空间信息常用数值计算方法 VC++ 实现

康建荣 编著

科学出版社

空间信息常用数值计算方法 VC++实现

康建荣 编著

科学出版社
北京

内 容 简 介

本书重点介绍空间信息技术常用数值计算方法的 VC++编程实现过程。全书分十章，主要包括解线性方程组的直接解法和迭代解法、拉格朗日插值、差分、差商、牛顿插值、埃尔米特插值、三次样条函数插值、梯形求积法、辛卜生求积法、龙贝格求积法、高斯求积法、三角形和四边形积分区域上二重积分、冒泡排序、选择排序、插入排序、快速排序、归并排序、希尔排序、堆排序、拓扑排序、基数排序、一元线性回归、多元线性回归、非线性回归、多项式回归、逐步回归分析、矢量叉积、折线段转向判断、点在线段内判断、两线段交点求取、点在区域内判断、点到线段垂直线交点求取、线段端点垂直点坐标计算、线段平行线求取、两线段平行线交点坐标计算、方位角计算、两边夹角计算、多边形方向判断、多边形自相交判断、任意多边形裁剪、凸包求取、曲线拟合、曲面拟合、矩形网格法和不规则三角网法绘制等值线、等值线光滑处理和等高值注记的编程实现算法。

本书每节内容分两部分，一部分是对算法原理的基本描述；另一部分是相应算法的 VC++实现，这部分也是本书的重点，是根据作者在科学研究过程中实际应用编写的算法实现代码。全书共提供了 137 个 VC++实现函数，所有算法都经过测试和应用。

本书可供从事测绘技术、土木工程设计以及工程应用领域的技术研究人员学习、参考，也可作为高等院校有关专业的教材或教学参考书。

图书在版编目(CIP)数据

空间信息常用数值计算方法 VC++实现/康建荣编著. —北京：科学出版社，2016.12

ISBN 978-7-03-051782-1

Ⅰ. ①空… Ⅱ. ①康… Ⅲ. ①空间信息技术-数值计算-C 语言-程序设计 Ⅳ. ①P208②TP312

中国版本图书馆 CIP 数据核字(2017)第 028871 号

责任编辑：周 丹 王 希/责任校对：赵桂芬

责任印制：张 倩/封面设计：许 瑞

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

新科印刷有限公司 印刷

科学出版社发行 各地新华书店经销

*

2016 年 12 月第 一 版 开本：787×1092 1/16

2016 年 12 月第一次印刷 印张：30 1/4

字数：720 000

定价：99.00 元

(如有印装质量问题，我社负责调换)

前　言

随着计算机技术的飞速发展，科学计算已应用到科学技术和社会生活的各个领域中。数值计算方法是工程领域非常重要的基础之一，计算已经成为不可缺少的工具。作者从事科学研究工作近三十年，深刻体会到计算方法在研究工作中的重要性，也深感到一本既有一定的理论描述又有详细算法实现的代码，使读者更容易、更深入地理解算法的参考书是十分必要的。通常我们在阅读参考文献时，往往会遇到理论描述较多，但如何实现没有给出，给参考者特别是初学者带来了困惑，不得不花较多时间去深入。本书的主旨就是让参考者既能明白理论知识，也能完全理解其实现的代码编写过程，从而减轻学习者的时间负担。

本书每节内容分两部分，一部分是对算法原理的基本描述，另一部分是相应算法的VC++实现，这部分也是本书的重点，是根据作者在科学研究过程中实际应用编写的算法实现代码。全书共提供了 137 个 VC++实现函数，所有算法都经过测试和应用，在编写代码时也考虑了算法的优化和效率。

在空间信息技术和工程应用领域，线性方程组求解是最常见的问题之一，其求解方法分为直接解法和迭代解法。通常来说，直接解法求解的精度较高，但在求解过程中需建立系数矩阵，当未知量数量较大时系数矩阵所需的存储空间非常大，往往会要求计算机内存较大，当计算机内存不能满足要求时，方程组就不能解算。但实际应用中，经常会遇到与某一未知量相关的其他未知量的数量不多，这时采用特殊存储结构可以减少系数矩阵对内存的占用，但不适合直接求解，因而迭代解法是这一问题的首选，但迭代法存在求解过程的收敛问题，故本书既给出了求解线性方程组常用的直接解法，同时也给出了迭代解法，并讨论了大型稀疏线性方程组求解的存储方法和实现算法，特别是，在迭代解法中增加了求解的收敛性判断，这样使迭代法应用起来更明确一些。

