{r_{11}}{|r_{11}|}, \cdots, \frac{r_{N-1,N-1}}{|r_{N-1,N-1}|}\right)$$

则瘦形 $\mathbf{QR}$ 分解可写成 $\mathbf{A} = (\mathbf{Q}_0\mathbf{D})(\mathbf{DR}_0)$，其中 $\mathbf{Q}_0\mathbf{D}$ 仍然是列正交的，$\mathbf{DR}_0$ 是对角元为正的上三角阵，我们有以下定理。

定理 3.4　设 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 为列满秩矩阵，有瘦形 $\mathbf{QR}$ 分解 $\mathbf{A} = \mathbf{Q}_0\mathbf{R}_0$，若 $\mathbf{R}_0$ 的对角元素均大于零，则分解是唯一的。

证明　由 $\mathbf{A}^T\mathbf{A} = (\mathbf{Q}_0\mathbf{R}_0)^T(\mathbf{Q}_0\mathbf{R}_0) = \mathbf{R}_0^T\mathbf{R}_0$ 可知，$\mathbf{A}^T\mathbf{A}$ 对称正定，且有 Cholesky 分解 $\mathbf{R}_0^T\mathbf{R}_0$，由定理 2.5 可知 $\mathbf{R}_0$ 是唯一的，又因为 $\mathbf{Q}_0 = \mathbf{AR}_0^{-1}$，所以 $\mathbf{Q}_0$ 也是唯一的。 □

对于 $\mathbf{QR}$ 分解的稳定性，我们不加证明地给出以下结论。矩阵的最大奇异值与最小奇异值（奇异值的定义请见 §4.1）的比值

$$\kappa_2(\mathbf{A}) = \frac{\sigma_{\max}}{\sigma_{\min}} \tag{55}$$

称为 2– 范数下的矩阵条件数，在后面的讨论我们会知道，这与之前给出的方阵条件数定义是一致的。G. W. Stewart 证明了 $\mathbf{A}$ 中 $O(\epsilon)$ 的相对误差会引起 $\mathbf{R}_0$ 和 $\mathbf{Q}_0$ 有 $O(\epsilon\kappa_2(\mathbf{A}))$ 的相对误差 [5]。因此，对于条件数不大的良态矩阵，其 $\mathbf{QR}$ 分解是相对稳定的。

3.5 选主列的 QR 分解

由定理 3.3 可知，若 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 是列满秩的，则 **QR** 分解可以给出 range($\mathbf{A}$) 的一组正交基，但对于秩亏损的情形，则 **QR** 分解就不一定能产生 range($\mathbf{A}$) 的正交基。我们来看 [5] 给出的一个例子，如果 $\mathbf{A}$ 有三列且

$$\mathbf{A} = (\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2) = (\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2) \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

是其 **QR** 分解，则 rank($\mathbf{A}$) $= 2$，但显然 $\{\mathbf{q}_0, \mathbf{q}_1\}$ 并不是 range($\mathbf{A}$) 的一组基，因为 $\mathbf{a}_2 = \mathbf{q}_0 + \mathbf{q}_1 + \mathbf{q}_2$。

这个问题可通过选主列的办法解决，且我们只需对用 Householder 变换的 **QR** 分解稍作修改就可以生成 range($\mathbf{A}$) 的正交基。下面我们介绍选主列的 **QR** 分解。

对于 $\mathbf{A} \in \mathbb{R}^{M\times N}$，首先我们在 $\mathbf{A}$ 的列 $\{\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_{N-1}\}$ 中选出 $\mathbf{a}_{k_0}$，使得

$$\|\mathbf{a}_{k_0}\|_2 = \max\{\|\mathbf{a}_0\|_2, \|\mathbf{a}_1\|_2, \cdots, \|\mathbf{a}_{N-1}\|_2\}$$

这称为选出 $\mathbf{A}$ 的主列，交换第 0 与第 k_0 列，相当于右乘初等排列矩阵 $\mathbf{\Pi}_0 = \mathbf{I}_{0k_0}$，然后，按照以前的 Householder **QR** 分解一样进行第 0 步计算，即计算 Householder 矩阵 $\mathbf{P}_0$，使得 $\mathbf{P}_0\mathbf{A}\mathbf{\Pi}_0$ 的第 0 列除对角元外均为 0。假设对某个 $j > 0$，有 Householder 矩阵 $\mathbf{P}_{j-1}\cdots\mathbf{P}_0$ 和排列矩阵 $\mathbf{\Pi}_{j-1}\cdots\mathbf{\Pi}_0$ 使得

$$\mathbf{A}^{(j-1)} = \mathbf{P}_{j-1}\cdots\mathbf{P}_0\mathbf{A}\mathbf{\Pi}_0\cdots\mathbf{\Pi}_{j-1} = \begin{pmatrix} \mathbf{R}^{(j-1)} & \mathbf{B}^{(j-1)} \\ \mathbf{0} & \bar{\mathbf{A}}^{(j-1)} \end{pmatrix}$$

其中 $\mathbf{R}^{(j-1)}$ 为 j 阶上三角方阵，$\bar{\mathbf{A}}^{(j-1)} \in \mathbb{R}^{(M-j)\times(N-j)}$。这时，我们在 $\bar{\mathbf{A}}^{(j-1)}$ 中选出主列，设 $\{\bar{\mathbf{a}}_j^{(j-1)}, \bar{\mathbf{a}}_{j+1}^{(j-1)}, \cdots, \bar{\mathbf{a}}_{N-1}^{(j-1)}\}$ 是其按列的划分，则

$$\left\|\bar{\mathbf{a}}_{k_j}^{(j-1)}\right\|_2 = \max\left\{\left\|\bar{\mathbf{a}}_j^{(j-1)}\right\|_2, \left\|\bar{\mathbf{a}}_{j+1}^{(j-1)}\right\|_2, \cdots, \left\|\bar{\mathbf{a}}_{N-1}^{(j-1)}\right\|_2\right\}$$

如果这个最大值 0，则 rank($\mathbf{A}$) $= j$，计算至此结束。否则，用排列阵 $\mathbf{\Pi}_j = \mathbf{I}_{\mathbf{jk_j}}$ 右乘 $\mathbf{A}^{(j-1)}$，从而交换其第 j 与第 k_j 列，然后计算 Householder 矩阵 $\mathbf{P}_j$，使得

$$\mathbf{P}_j\mathbf{A}^{(j-1)}\mathbf{\Pi}_j = \begin{pmatrix} \mathbf{R}^{(j)} & \mathbf{B}^{(j)} \\ \mathbf{0} & \bar{\mathbf{A}}^{(j)} \end{pmatrix}$$

其中 $\mathbf{R}^{(j)}$ 为 $j+1$ 阶上三角方阵。这样，按照先选主列，然后交换列，再施行 Householder 变换的步骤，经过 $r =$ rank($\mathbf{A}$) 步后即可得到

$$\mathbf{P}_{r-1}\cdots\mathbf{P}_0\mathbf{A}\mathbf{\Pi}_0\cdots\mathbf{\Pi}_{r-1} = \mathbf{R} \tag{56}$$

其中 $\mathbf{R} \in \mathbb{R}^{M\times N}$ 是上三角阵，且由于有选主列的步骤，$\mathbf{R}$ 的对角元按绝对值从大到小地排列，矩阵 $\mathbf{A}$ 的秩等于其非零对角元的个数。设 rank($\mathbf{A}$) $= r$，则 $\mathbf{R}$ 的第 r 至第 $M-1$ 行均为 $\mathbf{0}$，所以，由定理 3.3 的证明方法可知，

$$\text{range}(\mathbf{A}) = \text{span}\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{r-1}\}$$

令 $\mathbf{Q}=(\mathbf{P}_{r-1}\cdots\mathbf{P}_0)^T$，上述 $\mathbf{q}_i$ 是 $\mathbf{Q}$ 的第 i 列。再令 $\mathbf{\Pi}=\mathbf{\Pi}_0\cdots\mathbf{\Pi}_{r-1}$，则有选主列的 **QR** 分解式

$$\mathbf{A\Pi}=\mathbf{QR} \tag{57}$$

下面我们给出选主列 **QR** 分解的程序实现。对于矩阵的列交换，我们仍然使用排列数组的技巧，即我们使用一维数组 pos 记录矩阵列交换的情况，每次使用 A[i][pos[j]] 引用作列交换后的矩阵 $\mathbf{A}'$ 的第 i 行第 j 列元素。

在选主列时，我们需要计算列向量的 2– 范数，而 2– 范数需要开平方运算，故我们选用 2– 范数的平方，在 nr.h 中，我们定义函数 xxT 计算向量 2– 范数的平方。另外，注意到正交变换是等距变换，设 $\bar{\mathbf{A}}^{(j-1)}$ 交换列后的列划分仍然用 $\{\bar{\mathbf{a}}_j^{(j-1)},\bar{\mathbf{a}}_{j+1}^{(j-1)},\cdots,\bar{\mathbf{a}}_{N-1}^{(j-1)}\}$ 表示，那么我们在第 j 步计算 $\bar{\mathbf{A}}^{(j)}$ 的列范数，只需用

$$\|\bar{\mathbf{a}}_k^{(j)}\|_2^2=\|\bar{\mathbf{a}}_k^{(j-1)}\|_2^2-(a_{jk}^{(j)})^2\qquad k=j+1,j+2,\cdots,N-1 \tag{58}$$

