



北京化工大学

Beijing University of Chemical Technology

计算方法讲义 (五)

线性方程组

Cheng Yong

目录

第 1 章 线性方程组	1
1.1 高斯消元法	2
1.1.1 一般高斯消元法	2
1.1.2 列主元高斯消去法	7
1.2 矩阵分解法	11
1.2.1 杜利特尔 (Doolittle) 分解	12
1.2.2 克劳特 (Crout) 分解	15
1.2.3 平方根法分解	21
1.2.4 Cholesky 分解	21
1.3 追赶法	24
1.4 向量和矩阵的范数	29
1.5 迭代法	33
1.5.1 雅克比 (Jacobi) 迭代法	33
1.5.2 高斯-赛得尔迭代法	38
1.5.3 超松弛迭代技术	42

1.5.4 迭代法的收敛性	43
1.6 本章小结	44

创建日期: 2019 年 7 月 5 日
更新日期: 2020 年 2 月 11 日

第 1 章 线性方程组

在工程技术、自然科学和社会科学中，经常遇到的许多问题最终都可归结为解线性方程组，如电学中网络问题、用最小二乘法求实验数据的曲线拟合问题，工程中的三次样条函数的插值问题，经济运行中的投入产出问题以及大地测量、机械与建筑结构的设计计算问题等等，都归结为求解线性方程组或非线性方程组的数学问题。因此线性方程组的求解对于实际问题是极其重要的。

实际问题中的线性方程组分类：

- 按系数矩阵中零元素的个数 (稠密线性方程组和稀疏线性方程组);
- 按未知量的个数 (高阶线性方程组和低阶线性方程组);
- 按系数矩阵的形状 (对称正定方程组、三角形方程组和三对角方程组);

一般形式的 n 阶线性方程组为：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (1.1)$$

写成矩阵形式为：

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (1.2)$$

其中

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (1.3)$$

当系数矩阵 \mathbf{A} 非奇异 (即 $\det(\mathbf{A}) \neq 0$) 时，方程组有惟一解。

线性方程组的直接解法一般包括：

- Gauss 消元法
 1. Gauss 消元法;
 2. Gauss 列主元消元法;
- 矩阵分解法
 1. 杜利特尔 (Doolittle) 分解;

- 2. 克劳特 (Crout) 分解;
- 3. 乔累斯基 (Cholesky) 分解;
- 追赶法

1.1 高斯消元法

1.1.1 一般高斯消元法

高斯消元法将原方程组的增广矩阵通过初等变换，即：

$$\bar{\mathbf{A}} = (\mathbf{A}, \mathbf{b}) \Rightarrow (\mathbf{A}^{(1)}, \mathbf{b}^{(1)}) \Rightarrow \cdots \Rightarrow (\mathbf{A}^{(n)}, \mathbf{b}^{(n)}) \quad (1.4)$$

其中 \mathbf{A}^n 为上三角矩阵。

不难得到，方程组

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (1.5)$$

和

$$\mathbf{A}^n\mathbf{x} = \mathbf{b}^n \quad (1.6)$$

同解，以上求解线性方程组的方法称为 Gauss 消元法。

下三角形线性方程组求解

$$\mathbf{L}\mathbf{x} = \mathbf{b} \Rightarrow \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad (1.7)$$

通过从上到下代入，可以得到其解为：

$$\begin{cases} x_1 = \frac{b_1}{l_{11}} \\ x_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij}x_j}{l_{ii}} \quad i = 2, 3, 4, \dots, n \end{cases} \quad (1.8)$$

上三角形线性方程组求解

$$\mathbf{U}\mathbf{x} = \mathbf{b} \Rightarrow \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad (1.9)$$

通过从下到上代入，可以得到其解为：

$$\begin{cases} x_n = \frac{b_n}{u_{nn}} \\ x_i = \frac{b_i - \sum_{j=i+1}^n u_{ij}x_j}{u_{ii}} \quad i = n-1, n-2, n-3, \dots, 2, 1 \end{cases} \quad (1.10)$$

对于线性方程组

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (1.11)$$

如果 $\det(\mathbf{A}) \neq 0$ ，对其增广矩阵进行初等变换：

$$\bar{\mathbf{A}} = (\mathbf{A}, \mathbf{b}) = (\mathbf{A}^{(1)}, \mathbf{b}^{(1)}) = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{pmatrix} \quad (1.12)$$

如果 $a_{11}^{(1)} \neq 0$ ，定义行乘子

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \quad i = 2, 3, 4, \dots, n \quad (1.13)$$

然后第 i 行减去第 1 行 $\times m_{i1}$ ，有：

$$a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)} \quad i = 2, 3, \dots, n, j = 1, 2, 3, \dots, n \quad (1.14)$$

$$b_i^{(2)} = b_i^{(1)} - m_{i1}b_1^{(1)} \quad i = 2, 3, \dots, n \quad (1.15)$$

得到变换后的矩阵为：

$$(\mathbf{A}^{(1)}, \mathbf{b}^{(1)}) \Rightarrow (\mathbf{A}^{(2)}, \mathbf{b}^{(2)}) = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{pmatrix} \quad (1.16)$$

如果 $a_{11}^{(1)} = 0$ ，由于 $\det(\mathbf{A}) \neq 0$ ，因此第一列中至少有一个元素不为零。如果有 $a_{i_11}^{(1)} \neq 0$ ，则将增广矩阵 $(\mathbf{A}^{(1)}, \mathbf{b}^{(1)})$ 的第 1 行和第 i_1 行进行交换后再进行消元。

依次类推，经过 $n-1$ 步后， $(\mathbf{A}^{(1)}, \mathbf{b}^{(1)})$ 将变换为如下形式：

$$(\mathbf{A}^{(1)}, \mathbf{b}^{(1)}) \Rightarrow (\mathbf{A}^{(n)}, \mathbf{b}^{(n)}) = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(n)} & b_n^{(n)} \end{pmatrix} \quad (1.17)$$

由于 $\det(\mathbf{A}) \neq 0$, 因此, 上三角矩阵 $\mathbf{A}^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$ 有唯一解。
因此可得到线性方程组

$$\mathbf{Ax} = \mathbf{b} \quad (1.18)$$

的解。即:

$$\begin{cases} x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \\ x_i = \frac{b_i^{(i)} - \sum_{j=i+1}^n a_{ij}^{(i)} x_j}{a_{ii}^{(i)}} \quad i = n-1, n-2, n-3, \dots, 2, 1 \end{cases} \quad (1.19)$$

例 1. 用 Gauss 消元法求解方程组:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 6 \\ 2x_1 + 3x_2 + 4x_3 = 9 \\ x_1 + 3x_2 + 2x_3 = 6 \end{cases} \quad (1.20)$$

解: $n = 3, a_{11} = 1 \neq 0$,

$$m_{21} = a_{21}/a_{11} = 2/1 = 2 \quad (1.21)$$

$$m_{31} = a_{31}/a_{11} = 1/1 = 1 \quad (1.22)$$

将 $i(i=2, 3)$ 个方程减去 $m_{i1} \times$ 第 1 个方程, 完成第一步消元, 得到:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 6 \\ -x_2 - 2x_3 = -3 \\ x_2 - x_3 = 0 \end{cases} \quad (1.23)$$

此时 $a_{22}^{(1)} = -1 \neq 0, m_{32} = a_{32}^{(1)}/a_{22}^{(1)} = 1/-1 = -1$,

将 $i(i=3)$ 个方程减去 $m_{32} \times$ 第 2 个方程, 完成第二步消元, 得到:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 6 \\ -x_2 - 2x_3 = -3 \\ -3x_3 = -3 \end{cases} \quad (1.24)$$

通过以下回代可求得方程的解:

$$x_3 = -3/-3 = 1 \quad (1.25)$$

$$x_2 = -(-3 + 2x_3) = -(-3 + 2 \times 1) = 1 \quad (1.26)$$

$$x_1 = 6 - 2x_2 - 3x_3 = 6 - 2 \times 1 - 3 \times 1 = 1 \quad (1.27)$$

故求得的方程解为: $x_1 = x_2 = x_3 = 1$ 。

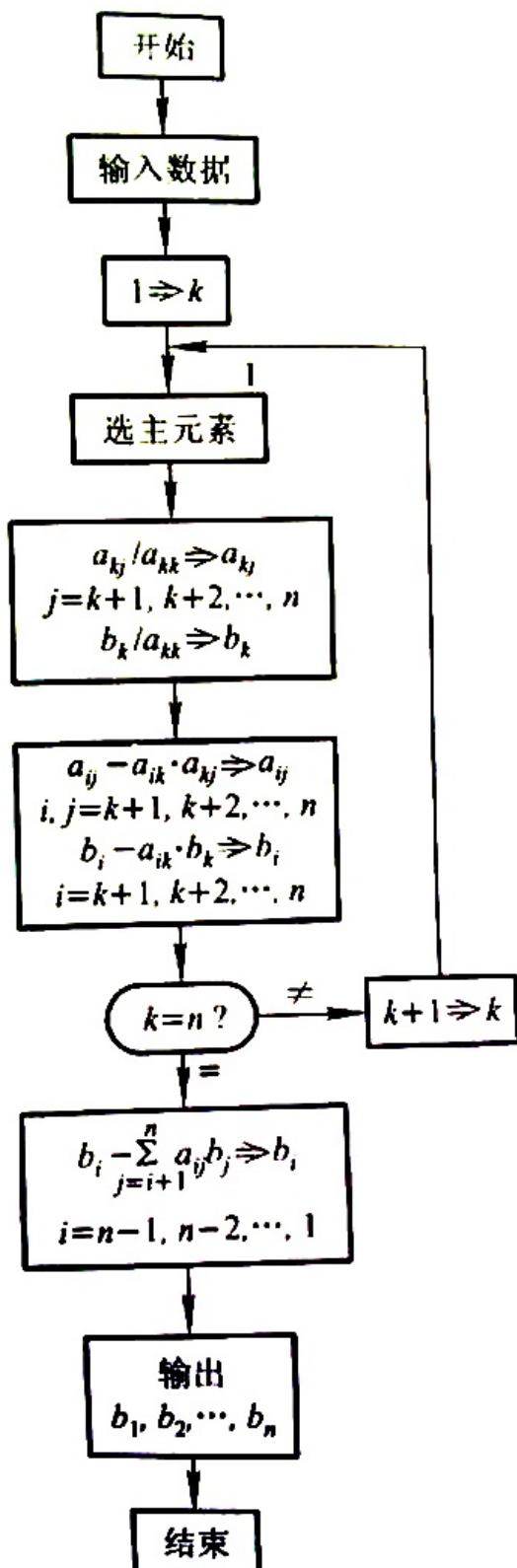


图 1.1: Gauss 消去法

```
1 #include <stdio.h>
```

```

2 #include <math.h>
3 #define MAXSIZE 50
4 void input(double a[MAXSIZE][MAXSIZE+1],long n);
5 void output(double x[MAXSIZE],long n);
6 void main(void) {
7     double a[MAXSIZE][MAXSIZE+1],x[MAXSIZE],s;
8     long n,i,j,k;
9     printf("\n请输入原方程组的阶数: ");
10    scanf("%ld",&n);
11    input(a,n);
12    for(k=0; k<=n-2; k++) {
13        for(i=k+1; i<=n-1; i++) {
14            a[i][k]/=-a[k][k];
15            for(j=k+1; j<=n; j++)
16                a[i][j]+=a[i][k]*a[k][j];
17        }
18        for(k=n-1; k>=0; k--) {
19            s=0;
20            for(j=k+1; j<=n-1; j++) s+=a[k][j]*x[j];
21            x[k]=(a[k][n]-s)/a[k][k];
22        }
23        output(x,n);
24    }
25
26 void input(double a[MAXSIZE][MAXSIZE+1],long n) {
27     long i,j;
28     printf("\n请输入原方程组的增广矩阵: \n");
29     for(i=1; i<=n; i++) {
30         for(j=1; j<=n+1; j++)
31             scanf("%lf",&a[i-1][j-1]);
32     }
33 void output(double x[MAXSIZE],long n) {
34     long k;
35     printf("\n原方程组的解为: \n");
36     for(k=1; k<=n; k++)
37         printf(" %lf",x[k-1]);
38 }
```