在插值算法中，除了根据通用方法编写了拉格朗日插值、差分、差商、牛顿插值、埃尔米特插值等算法外，特别地讨论了以一阶、二阶导数为参数的三次插值样条函数插值方法，改变了传统计算方法往往只以二阶导数为参数的分析，并且给出了非周期和周期两种情形的算法实现，目的就是使参考者更能深入理解三次样条插值方法。

在数值积分算法中，除讨论常用的一重积分算法外，主要增加了二重积分的算法实现，重点是增加了三角形、四边形和任意多边形积分区域上的二重积分的算法实现，从而使参考者可以更好地理解二重积分的数值分析方法。

排序是数值计算经常用到的算法，本书以模板类的方法设计了常用的九类排序算法，从而提高了算法的适用范围，提升了算法的使用效率。

回归分析方法是运用十分广泛的一种统计分析方法，本书对常用的回归分析方法：一元线性回归、多元线性回归、非线性回归、多项式回归和逐步回归分析进行了编程实现。

在现代工程和数学领域，计算几何在图形学、机器人技术、超大规模集成电路设计

和统计等诸多领域有着十分重要的应用。本书主要介绍计算几何常用算法，特别是深入分析两线段交点求取、点在区域内判断算法，增加点到线段垂直线交点求取、线段端点垂直点坐标计算、线段平行线求取、两线段平行线交点坐标计算、方位角计算、两边夹角计算的算法实现，提出多边形自相交判断算法，以链表数据结构实现 Weiler-Atherton 任意多边形裁剪算法，以分区网格方法实现凸包求取的快速实现算法。

在现代图形造型技术中，曲线拟合是一个重要的部分。在曲线拟合算法中，除通用方法外，重点讨论实用三次样条函数法的首末端补充点的方法、Bézier 函数法反求控制点的方法、B 样条函数法反求控制点的补充点条件方法及 deBoor 算法和基于矩阵表示的算法实现，从而使读者充分理解常用曲线拟合算法实现的关键问题。

曲面拟合法可以解决很多实际的工程问题。本书主要是针对离散点数据进行曲面拟合，主要介绍距离平方反比法、方位加权法、趋势面拟合法的算法实现，特别分析克里金法的算法实现方法。在趋势面拟合法中，给出任意次趋势面拟合的通用实现算法；在克里金法中，设计变异函数的参数求取算法，其目的就是使参考者深入理解算法的核心。

等值线是一种形和数的统一，在许多领域是成果数据表示的重要图件之一。常规的等值线绘制方法分为矩形网格法和不规则三角网法两种。矩形网格法具有算法简单的优点；对于非规则离散分布的特征点数据，也可直接建立非规则的网格，其中三角网法是较为简便和常用的方法。与规则网格法相比，不规则三角网法能够保持数据的原有精度，且程序简单、占用内存少，但是外推性较差，主要应用于专题地图等值线的自动绘制。本书特别设计了不规则三角网的三角形单元的存储结构，并根据三角网每一条边只能存在于相邻的两个三角形中的特点，编写了实现代码，不同于传统的设计方法，具有一定的特色，实现的效率明显提高。在等值线追踪过程中，可根据边所在的三角形序号，直接找到相邻三角形单元，不需进行查找比较，同时记录指向上坡的方向，为等高值注记明确了字头方向，使得算法效率较高。

基于形函数的图形校正模型和多边形区域信息最佳注记位置分析，也是作者认为在实际应用中常用的数值分析方法。在国民经济建设过程中，经常会用到地形方面图像和图形资料，一般这些资料经过摄像、扫描和矢量化后，往往产生平移、旋转、伸缩等变形，这种变形通常是复杂的非线性变形，为了能够正确使用资料，须对扫描图像和图形进行校正。本书提出一种基于形函数的快速图形校正方法，避免非线性问题的迭代解法，提高了解算速度且模型简单，并能有效地进行图形校正。注记是计算机地图制图和地理信息系统的关键问题之一。如在土地利用现状图中标注图斑号和土地类型，标注位置是这类问题的关键，直接影响着图的清晰和美观。土地利用现状的图斑在平面形状中是多边形，根据多边形的几何特点，分为凸多边形和凹多边形。本书基于土地利用现状图的信息注记对多边形的最佳标注位置进行分析。