修正旧的列范数来得到新的列范数，而不必每步重新计算列范数。这使得选主列的工作量由 $O(MN^2)$ 减少到 $O(MN)$。设 $\mathbf{A}$ 的秩为 r，选主列 **QR** 分解的总工作量是 $O(M^2r+MN)$ 级的。

我们把选主列的 **QR** 分解方法封装到函数 $qrdcmp\_with\_pivot$ 中，该函数除返回正交阵和上三角阵外，还返回矩阵的“秩”，不过由于计算精度的限制，这个所谓的秩是数值意义上的，我们称之为数值秩，关于数值秩的更多讨论，我们将留到 §4.2 中。

```
#include "nr.h"
int qrdcmp_with_pivot(int m, int n, double A[m][n], double Q[m][m], int pos[])
{
    int i, j, k, min, p, rank ← 0;
    double At[n][m], Qt[m][m], u[m], beta, tmp, temp[n];
    init_array(n, pos);
    transpose(m, n, A, At);
    min ← (m ≥ n) ? n : m;
    identy_matrix(m, Qt);
    for (j ← 0; j < n; j++) {//计算A的列范数。
        temp[j] ← xxT(m, At[j]);
    }
    p ← max(m, temp);//选主列。
    if (temp[p] < EPSILON)//若矩阵元素小于机器精度，则停止计算。
        return rank;
    swap_int(&pos[0], &pos[p]);
    swap_doub(&temp[0], &temp[p]);
    for (k ← 0; k < min − 1; k++) {
        householder(m, k, u, &beta, At[pos[k]]);
        for (j ← 0; j < m; j++) {
            for (i ← k, tmp ← 0.0; i < m; i++)
                tmp += u[i] * Qt[i][j];
            tmp *= beta;
```

```
            for (i ← k; i < m; i++)
                Qt[i][j] -= tmp * u[i];
        }
        for (j ← k + 1; j < n; j++) {
            for (i ← k, tmp ← 0.0; i < m; i++)
                tmp += u[i] * At[pos[j]][i];
            tmp *= beta;
            for (i ← k; i < m; i++)
                At[pos[j]][i] -= tmp * u[i];
        }
        for (j ← k + 1; j < n; j++) {
            temp[j] -= At[pos[j]][k] * At[pos[j]][k];
        }
        p ← max(m - k - 1, &temp[k + 1]); //选主列。
        if (temp[p + k + 1] < EPSILON) {//若最大的列范数小于机器精度，则停止计算。
            rank ← k + 1;
            break;
        }
        swap_int(&pos[k + 1], &pos[p + k + 1]);
        swap_doub(&temp[k + 1], &temp[p + k + 1]);
    }
    if (rank ≡ 0)
        rank ← min;
    transpose(n, m, At, A);
    transpose(m, m, Qt, Q);
    return rank;
}
```

应用上述程序，我们可对 5 阶的 Hilbert 矩阵 $\mathbf{H}_5$ 作选主列的 $\mathbf{QR}$ 分解，设有分解式 $\mathbf{H}_5\mathbf{\Pi} = \mathbf{QR}$，则 $\mathbf{\Pi} = (\mathbf{e}_0, \mathbf{e}_2, \mathbf{e}_4, \mathbf{e}_1, \mathbf{e}_3)$，保留小数点后六位有

$$\mathbf{Q} = \begin{pmatrix} -0.826584 & 0.522405 & -0.201452 & 0.056852 & 0.006000 \\ -0.413292 & -0.332239 & 0.699616 & -0.464838 & -0.115198 \\ -0.275528 & -0.458870 & 0.142621 & 0.662702 & 0.503992 \\ -0.206646 & -0.462841 & -0.306463 & 0.243620 & -0.767988 \\ -0.165317 & -0.438094 & -0.596402 & -0.531196 & 0.377994 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} -1.209798 & -0.492014 & -0.317759 & -0.688820 & -0.385411 \\ 0.000000 & -0.140424 & -0.122977 & -0.129845 & -0.133207 \\ 0.000000 & 0.000000 & -0.007888 & 0.007442 & -0.005000 \\ 0.000000 & 0.000000 & 0.000000 & -0.000653 & 0.000099 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 & -0.000004 \end{pmatrix}$$

由 $\mathbf{R}$ 的最后一个对角元 -0.000004 可知，如果我们的精度低于 10^{-5} 的话，则程序返回 $\mathbf{H}_5$ 的数值秩是 4，因此，$\mathbf{H}_5$ 是非常接近奇异的，是个病态矩阵。事实上，$\kappa_2(\mathbf{H}_5)$ 的数量级是 10^5（我们会在 §4.3 中计算其具体数值），正好是上述精度临界值的倒数，可见数值秩的准确性依赖于矩阵的条件数。

3.6 满秩的最小二乘问题

由于 $\mathbf{QR}$ 分解适用于更大的矩阵类，所以可用它讨论线性方程组（2）$M > N$ 的情形，这时（2）称为超定线性方程组，其解可能存在，也可能不存在。对于后者，我们试图求出满足（3）的近似解，这里我们使用 2– 范数度量其近似程度，即求 $\mathbf{x}$，使 $\|\mathbf{Ax}-\mathbf{b}\|_2$ 最小。这称为超定线性方程组的最小二乘问题，由线性代数的知识可知，该问题等价于求解法方程组

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b} \tag{59}$$

在此，我们考虑 $\mathbf{A}$ 为列满秩的情况，即 $\text{rank}(\mathbf{A}) = N$。考察（59）的系数矩阵 $\mathbf{A}^T\mathbf{A}$，对 $\mathbf{A}$ 作瘦形 $\mathbf{QR}$ 分解 $\mathbf{A} = \mathbf{Q}_0\mathbf{R}_0$，由于 $\mathbf{Q}_0$ 是列正交阵，所以

$$\mathbf{A}^T\mathbf{A} = \mathbf{R}_0^T\mathbf{Q}_0^T\mathbf{Q}_0\mathbf{R}_0 = \mathbf{R}_0^T\mathbf{R}_0 \tag{60}$$

由上一小节的讨论可知 $\mathbf{R}_0$ 是非奇异的，所以 $\mathbf{A}^T\mathbf{A}$ 为对称正定矩阵，故存在 Cholesky 分解，$\mathbf{R}_0$ 是其分解因子。因此我们可以用 Cholesky 方法求解（59），这称为法方程组法。或者，利用 $\mathbf{R}_0$ 的可逆性将（59）化简为

$$\mathbf{R}_0\mathbf{x} = \mathbf{Q}_0^T\mathbf{b} \tag{61}$$

因为 $\mathbf{R}_0$ 是上三角阵，所以用向后回代即可解出（61），这称为 $\mathbf{QR}$ 法。

对于法方程组法与 $\mathbf{QR}$ 法，稳定的 $\mathbf{QR}$ 法的比法方程组法有较高的精度；法方程组法在 $M \gg N$ 时只需进行 $\mathbf{QR}$ 法一半的运算，且不需要太大的储存空间；$\mathbf{QR}$ 方法适用于更大的矩阵类。详细的比较请参看 [5]。

再考虑 $M < N$ 的情形，这时方程组（2）称为欠定方程组，它要么无解要么有无穷多个解，当有无穷多解的时候，则我们希望求出解集合中一个有代表性的解 — 极小 2 范数解，即具有最小长度 $\|\mathbf{x}\|_2$ 的那个解。这里我们假定系数矩阵 $\mathbf{A}$ 行满秩，即 $\text{rank}(\mathbf{A}) = M$。

考虑 $\mathbf{A}^T$ 的 $\mathbf{QR}$ 分解

$$\mathbf{A}^T = \mathbf{QR} = \begin{pmatrix} \mathbf{R}_0 \\ \mathbf{0} \end{pmatrix} \qquad \mathbf{R}_0 \in \mathbb{R}^{M\times M}$$

则 $\mathbf{Ax} = \mathbf{b}$ 化为

$$(\mathbf{QR})^T\mathbf{x} = (\mathbf{R}_0^T\ \mathbf{0}) \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \end{pmatrix} = \mathbf{b} \tag{62}$$

其中

$$\mathbf{Q}^T\mathbf{x} = \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \end{pmatrix} \qquad \mathbf{z}_0 \in \mathbb{R}^M,\ \mathbf{z}_1 \in \mathbb{R}^{N-M} \tag{63}$$

若令 $\mathbf{z}_1 = \mathbf{0}$，可以验证，联立（62）和（63）式解得的

$$\mathbf{x} = \mathbf{Q}\begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \end{pmatrix} = \begin{pmatrix} (\mathbf{R}_0^T)^{-1}\mathbf{b} \\ \mathbf{0} \end{pmatrix} \tag{64}$$