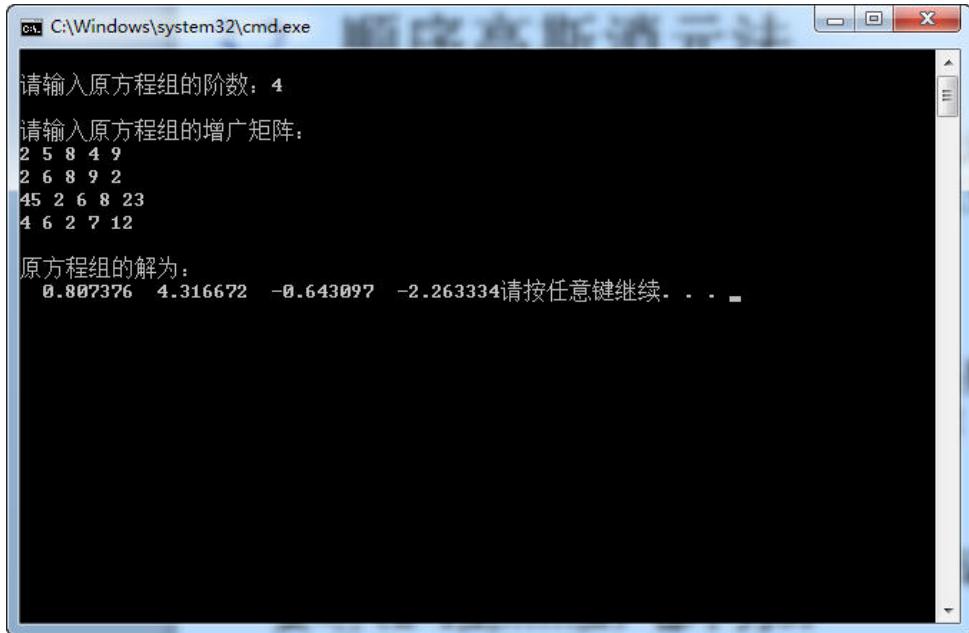


图 1.2: 高斯消去法运行结果

1.1.2 列主元高斯消去法

例 2. 用 *Gauss* 消去法解线性方程组 (用 3 位十进制浮点数计算)。

$$\begin{pmatrix} 10^{-8} & 2 & 3 \\ -1 & 3.712 & 4.623 \\ -2 & 1.072 & 5.643 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad (1.28)$$

解: 上述方程组和例 1 一样若用 *Gauss* 消去法计算会有小数作除数的现象, 若采用换行的技巧, 则可避免。

$$\bar{\mathbf{A}} = (\mathbf{A}, \mathbf{b}) = \begin{pmatrix} 10^{-8} & 2 & 3 & 1 \\ -1 & 3.712 & 4.623 & 2 \\ -2 & 1.072 & 5.643 & 3 \end{pmatrix} \quad (1.29)$$

因为 10^{-8} 很小, 绝对值最大的列元素为 $a_{13} = -2$, 因此可以将 1,3 行交换, 变换后结果为:

$$(\mathbf{A}^{(1)}, \mathbf{b}^{(1)}) = \begin{pmatrix} -2 & 1.072 & 5.643 & 3 \\ -1 & 3.712 & 4.623 & 2 \\ 10^{-8} & 2 & 3 & 1 \end{pmatrix} \quad (1.30)$$

求得 $m_{21} = 0.5, m_{31} = -0.5 * 10^{-8}$, 然后第一步消元, 结果如下:

$$(\mathbf{A}^{(2)}, \mathbf{b}^{(2)}) = \begin{pmatrix} -2 & 1.072 & 5.643 & 3 \\ 0 & 3.176 & 1.8015 & 0.5 \\ 0 & 2 & 3 & 1 \end{pmatrix} \quad (1.31)$$

这时 3.176 绝对值最大，不需要换行。取 $m_{32} = 0.62972292$ ，第二步消元，结果如下：

$$(\mathbf{A}^{(3)}, \mathbf{b}^{(3)}) = \begin{pmatrix} -2 & 1.072 & 5.643 & 3 \\ 0 & 3.176 & 1.8015 & 0.5 \\ 0 & 0 & 1.8655541 & 0.68513854 \end{pmatrix} \quad (1.32)$$

再回代，计算方程的解为：

$$x_3 = \frac{b_3^{(3)}}{a_{33}^{(3)}} = \frac{0.68513854}{1.8655541} = 0.36725739 \quad (1.33)$$

$$x_2 = \frac{b_2^{(2)} - a_{23}^{(2)}x_3}{a_{22}^{(2)}} = \frac{0.5 - 1.8015 \times x_3}{3.176} = -0.05088607 \quad (1.34)$$

$$x_1 = \frac{b_1^{(1)} - a_{12}^{(1)}x_2 - a_{13}^{(1)}x_3}{a_{11}^{(1)}} = -0.49105820 \quad (1.35)$$

方程的准确解为： $(-0.491058227, -0.050886075, 0.367257384)^\top$

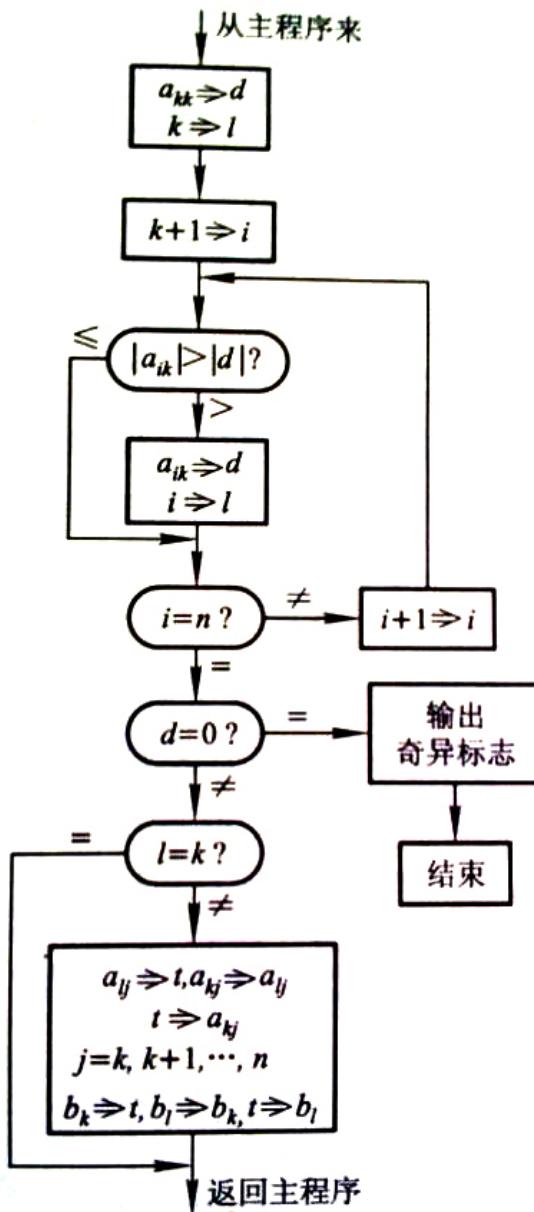


图 1.3: 列主元高斯消去法主元选取过程

```

1 #include<iostream>
2 #include<math.h>
3
4 using namespace std;
5
6 float *colPivot( float *c, int n ) {
7     int i,j,t,k;
8     float *x,p;
9     x=new float[n*sizeof(float)];
10    for( i=0; i<=n-2; i++ ) {
11        k=i;
12        for(j=i+1; j<=n-1; j++)

```

```

13         if(fabs(*(c+j*(n+1)+i))>(fabs(*(c+k*(n+1)+i))))
14             k=j;
15         if(k!=i)
16             for( j=i; j<=n; j++ ) {
17                 p=*(c+i*(n+1)+j);
18                 *(c+i*(n+1)+j)=*(c+k*(n+1)+j);
19                 *(c+k*(n+1)+j)=p;
20             }
21         for( j=i+1; j<=n-1; j++ ) {
22             p=(*(c+j*(n+1)+i))/(*(c+i*(n+1)+i));
23             for( t=i; t<=n; t++ )
24                 *(c+j*(n+1)+t)-=p*(*(c+i*(n+1)+t));
25         }
26     }
27     for( i=n-1; i>=0; i-- ) {
28         for( j=n-1; j>=i+1; j-- )
29             (*(c+i*(n+1)+n))-=x[j]*(*(c+i*(n+1)+j));
30         x[i]=*(c+i*(n+1)+n)/(*(c+i*(n+1)+i));
31     }
32     return x;
33 }
34
35 int main() {
36     int i;
37     float *x;
38     float c[3][4] = {0.101,2.304,3.555,1.183,
39                      -1.347,3.712,4.623,2.137,
40                      -2.835,1.072,5.643,3.035
41      };
42     float *ColPivot(float *,int);
43     x=colPivot(c[0],3);
44     for( i=0; i<=2; i++ )
45         cout<<"x("<<i<<")="<<x[i]<<endl;
46     return 0;
47 }
```

```
x<0>=-0.398234
x<1>=0.0137951
x<2>=0.335144
请按任意键继续...
```

图 1.4: 列主元高斯消元法运行结果

1.2 矩阵分解法

如果将线性方程组

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (1.36)$$

的系数矩阵 \mathbf{A} 分解为两个三角形矩阵 \mathbf{L} 和 \mathbf{U} 的乘积, 即:

$$\mathbf{A} = \mathbf{L}\mathbf{U} \quad (1.37)$$

则

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (1.38)$$

$$\mathbf{L}\mathbf{U}\mathbf{x} = \mathbf{b} \quad (1.39)$$

$$\mathbf{L}\mathbf{y} = \mathbf{b} \quad \mathbf{U}\mathbf{x} = \mathbf{y} \quad (1.40)$$

因为 \mathbf{L} 和 \mathbf{U} 都是三角形矩阵, 因此可以通过回代直接求解, 这种方法我们称为求解线性方程组的三角形分解法, 简称矩阵分解法。

定义 1. 定义 若方阵 \mathbf{A} 可以分解为一个下三角矩阵 \mathbf{L} 和一个上三角矩阵 \mathbf{U} 的乘积, 即 $\mathbf{A} = \mathbf{L}\mathbf{U}$, 则这种分解称为 \mathbf{A} 的一种三角分解或 LU 分解。如果 \mathbf{L} 为单位下三角矩阵, 则称为杜利特尔 (*Doolittle*) 分解; 若 \mathbf{U} 为单位上三角矩阵, 则称为克劳特 (*Cout*) 分解。

定理 1. 矩阵的 LU 分解定理 设 A 为 n 阶方阵, 如果 A 的顺序主子矩阵 A_1, A_2, \dots, A_{n-1} 均非奇异, 则 A 可分解为一个单位下三角矩阵 L 和一个上三角矩阵 U 的乘积, 即 $A = LU$, 且这种分解是唯一的。

1.2.1 杜利特尔 (Doolittle) 分解

根据上述定理, 如果 n 阶方阵 A 的各阶顺序主子式不为 0, 则存在唯一的 LU 分解。现考察 4 阶方阵, 有:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \quad (1.41)$$

右边两个矩阵相乘, 可得到如下等式:

$$a_{11} = u_{11}, a_{12} = u_{12}, a_{13} = u_{13}, a_{14} = u_{14} \quad (1.42)$$

$$a_{21} = l_{21}u_{11}, a_{22} = l_{21}u_{12} + u_{22}, a_{23} = l_{21}u_{13} + u_{23}, a_{24} = l_{21}u_{14} + u_{24} \quad (1.43)$$

$$a_{31} = l_{31}u_{11}, a_{32} = l_{31}u_{12} + l_{32}u_{22}, a_{33} = l_{31}u_{13} + l_{32}u_{23} + u_{33} \quad (1.44)$$

$$a_{34} = l_{31}u_{14} + l_{32}u_{24} + u_{34}, a_{41} = l_{41}u_{11}, a_{42} = l_{41}u_{12} + l_{42}u_{22} \quad (1.45)$$

$$a_{43} = l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33}, a_{44} = l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + u_{44} \quad (1.46)$$

对于上述等式, 我们可以按行设定计算顺序:

$$u_{11}, u_{12}, u_{13}, u_{14} \rightarrow l_{21} \rightarrow u_{22}, u_{23}, u_{24} \quad (1.47)$$

$$\rightarrow l_{31}, l_{32} \rightarrow u_{33}, u_{34} \rightarrow l_{41}, l_{42}, l_{43} \rightarrow u_{44} \quad (1.48)$$