本书得到国家自然科学基金项目(51574132)的资助，在此表示感谢！

本书的不足之处在所难免，敬请读者批评指正！

康建荣

2016 年 7 月

目　录

第 1 章　线性方程组求解算法

1.1　高斯消元法

1.1.1　算法原理

高斯消元法在数学原理上可等价于高斯-约当消元法，解线性方程组的算法思路就是将系数矩阵化成单位阵。具体做法就是逐次将第 i 行乘以某一系数，然后加到其后的所有行上，使第 i 行之后所有行的第 i 列的系数为零，这个过程也称为消元过程。

设有 n 阶线性方程组：

$$\boldsymbol{A}_{n\times n}\boldsymbol{X}_{n\times 1}=\boldsymbol{B}_{n\times 1}$$

其中，

$$\boldsymbol{A}=\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix},\quad \boldsymbol{X}=\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

解算此方程组的高斯消元法分两步进行。

第一步：消去过程

从 $k=0$ 到 $k=n-2$ 的循环：

(1) 若 $a_{k,k}^{(k)}\neq 0$，令 $t_{i,k}=a_{i,k}^{(k)}/a_{k,k}^{(k)}$ $(k=0,1,\cdots,n-2)$，$(i=k+1,k+2,\cdots,n-1)$

(2) 计算：$a_{i,j}^{(k+1)}=a_{i,j}^{(k)}-t_{i,k}\cdot a_{k,j}^{(k)}$ $(i=k+1,k+2,\cdots,n-1)$，$(j=k+1,k+2,\cdots,n-1)$

(3) 计算：$b_i^{(k+1)}=b_i^{(k)}-t_{i,k}\cdot b_k^{(k)}$ $(i=k+1,k+2,\cdots,n-1)$

第二步：回代过程

若 $a_{n-1,n-1}^{(n-1)}\neq 0$，则逐步回代得到原方程组的解：

(1) 计算：$x_{n-1}=b_{n-1}^{(n-1)}/a_{n-1,n-1}^{(n-1)}$

(2) 计算：$x_k=(b_k^{(k)}-\sum_{j=k+1}^{n-1}a_{k,j}^{(k)}\cdot x_j)/a_{k,k}^{(k)}$ $(k=n-2,n-3,\cdots,1,0)$

高斯消元法的算法复杂度是 $O(n^3)$，这就是说，如果系数矩阵是 $n\times n$，那么高斯消元法所需要的计算量大约与 n^3 成比例。

高斯消元法可用在任何域中。高斯消元法对于一些矩阵来说是稳定的，对于普遍的矩阵来说，高斯消元法在应用上通常也是稳定的，不过也有例外。

1.1.2　列主元高斯消元法

在实际应用中，经常会遇到系数矩阵中对角主元系数很小的方程组，即出现小主元，

这时将其作为除数进行消元，会带入大的舍入误差，这种舍入误差在计算过程中传播扩大而不能控制时，会对计算结果造成很不利的影响，造成计算解与其解相差甚远的情形，因而在高斯消元法中要避免出现小主元的情形，这就需要采用“选主元”技术。列主元就是在列中选取绝对值最大元的作为主元，然后通过行交换进行高斯消元法计算。

1.1.3 全选主元高斯消元法

为了使线性方程组解算更加稳定，经常将系数矩阵的对角线上主元交换为所在行与列中绝对值为最大值，这样就形成了全选主元的高斯消元法。具体做法就是从系数矩阵的第 k 行、第 k 列开始的右下角部分中选取绝对值最大的元素，并通过行交换与列交换将它交换到主元位置上(k=1, 2, …)。

通过选主元的方法弥补了顺序高斯消元法的弊端，由线性代数行列变换的知识知道，系数矩阵行变换时，方程的解不变，但是列变换时，未知数的顺序发生了交换，因此需要记录列交换的信息，以便求解后还原。

1.1.4 实现算法

1. 函数定义

返回值类型：BOOL

函数名：Gauss(顺序高斯消元法)

ColGauss(列主元高斯消元法)

MainGauss(全选主元高斯消元法)