是欠定方程组的一个解，下面我们进一步证明 $\mathbf{x}$ 是极小范数解。

证明 假设 $\mathbf{x}' \in \text{null}(\mathbf{A})$，则欠定方程组的解可用 $\mathbf{x}+\mathbf{x}'$ 表示。我们有

$$\|\mathbf{x}+\mathbf{x}'\|_2 = \left\|\mathbf{Q}\left[\begin{pmatrix}\mathbf{z}_0\\ \mathbf{0}\end{pmatrix} + \mathbf{Q}^T\mathbf{x}'\right]\right\|_2 = \left\|\begin{pmatrix}\mathbf{z}_0\\ \mathbf{0}\end{pmatrix} + \mathbf{Q}^T\mathbf{x}'\right\|_2$$

观察 $\mathbf{Q}^T\mathbf{x}'$，由线性代数可知 $\text{null}(\mathbf{A})^\perp = \text{range}(\mathbf{A}^T)$，又由定理 3.3 可知，$\text{null}(\mathbf{A})^\perp = \text{range}(\mathbf{A}^T) = \text{span}\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{M-1}\}$，$\text{null}(\mathbf{A}) = \text{span}\{\mathbf{q}_M, \mathbf{q}_{M+1}, \cdots, \mathbf{q}_{N-1}\}$，其中 $\mathbf{q}_i$ 是 $\mathbf{Q}$ 的第 i 列，所以，$\mathbf{Q}^T\mathbf{x}'$ 的前 M 个分量为 0，当 $\mathbf{x}' \neq \mathbf{0}$ 时后 $N-M$ 个分量不全为 0。因此，只有当 $\mathbf{x}' = \mathbf{0}$ 时，解的长度最小，即 $\mathbf{x}$ 是极小范数解。 □

以上是满秩最小二乘问题的简单讨论，之所以假设满秩，是因为在实际应用中，被测量的变量应该是相互独立的，但是，在某些情况下，系数矩阵会出现非常接近秩亏的情形，这时由于条件数很大，$\mathbf{QR}$ 分解的方法将会变得不稳定。在下一节中，我们将使用矩阵奇异值分解的技巧进一步讨论秩亏损和接近秩亏损的情形。

3.7 矩阵特征值的计算

基于矩阵 $\mathbf{QR}$ 分解的 $\mathbf{QR}$ 方法是求实矩阵全部特征值的一种有效方法。对任意 $\mathbf{A} \in \mathbb{R}^{N\times N}$，由定理（3.1）有，$\mathbf{A} = \mathbf{QR}$，令 $\mathbf{A}_1 = \mathbf{RQ}$，则有 $\mathbf{A}_1 = \mathbf{Q}^T\mathbf{AQ}$，所以 $\mathbf{A}_1$ 与 $\mathbf{A}$ 有相同的特征值。继续对 $\mathbf{A}_1$ 作 $\mathbf{QR}$ 分解，可得

$$\begin{aligned} \mathbf{A}_0 &= \mathbf{A} \\ \mathbf{A}_k &= \mathbf{Q}_k\mathbf{R}_k \\ \mathbf{A}_{k+1} &= \mathbf{R}_k\mathbf{Q}_k \qquad k = 0, 1, \cdots\cdots \end{aligned} \tag{65}$$

由（65）得到序列 $\{\mathbf{A}_k\}$ 的方法称为 $\mathbf{QR}$ 方法。以下是关于 $\mathbf{QR}$ 方法收敛性的定理，其证明可参见 [4]

定理 3.5 $\mathbf{A} \in \mathbb{R}^{N\times N}$，设其特征值满足

$$|\lambda_0| > |\lambda_1| > \cdots > |\lambda_{N-1}| > 0$$

则用 $\mathbf{QR}$ 方法产生的序列 $\{\mathbf{A}_k\}$ 基本收敛到上三角阵，其对角元的极限为

$$\lim_{k\to\infty} a_{ii}^{(k)} = \lambda_i \qquad i = 0, 1, \cdots, N-1$$

其中，当 $k \to \infty$，若 $\mathbf{A}_k$ 的对角元均收敛，且严格下三角部分元素收敛到 0，则称 $\{\mathbf{A}_k\}$ 基本收敛到上三角阵。基本收敛的概念并未指出 $\{\mathbf{A}_k\}$ 严格上三角部分元素是否收敛，但这对求特征值而言，已经足够了。

$\mathbf{QR}$ 方法是一种迭代的方法，每次迭代都要作一次 $\mathbf{QR}$ 分解，可见 $\mathbf{QR}$ 分解对该算法起着重要的作用。为提高迭代的速度，我们希望有更快的 $\mathbf{QR}$ 分解，不妨考虑定义 3.2 描述的上 Hessenberg 矩阵。

因为上 Hessenberg 阵 $\mathbf{H}$ 的次对角线以下元素均为 0，所以只需用 $N-1$ 次 Givens 变换就可完成其 $\mathbf{QR}$ 分解，即 $\mathbf{H} = \mathbf{QR}$，其中

$$\mathbf{Q}^T = \mathbf{G}(N-2, N-1, \theta_{N-2})\mathbf{G}(N-3, N-2, \theta_{N-3})\cdots\mathbf{G}(0, 1, \theta_0)$$

可见对 Hessenberg 阵作 **QR** 分解的工作量是较少的（工作量是 $O(N^2)$ 级）。而且可以验证，当 $\mathbf{H}$ 是一个上 Hessenberg 阵。在作 **QR** 迭代时，下一步计算 $\mathbf{H}_1 = \mathbf{RQ}$，容易验证 **RQ** 也是一个上 Hessenberg 阵，这说明了 **QR** 方法保持上 Hessenberg 结构形式，使得每次迭代都能作工作量较少的 **QR** 分解，与一般矩阵 **QR** 分解的工作量 $O(N^3)$ 相比，大大节省运算工作量。

得知 Hessenberg 阵的这一性质后，我们容易想到在使用 **QR** 方法之前，希望能先将矩阵通过相似变换化成上 Hessenberg 矩阵，然后再使用 **QR** 方法。事实上，对任意给定的方阵，我们都可用正交相似变换将矩阵化成上 Hessenberg 矩阵，具体步骤可参见 [4]。

4. 矩阵的奇异值分解及其应用

对线性方程组（2），我们经常会遇到 $M > N$，或 $M = N$ 但 $\mathbf{A}$ 奇异的情形，这时（2）是欠定线性方程组。对于欠定线性方程组，**LU** 分解方法不能求解。但更多的情况是当方程组的系数矩阵在数值上非常接近奇异时，由于舍入误差的影响，**LU** 分解方法也无法进行，或者得出一个不令人满意的形式解。因此在应用中，我们要判断矩阵是否奇异或接近奇异，或者需要求出矩阵的秩。我们知道，**QR** 分解方法在矩阵非常接近奇异时也会表现得不太稳定，因此 **QR** 分解方法（或选主列的 **QR** 分解方法）只可求解不接近秩亏的情形，然而矩阵的奇异值分解（Singular Value Decomposition，简称 SVD）技术可以统一地解决上述所有问题，并且能使我们更深刻地理解问题的本质。所以，SVD 在理论分析和工程技术中有着广泛的应用。

在本节中，我们假设 $\mathbf{A} \in \mathbb{R}^{M\times N}$，且 $M \geq N$。（这只是为了方便而已，所有结论对 $M < N$ 时仍然成立。）所谓的奇异值分解是指 $\mathbf{A}$ 有分解式

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{66}$$

其中 $\mathbf{U} \in \mathbb{R}^{M\times M}$，$\mathbf{V} \in \mathbb{R}^{N\times N}$ 均为正交矩阵，分别称为左奇异向量矩阵和右奇异向量矩阵，$\mathbf{\Sigma} \in \mathbb{R}^{M\times N}$ 除对角元为非负数外，其余元素均为零。

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_0 & & & \\ & \sigma_1 & & \\ & & \ddots & \\ & & & \sigma_{N-1} \\ & & & \end{pmatrix} \qquad \sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{N-1} \geq 0$$

σ_i（$i = 0, 1, \cdots, N-1$）称为 $\mathbf{A}$ 的奇异值。以下，我们将证明任意实矩阵均可进行奇异值分解，且奇异值是被分解唯一确定的。

4.1 矩阵的奇异值分解

定理 4.1　对任意的 $\mathbf{A} \in \mathbb{R}^{M\times N}$，$\mathbf{A}$ 存在奇异值分解。

证明 $\mathbf{A}^T\mathbf{A}$ 是对称半正定矩阵，由线性代数的知识可知，存在正交矩阵 $\mathbf{V} \in \mathbb{R}^{N\times N}$，使

$$\mathbf{V}^T(\mathbf{A}^T\mathbf{A})\mathbf{V} = \text{diag}(\lambda_0, \lambda_1, \cdots, \lambda_{N-1})$$

