

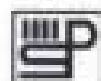
常 用 数 值 算 法 丛 书



Visual Fortran

常 用 数 值 算 法 集

何光渝 高永利 编著



科 学 出 版 社

(TP-1723.0101)

责任编辑: 陈晓萍

封面设计: 王 磊

· 常 用 数 值 算 法 丛 书 ·

Delphi

常用数值算法集

Visual Basic

常用数值算法集

Visual C++

常用数值算法集

▶▶ Visual Fortran

常用数值算法集

全书列举了数值计算中常用的Visual Fortran子过程近200个

解线性代数方程组

数据拟合

插值

方程求根和非线性方程组求解

数值积分

函数的极值和最优化

特殊函数

傅里叶变换谱方法

函数逼近

数据的统计描述

随机数

解常微分方程组

排序

两点边值问题的解法

特征值问题

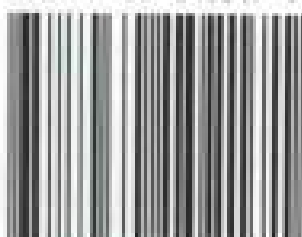
解偏微分方程组

每一个子程序都包括功能、方法、使用说明、子程序和例子五部分

所有子过程都在Visual Fortran 5.0 版本上进行过验证, 程序都能准确运行

内附光盘, 包括所有子过程、验证过程及所有验证这些过程的Visual Fortran工程项目

ISBN 7-03-010217-7



9 787030 102171 >

ISBN 7-03-010217-7/TP · 1723

定价: 65.00 元 (含光盘)

常用数值算法丛书

Visual Fortran 常用数值算法集

何光渝 高永利 编著



A0970594

科学出版社

2002

内 容 简 介

本书共有数值计算中常用的 Visual Fortran 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. 每一个子程序都包括功能、方法、使用说明、子程序和例子五部分. 本书的所有子过程都在 Visual Fortran 5.0 版本上进行过验证, 程序都能正确运行. 同时配书发行光盘, 包括所有子过程、验证过程及所有验证过程的 Visual Fortran 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 Fortran 常用数值算法集/何光渝, 高永利编著. —北京: 科学出版社, 2002

(常用数值算法丛书)

ISBN 7-03-010217-7

I. V... II. ①何... ②高... III. 数值计算—应用软件, Visual Fortran N. 0245

中国版本图书馆 CIP 数据核字(2002)第 12623 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 4 月第 1 版 开本: 720×1000 1/16

2002 年 4 月第一次印刷 印张: 44

印数: 1—4 000 字数: 816 000

定价: 65.00 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,从说明到实际使用,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.

前 言

Fortran 语言是第一个世界通用的计算机高级程序设计语言,由于它非常接近于自然语言(英语)和数学表达式,因而问世 40 余年来,一直被广泛应用,特别是在科学和工程计算领域,始终占据着主要地位.多年来,工程界各行业的技术人员编写出了大量实用的算法和工程应用软件.有的工程商业软件多达十多万行,至今仍在应用.随着计算机技术的迅速发展,Windows 已经逐渐取代了 DOS 而成为新一代操作系统.为了发挥 Fortran 在科学计算领域的优势,人们开发出像 Visual C++(以下简称 VC)甚至 Visual Basic(以下称简 VB)一样的基于 Windows 的应用程序.1997 年 Digital 公司正式推出 Digital Visual Fortran 5.0.显然,这为工程技术界进行科学计算和编写面向对象的工程实用软件的用户提供了极大的方便.熟悉 VB 或 VC 的读者可以很容易地掌握 Visual Fortran 的使用,进一步开发出自己专业领域的 Windows 下的界面友好的工程应用软件.

Visual Fortran 支持 Fortran 90(同时也支持 Fortran 77).Fortran 90 对以往的 Fortran 语言标准作了大量的改动,使之成为一种功能强大、具有现代语言特征的计算机语言,受到越来越多的程序开发人员的欢迎.本书所介绍的常用数值算法是利用 Fortran 90 语言编写的,可以使人们在制作应用程序中减少不必要的重复劳动,节约了时间,大大提高了计算机使用效率.

本书共有数值计算中常用的使用 Fortran 90 语言编写的子过程近 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、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,得到计算结果.本书的所有子过程都在 Visual Fortran 5.0 版本上进行了验证,准确无误.

本书配有光盘,内容包括书中全部子过程、验证程序及每种数值算法的 Visual Fortran 工程项目.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.