函数参数：

int n　　——方程组个数；

double a[]　——$n\times n$ 的系数矩阵；

double b[]　——常数项向量；

double x[]　——求出的解向量。

2. 实现函数

1) 高斯消元法算法

```
BOOL Gauss(double a[], double b[], double x[],int n)
{
    //a[]是n×n个系数,b[]是n个常数,x[]是求出的n个解  a[i][j]=a[i*n+j]
    int i,j,k;
    double tik;
    //为了不改变系数矩阵和常数矩阵，设置动态数组
    double *aa=new double[n*n];
    double *bb=new double[n];
    for(i=0;i<n*n;i++)
        aa[i]=a[i];
```

```
    for(i=0;i<n;i++)
        bb[i]=b[i];
    for(k=0;k<n-1;k++)//从k=0到k=n-1的循环
    {
        if(fabs(aa[k*n+k])<1.0e-30)
            return FALSE;
        for(i=k+1;i<n;i++)
        {
            tik=aa[i*n+k]/aa[k*n+k];// 若a(k)(k,k)≠0，t(i,k)=a(i,k)/a(k,k)
            for(j=k+1;j<n;j++)//计算：a(k+1)(i,j)=a(k)(i,j)-t(i,k)·a(k)(k,j)
            aa[i*n+j]=aa[i*n+j]-tik*aa[k*n+j];
            aa[i*n+k]=0.0;
            bb[i]=bb[i]-tik*bb[k];// 计算：b(k+1)(i)=b(k)(i)-t(i,k)·b(k)(k)
        }
    }
    if(fabs(aa[(n-1)*n+n-1])<1.0e-30)
        return FALSE;
    x[n-1]=bb[n-1]/aa[(n-1)*n+n-1];// 计算：x(n-1)=b(n-1)(n-1)/a(n-1)(n-1,n-1)
    for(k=n-2;k>=0;k--)//计算：x(k)=(b(k)(k)-Σ(j=k+1,n-1) a(k)(k,j)·x(j))/a(k)(k,k)
    {
        double akj=0;
        for(j=k+1;j<n;j++)//计算：Σ(j=k+1,n-1) a(k)(k,j)·x(j)
            akj=akj+aa[k*n+j]*x[j];
        x[k]=(bb[k]-akj)/aa[k*n+k];
    }
    delete []aa;
    delete []bb;
    return TRUE;
}
```

2) 列主元高斯消元法算法

```
BOOL ColGauss(double a[], double b[], double x[], int n)
{
    int i,j,k;
    int maxrow;
    double tik,maxa,temp;
    double *aa=new double[n*n];
    double *bb=new double[n];
    for(i=0;i<n*n;i++)
        aa[i]=a[i];
```

```
for(i=0;i<n;i++)
    bb[i]=b[i];
for(k=0;k<n;k++)
{
    maxrow=k;
    maxa=aa[k*n+k];
    for(i=k+1;i<n;i++)//查找k行以下k列上的绝对值最大元素
    {
        if(fabs(aa[i*n+k])>fabs(maxa))
        {
            maxrow=i;
            maxa=aa[i*n+k];
        }
    }
    if(maxrow>k)//若找到，进行行交换
    {
        for(j=0;j<n;j++)
        {
            temp=aa[k*n+j];
            aa[k*n+j]=aa[maxrow*n+j];
            aa[maxrow*n+j]=temp;
        }
        temp=bb[k];//常向量进行行交换
        bb[k]=bb[maxrow];
        bb[maxrow]=temp;
    }
    //以下与顺序高斯消元法相同
    if(fabs(aa[k*n+k])<1.0e-30)
        return FALSE;
    for(i=k+1;i<n;i++)//i=2,n
    {
        tik=aa[i*n+k]/aa[k*n+k];
        for(j=k+1;j<n;j++)
            aa[i*n+j]=aa[i*n+j]-tik*aa[k*n+j];
        aa[i*n+k]=0.0;
        bb[i]=bb[i]-tik*bb[k];
    }
}
if(fabs(aa[(n-1)*n+n-1])<1.0e-30)
    return FALSE;
x[n-1]=bb[n-1]/aa[(n-1)*n+n-1];
for(k=n-2;k>=0;k--)
```

```
    {
        double akj=0;
        for(j=k+1;j<n;j++)
            akj=akj+aa[k*n+j]*x[j];
        x[k]=(bb[k]-akj)/aa[k*n+k];
    }
    delete []aa;
    delete []bb;
    return TRUE;
}
```