其中 λ_j（$j = 0, 1, \cdots, N-1$）是 $\mathbf{A}^T\mathbf{A}$ 的特征值，且有

$$\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{N-1} \geq 0$$

令

$$\sigma_j = \sqrt{\lambda_j} \qquad j = 0, 1, \cdots, N-1 \tag{67}$$

设 $\mathbf{A}$ 的秩为 r，故 $\mathbf{A}^T\mathbf{A}$ 的秩也为 r，因此

$$\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{r-1} > \lambda_r = \lambda_{r+1} = \cdots = \lambda_{N-1} = 0$$
$$\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > \sigma_r = \sigma_{r+1} = \cdots = \sigma_{N-1} = 0$$

记 $\mathbf{V}$ 的第 i 列为 $\mathbf{v}_i$，并令 $\mathbf{V}_0 = (\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{r-1})$，$\mathbf{V}_1 = (\mathbf{v}_r, \mathbf{v}_{r+1}, \cdots, \mathbf{v}_{N-1})$。定义

$$\mathbf{\Sigma_0} = \text{diag}(\sigma_0, \sigma_1, \cdots, \sigma_{r-1}) \in \mathbb{R}^{r\times r} \tag{68}$$

记 $\mathbf{\Sigma} \in \mathbb{R}^{M\times N}$ 为

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma_0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

因为 $\mathbf{V}_1$ 的列是 $\mathbf{A}^T\mathbf{A}$ 对应特征值 0 的特征向量，即

$$\mathbf{A}^T\mathbf{A}\mathbf{v}_j = \mathbf{0} \qquad j = r, r+1, \cdots, N-1$$

即 $\mathbf{V}_1$ 的列构成 null$(\mathbf{A}^T\mathbf{A})$ 的正交基，所以 null$(\mathbf{A})$ = span$\{\mathbf{v}_r, \mathbf{v}_{r+1}, \cdots, \mathbf{v}_{N-1}\}$，有

$$\mathbf{A}\mathbf{V}_1 = \mathbf{0}$$

由 $\mathbf{V}$ 的正交性可知，$\mathbf{I} = \mathbf{V}\mathbf{V}^T = \mathbf{V}_0\mathbf{V}_0^T + \mathbf{V}_1\mathbf{V}_1^T$，有

$$\mathbf{A} = \mathbf{A}\mathbf{I} = \mathbf{A}\mathbf{V}_0\mathbf{V}_0^T + \mathbf{A}\mathbf{V}_1\mathbf{V}_1^T = \mathbf{A}\mathbf{V}_0\mathbf{V}_0^T \tag{69}$$

定义

$$\mathbf{u}_j = \frac{1}{\sigma_j}\mathbf{A}\mathbf{v}_j \qquad j = 0, 1, \cdots, r-1 \tag{70}$$

记 $\mathbf{U}_0 = (\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{r-1})$，于是有

$$\mathbf{A}\mathbf{V}_0 = \mathbf{U}_0\mathbf{\Sigma_0} \tag{71}$$

由（70）及 $\mathbf{V}$ 的正交性得

$$\mathbf{u}_i^T\mathbf{u}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad 0 \leq i, j \leq r-1$$

故 $\mathbf{U}_0$ 的列向量是标准正交向量组，可作为 range($\mathbf{A}$) 的一组基，range($\mathbf{A}$) = span$\{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{r-1}\}$，设其正交补 range($\mathbf{A}$)$^\perp$ 的一组标准正交基为 $\{\mathbf{u}_r, \mathbf{u}_{r+1}, \cdots, \mathbf{u}_{M-1}\}$，令 $\mathbf{U}_1 = (\mathbf{u}_r, \mathbf{u}_{r+1}, \cdots, \mathbf{u}_{M-1})$ 及 $\mathbf{U} = (\mathbf{U}_0\ \mathbf{U}_1)$，故 $\mathbf{U}$ 是正交矩阵，$\mathbf{U}$ 的列向量构成 $\mathbb{R}^M$ 的一组标准正交基。由（71）和（69）得

$$\begin{aligned}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T &= (\mathbf{U}_0\ \mathbf{U}_1)\begin{pmatrix}\mathbf{\Sigma}_0 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{V}_0^T\\ \mathbf{V}_1^T\end{pmatrix}\\ &= \mathbf{U}_0\mathbf{\Sigma}_0\mathbf{V}_0^T\\ &= \mathbf{A}\mathbf{V}_0\mathbf{V}_0^T\\ &= \mathbf{A}\end{aligned} \tag{72}$$

因此，分解是存在的，且奇异值由（67）由唯一确定（但矩阵 $\mathbf{U}$,$\mathbf{V}$ 并不唯一），定理得证。 □

上述证明出自 [7]，另外，由 $\mathbf{\Sigma_0}$ 的定义（68）及（72），我们得到 SVD 的另一种形式

$$\mathbf{A} = \mathbf{U}_0\mathbf{\Sigma_0}\mathbf{V}_0^T \tag{73}$$

这种形式称为 $\mathbf{A}$ 的瘦形奇异值分解，它有着广泛的应用。

定理 4.1 可推广到复数域的情形，但本文不打算讨论，详情可在一些数值分析教材中找到。另外，除了上述的 SVD 分解式外，还有很多的 SVD 分解形式，实际应用中应根据具体情况选用适当的形式。

4.2 由 SVD 得到的矩阵性质

定理 4.2 $\mathbf{A}$ 的秩等于其非零奇异值的个数。

证明 因为在分解式 $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ 中，$\mathbf{U}$ 和 $\mathbf{V}$ 是非奇异的，所以 rank($\mathbf{A}$) = rank($\mathbf{\Sigma}$)，$\mathbf{\Sigma}$ 的秩等于其非零对角元（奇异值）个数。 □

因此，SVD 可以计算出矩阵的秩。然而，因为计算机的精度是有限，所以在实际当中，我们需要引入数值秩的概念。在 MATLAB 中，数值秩是这样定义的：

定义 4.1 对矩阵 $\mathbf{A} \in \mathbb{R}^{M\times N}$，其大于 $\sigma_0 \max(M,N)\epsilon_M$ 的奇异值数目称为它的数值秩，其中 σ_0 是它的最大奇异值，ϵ_M 是机器精度。

在我们的 SVD 程序中，我们定义 $\epsilon_M = 10^{-16}$，函数 svd 返回上述定义的数值秩。

SVD 除了给出矩阵的秩外，还给出矩阵及其转置矩阵的值域和零空间。如果 rank($\mathbf{A}$) = r，有

定理 4.3

$$\begin{aligned}\text{range}(\mathbf{A}^T) &= \text{span}\{\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v_{r-1}}\}\\ \text{null}(\mathbf{A}) &= \text{span}\{\mathbf{v}_r, \mathbf{v}_{r+1}, \cdots, \mathbf{v}_{N-1}\}\\ \text{range}(\mathbf{A}) &= \text{span}\{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{r-1}\}\\ \text{null}(\mathbf{A}^T) &= \text{span}\{\mathbf{u}_r, \mathbf{u}_{r+1}, \cdots, \mathbf{u}_{M-1}\}\end{aligned} \tag{74}$$

证明 定理 4.1 的证明过程已给出前两个等式；另外，由线性空间的理论可知，range($\mathbf{A}^T$) = null($\mathbf{A}$)$^\perp$，null($\mathbf{A}^T$) = range($\mathbf{A}$)$^\perp$，再由 $\mathbf{U}$，$\mathbf{V}$ 的正交性即可证明后两个等式。 □

定理 4.4 若 rank($\mathbf{A}$) = r，则 $\|\mathbf{A}\|_2 = \sigma_0$，$\|\mathbf{A}\|_F = \sqrt{\sigma_0^2 + \sigma_1^2 + \cdots + \sigma_{r-1}^2}$。

证明 由 2– 范数的性质可知，$\|\mathbf{A}\|_2 = [\rho(\mathbf{A}^T\mathbf{A})]^{1/2} = \sigma_0$，其中 $\rho(\mathbf{A}^T\mathbf{A})$ 表示 $\mathbf{A}^T\mathbf{A}$ 的特征值取绝对值后的最大者，称为 $\mathbf{A}^T\mathbf{A}$ 的谱半径。

由 Frobenius 范数在正交变换下的不变性 [7]，得

$$\|\mathbf{A}\|_F = \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\|_F = \|\mathbf{\Sigma}\|_F$$

因此

$$\|\mathbf{A}\|_F = \sqrt{\sigma_0^2 + \sigma_1^2 + \cdots + \sigma_{r-1}^2}$$

□

定理 4.5 若 $\text{rank}(\mathbf{A}) = r < N$，则对任意 $\epsilon > 0$，存在列满秩矩阵 $\mathbf{A}_\epsilon \in \mathbb{R}^{M\times N}$，使

$$\|\mathbf{A} - \mathbf{A}_\epsilon\|_2 < \epsilon$$

证明 设 $\mathbf{A}$ 有奇异值分解 $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$，令 $\mathbf{A}_\epsilon = \mathbf{U}\mathbf{\Sigma}_\epsilon\mathbf{V}^T$，其中

$$\mathbf{\Sigma}_\epsilon = \text{diag}\left(\sigma_0 + \frac{\epsilon}{2}, \cdots, \sigma_{r-1} + \frac{\epsilon}{2}, \frac{\epsilon}{2}, \cdots, \frac{\epsilon}{2}\right) \in \mathbb{R}^{M\times N}$$