很容易导出显示的计算公式:

$$u_{11} = a_{11}, u_{12} = a_{12}, u_{13} = a_{13}, u_{14} = a_{14}, \quad (1.49)$$

$$l_{21} = a_{21}/u_{11}, u_{22} = a_{22} - l_{21}u_{12}, \quad (1.50)$$

$$u_{23} = a_{23} - l_{21}u_{13}, u_{24} = a_{24} - l_{21}u_{14} \quad (1.51)$$

$$l_{31} = a_{31}/u_{11}, l_{32} = (a_{32} - l_{31}u_{12})/u_{22}, u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} \quad (1.52)$$

$$u_{34} = a_{34} - l_{31}u_{14} - l_{32}u_{24}, l_{41} = a_{41}/u_{11}, l_{42} = (a_{42} - l_{41}u_{12})/u_{22} \quad (1.53)$$

$$l_{43} = (a_{43} - l_{41}u_{13} - l_{42}u_{23})/u_{33}, \quad (1.54)$$

$$u_{44} = a_{44} - l_{41}u_{14} - l_{42}u_{24} - l_{43}u_{34} \quad (1.55)$$

可以进一步将上述结果推广到一般的 n 阶方阵，将 A 进行杜利特尔 (Doolittle) 分解，可以得到一个单位下三角矩阵 L 和一个上三角矩阵 U 的乘积。

其中 L 和 U 的计算公式为：

$$\begin{cases} u_{1j} = a_{1j}, & j = 1, 2, 3, \dots, n \\ l_{i1} = \frac{a_{i1}}{u_{11}}, & i = 2, 3, 4, \dots, n \\ u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, & j = i, i+1, \dots, n \\ l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}}{u_{jj}}, & j = 1, 2, \dots, i-1 \end{cases} \quad (1.56)$$

矩阵分解法有数值不稳定性。当 $u_{kk} = 0$ 时，三角分解计算中断；当 $u_{kk} \neq 0$ ，但 $|u_{kk}|$ 很小时，计算 l_{ik} 可能引起较大的舍入误差，使得计算结果不可靠。此时可采用与 Gauss 列主元消去法类似的方法，先选取列主元再进行三角分解。实际上，Doolittle 列主元分解法和 Gauss 列主元消去法在理论上是等价的。

例 3. 用 Doolittle 分解求解下列线性方程组。

$$\begin{pmatrix} 2 & 10 & 0 & -3 \\ -3 & -4 & -12 & 13 \\ 1 & 2 & 3 & -4 \\ 4 & 14 & 9 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 10 \\ 5 \\ -2 \\ 7 \end{pmatrix} \quad (1.57)$$

解，由 Doolittle 分解公式，可得到：

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1.5 & 1 & 0 & 0 \\ 0.5 & -\frac{3}{11} & 1 & 0 \\ 2 & -\frac{6}{11} & -9 & 1 \end{pmatrix} \begin{pmatrix} 2 & 10 & 0 & -3 \\ 0 & 11 & -12 & 8.5 \\ 0 & 0 & -\frac{3}{11} & -\frac{2}{11} \\ 0 & 0 & 0 & -4 \end{pmatrix} \quad (1.58)$$

因此有： $Ly = b$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1.5 & 1 & 0 & 0 \\ 0.5 & -\frac{3}{11} & 1 & 0 \\ 2 & -\frac{6}{11} & -9 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 10 \\ 5 \\ -2 \\ 7 \end{pmatrix} \quad (1.59)$$

得到 $y = (10 \quad 20 \quad -\frac{17}{11} \quad -16)^T$ 。

又有 $Ux = y$,

$$\begin{pmatrix} 2 & 10 & 0 & -3 \\ 0 & 11 & -12 & 8.5 \\ 0 & 0 & -3/11 & -2/11 \\ 0 & 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \\ -17/11 \\ -16 \end{pmatrix} \quad (1.60)$$

得到 $x = (1 \quad 2 \quad 3 \quad 4)^T$ 。

```

1 #include <stdio.h>
2 #include <iostream>
3 #include <math.h>
4
5 #define MAX_N 20
6
7 int main(int argc, char* argv[]) {
8     int n; //未知数个数
9     int i, j, k;
10    static double a[MAX_N][MAX_N], b[MAX_N], x[MAX_N], y[MAX_N];
11    static double l[MAX_N][MAX_N], u[MAX_N][MAX_N];
12    printf("\nInput n value(dim of Ax=b): "); //输入系数矩阵维度
13    scanf("%d", &n);
14    if(n > MAX_N) {
15        printf("The input n is larger than MAX_N, please redefine the MAX_N.\n");
16        return 1;
17    }
18    if(n <= 0) {
19        printf("Please input a number between 1 and %d.\n", MAX_N);
20        return 1;
21    }
22    printf("Now input the matrix a(i, j), i, j = 0, ..., %d:\n", n-1); //输入系数矩阵
23    for (i=0; i<n; i++) {
24        for (j=0; j<n; j++)
25            scanf("%lf", &a[i][j]);
26    printf("Now input the matrix b(i), i = 0, ..., %d:\n", n-1); //输入常数项
27    for(i=0; i<n; i++)
28        scanf("%lf", &b[i]);
29    for(i=0; i<n; i++)
30        l[i][i] = 1;
31    for(k=0; k<n; k++) {
32        for(j=k; j<n; j++) { // dolittle分解
33            u[k][j]=a[k][j];
34            for(i=0; i<=k-1; i++)
35                u[k][j]-=(l[k][i]*u[i][j]);
36            printf("%f\n", u[k][j]);
37        }
38        for(i=k+1; i<n; i++) {
39            l[i][k]=a[i][k];
40            for(j=0; j<=k-1; j++)
41                l[i][k]-=(l[i][j]*u[j][k]);
42            l[i][k]/=u[k][k];
43            printf("%f\n", l[i][k]);
44        }
45    }
46    for(i=0; i<n; i++) { //解Ly = b
47        y[i] = b[i];
48    }
}

```

```

49     for(j=0; j<=i-1; j++)
50         y[i] -= (l[i][j]*y[j]);
51     }
52
53     for(i=n-1; i>=0; i--){ // 解 UX = Y
54         x[i]=y[i];
55         for(j=i+1; j<n; j++)
56             x[i] -= (u[i][j]*x[j]);
57         x[i]/=u[i][i];
58     }
59
60     printf("Solve...x_i = \n"); // 输出结果
61     for(i=0; i<n; i++)
62         printf("%f\n", x[i]);
63     system("pause");
64     return 0;
65 }
```

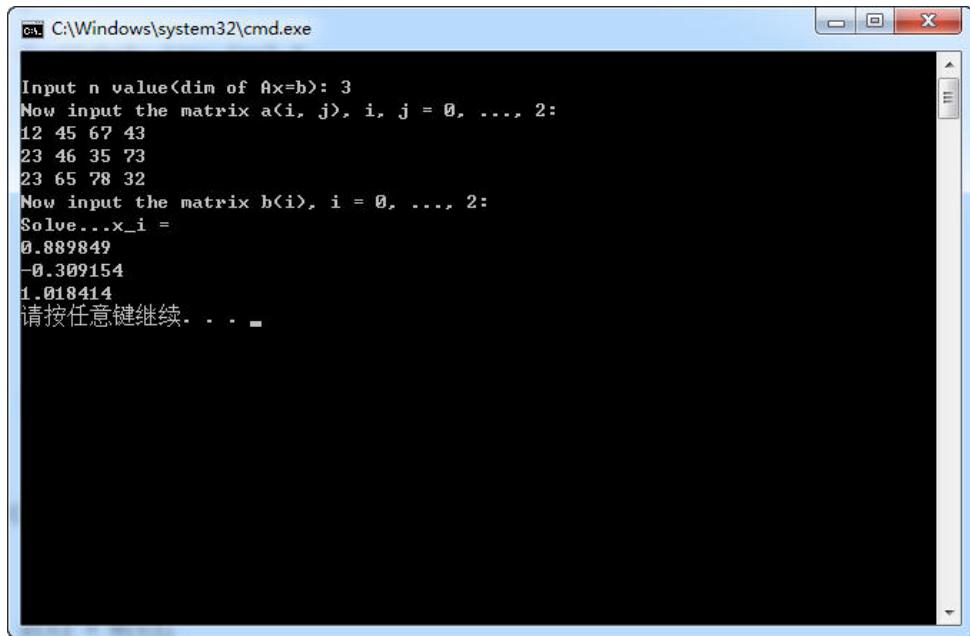


图 1.5: Doolittle 分解算法运行结果

1.2.2 克劳特 (Crout) 分解

根据上述定理, 如果 n 阶方阵 A 的各阶顺序主子式不为 0, 则存在唯一的 LU 分解。现考察 4 阶方阵, 有:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1.61)$$

右边两个矩阵相乘，可得到如下等式：

$$a_{11} = l_{11}, a_{12} = l_{11}u_{12}, a_{13} = l_{11}u_{13}, a_{14} = l_{11}u_{14} \quad (1.62)$$

$$a_{21} = l_{21}, a_{22} = l_{21}u_{12} + l_{22}, a_{23} = l_{21}u_{13} + l_{22}u_{23}, a_{24} = l_{21}u_{14} + l_{22}u_{24} \quad (1.63)$$

$$a_{31} = l_{31}, a_{32} = l_{31}u_{12} + l_{32}, a_{33} = l_{31}u_{13} + l_{32}u_{23} + l_{33} \quad (1.64)$$

$$a_{34} = l_{31}u_{14} + l_{32}u_{24} + l_{33}u_{34}, a_{41} = l_{41}, a_{42} = l_{41}u_{12} + l_{42} \quad (1.65)$$

$$a_{43} = l_{41}u_{13} + l_{42}u_{23} + l_{43}, a_{44} = l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + l_{44} \quad (1.66)$$

对于上述等式，我们可以按列设定计算顺序：

$$l_{11}, l_{21}, l_{31}, l_{41} \rightarrow u_{12}, u_{13}, u_{14} \rightarrow l_{22}, l_{32}, l_{42} \quad (1.67)$$

$$\rightarrow u_{23}, u_{24} \rightarrow l_{33}, l_{43} \rightarrow u_{34} \rightarrow l_{44} \quad (1.68)$$

很容易导出显示的计算公式：

$$l_{11} = a_{11}, l_{21} = a_{21}, l_{31} = a_{31}, l_{41} = a_{41}, u_{12} = a_{12}/l_{11}, \quad (1.69)$$

$$u_{13} = a_{13}/l_{11}, u_{14} = a_{14}/l_{11}, l_{22} = a_{22} - l_{21}u_{12}, \quad (1.70)$$

$$l_{32} = a_{32} - l_{31}u_{12}, l_{42} = a_{42} - l_{41}u_{12}, \quad (1.71)$$

$$u_{23} = (a_{23} - l_{21}u_{13})/l_{22}, u_{24} = (a_{24} - l_{21}u_{14})/l_{22}, \quad (1.72)$$

$$l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}, l_{43} = a_{43} - l_{41}u_{13} - l_{42}u_{23}, \quad (1.73)$$

$$u_{34} = (a_{34} - l_{31}u_{14} - l_{32}u_{24})/l_{33}, \quad (1.74)$$

$$l_{44} = a_{44} - l_{41}u_{14} - l_{42}u_{24} - l_{43}u_{34} \quad (1.75)$$

可以进一步将上述结果推广到一般的 n 阶方阵，将 \mathbf{A} 进行克劳特 (Crout) 分解，可以得到一个下三角矩阵 \mathbf{L} 和一个单位上三角矩阵 \mathbf{U} 的乘积。

其中 \mathbf{L} 和 \mathbf{U} 的计算公式为：

$$\begin{cases} l_{i1} = a_{i1}, & i = 1, 2, 3, \dots, n \\ u_{1j} = \frac{a_{1j}}{l_{11}}, & j = 2, 3, 4, \dots, n \\ l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}, & j = 1, 2, \dots, i \\ u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}}{l_{ii}}, & j = i+1, i+2, \dots, n \end{cases} \quad (1.76)$$

例 4. 用 Crout 分解对下面的矩阵进行分解。

$$\begin{pmatrix} 2 & 10 & 0 & -3 \\ -3 & -4 & -12 & 13 \\ 1 & 2 & 3 & -4 \\ 4 & 14 & 9 & -13 \end{pmatrix} \quad (1.77)$$