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	7
1.3 追赶法	13
1.4 五对角线性方程组解法	16
1.5 线性方程组解的迭代改善	21
1.6 范德蒙(Vandermonde)方程组解法	24
1.7 托伯利兹(Toeplitz)方程组解法	28
1.8 奇异值分解	34
1.9 线性方程组的共轭梯度法	46
1.10 对称方程组的乔列斯基(Cholesky)分解法	52
1.11 矩阵的 QR 分解	56
1.12 松弛迭代法	62
第 2 章 插值	66
2.1 拉格朗日插值	67
2.2 有理函数插值	71
2.3 三次样条插值	75
2.4 有序表的检索法	82
2.5 插值多项式	90
2.6 二元拉格朗日插值	98
2.7 双三次样条插值	101
第 3 章 数值积分	107
3.1 梯形求积法	108
3.2 辛普森(Simpson)求积法	112
3.3 龙贝格(Romberg)求积法	115
3.4 反常积分	118
3.5 高斯(Gauss)求积法	129

3.6	三重积分	134
第4章	特殊函数	140
4.1	Γ 函数、贝塔函数、阶乘及二项式系数	140
4.2	不完全 Γ 函数、误差函数	150
4.3	不完全贝塔函数	165
4.4	零阶、一阶和任意整数阶的第一、二类贝塞尔函数	170
4.5	零阶、一阶和任意整数阶的第一、二类变形贝塞尔函数	186
4.6	分数阶第一类贝塞尔函数和变形贝塞尔函数	201
4.7	指数积分和定指数积分	211
4.8	连带勒让德函数	218
	附录	222
第5章	函数逼近	237
5.1	级数求和	237
5.2	多项式和有理函数	241
5.3	切比雪夫逼近	247
5.4	积分和导数的切比雪夫逼近	254
5.5	用切比雪夫逼近求函数的多项式逼近	260
第6章	随机数	267
6.1	均匀分布随机数	267
6.2	变换方法 —— 指数分布和正态分布随机数	279
6.3	舍选法 —— F 分布、泊松分布和二项式分布随机数	285
6.4	随机位的产生	297
6.5	蒙特卡罗积分法	304
第7章	排序	307
7.1	直接插入法和 Shell 方法	307
7.2	堆排序	315
7.3	索引表和等级表	322
7.4	快速排序	331
7.5	等价类的确定	335
	附录	341
第8章	特征值问题	342
8.1	对称矩阵的雅可比变换	343
8.2	变实对称矩阵为三对角对称矩阵	352
8.3	三对角矩阵的特征值和特征向量	357

8.4	变一般矩阵为赫申伯格矩阵	363
8.5	实赫申伯格矩阵的 QR 算法	371
第 9 章	数据拟合	379
9.1	直线拟合	379
9.2	线性最小二乘法	384
9.3	非线性最小二乘法	405
9.4	绝对值偏差最小的直线拟合	417
第 10 章	方程求根和非线性方程组的解法	423
10.1	图解法	423
10.2	逐步扫描法和二分法	426
10.3	割线法和试位法	434
10.4	布伦特(Brent)方法	440
10.5	牛顿-拉斐森(Newton-Raphson)法	444
10.6	求复系数多项式根的拉盖尔(Laguerre)方法	450
10.7	求实系数多项式根的贝尔斯托(Bairstou)方法	458
10.8	非线性方程组的牛顿-拉斐森方法	462
第 11 章	函数的极值和最优化	468
11.1	黄金分割搜索法	468
11.2	不用导数的布伦特(Brent)法	476
11.3	用导数的布伦特(Brent)法	482
11.4	多元函数的下山单纯形法	489
11.5	多元函数的包维尔(Powell)法	495
11.6	多元函数的共轭梯度法	503
11.7	多元函数的变尺度法	508
11.8	线性规划的单纯形法	513
第 12 章	傅里叶变换谱方法	527
12.1	复数据快速傅里叶变换算法	527
12.2	实数据快速傅里叶变换算法(一)	536
12.3	实数据快速傅里叶变换算法(二)	540
12.4	快速正弦变换和余弦变换	547
12.5	卷积和逆卷积的快速算法	557
12.6	离散相关和自相关的快速算法	561
12.7	多维快速傅里叶变换算法	565
第 13 章	数据的统计描述	571

13.1	分布的矩——均值、平均差、标准差、方差、斜差和峰态·····	571
13.2	中位数的搜索·····	574
13.3	均值与方差的显著性检验·····	579
13.4	分布拟合的 χ^2 检验 ·····	591
13.5	分布拟合的 K-S 检验法 ·····	597
第 14 章	解常微分方程组 ·····	605
14.1	定步长四阶龙格-库塔(Runge-Kutta)法·····	605
14.2	自适应变步长的龙格-库塔法 ·····	612
14.3	改进的中点法·····	621
14.4	外推法·····	626
第 15 章	两点边值问题的解法 ·····	640
15.1	打靶法(一)·····	640
15.2	打靶法(二)·····	649
15.3	松弛法·····	657
第 16 章	偏微分方程的解法 ·····	679
16.1	解边值问题的松弛法·····	679
16.2	交替方向隐式方法(ADI) ·····	684
参考文献	·····	692
编后记	·····	693

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 将不仅诊断问题, 而且也解决问题.