因此

$$\begin{aligned}\|\mathbf{A} - \mathbf{A}_\epsilon\|_2 &= \|\mathbf{U}(\mathbf{\Sigma} - \mathbf{\Sigma}_\epsilon)\mathbf{V}^T\|_2 \le \|\mathbf{U}\|_2 \|\text{diag}(-\tfrac{\epsilon}{2}, \cdots, -\tfrac{\epsilon}{2})\|_2 \|\mathbf{V}^T\|_2 \\ &= \|\text{diag}(-\tfrac{\epsilon}{2}, \cdots, -\tfrac{\epsilon}{2})\|_2 = \frac{\epsilon}{2} < \epsilon\end{aligned}$$

由定理 4.2 可知，$\text{rank}(\mathbf{A}_\epsilon) = \text{rank}(\mathbf{\Sigma}_\epsilon) = N$，故 $\mathbf{A}_\epsilon$ 是满秩。 □

若令 $\mathbf{\Sigma}_j$ 表示除第 j 行 j 列元素为 σ_j 外其余元素均为 0 的矩阵，则 $\mathbf{A}$ 的 SVD 又可写成以下外积展开形式

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{U}(\mathbf{\Sigma}_0 + \mathbf{\Sigma}_1 + \cdots + \mathbf{\Sigma}_{N-1})\mathbf{V}^T = \sum_{j=0}^{N-1} \sigma_j \mathbf{u}_i \mathbf{v}_i^T \tag{75}$$

令 $\mathbf{A}_k = \sum_{j=0}^{k-1} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$，且 $0 < k < N$，有以下定理

定理 4.6 若 $\text{rank}(\mathbf{A}) = r$，$\mathbf{A}_k$ 如上定义，$k < r$，则

$$\min_{\text{rank}(\mathbf{B})\le k} \|\mathbf{A} - \mathbf{B}\|_2 = \|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_k \tag{76}$$

证明 由 $\mathbf{A}_k$ 的构造可知，$\text{rank}(A_k) = k$。因为 $\mathbf{A} - \mathbf{A}_k$ 的最大特征值是 σ_k，由定理 4.4 可知，$\sigma_k = \|\mathbf{A} - \mathbf{A}_k\|_2$。

假设存在 $\mathbf{B}$ 满足 $\text{rank}(\mathbf{B}) \le k$，则存在一个 $N - k$ 维的子空间 $W \subseteq \text{null}(\mathbf{B})$，由 $k + 1$ 个右奇异向量 $\{\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_k\}$ 张成的空间显然是 $k + 1$ 维的，因为这两个空间的维数之和为 $(N - k) + (k + 1) > N$，所以这两个空间的交集非空，设 $\mathbf{y}$ 为其交集的一个单位向量，则

$$\|\mathbf{A} - \mathbf{B}\|_2^2 \ge \|(\mathbf{A} - \mathbf{B})\mathbf{y}\|_2^2 = \|\mathbf{A}\mathbf{y}\|_2^2 = \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{y}\|_2^2 = \|\mathbf{\Sigma}\mathbf{V}^T\mathbf{y}\|_2^2$$

因为 $\mathbf{y} \in \text{span}\{\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_k\}$，所以 $\mathbf{V}^T\mathbf{y}$ 的后 $N - k - 1$ 个分量均为 0。因此，只需考虑 $\mathbf{\Sigma}\mathbf{V}^T\mathbf{y}$ 的前 $k + 1$ 个分量，有

$$\|\mathbf{A} - \mathbf{B}\|_2^2 \ge \|\mathbf{\Sigma}\mathbf{V}^T\mathbf{y}\|_2^2 \ge \sigma_k^2 \|\mathbf{V}^T\mathbf{y}\|_2^2 = \sigma_k^2$$

故定理成立。 □

若 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 是列满秩的，即 $r = N$，由以上定理有

$$\sigma_{N-1} = \min_{\text{rank}(\mathbf{B})\le N-1} \|\mathbf{A} - \mathbf{B}\|_2$$

所以任意满足 $\|\mathbf{A} - \mathbf{B}\|_2 < \sigma_{N-1}$ 的矩阵 $\mathbf{B} \in \mathbb{R}^{M\times N}$ 必为列满秩。

对于 Frobenius 范数，我们不加证明地给出类似定理 4.6 的结果（证明可参见 [7]）。

定理 4.7　若 $\text{rank}(\mathbf{A}) = r$，$\mathbf{A}_k$ 如上定义，$k < r$，则

$$\min_{\text{rank}(\mathbf{B})\le k} \|\mathbf{A} - \mathbf{B}\|_F = \|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sigma_k^2 + \sigma_{k+1}^2 + \cdots + \sigma_{r-1}^2} \tag{77}$$

由以上的定理 4.5 和定理 4.6 可以看到，每个秩亏损的矩阵必有列满秩矩阵充分接近它，而每个列满秩矩阵附近又围满了列满秩矩阵，故列满秩矩阵的集合是 $\mathbb{R}^{M\times N}$ 上的一个开的稠密子集。对于 $M < N$ 的情形，可作类似分析。这些的分析都是从 [4] 的相关结论中推广得到的。

在实际当中，由于机器表示精度或计算模型的误差，矩阵的元素有舍入误差，我们称矩阵有扰动。若矩阵是满秩的，那么微小的扰动一般不会改变矩阵满秩的性质；但对不满秩的矩阵而言，即使是微小的扰动，也有可能使矩阵变成满秩，因为满秩矩阵是稠密的，故这时分析问题对扰动的敏感性是十分必要的。

在 §2.4 中，我们曾经不加证明地指出，若矩阵 $\mathbf{A} \in \mathbb{R}^{N\times N}$ 是非常接近非奇异的，那么在数值上，方程组 $\mathbf{Ax} = \mathbf{b}$ 求解对于原始数据的变化会非常敏感。下面我们用矩阵的 SVD 分析该问题，由展开式（75）可得

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = (\mathbf{U\Sigma V}^T)^{-1}\mathbf{b} = \mathbf{V\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \sum_{i=0}^{N-1} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i \tag{78}$$

上式表明，当 σ_i 很小时，$\mathbf{A}$ 对 $\mathbf{b}$ 的微小扰动可导致 $\mathbf{x}$ 的较大变化。因此当系数矩阵非常接近奇异时，矩阵就会有很小的奇异值，这时用 $\mathbf{LU}$ 分解方法可能因扰动而得到一个不令人满意的解。

另外，值得注意的是，我们不能以矩阵的行列式与 0 的接近程度衡量矩阵接近奇异的程度，如下列两个 N 阶方阵

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & \cdots & -1 \\ & 1 & \cdots & -1 \\ & & \ddots & \vdots \\ & & & 1 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 1 & -1 & \cdots & -1 \\ & 1 & \cdots & -1 \\ & & \ddots & \vdots \\ \frac{-1}{2^{N-2}} & & & 1 \end{pmatrix}$$

虽然 $\det(\mathbf{A}) = 1$，但当 N 充分大时，奇异矩阵 $\mathbf{B}$ 可以任意地接近 $\mathbf{A}$，因为由定理 4.7 可知

$$\sigma_{N-1} = \min_{\text{rank}(\mathbf{X})<N} \|\mathbf{A} - \mathbf{X}\|_F \le \|\mathbf{A} - \mathbf{B}\|_F = \frac{1}{2^{N-2}}$$

所以 $\mathbf{A}$ 随着 N 的增大而趋向奇异，这个经典例子可在 [5][7] 等见到。由（55）可知，在 2–范数的意义下，矩阵的条件数 $\kappa_2(\mathbf{A})$ 定义为其最大奇异值与最小奇异值的比值。如果一个矩阵的条件数无穷大，则该矩阵是奇异的；如果一个矩阵的条件数太大，或其倒数超出了机器的精度，则称该矩阵是“病态的”。对非奇异 N 阶方阵，根据定理 4.4，其条件数又可写为

$$\kappa_2(\mathbf{A}) = \frac{\sigma_0}{\sigma_{N-1}} = \|\mathbf{A}\|_2\|\mathbf{A}^{-1}\|_2$$

可见，这与之前在 §2.4 定义的条件数是一致的。

4.3 奇异值分解的计算

在前两节中，我们讨论了 SVD 的相关理论，现在讨论如何计算它。SVD 的计算涉及矩阵特征值问题的计算，而特征值问题的计算是一个十分深广的问题，本文不打算详细讨论，详细讨论可参阅 J. H. Wilkinson[1] 或 G. H. Golub 与 C. F. Van Loan[5] 等的经典著作。因此，我们在此使用了较简单的 Jacobi 方法计算特征值和特征向量，方法虽然粗糙，但旨在说明 SVD 的计算。