解, 由 Crout 分解公式, 可得到:

$$l_{11} = 2, l_{21} = -3, l_{31} = 1, l_{41} = 4, \quad (1.78)$$

$$u_{12} = 10/2, u_{13} = 0/2 = 0, u_{14} = -\frac{3}{2}, \quad (1.79)$$

$$l_{22} = -4 - (-3 \cdot 5) = 11, l_{32} = 2 - 1 \cdot 5 = -3, \quad (1.80)$$

$$l_{42} = 14 - 4 \cdot 5 = -6, u_{23} = \frac{-12 - (-3 \cdot 0)}{11} = -\frac{12}{11}, \quad (1.81)$$

$$u_{24} = \frac{13 - (-3 \cdot (-1.5))}{11} = \frac{17}{22}, \quad (1.82)$$

$$l_{33} = 3 - 1 \cdot 0 - (-3) \cdot \left(-\frac{12}{11}\right) = -\frac{3}{11}, \quad (1.83)$$

$$l_{43} = 9 - 4 \cdot 0 - (-6) \cdot \left(-\frac{12}{11}\right) = \frac{27}{11}, \quad (1.84)$$

$$u_{34} = \frac{-4 - 1 \cdot (-1.5) - (-3) \cdot \frac{17}{22}}{-\frac{3}{11}} = \frac{2}{3}, \quad (1.85)$$

$$l_{44} = -13 - 4 \cdot (-1.5) - (-6) \cdot \frac{17}{22} - \frac{27}{11} \cdot \frac{2}{3} = -4 \quad (1.86)$$

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -3 & 11 & 0 & 0 \\ 1 & -3 & -\frac{3}{11} & 0 \\ 4 & -6 & \frac{27}{11} & -4 \end{pmatrix} \begin{pmatrix} 1 & 5 & 0 & -\frac{3}{2} \\ 0 & 1 & -\frac{12}{11} & \frac{17}{22} \\ 0 & 0 & 1 & \frac{2}{3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1.87)$$

```

1 #include "stdio.h"
2 #include "math.h" //头文件
3 #define N 20 //自定义N=20
4
5 int main() { //主函数
6     int i,j,k;
7     int size;
8     float a[N][N],l[N][N],u[N][N];
9     float b[N],x[N],y[N]; //定义变量
10    printf("\t\t\tCrout分解法解方程组\n");

```

```

11     printf("请输入方阵A的n: ");
12     scanf("%d",&size);
13     printf("\n");
14     printf("请输入方程组的系数:\n");
15     for(i=0; i<size; i++) {
16         for(j=0; j<size; j++) {
17             scanf("%f",&a[i][j]); //输入方程组系数矩阵a[][]
18         }
19     }
20     printf("\n请输入方程组的y:\n");
21     for(i=0; i<size; i++) {
22         scanf("%f",&b[i]); //输入结果矩阵b[]
23     }
24     printf("\n方阵A[][]为: \n");
25     for(i=0; i<size; i++) {
26         for(j=0; j<size; j++) {
27             printf("%f ",a[i][j]); //输出a[][]
28         }
29     }
30     printf("\n");
31     printf("\n方程组y为:\n");
32     for(i=0; i<size; i++) {
33         printf("%f ",b[i]); //输出b[]
34     }
35     for(i=0; i<size; i++) {
36         u[i][i]=1; //定初始值 令u[i][i]=1
37     }
38     for(i=0; i<size; i++) {
39         for(j=i+1; j<size; j++) {
40             l[i][j]=0; //定初始值 令l[i][j]=0
41         }
42     }
43     for(j=0; j<size; j++) {
44         for(i=j+1; i<size; i++) {
45             u[i][j]=0; //定初始值 令u[i][j]=0
46         }
47     }
48     l[0][0]=a[0][0];
49     for(i=1; i<size; i++) {
50         l[i][0]=a[i][0]; //计算第一行的l[]
51         u[0][i]=a[0][i]/l[0][0]; //计算第一列的u[]
52     }
53     for(i=1; i<size-1; i++) {
54         for(j=1; j<=i; j++) { //计算第2行到第size-1行的l[]
55             l[i][j]=a[i][j];
56             for(k=0; k<j; k++) {
57                 l[i][j]=l[i][j]-l[i][k]*u[k][j];
58             }
59         }
60     }
61     printf("\n");

```

```

59     for(j=i+1; j<size; j++) { //计算第2行到第size行的u[][]
60         u[i][j]=a[i][j];
61         for(k=0; k<=i-1; k++) {
62             u[i][j]=u[i][j]-l[i][k]*u[k][j];
63         }
64         u[i][j]=u[i][j]/l[i][i];
65     }
66     printf("\n");
67 }
68 }
69 for(j=1; j<size; j++) { //计算第size行的l[][]
70     l[size-1][j]=a[size-1][j];
71     for(k=0; k<=j-1; k++) {
72         l[size-1][j]=l[size-1][j]-l[size-1][k]*u[k][j];
73     }
74 }
75 printf("\n");
76 printf("输出矩阵L[i][j]\n");
77 for(i=0; i<size; i++) {
78     for(j=0; j<size; j++) {
79         printf("%f",l[i][j]);
80         printf(" ");//输出下三角矩阵l[][]
81     }
82     printf("\n");
83 }
84 printf("输出矩阵U[i][j]\n");
85 for(i=0; i<size; i++) {
86     for(j=0; j<size; j++) {
87         printf("%f",u[i][j]);
88         printf(" ");//输出单位上三角矩阵u[][]
89     }
90     printf("\n");
91 }
92 y[0]=b[0]/l[0][0];//给y[0]初始值
93 for(i=1; i<size; i++) { //计算y[i]的值
94     y[i]=b[i];
95     for(k=0; k<=i-1; k++) {
96         y[i]=y[i]-l[i][k]*y[k];//计算公式
97     }
98     y[i]=y[i]/l[i][i];
99 }
100 printf("\n");
101 printf("y值:\n");
102 for(i=0; i<size; i++)
103     printf("y[%d]=%f ",i+1,y[i]);//输出y[i]的结果
104 printf("\n\n");
105 printf("x的值:\n");
106 x[size-1]=y[size-1];//给x[size-1]赋值

```

```

107     for(i=size-2; i>=0; i--) {
108         x[i]=y[i];
109         for(k=i+1; k<size; k++) {
110             x[i]=x[i]-u[i][k]*x[k]; //计算x[]
111         }
112     }
113     for(i=0; i<size; i++) {
114         printf("x[%d]=%f\n", i+1, x[i]); //输出x[i]的结果
115     }
116 }

```

The screenshot shows a command-line interface window titled "C:\Windows\system32\cmd.exe". The title bar also displays "Crout分解法解方程组". The window contains the following text output:

```

请输入方阵A的n: 4
请输入方程组的系数:
34 67 34 67
23 54 876 12 54
45 23 61 28
23 17 83 59

请输入方程组的y:
4 87 23 92

方阵A[][]为:
34.000000 67.000000 34.000000 67.000000
23.000000 54.000000 876.000000 12.000000
54.000000 45.000000 23.000000 61.000000
28.000000 23.000000 17.000000 83.000000

方程组y为:
59.000000 4.000000 87.000000 23.000000

输出矩阵L[i][j]
34.000000 0.000000 0.000000 0.000000
23.000000 8.676472 0.000000 0.000000
54.000000 -61.411762 6006.503906 0.000000
28.000000 -32.176468 3152.328125 51.862362

输出矩阵U[i][j]
1.000000 1.970588 1.000000 1.970588
0.000000 1.000000 98.311852 -3.840677
0.000000 0.000000 1.000000 -0.046828
0.000000 0.000000 0.000000 1.000000

y值:
y[1]=1.735294 y[2]=-4.138983 y[3]=-0.043434 y[4]=-0.421250

x的值:
x[1]=1.736721
x[2]=0.452577
x[3]=-0.063161
x[4]=-0.421250
请按任意键继续... .

```

图 1.6: Crout 分解算法运行结果

1.2.3 平方根法分解

在工程计算中，如应用有限元法解结构力学问题、应用差分方法解椭圆型偏微分方程问题等，最后都归结为求解系数矩阵为对称正定线性方程组的问题。对于这类矩阵有更好的解决方案。

定理 2. 定理 设 A 为对称正定矩阵，则存在三角矩阵 L ，使得 $A = LL^\top$ 成立，如限制 L 的主对角线元素取正值，则这种分解是唯一的。

此时，按照 $l_{11} \rightarrow l_{21} \rightarrow l_{22} \rightarrow l_{31} \rightarrow l_{32} \rightarrow l_{33} \rightarrow \dots$ 逐行求出分解矩阵 L 的元素，计算公式为：

$$\begin{cases} l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}, & j = 1, 2, 3, \dots, i-1 \\ l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{\frac{1}{2}}, & i = 1, 2, 3, 4, \dots, n \end{cases} \quad (1.88)$$

这种矩阵分解公式由于含有开方运算而被称作平方根分解。

例 5. 用平方根法对下面的对称正定矩阵进行分解。

$$\begin{pmatrix} 6 & 7 & 5 \\ 7 & 13 & 8 \\ 5 & 8 & 6 \end{pmatrix} \quad (1.89)$$

解：由平方根法分解公式，可得到：

$$L = \begin{pmatrix} \sqrt{6} & 0 & 0 \\ \frac{7}{\sqrt{6}} & \sqrt{\frac{29}{6}} & 0 \\ \frac{5}{\sqrt{6}} & \frac{13}{\sqrt{174}} & \sqrt{\frac{25}{29}} \end{pmatrix} \quad (1.90)$$

因此

$$A = LL^\top \quad (1.91)$$

1.2.4 Cholesky 分解

定理 3. 定理 设对称正定矩阵 A 可分解成 $A = LDL^\top$ 的形式，其中 D 为对角阵，而 L 是单位下三角矩阵。

此时，可按 $d_1 \rightarrow l_{21} \rightarrow d_2 \rightarrow l_{31} \rightarrow l_{32} \rightarrow d_3 \rightarrow \dots$ 逐行来计算，具体分解公式如下：

$$\begin{cases} l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} d_k l_{ik} l_{jk}}{d_j}, & j = 1, 2, 3, \dots, i-1 \\ d_i = a_{ii} - \sum_{k=1}^{i-1} d_k l_{ik}^2, & i = 1, 2, 3, 4, \dots, n \end{cases} \quad (1.92)$$

这种关于对称正定矩阵的分解称作改进的平方根法，或乔累斯基 (Cholesky) 分解。这种方法具有较高的计算效率，总的计算量约为 $n^3/6$ 。

```

1 #include<iostream>
2 #include<math.h>
3 #include<process.h>
4 using namespace std;
5
6 /*
7  * 要求对称系数矩阵。
8 */
9
10
11 class Cholesky {
12 private:
13     int i,j,k,n;
14     double sum,*b,*d,*x,**a,eps;
15 public:
16     void choleskyInput();
17     void choleskyDecomposition();
18     void choleskyOutput();
19 ~Cholesky() {
20     delete []b;
21     delete []d;
22     delete []x;
23     for(i=0; i<n; i++) {
24         delete [] a[i];
25     }
26     delete []a;
27 }
28
29 };
30
31 void main() {
32     Cholesky solution;
33     solution.choleskyInput();
34     solution.choleskyDecomposition();
35     solution.choleskyOutput();
36 }
37
38 void Cholesky::choleskyInput() {
39     cout<<"输入方程的个数";
40     cin>>n;
41     b=new double[n];
42     d=new double[n];
43     x=new double[n];
44     a=new double*[n];
45
46     for(i=0; i<n; i++) {
47         a[i] = new double[n];
48     }

```

```

49
50     for(i=0; i<n; i++) {
51         for(j=0; j<n; j++) {
52             cout<<"/n输入a["<<i<<"]["<<j<<"]=";
53             cin>>a[i][j];
54         }
55     for(i=0; i<n; i++) {
56         for(j=0; j<n; j++) {
57             if(a[i][j] != a[j][i]) {
58                 cout<<"/n系数矩阵不对称.失败..."<<endl;
59                 exit(0);
60             }
61         }
62     for(i=0; i<n; i++) {
63
64         cout<<"/n输入 b["<<i<<"]=";
65         cin>>b[i];
66     }
67     cout<<"/n输入最小主元素";
68     cin>>eps; //输入段结束
69 }
70 void Cholesky::choleskyDecomposition() {
71     for(i=0; i<n; i++) {
72         for(j=0; j<n; j++) {
73             sum=a[i][j];
74             for(k=0; k<i; k++) {
75                 sum -= a[i][k]*a[j][k];
76             }
77             if( i==j) {
78                 if(sum <= 0) {
79                     cout<<"/n矩阵非正定.失败..."<<endl;
80                     exit(0);
81                 }
82                 d[i]=sqrt(sum);
83             } else {
84                 a[j][i] = sum/d[i];
85             }
86         }
87     }
88
89     for(i=0; i<n; i++) {
90         sum=b[i];
91         for(k=0; k<i; k++) {
92             sum -= a[i][k]*x[k];
93         }
94         x[i] = sum/d[i];
95     }
96 }
```

```

97     for(i=(n-1); i>=0; i--) {
98         sum = x[i];
99         for(k=(i+1); k<n; k++) {
100             sum -= a[k][i]*x[k];
101         }
102         x[i] = sum/d[i];
103     }
104 }
105 void Cholesky::choleskyOutput() {
106     cout<<"/n:结果是: "<<endl;
107     for(i=0; i<n; i++) {
108         cout<<"x["<<i<<"]="<<x[i]<<endl;
109     }
110 }
```

