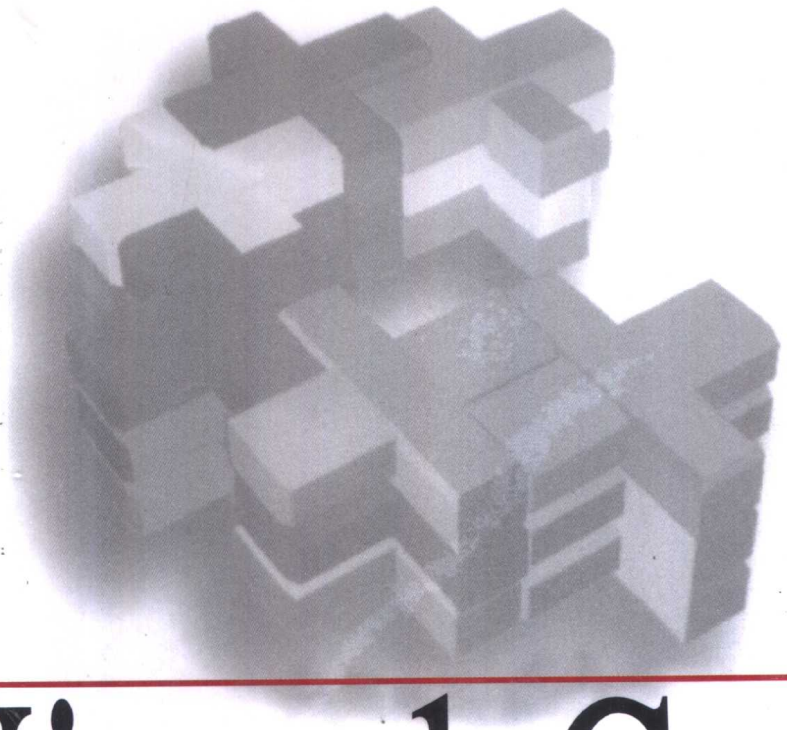
常 用 数 值 算 法 丛 书

内附源代



Visual C++

常用数值算法集

何光渝 编著

科学出版社

常用数值算法丛书

Visual C++常用数值算法集

何光渝　编著

科学出版社

2002

内 容 简 介

本书共有数值计算中常用的 Visual C++子过程近 200 个，内容包括：解线性代数方程组、插值、数值积分、特殊函数、函数逼近、随机数、排序、特征值问题、数据拟合、方程求根和非线性方程组求解、函数的极值和最优化、傅里叶变换谱方法、数据的统计描述、解常微分方程组、两点边值问题的解法和解偏微分方程组. 每一个子过程都包括功能、方法、使用说明、过程和例子五部分. 本书的所有子过程都在 Visual C++6.0 版本上进行过验证，程序都能正确运行. 同时配有光盘，包括所有子过程、验证过程及所有验证过程的 Visual C++工程项目.

本书可供大专院校师生和科研院所、工矿企业的工程技术人员使用.

图书在版编目(CIP)数据

Visual C++常用数值算法集/何光渝编著. —北京：科学出版社，2002

(常用数值算法丛书)

ISBN 7-03-010498-6

Ⅰ.V… Ⅱ.何… Ⅲ.C++语言—程序设计—数值计算—计算方法 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2002)第 037502 号

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

新蕾印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2002 年 7 月第 一 版 开本：720×1000 1/16

2002 年 7 月第一次印刷 印张：54 1/2

印数：1—5 000 字数：1 049 000

定价：78.00 元(含光盘)

(如有印装质量问题，我社负责调换〈路通〉)

序

在计算机技术迅猛发展的今天，面向对象技术日益成熟和完善，越来越多的非计算机专业的科研人员和工程技术人员，在各自的专业领域中采用各种类型的面向对象程序设计语言，设计出功能强大、实用的工程设计专业软件. 由于采用了面向对象技术中的编程机制和新颖、易用的可视化工具，此类软件简便、实用，易于推广，因而在科研单位、厂矿企业的科研开发和生产管理中发挥了重要作用.