事实上，奇异值分解定理的证明过程是构造性的，它给出了 SVD 最直接的计算方法。首先，我们要找能使 $\mathbf{A}^T\mathbf{A}$ 对角化的正交矩阵 $\mathbf{V}$，以及 $\mathbf{A}^T\mathbf{A}$ 的特征值。因为 $\mathbf{A}^T\mathbf{A}$ 是对称矩阵，这自然使我们想到对称特征值问题的计算方法，而对称特征值问题的方法很多，我们在此只选用较简单的循环 Jacobi 方法，因为我们不想涉及太多关于特征值问题的计算。Jacobi 方法能同时计算出特征值和特征向量，所以可得到奇异值和右奇异向量矩阵 $\mathbf{V}$。然后，用（70）即可计算出左奇异向量矩阵 $\mathbf{U}$ 的子矩阵 $\mathbf{U}_0$，若有需要的话，可将 $\mathbf{U}_0$ 扩充成 $\mathbf{U}$。这样，我们就可计算矩阵的 SVD。

用 Jacobi 方法计算 SVD 是很显然的，但在实际当中，这种方法是较慢的，是二次收敛的 [5]。然而，几乎所有的数值分析或科学计算的文献仍然讨论 Jacobi 方法，因为对某些类型的矩阵，它比其他的方法能更加精确地计算奇异值和奇异向量 [6]；另外，Jacobi 方法具有并行性 [5], 这对当今的算法设计有重大的意义。因此，Jacobi 方法还是有吸引力的。我们把 Jacobi 方法封装到函数 *jacobi* 中，其具体实现请见附录。

我们把 SVD 的计算封装到函数 *svd* 中，其返回值是被分解矩阵的一个数值秩（定义 4.1），我们在 nr.h 中定义机器精度 EPSILON 为 1e−16。

```
#include “nr.h”
int svd(int m, int n, double A[m][n], double U[][m], double V[][n],
    double sv[n])
{
    int i, j, k, rank, ord[n];
    double Vt[n][n], AtA[n][n], tmp[n];
    for (i ← 0; i < n; i++) {
        for (j ← i; j < n; j++) {
            for (k ← 0, tmp[j] ← 0.0; k < m; k++)
                tmp[j] += A[k][i] * A[k][j];
        }
```

```
        for (j ← i; j < m; j++)
            AtA[j][i] ← AtA[i][j] ← tmp[j];
    }
    jacobi(n, AtA, Vt, sv); //用循环Jacobi方法计算A^T A的特征值及特征向量。
    eigsrt(n, sv, ord); //将特征值按降序排列。
    for (i ← 0; i < n; i++) //计算奇异值。
        sv[i] ← sqrt(sv[i]);
    k ← m > n ? m : n;
    rank ← 0;
    while (rank < n) {//计算定义4.1中的数值秩。
        if (sv[rank] > sv[0] * k * EPSILON)
            rank += 1;
        else
            break;
    }
    for (j ← 0; j < n; j++) {//计算V。
        for (i ← 0; i < n; i++)
            V[i][j] ← Vt[i][ord[j]];
    }
    for (j ← 0; j < rank; j++) {//用(70)式逐列计算U_0。
        for (i ← 0; i < m; i++) {
            for (k ← 0, U[i][j] ← 0.0; k < n; k++)
                U[i][j] += A[i][k] * V[k][j];
            U[i][j] ← U[i][j] / sv[j];
        }
    }
    for (j ← rank; j < m; j++) {//我们没有计算U_1，若需要的话，可修改这部分。
        for (i ← 0; i < m; i++)
            U[i][j] ← 0.0;
    }
    return rank;
}
```

应用上述程序，我们可对 5 阶 Hilbert 矩阵 $\mathbf{H}_5$ 作奇异值分解，设有分解式 $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$，保留小数点后六位有

$$\mathbf{U} = \mathbf{V} = \begin{pmatrix} 0.767855 & -0.601871 & 0.214214 & -0.047162 & 0.006172 \\ 0.445791 & 0.275913 & -0.724102 & 0.432671 & -0.116678 \\ 0.321578 & 0.424877 & -0.120453 & -0.667367 & 0.506141 \\ 0.253439 & 0.443903 & 0.309574 & -0.232999 & -0.767199 \\ 0.209823 & 0.429013 & 0.565193 & 0.557587 & 0.376264 \end{pmatrix}$$

$$\mathbf{\Sigma} = \text{diag}(1.567051 \quad 0.208534 \quad 0.011407 \quad 0.000306 \quad 0.000003)$$

因为 Hilbert 阵是对称的，所以 $\mathbf{U} = \mathbf{V}$。另外，我们可以计算出 $\kappa_2(\mathbf{H}_5) = 4.766043 \times 10^5$，可见 Hilbert 矩阵是个“病态”矩阵，事实上，其病态程度会随其阶数的增大而增大，[4] 中

列出了某些 Hilbert 阵的条件数（文中有一个笔误，10 阶 Hilbert 阵 $\mathbf{H}_{10}$ 的条件数 $\kappa_2(\mathbf{H}_{10})$ 应为 1.6025×10^{13}）。

4.4 最小二乘问题的进一步讨论

对于线性方程组

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2}$$

我们前面已讨论过系数矩阵 $\mathbf{A} \in \mathbb{R}^{N\times N}$ 为非奇异方阵和 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 为列（或行）满秩的情形，现在我们对这两种情形作进一步的讨论。

先讨论奇异方阵的情形。设 N 阶方阵有奇异值分解式 $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$，其中对角阵 $\mathbf{\Sigma}$ 为奇异方阵，设 $\text{rank}(\mathbf{A}) = r$，我们定义它的伪逆为

$$\mathbf{\Sigma}^{-1} = \text{diag}\left(\frac{1}{\sigma_0}, \cdots, \frac{1}{\sigma_{r-1}}, 0, \cdots, 0\right)$$

也就是若 $\sigma_i = 0$，则将 $1/\sigma_i$ 替换成 0。在下面的讨论中，我们将用到伪逆。

首先，对于齐次线性方程组，即 $\mathbf{b} = \mathbf{0}$，由定理 4.3 知，可用 SVD 直接得到其解，对应于奇异值为零的右奇异向量 $\{\mathbf{v}_r, \mathbf{v}_{r+1}, \cdots, \mathbf{v}_{N-1}\}$ 组成其解空间得一组基。

当右边项 $\mathbf{b} \neq \mathbf{0}$ 时，若 $\mathbf{b} \in \text{range}(\mathbf{A})$，则有解，且解不止一个，因为 $\text{null}(\mathbf{A})$ 中的任意向量都可加到 $\mathbf{x}$ 上。因此，我们希望求出其极小范数解。与非奇异的情形（78）类似，

$$\mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} = \sum_{i=0}^{r-1} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i \tag{79}$$

也是方程组的一个解，只不过这时的 $\mathbf{\Sigma}^{-1}$ 是 $\mathbf{\Sigma}$ 的伪逆。下面证明（79）是方程组的极小范数解。

证明 考虑 $\|\mathbf{x} + \mathbf{x}'\|_2$，其中 $\mathbf{x}' \in \text{null}(\mathbf{A})$，有

$$\begin{aligned}\|\mathbf{x} + \mathbf{x}'\|_2 &= \|\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} + \mathbf{x}'\|_2 \\ &= \|\mathbf{V}(\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} + \mathbf{V}^T\mathbf{x}')\|_2 \\ &= \|\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b} + \mathbf{V}^T\mathbf{x}'\|_2\end{aligned}$$

观察右边两项，因为 $\mathbf{b} \in \text{range}(\mathbf{A})$，由定理 4.3 可知，$\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}$ 的后 $N - r$ 个分量均为 0；对于第二项，由于 $\mathbf{x}' \in \text{null}(\mathbf{A})$，又由定理 4.3 可知，$\mathbf{V}^T\mathbf{x}'$ 的前 r 个分量全为 0，而后 $N - r$ 个分量不全为 0。因此，只有当 $\mathbf{x}' = \mathbf{0}$ 时，$\|\mathbf{x} + \mathbf{x}'\|_2 = \|\mathbf{x}\|_2$ 最小。 □

但如果 $\mathbf{b} \notin \text{range}(\mathbf{A})$，则方程组无解。幸好的是（79）还是可以构造出在最小二乘意义上的最近似解，我们有以下定理。

定理 4.8 对任意的 $\mathbf{b}$，（79）求得的 $\mathbf{x}$，使得残差 $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ 最小。