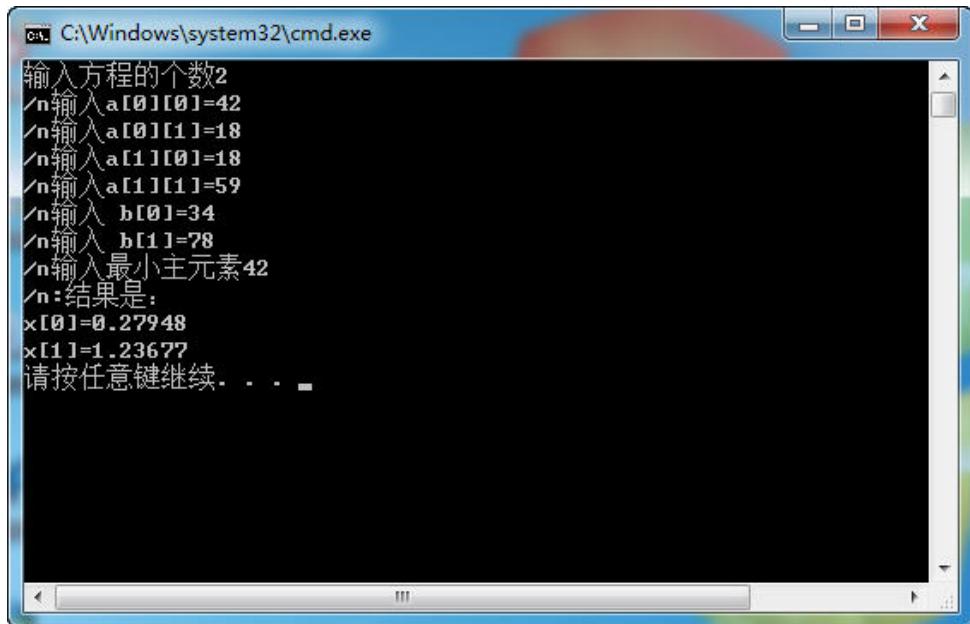


图 1.7: Cholesky 算法运行结果

1.3 追赶法

在科学计算中，如三次样条插值（教材 pp.35）中，非零元素集中在三条对角线上，如下所示。对于这种三对角矩阵的方程组，可以使用追赶法进行求解。

$$\left\{ \begin{array}{l} b_1 x_1 + c_1 x_2 = f_1 \\ a_2 x_1 + b_2 x_2 + c_2 x_3 = f_2 \\ \dots \\ a_{n-1} x_{n-2} + b_{n-1} x_{n-1} + c_{n-1} x_n = f_{n-1} \\ a_n x_{n-1} + b_n x_n = f_n \end{array} \right. \quad (1.93)$$

可以将其系数记为:

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & a_n & b_n \end{bmatrix} \quad (1.94)$$

定理 4. 定理 设上述三对角矩阵为对角占优矩阵, 即满足如下条件:

$$\begin{cases} |b_1| > |c_1| \\ |b_i| > |a_i| + |c_i| \quad i = 2, 3, 4, \dots, n-1 \\ |b_n| > |a_n| \end{cases} \quad (1.95)$$

则它是非奇异的, 这时方程有唯一解。

对于上述三对角矩阵, 可以先从方程 (2) 中消去 x_1 , 方程 (3) 中消去 x_2 , 方程 (4) 中消去 x_3 , 以此类推, 从方程 (n) 中消去 x_{n-1} , 因此得到如下形式的方程组。

(1) 消元过程

$$\begin{cases} x_1 + u_1 x_2 = g_1 \\ x_2 + u_2 x_3 = g_2 \\ \dots \\ x_{n-1} + u_{n-1} x_n = g_{n-1} \\ x_n = g_n \end{cases} \quad (1.96)$$

其中系数为,

$$\begin{cases} u_1 = c_1/b_1, g_1 = f_1/b_1 \\ u_i = \frac{c_i}{b_i - u_{i-1} a_i} \quad i = 2, 3, \dots, n-1 \\ g_i = \frac{f_i - g_{i-1} a_i}{b_i - u_{i-1} a_i} \quad i = 2, 3, \dots, n \end{cases} \quad (1.97)$$

(2) 回代过程

使用下面的递推式来求得方程的解:

$$\begin{cases} x_n = g_n \\ x_i = g_i - u_i x_{i+1} \quad i = n-1, n-2, \dots, 1 \end{cases} \quad (1.98)$$

综上所述, 解三对角方程组的追赶法分为“追”和“赶”两个环节:

1. **追的过程 (消元过程)**。计算系数 $u_1, u_2 \rightarrow \dots \rightarrow u_{n-1}$ 和 $g_1, g_2 \rightarrow \dots \rightarrow g_n$;
2. **赶的过程 (回代过程)**。按逆序求得解 $x_n \rightarrow x_{n-1} \rightarrow \dots \rightarrow x_1$;

例 6. 用追赶法求解下列线性方程组。

$$\begin{pmatrix} 3 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad (1.99)$$

解：首先进行“追”的过程，得到如下方程组：

$$u_1 = c_1/b_1 = \frac{1}{3} \qquad g_1 = \frac{1}{3} \quad (1.100)$$

$$u_2 = \frac{c_2}{b_2 - u_1 \cdot a_2} = \frac{3}{7} \qquad g_2 = -\frac{2}{7} \quad (1.101)$$

$$u_3 = \frac{c_3}{b_3 - u_2 \cdot a_3} = \frac{7}{15} \qquad g_3 = \frac{11}{15} \quad (1.102)$$

$$g_4 = -\frac{11}{38} \quad (1.103)$$

$$\begin{pmatrix} 1 & 1/3 & 0 & 0 \\ 0 & 1 & 3/7 & 0 \\ 0 & 0 & 1 & 7/15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1/3 \\ -2/7 \\ 11/15 \\ -11/38 \end{pmatrix} \quad (1.104)$$

通过“赶的”回代过程，可得到方程的解：

$$\mathbf{x} = \begin{pmatrix} \frac{21}{38} \\ -\frac{25}{38} \\ \frac{33}{38} \\ -\frac{11}{38} \end{pmatrix} \quad (1.105)$$

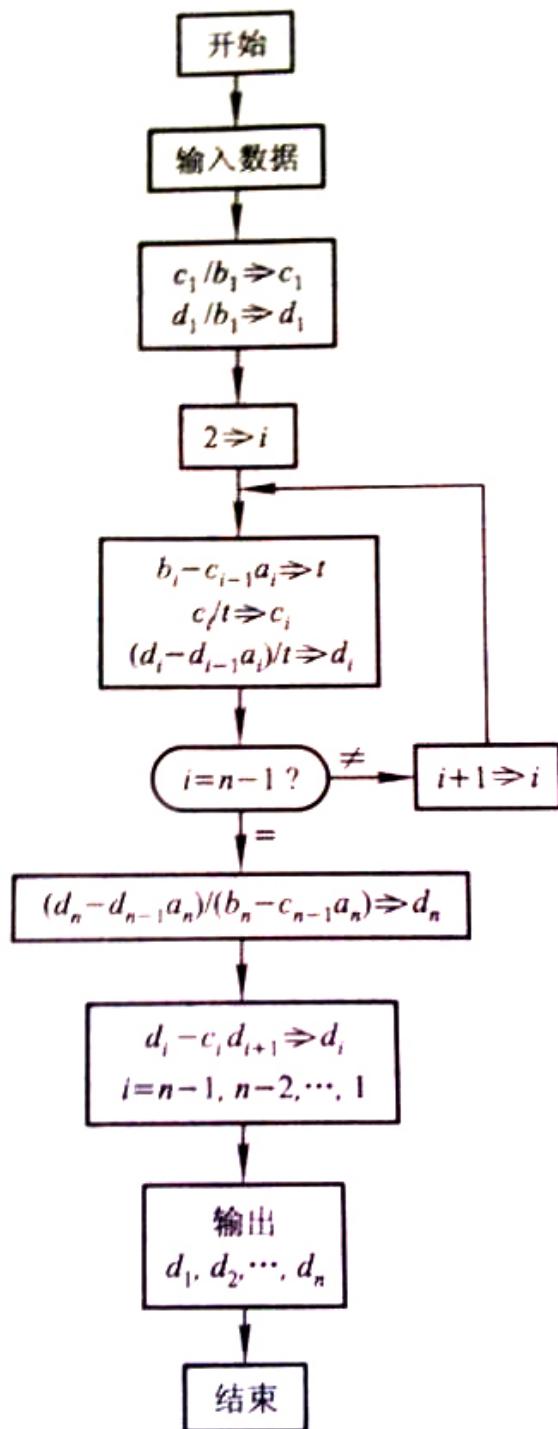


图 1.8: 追赶法

```

1 #include <stdio.h>
2 #include <conio.h>
3 #include <dos.h>
4
5 void main() {
6     char play;
  
```

```

7   int i,n;
8   float a[100],b[100],c[100],d[100];
9   float u[100],l[100],y[100],x[100];
10
11  do {
12      printf("Please input the square distance set of piece n:\r\n");
13      scanf("%d",&n);
14
15      printf("Please input a2--an:\r\n");
16      for(i=2; i<=n; i++) {
17          printf("a%d=%",i);
18          scanf("%f",&a[i]);
19      }
20
21      printf("Please input b1--bn:\r\n");
22      for(i=1; i<=n; i++) {
23          printf("b%d=%",i);
24          scanf("%f",&b[i]);
25      }
26
27      printf("Please input c1--c(n-1):\r\n");
28      for(i=1; i<n; i++) {
29          printf("c%d=%",i);
30          scanf("%f",&c[i]);
31      }
32
33      printf("Please input d1--dn:\r\n");
34      for(i=1; i<=n; i++) {
35          printf("d%d=%",i);
36          scanf("%f",&d[i]);
37      }
38
39      u[1]=b[1];
40      y[1]=d[1];
41      for(i=2; i<=n; i++) {
42          l[i]=a[i]/u[i-1];
43          u[i]=b[i]-l[i]*c[i-1];
44          y[i]=d[i]-l[i]*y[i-1];
45      }
46      x[n]=y[n]/u[n];
47      for(i=n-1; i>0; i--)
48          x[i]=(y[i]-c[i]*x[i+1])/u[i];
49      cprintf("The result:\r\n\r\n");
50      for(i=n; i>=1; i--)
51          printf(" x%d=%f\r\n",i,x[i]);
52      getche();
53      printf("\r\n");
54      printf("continue?(y/n)");

```

```

55     play=getche();
56 } while(play=='y'||play=='Y');
57
58 }

```

```

C:\Windows\system32\cmd.exe
Please input the square distance set of piece n:
3
Please input a2--an:
a2=2
a3=4
Please input b1--bn:
b1=6
b2=7
b3=9
Please input c1--c<n-1>:
c1=2
c2=5
Please input d1--dn:
d1=2
d2=5
d3=8
The result:
x3=0.900901
x2=-0.027027
x1=0.342342

```

图 1.9: 追赶法运行结果

1.4 向量和矩阵的范数

为了研究线性方程组近似解的误差估计和迭代法的收敛性，有必要对向量及矩阵的“大小”引进某种度量，即范数的概念。向量范数是用来度量向量长度的，它可以看成是二、三维解析几何中向量长度概念的推广。用 R^n 表示 n 维实向量空间。

定义 2. 定义对任一向量 $\mathbf{X} \in R^n$ ，按照一定规则确定一个实数与它对应，该实数记为 $\|\mathbf{X}\|$ 。若 $\|\mathbf{X}\|$ 满足下面三个性质：

1. $\|\mathbf{X}\| \geq 0$, $\|\mathbf{X}\| = 0$ 当且仅当 $\mathbf{X} = 0$;
2. 对任意实数 λ , $\|\lambda\mathbf{X}\| = |\lambda|\|\mathbf{X}\|$;
3. 对任意向量 $\mathbf{Y} \in R^n$, $\|\mathbf{X} + \mathbf{Y}\| \leq \|\mathbf{X}\| + \|\mathbf{Y}\|$;

则称该实数 $\|\mathbf{X}\|$ 为向量 \mathbf{X} 的范数。

常见的范数在 R^n 中，常见的范数有以下几种：

1. $\|\mathbf{X}\|_1 = |x_1| + |x_2| + \cdots + |x_n| = \sum_{i=1}^n |x_i|;$
2. $\|\mathbf{X}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^n x_i^2};$
3. $\|\mathbf{X}\|_\infty = \max\{|x_1|, |x_2|, \dots, |x_n|\} = \max_{1 \leq i \leq n} \{|x_i|\};$