3) 全选主元高斯消元法算法

```
BOOL MainGauss(double a[], double b[], double x[], int n)
{
    int *js,k,i,j,is;
    double maxa,temp,tik;
    double *aa=new double[n*n+1];
    double *bb=new double[n+1];
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            aa[i*n+j]=a[i*n+j];
        bb[i]=b[i];
    }
    js=new int[n]; //开辟用于记忆列交换信息的动态空间
    for(k=0;k<n-1;k++)
    {
        is=k;
        js[k]=k;
        maxa=aa[k*n+k];
        for(i=k+1;i<n;i++)//查找k行以下k列以后区域内的绝对值最大元素
        {
            for(j=k+1;j<n;j++)
            {
                if(fabs(aa[i*n+j])>fabs(maxa))
                {
                    is=i;
                    js[k]=j;
                    maxa=aa[i*n+j];
                }
            }
        }
        if(is!=k)//若绝对值最大元素的行与k不同，则进行行交换
```

```
    {
        for(j=0;j<n;j++)
        {
            temp=aa[k*n+j];
            aa[k*n+j]=aa[is*n+j];
            aa[is*n+j]=temp;
        }
        temp=bb[k]; //常向量进行行交换
        bb[k]=bb[is];
        bb[is]=temp;
    }
    if(js[k]!=k)//若绝对值最大元素的列与k不同，则进行列交换
    {
        for(i=0;i<n;i++)
        {
            temp=aa[i*n+k];
            aa[i*n+k]=aa[i*n+js[k]];
            aa[i*n+js[k]]=temp;
        }
    }
    //以下与顺序高斯消元法相同
    if(fabs(aa[k*n+k])<1.0e-30)
        return FALSE;
    for(i=k+1;i<n;i++)//i=2,n
    {
        tik=aa[i*n+k]/aa[k*n+k];
        for(j=k+1;j<n;j++)
            aa[i*n+j]=aa[i*n+j]-tik*aa[k*n+j];
        aa[i*n+k]=0.0;
        bb[i]=bb[i]-tik*bb[k];
    }
}
if(fabs(aa[(n-1)*n+n-1])<1.0e-30)
    return FALSE;
x[n-1]=bb[n-1]/aa[(n-1)*n+n-1];
for(k=n-2;k>=0;k--)
{
    double akj=0;
    for(j=k+1;j<n;j++)
        akj=akj+aa[k*n+j]*x[j];
    x[k]=(bb[k]-akj)/aa[k*n+k];
}
```

```
    js[n-1]=n-1;
    for(k=n-1;k>=0;k--)//因为进行了列交换，则需恢复解向量顺序
    {
        if(js[k]!=k)
        {
            temp=x[k];
            x[k]=x[js[k]];
            x[js[k]]=temp;
        }
    }//解向量行变换
    delete []js;//释放动态空间
    delete []aa;
    delete []bb;
    return TRUE;//返回正常标志
}
```

1.1.5　验证实例

例 1：

```
    double a[]={1,2,3,4,1,4,9,16,1,8,27,64,1,16,81,256};
    double b[]={2,10,44,190};
    int n=4;
    double X[6];
    if(Gauss(a,b,X,n))
    {
        CString cs,ccs="";
        for(int i=0;i<n;i++)
        {
            cs.Format("x[%d]=%f\r\n",i,X[i]);
            ccs+=cs;
        }
        AfxMessageBox(ccs);
    }
    else
        AfxMessageBox("No!");
```

运行结果：x[0]=-1.0, x[1]=1.0, x[2]=-1.0, x[3]=1.0。

例 2：

```
    double a[]={0.2368,0.2471,0.2568,1.2671,0.1968,0.2071,1.2168,0.2271, 0.1581,
1.1675,0.1768,0.1871,1.1161,0.1254,0.1397,0.1490};
    double b[]={1.8471,1.7471,1.6471,1.5471};
```

运行结果：x[0]=1.040577, x[1]=0.987051, x[2]=0.935040, x[3]=0.881282。

1.2　矩阵三角分解法

1.2.1　算法公式

在解线性方程组时，可以将系数矩阵 $\boldsymbol{A}$ 分解为上三角阵 $\boldsymbol{U}$ 和单位下三角阵 $\boldsymbol{L}$，则求解 $\boldsymbol{A}_{n\times n}\boldsymbol{X}_{n\times 1}=\boldsymbol{B}_{n\times 1}$ 的问题就等价于求解两个三角形方程组：