证明 对任意 $\mathbf{x}'$，设 $\mathbf{A}\mathbf{x}' = \mathbf{b}'$，则

$$\begin{aligned}\|\mathbf{A}(\mathbf{x} + \mathbf{x}') - \mathbf{b}\|_2 &= \|(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{b}) - \mathbf{b} + \mathbf{b}'\|_2 \\ &= \|(\mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^{-1}\mathbf{U}^T - \mathbf{I}) + \mathbf{b}'\|_2 \\ &= \|\mathbf{U}[(\mathbf{\Sigma}\mathbf{\Sigma}^{-1} - \mathbf{I})\mathbf{U}^T\mathbf{b} + \mathbf{U}^T\mathbf{b}']\|_2 \\ &= \|(\mathbf{\Sigma}\mathbf{\Sigma}^{-1} - \mathbf{I})\mathbf{U}^T\mathbf{b} + \mathbf{U}^T\mathbf{b}'\|_2\end{aligned}$$

$(\mathbf{\Sigma}\mathbf{\Sigma}^{-1}-\mathbf{I})$ 为对角阵，只有当 $\sigma_i=0$ 时，其第 i 个分量为非零；由于 $\mathbf{b}'\in\text{range}(\mathbf{A})$，故只有 $\sigma_i\neq 0$ 时，$\mathbf{U}^T\mathbf{b}'$ 的第 i 个分量为非零。因此，只有令 $\mathbf{b}'=\mathbf{0}$ 时，残差最小。 □

以上两个简洁的证明出自 [3]。由讨论可知，无论方阵奇异与否，(79) 都是极小范数解。然而，从数值上看，更一般的情况是，系数矩阵存在很小的但不为零的 σ_i，即方程组是“病态”的。这时用 **LU** 分解直接法求解，则可能给出方程组的形式解，但解向量可能有非常大的分量，当与 $\mathbf{A}$ 相乘时，由于相抵消而得到右边项 $\tilde{\mathbf{b}}$ 的一个很差的解。遇到这些情形时，将这些小的 σ_i 化为 0，再用 (79) 求解，得到的 $\mathbf{x}$ 在使残差尽可能小的意义上看，比用直接法和保留这些小 σ_i 的 SVD 解法都要好得多。但这种将小奇异值化零的 SVD 方法不能盲目使用，必须谨慎确定将小奇异值化零的阀值，且需考虑多大的计算残差可以接受 [3]。

对于非方阵的情形，我们已在 §3.6 讨论过满秩的情形，类似方阵的情形，我们可证明 (79) 仍然是最小二乘解，且是极小范数解，将小奇异值零化的 SVD 方法仍然是处理接近秩亏损情形的很好做法，详细讨论可见 [3] [5] 等。

由上述讨论可见，SVD 方法似乎是永远有效的方法，至少在理论上说是这样。但从 SVD 的计算可知，其计算量要比法方程组法或 **QR** 法等大得多，所以我们说，速度与精确性往往是矛盾的，不可能存在一种完美的算法，具体问题应具体分析。

我们把 (79) 封装成“回代”程序 *svbksb*，我们在这里使用定义 4.1 的数值秩 $\tilde{r}$ 作为零化小奇异值的阀值，即将 $\sigma_{\tilde{r}-1}$ 以后的小奇异值零化。有关数值秩的更多讨论，可参看 [5]。

```
void svbksb(int m, int n, int rank, double U[][m], double V[][n],
            double b[], double sv[], double x[])
{
    int i, j;
    double tmp[rank];
    for (i ← 0; i < rank; i++) {
        for (j ← 0, tmp[i] ← 0.0; j < m; j++)
            tmp[i] += U[j][i] * b[j];
        tmp[i] ← tmp[i] / sv[i];
    }
    for (i ← 0; i < n; i++) {
        for (j ← 0, x[i] ← 0.0; j < rank; j++)
            x[i] += V[i][j] * tmp[j];
    }
}
```

4.5 矩阵的低秩逼近

设矩阵 $\mathbf{A}$ 有奇异值分解式 $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$，它可写成外积展开式 (75)

$$\mathbf{A}=\sigma_0\mathbf{u}_0\mathbf{v}_0^T+\sigma_1\mathbf{u}_1\mathbf{v}_1^T+\cdots+\sigma_{N-1}\mathbf{u}_{N-1}\mathbf{v}_{N-1}^T$$

如果 $\mathbf{A}$ 的奇异值 σ_k 非常小或等于 0，则由定理 (4.6) 或 (4.7) 可知，仅取其外积展开式前 k 项得到的

$$\mathbf{A}_k=\sigma_0\mathbf{u}_0\mathbf{v}_0^T+\sigma_1\mathbf{u}_1\mathbf{v}_1^T+\cdots+\sigma_{k-1}\mathbf{u}_{k-1}\mathbf{v}_{k-1}^T$$

将是 $\mathbf{A}$ 很好近似。这意味仅需保存少量的 $\mathbf{U}$ 和 $\mathbf{V}$ 的列（前 k 列）就可恢复整个矩阵，且精度很高。这种使用低秩逼近压缩矩阵的技术对减少储存空间和运算量有着重要作用。

我们知道，一幅黑白的图片可用一个二维数组表示，数组元素的值表示像素点的灰度，即可用矩阵 $\mathbf{A} \in \mathbb{R}^{M\times N}$ 表示，其储存空间为 MN。但对一般图像来说，我们注意到，相邻两个像素点的灰度值是很接近的，故矩阵 $\mathbf{A}$ 的相邻行（列）的相关性很大，造成 $\mathbf{A}$ 有很多很小的奇异值，这使得运用低秩逼近压缩图像成为可能。压缩后的图像用 $\mathbf{A}_k$ 表示，这时储存空间变为 $k(M+N+1)$。当然，我们是在保证图像质量可满足要求的前提下选择 k 的，不同的图片可能有不同的 k 值，灰度值变化小的图像显然可选取较小的 k，因为此时的图像较为单调，信息量少，故压缩效果较佳。

在某些情况中，储存空间的减少意味着运算量的减少。一个显然的例子是，用近似的矩阵 $\mathbf{A}_k$ 左乘向量 $\mathbf{x}$ 是非常高效的，可将先 $\mathbf{x}$ 与 $\mathbf{V}$ 的列作点乘

$$\mathbf{v}_i^T\mathbf{x} \qquad i = 0, 1\cdots, k-1$$

并乘以相应的奇异值，然后分别乘以相应的 $\mathbf{U}$ 的列，并求累加和即可。这时，用近似矩阵计算 $\mathbf{Ax}$ 仅需 $k(M+N)$ 次乘法，而不是原来的 MN 次。

当在 Google 上搜索一篇文章时，若搜索引擎发现有该文章的网站很多时，它会自动隐藏相同的结果，这是因为 Google 事先对所有文章作好了分类，而这些分类都是自动的。文章的相关性判断可归结为向量相关性的判断，向量的分量是某一词语在文章中的加权词频，我们可以用余弦定理计算向量间的夹角余弦值，从而判断其相关性 [8]，这个过程需要用到向量的点乘运算。现把数以亿计的文章构成一个巨大的矩阵 $\mathbf{A}$，该矩阵的每一行代表一篇文章，每一列对应一个词，其元素的值为加权词频。当要将一篇文章（设其对应向量为 $\mathbf{x}$）与其余所有文章比较时，我们需要计算 $\mathbf{Ax}$，这时我们就想到用低秩逼近的技巧降低运算。

问题是，若希望低秩逼近有较好的效果，则需要矩阵 $\mathbf{A}$ 有较多很小的奇异值，即文章的相关性较大，所以需要事先对文章作大致的分类，然后将一些可能相关的文章构成矩阵，再用低秩逼近技术，于是效果就有望提高。为此，我们可在开始时先用较少的词汇对文章作大的分类，然后对每个子类再选择与其有关的更多词汇作进一步的分类，如此类推，我们便可作出越来越细致的分类，因为得益于前面的分类，低秩逼近技术的效果会逐渐显现。同时，注意到问题不断被分解为子问题，因此并行处理也成为可能。

当然，上述方法只是本人的一个简单猜测，Google 对此应该会有更高明的办法，但 [9] 并无透露更多的技术细节，该文还提到，Google 内部使用的并行 SVD 算法是 Google 中国的张智威博士和几个中国的工程师及实习生设计的。

5. 结语

从上面的讨论可以知道，矩阵 $\mathbf{A}$ 的分解是把 $\mathbf{A}$ 表示成几个较简单的矩阵之积，它使所讨论的问题变得容易求解，这是种思想在数值算法设计中尤为重要。另外，这些矩阵分解方法均有其对应的并行算法，这对当今算法设计有着重大的意义，关于这方面的讨论可参看 [5]。