其中 x_1, x_2, \dots, x_n 分别是 \mathbf{X} 的 n 个分量。以上定义的范数分别称为 1-范数, 2-范数和 ∞ -范数。可以验证它们都是满足范数性质的, 其中 $\|\mathbf{x}\|_2$ 是由内积导出的向量范数。

上述范数都是 p 范数

$$\|\mathbf{X}\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \quad (1.106)$$

当不需要指明使用哪一种向量范数时, 就用记号 $\|\cdot\|$ 泛指任何一种向量范数。

∞ -范数

定理 5. 定理 对任意向量 $\mathbf{x} \in R^n$, 有 $\lim_{p \rightarrow \infty} \|\mathbf{x}\|_p = \|\mathbf{x}\|_\infty$ 。

证明. 由于 $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$

所以

$$\|\mathbf{x}\|_\infty = \left(\max_{1 \leq i \leq n} |x_i|^p \right)^{\frac{1}{p}} \leq \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \leq \left(n \max_{1 \leq i \leq n} |x_i|^p \right)^{\frac{1}{p}} \quad (1.107)$$

即:

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_p \leq n^{\frac{1}{p}} \|\mathbf{x}\|_\infty \quad (1.108)$$

当 $p \rightarrow \infty$, $n^{\frac{1}{p}} \rightarrow 1$ 。

所以:

$$\lim_{p \rightarrow \infty} \|\mathbf{x}\|_p = \|\mathbf{x}\|_\infty \quad (1.109)$$

□

例 7. 设 $\mathbf{x} = (1, 0, -1, 2)^\top$, 试计算 1-范数, 2-范数和 ∞ -范数。

解:

$$\|\mathbf{x}\|_1 = 1 + 0 + |-1| + 2 = 4 \quad (1.110)$$

$$\|\mathbf{x}\|_2 = \sqrt{1^2 + 0^2 + (-1)^2 + 2^2} \quad (1.111)$$

$$= \sqrt{6} \quad (1.112)$$

$$\|\mathbf{x}\|_\infty = \max\{1, 0, |-1|, 2\} \quad (1.113)$$

$$= 2 \quad (1.114)$$

矩阵的范数

定义 3. 定义 如果矩阵 $\mathbf{A} \in R^{n \times n}$ 的某个非负实值函数 $N(\mathbf{A}) = \|\mathbf{A}\|$, 满足:

1. $\|\mathbf{A}\| \geq 0$, 当且仅当 $\mathbf{A} = 0$ 时, $\|\mathbf{A}\| = 0$;
2. 对任意实数 λ , $\|\lambda\mathbf{A}\| = |\lambda|\|\mathbf{A}\|$;
3. 对任意矩阵 $\mathbf{A}, \mathbf{B} \in R^{n \times n}$, $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$;
4. $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|$

则称 $N(\mathbf{A})$ 是 $R^{n \times n}$ 的一个矩阵范数 (或模)。

对 n 阶方阵 $\mathbf{A} = (a_{ij})_n$, 其 $1, 2, \infty$ 范数计算如下。其中, $\lambda_{\max}(\mathbf{A}^\top \mathbf{A})$ 表示 $\mathbf{A}^\top \mathbf{A}$ 的最大特征值, 即满足

$$f(\lambda) = |\lambda \mathbf{I}_n - \mathbf{A}^\top \mathbf{A}| = 0 \quad (1.115)$$

定义 4. 定义

1. $\|\mathbf{A}\|_\infty$ 范数 (称为 \mathbf{A} 的行范数):

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}| \quad (1.116)$$

2. $\|\mathbf{A}\|_1$ 范数 (称为 \mathbf{A} 的列范数):

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}| \quad (1.117)$$

3. $\|\mathbf{A}\|_2$ 范数 (称为 \mathbf{A} 的 2 范数或谱范数):

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^\top \mathbf{A})} \quad (1.118)$$

例 8. 设 $A = \begin{pmatrix} 1 & -2 \\ -3 & 4 \end{pmatrix}$, 计算 A 的各种范数。

$$\|A\|_1 = \max\{|1| + |-3|, |-2| + |4|\} = 6 \quad (1.119)$$

$$\|A\|_\infty = \max\{|1| + |-2|, |-3| + |4|\} = 7 \quad (1.120)$$

$$\|A\|_F = \sqrt{1^2 + (-2)^2 + (-3)^2 + 4^2} \approx 5.477 \quad (1.121)$$

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^\top A)} = \sqrt{15 + \sqrt{221}} \approx 5.46 \quad (1.122)$$

例 9. 计算方阵

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 4 \\ 0 & -2 & 4 \end{pmatrix} \quad (1.123)$$

的三种常用范数。

解：

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}| = \max\{1, 4, 8\} = 8 \quad (1.124)$$

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}| = \max\{1, 6, 6\} = 6 \quad (1.125)$$

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^\top \mathbf{A})} = \sqrt{32} = 4\sqrt{2} \quad (1.126)$$

矩阵的谱半径

定义 5 (谱半径). 设 $\mathbf{A} \in R^{n \times n}$ 的特征值为 $\lambda_i (i = 1, 2, 3, \dots, n)$, 称

$$\rho(\mathbf{A}) = \max_{1 \leq i \leq n} |\lambda_i| \quad (1.127)$$

为 \mathbf{A} 的谱半径。

定理 6. 设 $\|\cdot\|$ 是 $R^{n \times n}$ 上的任意一个矩阵范数, $A \in R^{n \times n}$, 则有:

$$\rho(A) \leq \|A\| \quad (1.128)$$

即 A 的谱半径是 A 的任意一种范数的下界。

定理 7. 如果 $A \in R^{n \times n}$ 为对称矩阵, 则有:

$$\rho(A) = \|A\|_2 \quad (1.129)$$

定理 8 (迭代法收敛性定理). 设有线性方程组 $\mathbf{AX} = \mathbf{b}$, 则对于任意的初始向量 $\mathbf{X}^{(0)}$, 迭代法

$$\mathbf{X}^{(k+1)} = \mathbf{B}\mathbf{X}^{(k)} + \mathbf{b} \quad (1.130)$$

收敛的充分必要条件是

$$\rho(\mathbf{B}) < 1 \quad (1.131)$$

上述定理表明, 线性方程组迭代法收敛与否与 $\mathbf{X}^{(0)}$ 和 \mathbf{b} 无关, 而只与迭代矩阵 \mathbf{B} 的性质有关。

1.5 迭代法

线性方程组如下：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \cdots \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (1.132)$$

是人们熟知的计算模型，如果方程组的系数行列式不为 0，则该方程组具有唯一解。

方程组的求解难易程度取决于系数矩阵的复杂程度，如果系数矩阵为对角矩阵，则可以立即获得方程组的解。如果系数矩阵为上三角矩阵，则可以通过从上而下逐步回代的方式，可以顺序得到它的解。同理，对于下三角矩阵，也可以很容易求得方程组的解。

1.5.1 雅克比 (Jacobi) 迭代法

设有 n 阶方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (1.133)$$

写成矩阵形式为：

$$Ax = b \quad (1.134)$$

其中

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (1.135)$$

若系数矩阵 A 非奇异，且 $a_{ii} \neq 0$ ($i = 1, 2, \dots, n$)，则可将上述方程组改写成如下形式：

$$\begin{cases} x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) \\ x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n) \\ \cdots \\ x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}) \end{cases} \quad (1.136)$$

改写为迭代格式如下：

$$\begin{cases} x_1^{k+1} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^k - a_{13}x_3^k - \cdots - a_{1n}x_n^k) \\ x_2^{k+1} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^k - a_{23}x_3^k - \cdots - a_{2n}x_n^k) \\ \dots \\ x_n^{k+1} = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^k - a_{n2}x_2^k - \cdots - a_{n,n-1}x_{n-1}^k) \end{cases} \quad (1.137)$$

将上述迭代格式可简写为：

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j^k \right) \quad i = 1, 2, 3, \dots, n \quad (1.138)$$

终止迭代条件为：

$$\max_{1 \leq i \leq n} |x_i^{k+1} - x_i^k| < \varepsilon \quad (1.139)$$

其中 ε 为事前给定的阈值。

雅克比 (Jacobi) 迭代法的矩阵形式

矩阵按区域分为三部分 L, D, U , 如下：

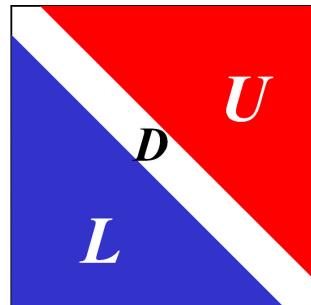


图 1.10: 雅克比迭代法

$$Ax = b \Leftrightarrow (D + L + U)x = b \quad (1.140)$$

$$\Leftrightarrow Dx = -(L + U)x + b \quad (1.141)$$

$$\Leftrightarrow x = \underbrace{-D^{-1}(L + U)x}_{B} + \underbrace{D^{-1}b}_{f} \quad (1.142)$$

因此，雅克比 (Jacobi) 迭代法的矩阵形式为：

$$x^{k+1} = Bx^k + f \quad (1.143)$$

$$= -D^{-1}(L + U)x^k + D^{-1}b \quad (1.144)$$

例 10. 用 *Jacobi* 迭代法解方程组

$$\begin{cases} 10x_1 + 3x_2 + x_3 = 14 \\ 2x_1 - 10x_2 + 3x_3 = -5 \\ x_1 + 3x_2 + 10x_3 = 14 \end{cases} \quad (1.145)$$

初值 $x_1^0 = x_2^0 = x_3^0 = 0$, 要求精度 $\varepsilon = 0.02$ 。

解: 迭代格式:

$$\begin{cases} x_1^{k+1} = -0.3x_2^k - 0.1x_3^k + 1.4 \\ x_2^{k+1} = 0.2x_1^k + 0.3x_3^k + 0.5 \\ x_3^{k+1} = -0.1x_1^k - 0.3x_2^k + 1.4 \end{cases} \quad (1.146)$$

反复迭代, 计算结果见下表。

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0	0	0
1	1.4	0.5	1.4
2	1.11	1.20	1.11
3	0.929	1.055	0.929
4	0.9906	0.9645	0.9906
5	1.01159	0.9953	1.01159
6	1.000251	1.005795	1.000251

图 1.11: 迭代计算结果

可以看到, 当迭代次数 k 增大时, 迭代值 x_1^k, x_2^k, x_3^k 会越来越接近解 $x_1 = x_2 = x_3 = 1$ 。

通常用迭代偏差 $\max_{1 \leq i \leq n} |x_i^{(k+1)} - x_i^{(k)}|$ 刻画迭代的精度, 为了防止迭代过程的不收敛, 或者收敛速度过慢, 可以设置最大迭代次数 N , 如果超过迭代次数 N , 或达不到迭代精度, 则宣称“迭代失败”。