科学研究和工程技术中的应用软件的编制需要较深的专业知识和丰富的实践经验，一般情况下只能由专业技术人员自己动手研制开发. 在科学研究和工程技术的专业应用软件的开发中，数值计算是必不可少的，数值计算中所使用的计算方法及其程序(简称算法)一方面以计算数学为理论依据，另一方面以计算机作计算工具. 近些年来，计算机专业科研人员大力研究、发明了各种新的算法，并不断地完善和标准化. 新的算法一经提出，并证明为有效后就被大量、广泛、重复地使用. 为了缩短应用软件的编制周期，减少重复劳动，发展计算方法，我们组织编写了这套丛书.

编写本套丛书的指导思想是：以目前最常用的面向对象技术的各类软件程序语言为平台，简略介绍算法的数学理论，偏重于程序；尽量汇编新的算法，也选编了有效的经典算法；对每一个算法都必须编制验证程序，并建立工程.

“常用数值算法丛书”将为千千万万非计算机专业的工程技术人员架起一座方便快捷的桥梁，带领读者轻松而快速地步入计算机应用的最新领域.

“常用数值算法丛书”是独树一帜的，它几乎包括了当前流行的所有面向对象技术软件，从理论到编程，从说明到实际应用，都编入丛书. 随着计算方法的不断发展，配合最新、最流行、最实用的软件，我们将不断推出新书以奉献给广大的读者.

“常用数值算法丛书”力求把最实用、最重要的知识讲清楚、讲透彻，引导读者轻松入门，迅速应用. 丛书中所有的算法都在计算机上验证通过，并随书发行光盘，省去了读者录入程序的繁琐，达到事半功倍的效果.

愿“常用数值算法丛书”能成为广大读者的有力工具，并衷心地希望广大读者对本丛书的不足或缺点提出批评，对今后的发展提出宝贵意见.

丛书编委会

前　言

目前，C 语言已成为高校和科研院所最常用的计算机语言之一，随着软件工程技术的不断发展，应用面向对象的编程技术已经成为当今软件开发的重要手段. 计算机可视化技术得到了越来越广泛的发展和应用，越来越多的计算机专业人员甚至非专业人员都开始从事可视化的研究和开发. 1998 年 Microsoft 公司推出了 Microsoft Visual Studio 6.0 组件，Microsoft Visual C++ 6.0 是其中得意的核心产品. 受到广大开发人员的青睐. 使用 Visual C++ 语言所编写的程序，与 Basic 和 Pascal 程序相比，具有速度快、移植性好等优势. 本书所介绍的常用数值算法，可以使人们在制作应用软件中减少不必要的重复劳动，节约时间，大大提高计算机使用效率.

本书共有数值计算中常用的 Visual C++子过程近 200 个. 内容包括：解线性代数方程组、插值、数值积分、特殊函数、函数逼近、随机数、排序、特征值问题、数据拟合、方程求根和非线性方程组求解、函数的极值和最优化、数据的统计描述、傅里叶变换谱方法、解常微分方程组、两点边值问题的解法和解偏微分方程组. 每一个子过程都包括功能、方法、使用说明、子过程和例子五部分. 每种算法都给出了例子和验证程序，帮助读者了解如何调用子过程、怎样输入数据，并得到计算结果. 本书的所有子过程都在 Visual C++ 6.0 版本上进行了验证，准确无误.

配合本书的出版，将同时发行书中全部子过程、验证程序及每种数值算法的 Visual C++工程项目的光盘. 一旦查阅书中的子过程，遇到困难，请直接调用光盘中的工程，仔细浏览即能解决问题，同时也省去读者敲击程序的繁琐.

由于编者水平有限，缺点错误在所难免，恳请广大读者批评指正.

编　者

目　录

第 1 章　线性代数方程组的解法

本章包括线性代数方程组的求解、矩阵求逆、行列式计算、奇异值分解和线性最小二乘问题等的算法和子过程，所给算法具有广泛的适用性和很强的通用性.

