

常 用 数 值 算 法 丛 书



Java

常 用 数 值 算 法 集

何光渝 高永利 编著



科 学 出 版 社

常用数值算法丛书

Java 常用数值算法集

何光渝 高永利 编著

科学出版社

北 京

内 容 简 介

本书共有数值计算中常用的 Java 方法近 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,每一个方法都包括功能、算法、使用说明、方法和例子五部分,本书的所有方法都在 Java 2 开发工具包 JDK 1.3 版本上进行了验证,准确无误,配书同时发行光盘,包括所有方法和验证程序。

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

Java 常用数值算法集/何光渝,高永利编著. —北京:科学出版社, 2003

(常用数值算法丛书)

ISBN 7-03-010928-7

I. J... II. ①何... ②高... III. JAVA 语言—数值计算—计算方法 N. TP312

中国版本图书馆 CIP 数据核字(2002)第 084915 号

责任编辑:陈晓萍/责任校对:赵慧玲

责任印制:吕春珉/封面设计:科学设计集体

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

*

2003 年 1 月第 一 版 开本:720×1000 1/16

2003 年 1 月第一次印刷 印张:57 1/2

印数:1 4 000 字数:1 080 000

定价:80.00 元(含光盘)

(如有印装质量问题,我社负责调换〈环伟〉)

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

Java 语言是目前推广速度最快的程序设计语言,它采用面向对象编程技术,功能强大而又简单易学,深受广大程序设计人员的偏爱. Java 伴随着 Internet 的发展而成熟. Java 是精心设计的语言,它具有简单性、面向对象性、平台无关性、安全性和健壮性等诸多特点,内置了多线程和网络支持能力,可以说它是网络世界的通用语言. 2000 年 5 月 Sun 推出了 Java 2 开发工具包 JDK 1.3,受到广大开发人员的青睐. 该软件可以免费下载安装,下载地址为:<http://java.sun.com/products/jdk>. 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,节约了时间,大大提高计算机使用效率.

本书共有数值计算中常用的 Java 方法近 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. 每一个方法都包括功能、算法、使用说明、方法和例子五部分. 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方法,怎样输入数据,得到计算结果. 本书的所有方法都在 JDK 1.3 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方法、验证程序的光盘. 一旦应用书中的方法,遇到困难,请直接调用光盘中的验证程序,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

2002 年 8 月

目 录

序

前言

第 1 章 线性代数方程组的解法	1
1.1 全主元高斯-约当(Gauss-Jordan)消去法	2
1.2 LU 分解法	8
1.3 追赶法	16
1.4 五对角线性方程组解法	20
1.5 线性方程组解的迭代改善	27
1.6 范德蒙(Vandermonde)方程组解法	31
1.7 托伯利兹(Toeplitz)方程组解法	37
1.8 奇异值分解	44
1.9 线性方程组的共轭梯度法	60
1.10 对称方程组的乔列斯基(Cholesky)分解法	67
1.11 矩阵的 QR 分解	74
1.12 松弛迭代法	82
第 2 章 插值	88
2.1 拉格朗日插值	89
2.2 有理函数插值	94
2.3 三次样条插值	99
2.4 有序表的检索法	109
2.5 插值多项式	120
2.6 二元拉格朗日插值	131
2.7 双三次样条插值	134
第 3 章 数值积分	141
3.1 梯形求积法	142
3.2 辛普森(Simpson)求积法	148
3.3 龙贝格(Romberg)求积法	150
3.4 反常积分	154
3.5 高斯(Gauss)求积法	168

3.6	三重积分	175
第 4 章	特殊函数	183
4.1	Γ 函数、贝塔函数、阶乘及二项式系数	183
4.2	不完全 Γ 函数、误差函数	199
4.3	不完全贝塔函数	222
4.4	零阶、一阶和任意整数阶的第一、二类贝塞尔函数	228
4.5	零阶、一阶和任意整数阶的第一、二类变形贝塞尔函数	251
4.6	分数阶第一类贝塞尔函数和变形贝塞尔函数	273
4.7	指数积分和定指数积分	291
4.8	连带勒让德函数	301
第 5 章	函数逼近	321
5.1	级数求和	321
5.2	多项式和有理函数	325
5.3	切比雪夫逼近	333
5.4	积分和导数的切比雪夫逼近	341
5.5	用切比雪夫逼近求函数的多项式逼近	348
第 6 章	随机数	356
6.1	均匀分布随机数	356
6.2	变换方法——指数分布和正态分布随机数	373
6.3	舍选法—— Γ 分布、泊松分布和二项式分布随机数	382
6.4	随机位的产生	396
6.5	蒙特卡罗积分法	405
第 7 章	排序	408
7.1	直接插入法和 Shell 方法	408
7.2	堆排序	422
7.3	索引表和等级表	433
7.4	快速排序	447
7.5	等价类的确定	454
第 8 章	特征值问题	463
8.1	对称矩阵的雅可比变换	464
8.2	变实对称矩阵为三对角对称矩阵	476
8.3	三对角矩阵的特征值和特征向量	483
8.4	变一般矩阵为赫申伯格矩阵	491
8.5	实赫申伯格矩阵的 QR 算法	501