(1) $\boldsymbol{L}\cdot\boldsymbol{y}=\boldsymbol{B}$，求 $\boldsymbol{y}$；

(2) $\boldsymbol{U}\cdot\boldsymbol{x}=\boldsymbol{y}$，求 $\boldsymbol{x}$。

分解后的系数矩阵为

$$\boldsymbol{A}=\begin{bmatrix}1 & & & \\ l_{1,0} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n-1,0} & l_{n-1,1} & \cdots & 1\end{bmatrix}\begin{bmatrix}u_{0,0} & u_{0,1} & \cdots & u_{0,n-1} \\ & u_{1,1} & \cdots & u_{1,n-1} \\ & & \ddots & \vdots \\ & & & u_{n-1,n-1}\end{bmatrix}$$

算法过程如下：

(1) $u_{0,i}=a_{0,i}\,(i=0,1,\cdots,n-1)$

$l_{i,0}=a_{i,0}/u_{0,0}\,(i=1,2,\cdots,n-1)$

(2) 计算 $\boldsymbol{U}$ 的第 r 行，$\boldsymbol{L}$ 的第 r 列元素。

$$u_{r,i}=a_{r,i}-\sum_{k=0}^{r-1}l_{r,k}\cdot u_{k,i}\ (r=1,2,\cdots,n-1),\ \ (i=r,r+1,\cdots,n-1)$$

$$l_{i,r}=(a_{i,r}-\sum_{k=0}^{r-1}l_{i,k}\cdot u_{k,r})/u_{r,r}\ (r=1,2,\cdots,n-1),\ \ (i=r+1,\cdots,n-1,\ \ r\neq n-1)$$

(3) 进行求解。

$$y_0=b_0$$

$$y_i=b_i-\sum_{k=0}^{i-1}l_{i,k}\cdot y_k\ (i=1,2,\cdots,n-1)$$

$$x_{n-1}=y_{n-1}/u_{n-1,n-1}$$

$$x_i=(y_i-\sum_{k=i+1}^{n-1}u_{i,k}\cdot x_k)/u_{i,i}\ (i=n-2,n-3,\cdots,1,0)$$

1.2.2　实现算法

1. 函数定义

返回值类型：BOOL

函数名：TriMatrix

函数参数：

```
int n        ——方程组个数；
double a[]   ——n×n的系数矩阵；
double b[]   ——常数项向量；
```

double x[]　——求出的解向量。

2. 实现函数

```
BOOL  TriMatrix(double a[], double b[], double x[],int n)
{
    double *l=new double[n*n+1];
    double  *u=new double[n*n+1];
    double  *y=new double[n+1];
    int i,k,r;
    for(i=0;i<n;i++)//计算 u_{0,i} = a_{0,i}
        u[i]=a[i];
    if(fabs(u[0])<1.0e-10)
        return FALSE;
    for(i=1;i<n;i++)//计算 l_{i,0} = a_{i,0} / u_{0,0}
        l[i*n]=a[i*n]/u[0];
    for(r=1;r<n;r++)
    {
        for(i=r;i<n;i++)//计算 u_{r,i} = a_{r,i} - sum_{k=0}^{r-1} l_{r,k} · u_{k,i}
        {
            double lu=0.0;
            for(k=0;k<=r-1;k++)
                lu+=(l[r*n+k]*u[k*n+i]);
            u[r*n+i]=a[r*n+i]-lu;
        }
        if(r!=n-1)
        {
            for(i=r+1;i<n;i++)//计算 l_{i,r} = (a_{i,r} - sum_{k=0}^{r-1} l_{i,k} · u_{k,r}) / u_{r,r}
            {
                double lu=0.0;
                for(k=0;k<=r-1;k++)
                    lu+=(l[i*n+k]*u[k*n+r]);
                if(fabs(u[r*n+r])<1.0e-10)
                    return FALSE;
                l[i*n+r]=(a[i*n+r]-lu)/u[r*n+r];
            }
        }
    }
    y[0]=b[0];
    for(i=1;i<n;i++)//计算 y_i = b_i - sum_{k=0}^{i-1} l_{i,k} · y_k
```