1. 一般实矩阵

高斯-约当全主元消去法(见 1.1 节)具有数值稳定的特点，所给过程在得到解的同时还得到系数矩阵的逆，但计算量大，对于方程组阶数不高而要求精度较高时，可采用此方法；*LU* 分解法采用隐式的部分选主元方法，数值稳定性好，存储量小，特别对于要解系数矩阵相同的多个方程组时最为适用，它还可用于求矩阵的逆和行列式. *LU* 分解法的计算量大约是 $n^3/3$，与列主元消去法相当，而高斯-约当消去法的计算量大约是它们的 3 倍，即大约是 n^3. 对于对称矩阵，特别是正定矩阵宜采用乔列斯基分解法(见 1.10 节)，它的程序简单，计算量小. *QR* 分解法即正交三角分解法(见 1.11 节)，由于其数值稳定性非常好，因此现在已越来越多地应用于各种数值求解中，现常用 *QR* 分解代替 *LU* 分解. 缺点是计算量和存储量均较大，计算速度亦较慢.

2. 病态矩阵

病态矩阵即条件数很大的矩阵. 对于病态矩阵，高斯消去法和 *LU* 分解法都不能给出满意的结果，*QR* 方法有时也同样不能给出满意的解，通常采用以下的处理办法：

(1) 增加计算的有效位数，如采用双精度(双倍字长)计算，这是一个比较有效的措施. 但这样做会使计算时间增加，且所需存储单元也会增到近两倍.

(2) 采用迭代改善的办法(见 1.5 节)，它是成功地改进解的精度的办法之一. 该方法的基本思想是在消去法的基础上利用迭代逐步改善方程组的解(关键在于在迭代过程中有些运算必须用双精度).

(3) 采用奇异值分解(*SVD*)法或共轭斜量法(见 1.8 和 1.9 节). 实验表明，共轭斜量法对病态矩阵常常是一种有效的方法.

3. 特殊形式的矩阵

这里包括三对角矩阵(见 1.3 节)和五对角矩阵(见 1.4 节)的追赶法、范德蒙矩阵的 G. Rybicki 方法和托伯利兹矩阵的 Rybicki 推广的 Levinson 方法(见 1.6 和 1.7 节). 对于以这些特殊矩阵为系数矩阵的方程组,若用一般矩阵的方法,效率太低,时间和空间的浪费也很大,因此对它们有专门有效的方法.

4. 稀疏矩阵

对于大稀疏矩阵的方程组,常用迭代法求解,这里我们给出两种迭代法:共轭斜量法和松弛迭代法(见 1.9 和 1.12 节). 它们均不要求矩阵具有任何特殊结构,因此可用于一般稀疏矩阵方程组的求解. 其中松弛迭代法当取松弛因子为 1.0 时,即为高斯-塞德尔迭代法. 当然要注意迭代可能不收敛. 具体应用可参考第 16 章.

5. 奇异值分解(*SVD*)和最小二乘问题

SVD 对于奇异矩阵或数值上很接近奇异的矩阵是一个非常有效的方法,它可以精确地诊断问题. 在某些情形下,*SVD* 不仅诊断问题,而且还能解决问题.

对于最小二乘问题,*SVD* 也是一个常选用的方法.

对于解方程组,*LU* 分解法和 *SVD* 都是先对系数矩阵作分解,然后再用分解矩阵求解. 它们的重大差别是用 *SVD* 解方程组之前即调用子过程 SVBKSB 之前要对奇异值进行剪辑,请参考 SVBKSB 的验证程序 D1R8. *SVD* 的用法细节可参考第 9 章.

1.1 全主元高斯-约当(Gauss-Jordan)消去法

1. 功能

用高斯-约当消去法求解 $\boldsymbol{A}[\boldsymbol{XY}]=[\boldsymbol{BI}]$,其中 $\boldsymbol{A}$ 为 $n\times n$ 非奇异矩阵,$\boldsymbol{B}$ 为 $n\times m$ 矩阵,均已知;$\boldsymbol{X}_{n\times m}$,$\boldsymbol{Y}_{n\times n}$未知. 由于消去过程是在全矩阵中选主元(绝对值最大的元素)来进行的,故可使舍入误差对结果的影响减到最小.