第 9 章 数据拟合	511
9.1 直线拟合	511
9.2 线性最小二乘法	517
9.3 非线性最小二乘法	546
9.4 绝对值偏差最小的直线拟合	563
第 10 章 方程求根和非线性方程组的解法	571
10.1 图解法	571
10.2 逐步扫描法和二分法	575
10.3 割线法和试位法	586
10.4 布伦特(Brent)方法	594
10.5 牛顿-拉斐森(Newton-Raphson)法	600
10.6 求复系数多项式根的拉盖尔(Laguerre)方法	608
10.7 求实系数多项式根的贝尔斯托(Bairstou)方法	625
10.8 非线性方程组的牛顿-拉斐森方法	630
第 11 章 函数的极值和最优化	637
11.1 黄金分割搜索法	637
11.2 不用导数的布伦特(Brent)法	648
11.3 用导数的布伦特(Brent)法	656
11.4 多元函数的下山单纯形法	666
11.5 多元函数的包维尔(Powell)法	675
11.6 多元函数的共轭梯度法	685
11.7 多元函数的变尺度法	692
第 12 章 傅里叶变换谱方法	699
12.1 复数据快速傅里叶变换算法	699
12.2 实数据快速傅里叶变换算法(一)	708
12.3 实数据快速傅里叶变换算法(二)	715
12.4 快速正弦变换和余弦变换	723
12.5 卷积和逆卷积的快速算法	735
12.6 离散相关和自相关的快速算法	741
12.7 多维快速傅里叶变换算法	746
第 13 章 数据的统计描述	752
13.1 分布的矩——均值、平均差、标准差、方差、斜差和峰态	752
13.2 中位数的搜索	757
13.3 均值与方差的显著性检验	765

13.4	分布拟合的 χ^2 检验	782
13.5	分布拟合的 K-S 检验法	790
第 14 章	解常微分方程组	801
14.1	定步长四阶龙格-库塔(Runge-Kutta)法.....	801
14.2	自适应变步长的龙格-库塔法	809
14.3	改进的中点法.....	821
14.4	外推法.....	827
第 15 章	两点边值问题的解法	846
15.1	打靶法(一).....	846
15.2	打靶法(二).....	855
15.3	松弛法.....	865
第 16 章	偏微分方程的解法	891
16.1	解边值问题的松弛法.....	891
16.2	交替方向隐式方法(ADI)	897
参考文献		907
编后记		908

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方法:

(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一办法之一. 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 SVBKS B 之前要对奇异值进行剪辑, 请参考 SVBKS B 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 $A[XY]=[BI]$, 其中 A 为 $n \times n$ 非奇异矩阵, B 为 $n \times m$ 矩阵, 均已知; $X_{n \times m}, Y_{n \times n}$ 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. 算法

(1) 施行初等变换把 A 变为单位矩阵, 则

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

(2) 算法. 记

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

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

第 k 步的计算为

① 选主元, 设为 $a_{i_0 j_0}^{(k-1)}$.

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

③ 消元过程计算公式:

$$p_k = 1/a_{j_0 j_0}^{(k-1)}$$

$$a_{j_0 j}^{(k)} = a_{j_0 j}^{(k-1)} \cdot p_k, \quad j = 1, \dots, n$$

$$b_{j_0 l}^{(k)} = b_{j_0 l}^{(k-1)} \cdot p_k, \quad l = 1, \dots, m$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{j_0 j}^{(k)} \cdot a_{i j_0}^{(k-1)}, \quad j = 1, \dots, n$$

$$b_{il}^{(k)} = b_{il}^{(k-1)} - b_{j_0 l}^{(k)} \cdot a_{i j_0}^{(k-1)}, \quad l = 1, \dots, m$$