我们总结并对比一下本文提及的矩阵分解方法，列成表 1：

表 1 矩阵分解方法的对比

分解方法	适用矩阵	分解式	求解线性系统	计算复杂性
LU 分解	$\mathbf{A} \in \mathbb{R}^{N\times N}$ 且顺序主子式均不等于 0	$\mathbf{A}=\mathbf{LU}$	以适用矩阵为系数阵的良态线性方程组	$O(N^3)$
选主元的 **LU** 分解	$\mathbf{A} \in \mathbb{R}^{N\times N}$ 且 $\mathbf{A}$ 非奇异	$\mathbf{PA}=\mathbf{LU}$	非奇异的良态线性方程组	$O(N^3)$
QR 分解	$\mathbf{A} \in \mathbb{R}^{M\times N}$	$\mathbf{A}=\mathbf{QR}$	非奇异线性方程组，满秩（且不接近秩亏）的最小二乘问题	$O(M^2N)$
选主列的 **QR** 分解	$\mathbf{A} \in \mathbb{R}^{M\times N}$	$\mathbf{A\Pi}=\mathbf{QR}$	除接近秩亏外所有线性方程组和最小二乘问题	$O(M^2r+MN)$
SVD	$\mathbf{A} \in \mathbb{R}^{M\times N}$	$\mathbf{A}=\mathbf{U\Sigma V}^T$	所有线性方程组和最小二乘问题	迭代法

由上表可见，从上到下的矩阵分解方法对线性系统的求解能力越来越强，但是计算复杂性也越来越大，这种矛盾性在工程科学中经常会遇到，完美的算法似乎是不存在的，我们总是要视问题的具体情况来选择适当的算法进行求解。我们不知道上述的论断是否必然，但至少大量的实践经验告诉我们，分析问题的特性还是有好处的，如对某些特殊的矩阵，我们有特殊的分解：

表 2 特殊矩阵的分解

特殊矩阵	分解方法	计算复杂性
三对角矩阵	**LU** 分解	$O(N)$
对称正定矩阵	**LU** 分解	$O(\frac{1}{6}N^3)$
Hessenberg 矩阵	**QR** 分解	$O(N^2)$

根据这些特殊矩阵的特殊性质，我们大大节省了计算的工作量，所以我们更乐意处理这些矩阵。将一般矩阵化为特殊矩阵，从而减少储存空间或计算量是算法设计中经常用到的技巧，至于如何转化，通常我们又会回到矩阵分解的问题上，例如通过正交相似变换把一般矩阵约化为 Hessenberg 矩阵等。总之，具体问题要具体分析。

有关矩阵分解的课题，还有很多精彩的讨论，本文只是抛砖引玉，旨在引起正在学习数值线性代数的同学的兴趣，更多的算法讨论可参阅 Golub 和 Van Loan 的著作 [5]，面向对象的数值算法实现可参阅 William H. Press 等人的著作 [3]。

参考文献：

[1] Wilkinson. J. H. The Algebraic Eigenvalue problem[M]. Oxford: Oxford University Press, 1965. 石钟慈, 邓健新译. 代数特征值问题 [M]. 北京: 科学出版社, 2001.

[2] J. H. Wilkinson. Rounding Errors In Algebraic Processes[M]. New York: Dover, 1963. 黄开斌译. 代数过程的舍入误差 [M]. 北京: 人民教育出版社, 1982.

[3] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. Numerical Recipes in C++: The Art of Scientific Computing[M], 3rd ed. London: Cambridge University Press, 2007.

[4] 关治, 陆金甫编. 数值分析基础 [M]. 北京: 高等教育出版社, 1998.

[5] G. H. Golub, C. F. Van Loan. Matrix Computation[M], 3rd ed. Baltimore: Hopkins University Press, 1996. 袁亚湘等译. 矩阵计算 [M]. 北京: 科学出版社, 2001.

[6] James W. Demmel. Applied Numerical Linear Algebra[M]. SIAM, 1997. 王国荣译. 应用数值线性代数 [M]. 北京: 人民邮电出版社, 2007: 104, 210.

[7] Steven J. Leon. Linear Algebra with Applications[M], 7th ed. Prentice Hall, 2006.

[8] 吴军. 余弦定理和新闻的分类 [OL]. Google 黑板报, 数学之美系列十二. http://googlechinablog.com/2006/07/12.html.

[9] 吴军. 矩阵运算和文本处理中的分类问题 [OL]. Google 黑板报, 数学之美系列十八. http://www.googlechinablog.com/2007/01/blog-post.html.

附录 A　nr.h

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define EPSILON 1e-16
#define MAXNUM 65535

void swap_doub(double *a, double *b)
{//交换两个浮点数的值。
    double tmp;
    tmp ← *a;
    *a ← *b;
    *b ← tmp;
}

void swap_int(int *a, int *b)
{//交换两个整形数的值。
    int tmp;
    tmp ← *a;
    *a ← *b;
    *b ← tmp;
}
```

```
int fabs_max(int n, double array[])
{//求n维数组中绝对值最大的元素的指标。
    int i, m ← 0;
    double tmp ← fabs(array[0]);
    for (i ← 1; i < n; i++) {
        if (fabs(array[i]) > tmp) {
            tmp ← fabs(array[i]);
            m ← i;
        }
    }
    return m;
}

int max(int n, double array[])
{//求n维数组中最大的元素的指标。
    int i, m ← 0;
    double tmp ← array[0];
    for (i ← 1; i < n; i++) {
        if (array[i] > tmp) {
            tmp ← array[i];
            m ← i;
        }
    }
    return m;
}

void init_array(int n, int array[])
{//初始化数组为{0,1,···,n−1}。
    int i;
    for (i ← 0; i < n; i++)
        array[i] ← i;
}

void init_matrix(int n, double A[][n])
{//将矩阵的元素初始化为0。
    int i, j;
    for (i ← 0; i < n; i++) {
        for (j ← 0; j < n; A[i][j] ← 0.0, j++);
    }
}

void identy_matrix(int n, double A[][n])
{//将矩阵初始化为单位矩阵。
    int i;
    init_matrix(n, A);
```

```
    for (i ← 0; i < n; A[i][i] ← 1.0, i++);
}

double xxT(int n, double x[])
{//对n维向量，求x^T x的值。
    int i;
    double sum;
    for (i ← 0, sum ← 0.0; i < n; i++)
        sum += x[i] * x[i];
    return sum;
}

double norm2(int n, double x[])
{//求n维向量的2–范数。
    return sqrt(xxT(n,x));
}

void transpose(int m, int n, double A[m][n], double At[n][m])
{//求矩阵A的转置A^T，保存在At上。
    int i, j;
    for (i ← 0; i < n; i++)
        for (j ← 0; j < m; j++)
            At[i][j] ← A[j][i];
}

void jacobi(int n, double A[][n], double V[][n], double ev[])
{//用循环Jacobi方法求解矩阵特征值及对应的特征向量。
    int i, j, k, noi ← 0;
    double tau, c, s, t, off ← 0.0, alpha ← 0.0, tmp1, tmp2;
    identy_matrix(n, V);
    for (i ← 0; i < n − 1; i++) {
        for (j ← i + 1; j < n; j++) {
            off += 2 * A[i][j] * A[i][j];
        }
    }
    alpha ← sqrt(off);
    while (off > EPSILON ∧ noi < MAXNUM) {
        alpha ← alpha / n;
        for (i ← 0; i < n − 1; i++) {
            for (j ← i + 1; j < n; j++) {
                if (fabs(A[i][j]) ≥ alpha) {
                    tau ← (A[i][i] − A[j][j]) / (2 * A[i][j]);
                    if (tau ≥ 0)
                        t ← 1 / (fabs(tau) + sqrt(1 + tau * tau));
                    else
```

```
                    t ← −1 / (fabs(tau) + sqrt(1 + tau ∗ tau));
                c ← 1 / sqrt(1 + t ∗ t);
                s ← t ∗ c;
                for (k ← 0; k < n; k++) {
                    if (k ≠ i ∧ k ≠ j) {
                        tmp1 ← c ∗ A[k][i] + s ∗ A[k][j];
                        tmp2 ← c ∗ A[k][j] − s ∗ A[k][i];
                        A[k][i] ← A[i][k] ← tmp1;
                        A[k][j] ← A[j][k] ← tmp2;
                    }
                }
                A[i][i] ← A[i][i] + t ∗ A[i][j];
                A[j][j] ← A[j][j] − t ∗ A[i][j];
                off −= 2 ∗ A[i][j] ∗ A[i][j];
                A[i][j] ← A[j][i] ← 0.0;
                for (k ← 0; k < n; k++) {
                    tmp1 ← c ∗ V[k][i] + s ∗ V[k][j];
                    tmp2 ← −s ∗ V[k][i] + c ∗ V[k][j];
                    V[k][i] ← tmp1;
                    V[k][j] ← tmp2;
                }
            }
        }
    }
    noi += 1;
}
    if (noi ≡ MAXNUM)
        printf("So Many Iterations!\n");
for (i ← 0; i < n; i++)
    ev[i] ← A[i][i];
}

void eigsrt(int n, double ev[], int ord[])
{//将数组ev的元素由大到小排序。
    int i, m;
    init_array(n, ord);
    for (i ← 0; i < n − 1; i++) {
        m ← max(n − i, &ev[i]);
        swap_int(&ord[i], &ord[m + i]);
        swap_doub(&ev[i], &ev[m + i]);
    }
}
```

致谢: 感谢指导老师林福荣教授以及其他对本文提供建议的同学!