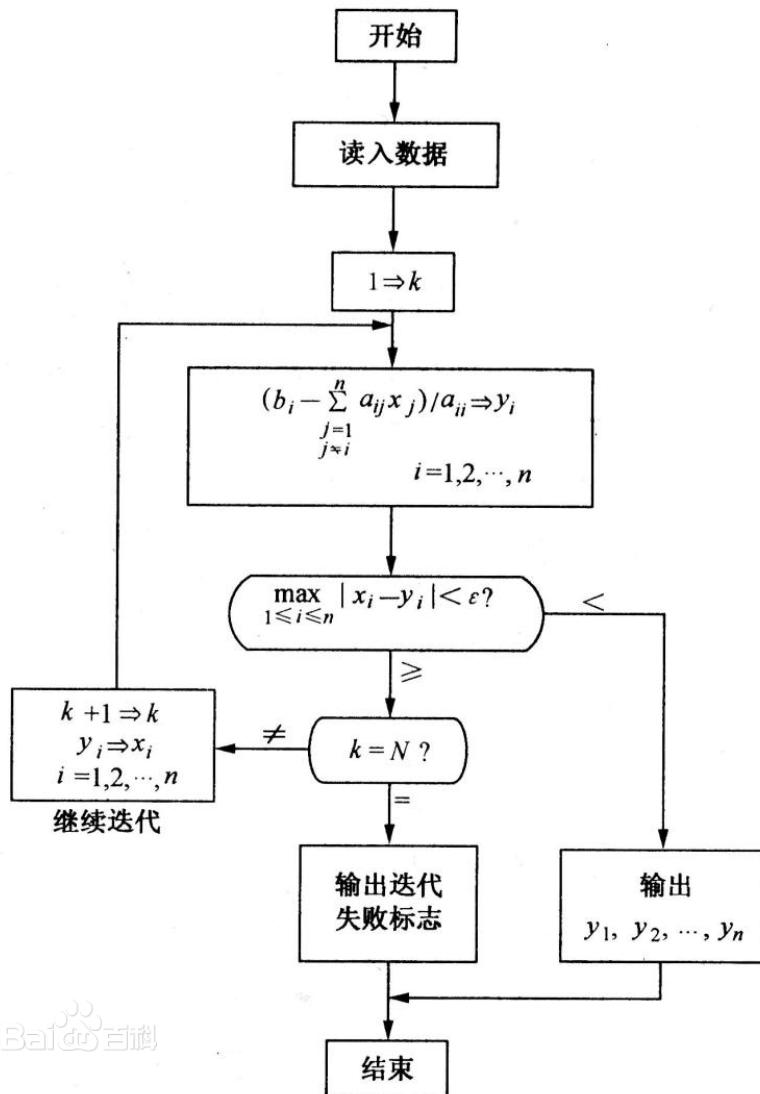


图 1.12: Jacobi 迭代法

```

1 #include "stdio.h"
2 #include "math.h"
3 #define MAX 100
4 #define n 3
5 #define exp 0.005
6 main() {
7     int i,j,k,m;
8     float temp,s;
9     float static a[n][n] = {{10,0,-1},{-2,10,-1},{0,-1,5}};
10    float static b[n] = {9,7,4};
11    float static x[n],B[n][n],g[n],y[n] = {0,0,0};
12
13    for(i=0; i<n; i++)
14        for(j=0; j<n; j++) {
  
```

```

15         B[i][j]=-a[i][j]/a[i][i];
16         g[i]=b[i]/a[i][i];
17     }
18     for(i=0; i<n; i++)
19         B[i][i]=0;
20     m=0;
21     do {
22         for(i=0; i<n; i++)
23             x[i]=y[i];
24         for(i=0; i<n; i++) {
25             y[i]=g[i];
26             for(j=0; j<n; j++)
27                 y[i]=y[i]+B[i][j]*x[j];
28         }
29         m++;
30         printf("\n%dth result is:",m);
31         printf("\nx0=%7.5f,x1=%7.5f,x2=%7.5f",y[0],y[1],y[2]);
32         temp=0;
33         for(i=0; i<n; i++) {
34             s=fabs(y[i]-x[i]);
35             if(temp<s) temp=s;
36         }
37         printf("\ntemp=%f",temp);
38     } while(temp>=exp);
39     printf("\n\nThe last result is:");
40     for(i=0; i<n; i++)
41         printf("\nx[%d]=%7.5f",i,y[i]);
42 }

```

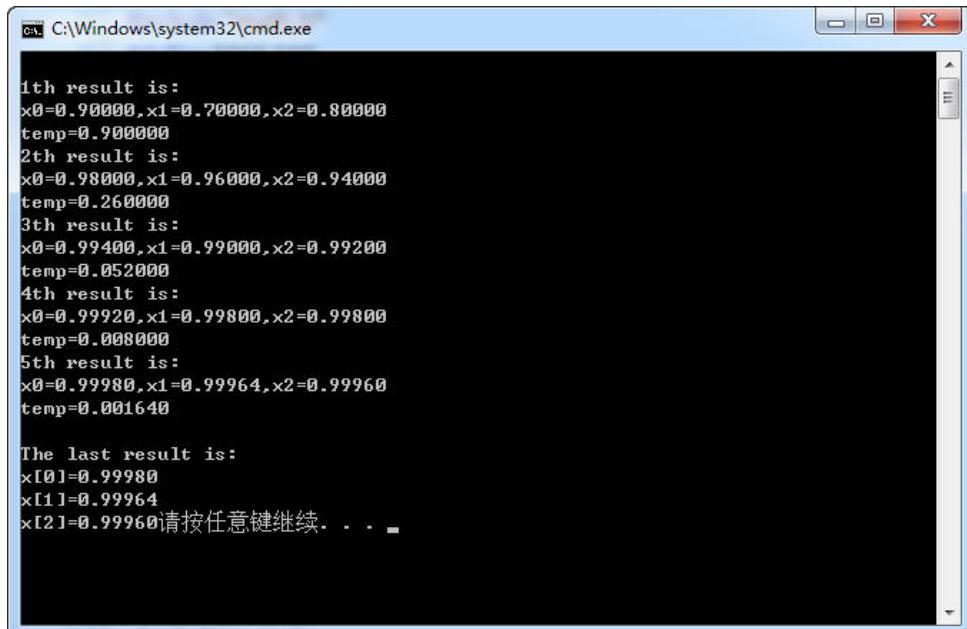


图 1.13: Jacobi 迭代法运行结果

1.5.2 高斯-赛得尔迭代法

与 Jacobi 公式不同, Gauss-Seidel 公式设定计算顺序为 $x_1^{(k+1)} \rightarrow x_2^{(k+1)} \rightarrow x_3^{(k+1)}$, 然后充分利用新信息用于计算, 如用 $x_1^{(k+1)}$ 取代 $x_1^{(k)}$ 计算 $x_2^{(k+1)}$, 再用 $x_2^{(k+1)}$ 取代 $x_2^{(k)}$ 计算 $x_3^{(k+1)}$ 等。由于 Gauss-Seidel 迭代充分利用新信息进行计算, 因此可以预料它的逼近效果要优于 Jacobi 迭代方法。

$$\left\{ \begin{array}{l} x_1^{k+1} = \frac{1}{a_{11}}(-a_{12}x_2^k - a_{13}x_3^k - a_{14}x_4^k - \cdots - a_{1n}x_n^k + b_1) \\ x_2^{k+1} = \frac{1}{a_{22}}(-a_{21}x_1^{k+1} - a_{23}x_3^k - a_{24}x_4^k - \cdots - a_{2n}x_n^k + b_2) \\ x_3^{k+1} = \frac{1}{a_{33}}(-a_{31}x_1^{k+1} - a_{32}x_2^{k+1} - a_{34}x_4^k - \cdots - a_{3n}x_n^k + b_3) \\ \vdots \\ x_n^{k+1} = \frac{1}{a_{nn}}(-a_{n1}x_1^{k+1} - a_{n2}x_2^{k+1} - a_{n3}x_3^{k+1} - \cdots - a_{n-1}x_{n-1}^{k+1} + b_n) \end{array} \right. \quad (1.147)$$

将上述迭代格式可简写为如下通式:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^n a_{ij}x_j^k \right) \quad i = 1, 2, 3, \dots, n \quad (1.148)$$

终止迭代条件为:

$$\max_{1 \leq i \leq n} |x_i^{k+1} - x_i^k| < \varepsilon \quad (1.149)$$

其中 ε 为事前给定的阈值。

高斯-赛得尔迭代法的矩阵形式

矩阵按区域分为三部分 L, D, U , 如下:

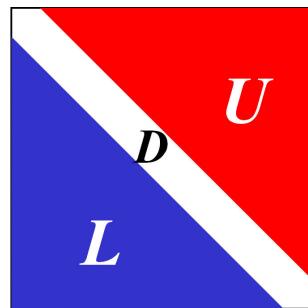


图 1.14: 高斯-赛得尔迭代法

$$\mathbf{x}^{k+1} = -D^{-1}(L\mathbf{x}^{k+1} + U\mathbf{x}^k) + D^{-1}\mathbf{b} \quad (1.150)$$

$$\Leftrightarrow (D + L)\mathbf{x}^{k+1} = -U\mathbf{x}^k + \mathbf{b} \quad (1.151)$$

$$\Leftrightarrow \mathbf{x}^{k+1} = \underbrace{-(D + L)^{-1}U}_{B}\mathbf{x}^k + \underbrace{(D + L)^{-1}\mathbf{b}}_{f} \quad (1.152)$$

因此，高斯-赛得尔迭代法的矩阵形式为：

$$\mathbf{x}^{k+1} = \mathbf{B}\mathbf{x}^k + \mathbf{f} \quad (1.153)$$

$$= -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^k + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b} \quad (1.154)$$

例 11. 用 *Gauss-Seidel* 迭代法解方程组

$$\begin{cases} 10x_1 + 3x_2 + x_3 = 14 \\ 2x_1 - 10x_2 + 3x_3 = -5 \\ x_1 + 3x_2 + 10x_3 = 14 \end{cases} \quad (1.155)$$

初值 $x_1^0 = x_2^0 = x_3^0 = 0$ ，要求精度 $\varepsilon = 0.05$ 。

解：迭代格式：

$$\begin{cases} x_1^{k+1} = -0.3x_2^k - 0.1x_3^k + 1.4 \\ x_2^{k+1} = 0.2x_1^{k+1} + 0.3x_3^k + 0.5 \\ x_3^{k+1} = -0.1x_1^{k+1} - 0.3x_2^{k+1} + 1.4 \end{cases} \quad (1.156)$$

反复迭代，计算结果见下表。

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0	0	0
1	1.4	0.78	1.026
2	1.06340	1.02048	0.98752
3	0.99510	0.99528	1.00191
4	1.00122	1.00082	0.99963

图 1.15: 迭代计算结果

可以看到，当迭代次数增大时，迭代值 x_1^k, x_2^k, x_3^k 会越来越接近解 $x_1 = x_2 = x_3 = 1$ 。一般来说，Gauss-Seidel 要比 Jacobi 迭代好，收敛速度快，但也有特例，甚至有时 Jacobi 迭代收敛而 Gauss-Seidel 迭代发散。