2. 方法

(1) 施行初等变换把 $\boldsymbol{A}$ 变为单位矩阵,则

$$\boldsymbol{X}=\boldsymbol{A}^{-1}\boldsymbol{B},\quad \boldsymbol{Y}=\boldsymbol{A}^{-1}$$

(2) 算法. 记

$$\boldsymbol{A}^{(0)} = (a_{ij}^{(0)}) = \boldsymbol{A} = (a_{ij}), \quad \boldsymbol{b}^{(0)} = \boldsymbol{b} = (b_{ij}^{(0)})$$

第 k 步的矩阵为 $\boldsymbol{A}^{(k)} = (a_{ij}^{(k)})_{n\times n}$, $\boldsymbol{b} = (b_{ij}^{(k)})_{n\times m}(k = 1, \cdots, n)$.

第 k 步的计算为

① 选主元,设为 $a_{i_0j_0}^{(k-1)}$.

② 若 $i_0 = j_0$,则转③,否则交换矩阵$[\boldsymbol{A}^{(k-1)}\boldsymbol{B}^{(k-1)}]$的第 i_0 行与第 j_0 行,则 $a_{i_0j_0}^{(k-1)}$ 移至矩阵 $\boldsymbol{A}^{(k-1)}$的对角线上,得到的矩阵仍记为$[\boldsymbol{A}^{(k-1)}\boldsymbol{B}^{(k-1)}] = [(a_{ij}^{(k-1)}) \cdot (b_{ij}^{(k-1)})]$,主元为 $a_{j_0j_0}^{(k-1)}$.

③ 消元过程计算公式:

$$\begin{aligned}
&p_k = 1/a_{j_0i_0}^{(k-1)} \\
&a_{j_0j}^{(k)} = a_{j_0j}^{(k-1)} \cdot p_k, && j = 1, \cdots, n \\
&b_{j_0l}^{(k)} = b_{j_0l}^{(k-1)} \cdot p_k, && l = 1, \cdots, m \\
&a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{j_0j}^{(k)} \cdot a_{ij_0}^{(k-1)}, && j = 1, \cdots, n \\
&b_{il}^{(k)} = b_{ij}^{(k-1)} - b_{j_0l}^{(k)} \cdot a_{ij_0}^{(k-1)}, && l = 1, \cdots, m \\
&i = 1, \cdots, n && i \neq j_0
\end{aligned}$$

3. 使用说明

GAUSSJ (A[],N,B[])

N　整型变量,输入参数,矩阵 $\boldsymbol{A}$ 的阶数

A[]　实型数组,输入、输出参数,输入时按列存放实方阵 $\boldsymbol{A}$,计算结束时输出逆矩阵 $\boldsymbol{A}^{-1}$

B[]　实型数组,输入、输出参数,输入时按列存放实方阵 $\boldsymbol{B}$,计算结束时输出解 $\boldsymbol{A}^{-1}\boldsymbol{B}$

4. 过程

子过程 GAUSSJ.

```
void gaussj(double a[], int n, double b[])
{
  int i,j,k,l,ll,irow,icol;
  double big,pivinv,dum;
  int ipiv[50], indxr[50], indxc[50];
  for (j=0; j<=n-1; j++)
```

```
{
  ipiv[j]=0;
}
for (i=0; i<=n-1; i++)
{
  big=0.0;
  for (j=0; j<=n-1; j++)
  {
    if (ipiv[j]! =1)
    {
      for (k=0; k<=n-1; k++)
      {
        if (ipiv[k]==0)
        {
          if (fabs(a[j*n+k])>=big)
          {
            big=fabs(a[j*n+k]);
            irow=j;
            icol=k;
          }
          else if(ipiv[k]>1)
          {
            cout<<"singular matrix";
          }
        }
      }
    }
  }
  ipiv[icol]=ipiv[icol]+1;
  if (irow! =icol)
  {
    for(l=0; l<=n-1; l++)
    {
      dum=(a[irow*n+l]);
      a[irow*n+l]=a[icol*n+l];
      a[icol*n+l]=dum;
    }
```

```
      dum=b[irow];
      b[irow]=b[icol];
      b[icol]=dum;
    }
    indxr[i]=irow;
    indxc[i]=icol;
    if (a[icol * n+icol]==0.0)
    {
      cout<< "singular matrix.";
    }
    pivinv=1.0/(a[icol * n+icol]);
    a[icol * n+icol]=1.0;
    for(l=0; l<=n-1; l++)
    {
      a[icol * n+l]=a[icol * n+l] * pivinv;
    }
    b[icol]=b[icol] * pivinv;
    for(ll=0; ll<=n-1; ll++)
    {
      if(ll! =icol)
      {
        dum=a[ll * n+icol];
        a[ll * n+icol]=0.0;
        for(l=0; l<=n-1; l++)
        {
          a[ll * n+l]=a[ll * n+l]-a[icol * n+l] * dum;
        }
        b[ll]=b[ll]-b[icol] * dum;
      }
    }
  }
  for(l=n-1;l<=0;l--)
  {
    if(indxr[l]! =indxc[l])
    {
      for(k=0; k<=n-1; k++)
      {
```

```
            dum=a[k*n+indxr[l]];
            a[k*n+indxr[l]]=a[k*n+indxc[l]];
            a[k*n+indxr[l]]=dum;
          }
        }
      }
    }
```