$$i = 1, \dots, n \quad i \neq j_0$$

3. 使用说明

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

N 整型变量, 输入参数, 矩阵 $\mathbf{A}$ 的阶数

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

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

4. 方法

方法 GAUSSJ.

```
public class d1r1F
{
    void gaussj(double a[], int n, double b[])
    {
        int i, j, k, l, ll, irow, icol;
        icol = 0;
        irow = 0;
```

```

double big, pivinv, dum;
int ipiv[] = new int[50];
int indxr[] = new int[50];
int indxc[] = new int[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 (Math.abs(a[j * n + k]) >= big)
                    {
                        big = Math.abs(a[j * n + k]);
                        irow = j;
                        icol = k;
                    }
                }
                else if (ipiv[k] > 1)
                {
                    {
                        System.out.println("singular matrix");
                    }
                }
            }
        }
    }
    ipiv[icol] = ipiv[icol] + 1;
    if (irow != icol)
    {
        for (l = 0; l <= n - 1; l++)

```

```

    {
        dum = (a[irow * n + 1]);
        a[irow * n + 1] = a[icol * n + 1];
        a[icol * n + 1] = dum;
    }
    dum = b[irow];
    b[irow] = b[icol];
    b[icol] = dum;
}
indxr[i] = irow;
indx[i] = icol;
if (a[icol * n + icol] == 0.0)
{
    System.out.println("singular matrix.");
    System.exit(1);
}
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 + indxc[l]] = dum;
            }
        }
    }
}
}
}

```

5. 例子

验证程序 D1R1 调用方法 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:

```

import java.text. * ;
public class d1r1
{
    public static void main (String[] args)
    {
        //program d1r1
        //driver program for routine gaussj
        int i, j, l, p=0;
        int n = 3;
        d1r1F g = new d1r1F();
        double a[][] = new double[4][4];
        double b[] = new double[4];
        double b1[] = new double[4];
        double a1[] = new double[10];
        DecimalFormat form = new DecimalFormat("0.000000");
        //输入已知的方程组的系数矩阵
        a[0][0] = 2; a[0][1] = 1; a[0][2] = 2;
        a[1][0] = 5; a[1][1] = -1; a[1][2] = 1;
    }
}

```

```
a[2][0] = 1; a[2][1] = -3; a[2][2] = -4;
//输入已知的方程组的右端向量
b[0] = 5;
b[1] = 8;
b[2] = -4;
System.out.println();
System.out.println("已知的方程组的右端向量");
System.out.println("  " + form.format(b[0]));
    System.out.println("  " + form.format(b[1]));
    System.out.println("  " + form.format(b[2]));
for (i = 0; i <= n - 1; i++)
{
    for (j = 0; j <= n - 1; j++)
    {
        a1[i * n + j] = a[i][j];
    }
}
g.gaussj(a1, n, b);
System.out.println();
System.out.println("计算出的方程组的解");
System.out.println("  " + form.format(b[0]));
System.out.println("  " + form.format(b[1]));
    System.out.println("  " + form.format(b[2]));
//将计算出的 b 乘以系数矩阵,以验证计算结果正确
for (l = 0; l <= n - 1; l++)
{
    b1[l] = 0.0;
    for (j = 0; j <= n - 1; j++)
    {
        b1[l] = b1[l] + a[l][j] * b[j];
    }
}
System.out.println();
System.out.println("计算出的解乘以系数矩阵的结果");
System.out.println("  " + form.format(b1[0]));
    System.out.println("  " + form.format(b1[1]));
System.out.println("  " + form.format(b1[2]));
```

```

    }
}

```

计算结果如下:

已知的方程组的右端向量

```

5.00000
8.00000
-4.00000

```

计算出的方程组的解

```

1.00000
-1.00000
2.00000

```

计算出的解乘以系数矩阵的结果

```

5.00000
8.00000
-4.00000

```

1.2 LU 分解法

1. 功能

求解系数矩阵为非奇异的线性代数方程组 $Ax=b$, 它能串联式地逐次解 A 相同 b 不同的方程组. 本方法也叫杜利特尔(Doolittle)方法, 它将高斯主元消去法中的中间结果的记录次数从约 $n^3/3$ 次减少到约 n^2 次. 方法 LUDCMP 将系数矩阵 A 分解为上三角矩阵和下三角矩阵. 方法 LUBKSB 利用 LUDCMP 的分解结果求得线性方程组 $Ax=b$ 的解.

2. 算法

(1) 采用隐式部分选主元的杜利特尔方法.

(2) 首先作 A 的 LU 分解:

$$A = LU = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ & \dots & & \dots & \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & \dots & & \dots & \\ & & & & u_{nn} \end{bmatrix}$$