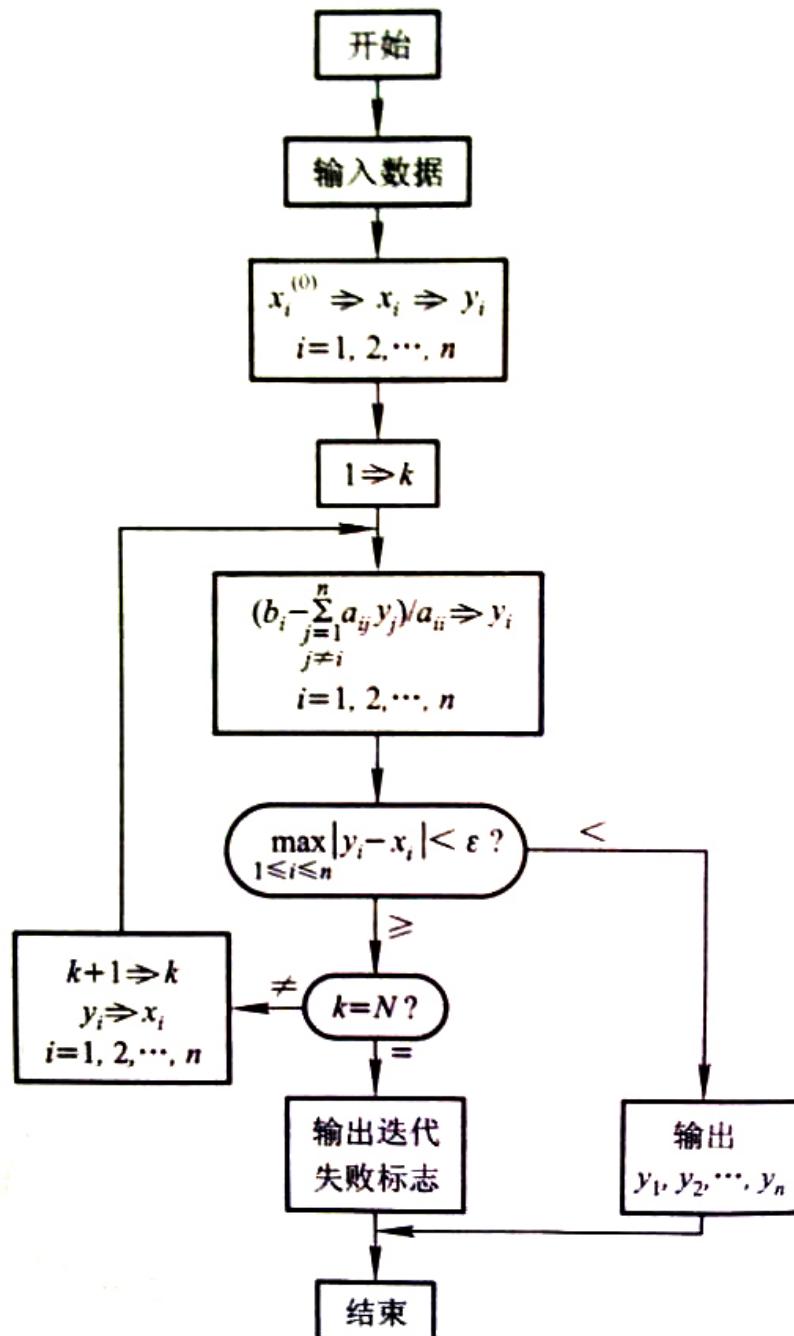


图 1.16: Gauss-Seidel 迭代法

```

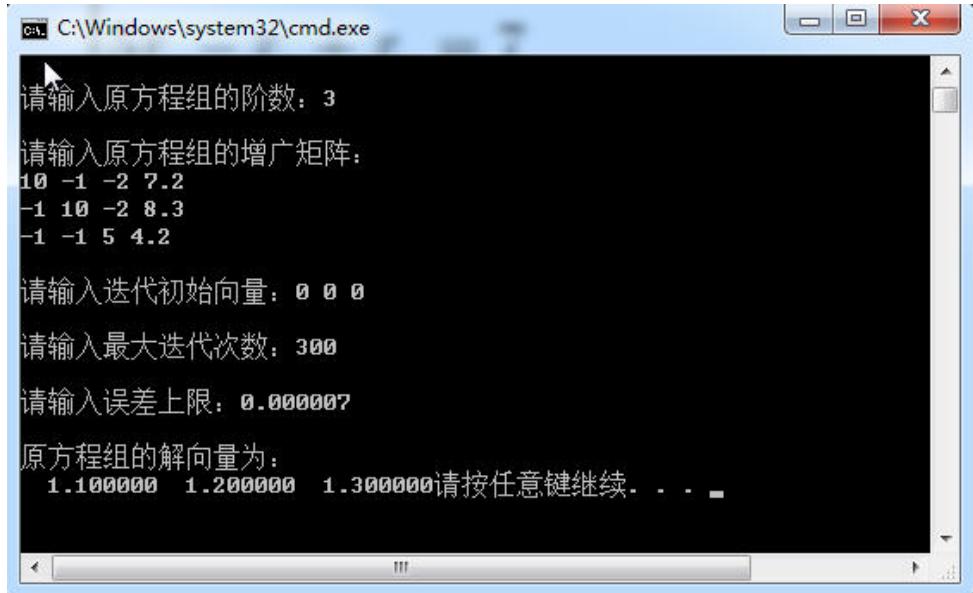
1 #include <stdio.h>
2 #include <math.h>
3 #define MAXSIZE 50
4
5 void input(double a[MAXSIZE][MAXSIZE], double b[MAXSIZE], long n);
6 void output(double x[MAXSIZE], long n);
7
8
  
```

```

9 void main(void) {
10    double a[MAXSIZE][MAXSIZE],b[MAXSIZE],x[MAXSIZE];
11    double epsilon,e,s,oldx;
12    long n,i,j,k,maxk;
13    printf("\n请输入原方程组的阶数: ");
14    scanf("%ld",&n);
15    input(a,b,n);
16    printf("\n请输入迭代初始向量: ");
17    for(i=0; i<=n-1; i++)
18        scanf("%lf",&x[i]);
19    printf("\n请输入最大迭代次数: ");
20    scanf("%ld",&maxk);
21    printf("\n请输入误差上限: ");
22    scanf("%lf",&epsilon);
23    for(k=1; k<=maxk; k++) {
24        e=0;
25        for(i=0; i<=n-1; i++) {
26            oldx=x[i];
27            s=0;
28            for(j=0; j<=n-1; j++)
29                if(j!=i) s+=a[i][j]*x[j];
30            x[i]=(b[i]-s)/a[i][i];
31            if(e<fabs(oldx-x[i]))
32                e=fabs(oldx-x[i]);
33        }
34        if(e<epsilon) break;
35    }
36    if(k<=maxk)
37        output(x,n);
38    else
39        printf("\n迭代次数已超过上限。");
40    }
41
42 void input(double a[MAXSIZE][MAXSIZE],double b[MAXSIZE],long n) {
43    long i,j;
44    printf("\n请输入原方程组的增广矩阵: \n");
45    for(i=0; i<=n-1; i++) {
46        for(j=0; j<=n-1; j++)
47            scanf("%lf",&a[i][j]);
48        scanf("%lf",&b[i]);
49    }
50 }
51
52 void output(double x[MAXSIZE],long n) {
53    long i;
54    printf("\n原方程组的解向量为: \n");
55    for(i=0; i<=n-1; i++)
56        printf(" %lf\n",x[i]);

```

57 }



```

请输入原方程组的阶数: 3
请输入原方程组的增广矩阵:
1 0 -1 -2 7.2
-1 1 0 -2 8.3
-1 -1 5 4.2
请输入迭代初始向量: 0 0 0
请输入最大迭代次数: 300
请输入误差上界: 0.000007
原方程组的解向量为:
1.100000 1.200000 1.300000 请按任意键继续. . .

```

图 1.17: Gauss-Seidel 法迭代运行结果

一般来说, Gauss-Seidel 法迭代充分利用了迭代过程中产生的新的信息, 它的迭代效果要优于 Jacobi 迭代方法, 但情况并不总是这样, 有时候 Gauss-Seidel 方法迭代比 Jacobi 方法收敛更慢, 甚至可以举出 Jacobi 迭代收敛反而 Gauss-Seidel 方法发散的例子。

1.5.3 超松弛迭代技术

松弛法实质上是 Gauss-Seidel 迭代方法的一种加速方法, 这种方法将老的迭代值与 Gauss-Seidel 迭代的计算结果适当加权, 期望获得更好的近似值。

考察方程组 (1.132) 的 Gauss-Seidel 迭代公式, 据其第一个式子:

$$\tilde{x}_1^{(k+1)} = (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11} \quad (1.157)$$

求出迭代值 $\tilde{x}_1^{(k+1)}$ 后, 将它与老的值 $x_1^{(k)}$ 依松弛因子进行加权平均, 记改进值为 $x_1^{(k+1)}$

$$x_1^{(k+1)} = \omega \tilde{x}_1^{(k+1)} + (1 - \omega)x_1^{(k)} \quad (1.158)$$

第 2 步用改进值 $x_1^{(k+1)}$ 取代老值 $x_1^{(k)}$ 进行迭代, 求得:

$$\tilde{x}_2^{(k+1)} = (b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)})/a_{22} \quad (1.159)$$

然后再将迭代值 $\tilde{x}_2^{(k+1)}$ 与老值 $x_2^{(k)}$ 依松弛因子 ω 进行加权平均获得改进值为 $x_2^{(k+1)}$

$$x_2^{(k+1)} = \omega \tilde{x}_2^{(k+1)} + (1 - \omega)x_2^{(k)} \quad (1.160)$$

重复这种加工手续，第 3 步求得迭代值

$$\tilde{x}_3^{(k+1)} = (b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)})/a_{33} \quad (1.161)$$

然后进一步将它加工成改进值：

$$x_3^{(k+1)} = \omega\tilde{x}_3^{(k+1)} + (1 - \omega)x_3^{(k)} \quad (1.162)$$

这样设计出的迭代改进系统称作求解所给方程组的松弛迭代法。容易看出，Gauss-Seidel 迭代式松弛因子 $\omega = 1$ 的特殊情形，因此松弛迭代法可以看作是 Gauss-Seidel 迭代法的改进。为了保证松弛迭代法的收敛，要求松弛因子满足 $0 < \omega < 2$ 。通常情况下，常取 $1 < \omega < 2$ ，即所谓的超松弛法。

SOR 方法的关键在于选取合适的松弛因子，松弛因子的选择对收敛速度有极大的影响。

1.5.4 迭代法的收敛性

迭代法的收敛条件

定义 6. 对角占优矩阵定义 如果矩阵的每一行中，不在主对角线上的所有元素绝对值之和小于主对角线上元素的绝对值，即：

$$\sum_{j=1, j \neq i}^n |a_{ij}| < |a_{ii}| \quad i = 1, 2, 3, \dots, n \quad (1.163)$$

则称矩阵 A 按行严格对角占优。类似地，也有按列严格对角占优。

定理 9. 定理 若线性方程组 $Ax = b$ 的系数矩阵 A 按行严格对角占优，则雅克比迭代法和高斯-赛得尔迭代法对任意给定初值均收敛。

例 12. 解方程组

$$\begin{cases} 20x_1 + 2x_2 + 3x_3 = 24 \\ x_1 + 8x_2 + x_3 = 12 \\ 2x_1 - 3x_2 + 15x_3 = 30 \end{cases} \quad (1.164)$$

取 $x_1^0 = x_2^0 = x_3^0 = 0$ 。问：Jacobi 和 Gauss-Seidel 迭代法是否收敛？

解：由于

$$20 > 2 + 3 = 5 \quad (1.165)$$

$$8 > 1 + 1 = 2 \quad (1.166)$$

$$15 > 2 + |-3| = 5 \quad (1.167)$$

因此系数矩阵是按行严格对角占优矩阵，根据定理 Jacobi 和 Gauss-Seidel 迭代法应用于此方程组均收敛。

1.6 本章小结

本章主要介绍了解线性方程组的直接法。主要包括高斯消元法、矩阵分解法和追赶法。直接法是一种计算量小而精度高的方法(克莱姆算法也是一种直接法)。最具有代表性的算法是高斯(Gauss)消去法，其它算法大都是它的变型，这类方法是解具有稠密矩阵或非结构矩阵(零元分布无规律)方程组的有效方法。

选主元的算法有很好的数值稳定性。从计算简单出发实际中多选用列主元法。解三对角矩阵方程组(A 的对角元占优)的追赶法，解对称正定矩阵方程组的平方根法都是三角分解法，且都是数值稳定的方法，这些方法不选主元素，也具有较高的精度。

在实际应用中如何选择算法是一个重要问题，往往从解的精度、计算量和所需内存等三个方面考虑：但这些条件相互间是矛盾而不能兼顾的，因此实际计算时应根据问题的特点和要求及所用计算机的性能来选择算法。一般说，系数矩阵为中、小型满矩阵，用直接法较好；当系数矩阵为大型、稀疏矩阵时，有效的解法是前章讨论的迭代法。

线性方程组的迭代法，包括 Jacobi 方法、Gauss-Seidel 方法和超松弛迭代技术，另外还对上述方法进行了原理阐述和算法分析。

首先要把线性方程组表示成矩阵形式，设其系数矩阵为 A ，右端项为 b 。

- (1) 方程组是否能用 Jacobi 解， A 非奇异，则线性方程组有唯一解，这时才可以用 Jacobi 迭代；
- (2) Jacobi 迭代的矩阵表示，是把 A 分解为一个对角阵 D 、上三角阵 U 、下三角阵 L ， $A = D - L - U$ ，则 $M = D^{-1}(L + U)$ 称为 Jacobi 迭代矩阵 (D^{-1} 表示 D 的逆)。Jacobi 迭代是否收敛，取决于迭代矩阵 M ，而 M 只依赖于系数矩阵 A ；
- (3) 迭代矩阵 M 的所有特征值的模都小于 1，是 Jacobi 迭代收敛的充要条件；
- (4) M 的 1 范数、2 范数、无穷范数，这三者中的任何一个，如果小于 1，则 Jacobi 迭代收敛。范数越小，则收敛越快；
- (5) 若系数矩阵 A 按行(列)严格对角占优，且非奇异，则 Jacobi 迭代收敛。(如果 A 的对角线元素的模大于同行(列)中其他元素的模的和，则称 A 按行(列)严格对角占优)；

练习题

1. 编程实现 Doolittle 矩阵分解算法。
2. 编程实现高斯消元法。
3. 编程实现 Cholesky 算法。

4. 写出线性代数方程组

$$\begin{cases} 9x_1 - 2x_2 + x_3 = 6 \\ -x_1 + 8x_2 - x_3 = 8 \\ -x_1 + x_2 + 8x_3 = -3 \end{cases} \quad (1.168)$$

的 Gauss-Seidel 迭代式。

5. 使用 Doolittle 矩阵分解法求解下列线性方程组。

$$\begin{cases} 2x_1 + 10x_2 + 9x_3 = 5 \\ 3x_1 - x_2 + x_3 = 13 \\ -2x_1 - 2x_2 + 20x_3 = 3 \end{cases} \quad (1.169)$$

参考文献

- [1] Author. *Title*. <http://www.baidu.com>, 2019.
- [2] Author. *Title*. <http://www.baidu.com>, 2019.
- [3] Author. *Title*. <http://www.baidu.com>, 2019.
- [4] Author. *Title*. <http://www.baidu.com>, 2019.