5. 例子

验证程序 D1R1 调用子过程 GAUSSJ 可以对例子中的矩阵求出其解，并将解乘以已知系数矩阵检查是否和方程右端的向量相等. 验证程序 D1R1 如下：

```
#include "iostream.h"
#include "math.h"

void gaussj(double a[], int n, double b[])
{
  int i,j,k,l,ll,irow,icol;
  double big,pivinv,dum;
  int ipiv[50], indxr[50], indxc[50];
  for (j=0; j<=n-1; j++)
  {
    ipiv[j]=0;
  }
  for (i=0; i<=n-1; i++)
  {
    big=0.0;
    for (j=0; j<=n-1; j++)
    {
      if(ipiv[j]! =1)
      {
        for (k=0; k<=n-1; k++)
        {
          if(ipiv[k]==0)
          {
            if(fabs(a[j*n+k])>=big)
            {
```

```
            big=fabs(a[j * n+k]);
            irow=j;
            icol=k;
          }
          else if(ipiv[k]>1)
          {
            cout<<"singular matrix";
          }
        }
      }
    }
  }
  ipiv[icol]=ipiv[icol]+1;
  if(irow! =icol)
  {
    for(l=0;l<=n-1;l++)
    {
      dum=(a[irow * n+l]);
      a[irow * n+l]=a[icol * n+l];
      a[icol * n+l]=dum;
    }
    dum=b[irow];
    b[irow]=b[icol];
    b[icol]=dum;
  }
  indxr[i]=irow;
  indxc[i]=icol;
  if(a[icol * n+icol]==0.0)
  {
    cout<< "singular matrix.";
  }
  pivinv=1.0/(a[icol * n+icol]);
  a[icol * n+icol]=1.0;
  for(l=0;l<=n-1;l++)
  {
    a[icol * n+l]=a[icol * n+l] * pivinv;
  }
```

```
    b[icol]=b[icol]*pivinv;
    for(ll=0;ll<=n-1;ll++)
    {
      if(ll!=icol)
      {
        dum=a[ll*n+icol];
        a[ll*n+icol]=0.0;
        for(l=0;l<=n-1;l++)
        {
          a[ll*n+l]=a[ll*n+l]-a[icol*n+l]*dum;
        }
        b[ll]=b[ll]-b[icol]*dum;
      }
    }
  }
  for(l=n-1;l<=0;l--)
  {
    if(indxr[l]!=indxc[l])
    {
      for(k=0;k<=n-1;k++)
      {
        dum=a[k*n+indxr[l]];
        a[k*n+indxr[l]]=a[k*n+indxc[l]];
        a[k*n+indxr[l]]=dum;
      }
    }
  }
}

void main()
{
  //program d1r1
  //driver program for routine gaussj
  int i,j,l,n;
  n=3;
  double a[3][3], b[3], a1[9], b1[3];
  //输入已知的方程组的系数矩阵
```