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 DIR8. 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. 方法

(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:

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

3. 使用说明

GAUSSJ (A, N, B)

N 整型变量, 输入参数, 矩阵 \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. 过程

子过程 GAUSSJ.

```
SUBROUTINE gaussj(a,n,b)
  INTEGER n,NMAX
  REAL a(n,n),b(n)
  PARAMETER (NMAX=50)
  INTEGER i,col,irow,j,k,l,ll,indx(NMAX),&
    indxr(NMAX),ipiv(NMAX)
  REAL big,dum,pvinv
```

```

do j=1,n
    ipiv(j)=0
end do
do i=1,n
    big=0.
    do j=1,n
        if(ipiv(j)/=1) then
            do k=1,n
                if (ipiv(k) == 0) then
                    if (abs(a(j,k)) >= big) then
                        big=abs(a(j,k))
                        irow=j
                        icol=k
                    endif
                else if (ipiv(k) > 1) then
                    pause 'singular matrix in gaussj'
                endif
            end do
        endif
    end do
    if (big == 0) then
        pause 'singular matrix in gaussj'
    endif
    ipiv(icol)=ipiv(icol)+1
    if (irow/=icol) then
        do l=1,n
            dum=a(irow,l)
            a(irow,l)=a(icol,l)
            a(icol,l)=dum
        end do
        dum=b(irow)
        b(irow)=b(icol)
        b(icol)=dum
    endif
    indxr(i)=irow
    indxc(i)=icol
    if (a(icol,icol) == 0.) pause 'singular matrix in gaussj'
    pivinv=1./a(icol,icol)
    a(icol,icol)=1.
    do l=1,n

```

```

      a(icol,l)=a(icol,l)*pivinv
    end do
    b(icol)=b(icol)*pivinv
    do ll=1,n
      if(ll/=icol) then
        dum=a(ll,icol)
        a(ll,icol)=0.
        do l=1,n
          a(ll,l)=a(ll,l)-a(icol,l)*dum
        end do
        b(ll)=b(ll)-b(icol)*dum
      endif
    end do
  end do
do l=n,1,-1
  if(indxr(l)/=indxcl) then
    do k=1,n
      dum=a(k,indxcl)
      a(k,indxcl)=a(k,indxr(l))
      a(k,indxr(l))=dum
    end do
  endif
end do
END SUBROUTINE gaussj

```

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:

```

PROGRAM D1R1
! Driver program for routine GAUSSJ
PARAMETER(N=3)
DIMENSION A(3,3),B(3),A1(3,3),B1(3)
DATA A/2.,5.,1.,1.,-1.,-3.,2.,1.,-4./
DATA B/5.,8.,-4./
Print *, '已知的方程组的右端向量'
DO I=1,N

```

```

      WRITE( *, '(1X,3F12.6)') B(I)
    END DO
    DO I=1,N
      DO J=1,3
        A1(I,J)=A(I,J)
      END DO
    END DO
    Call GAUSSJ(A,N,B)
    WRITE( *, * )
    Print *, '计算出的方程组的解'
    DO I=1,N
      WRITE( *, '(1X,3F12.6)') B(I)
    END DO
    ! 将计算出的解 B 乘以系数矩阵,以验证计算结果正确
    DO L=1,N
      B1(L)=0.
      DO J=1,N
        B1(L)=B1(L)+A1(L,J)*B(J)
      END DO
    END DO
    WRITE( *, * )
    Print *, '计算出的解乘以系数矩阵的结果'
    DO I=1,N
      WRITE( *, '(1X,3F12.6)') B1(I)
    END DO
  END

```

计算结果如下:

已知的方程组的右端向量

5.000000

8.000000

-4.000000

计算出的方程组的解

1.000000

-1.000001

2.000000

计算出的解乘以系数矩阵的结果

5.000000
8.000000
-4.000000

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. 子过程 LUDCMP 将系数矩阵 A 分解为上三角矩阵和下三角矩阵. 子过程 LUBKSB 利用 LUDCMP 的分解结果求得线性方程组 $Ax=b$ 的解.

2. 方法

(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:

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

考虑到数值稳定性, 其中 l_{ij}, u_{ij} 计算如下:

① 选取 A 中每行中的主元(绝对值最大元) $\{u_{ij}\}$, $i=1, \cdots, n$.

② 取

$$|a_{i_1j}/a_{1j_1}| = \max_{1 \leq i \leq n} |a_{i1}/a_{1j_1}|$$

若 $i_1 \neq 1$, 则交换第 1 行与第 i_1 行得 $A=(a_{ij})$.

$$u_{1j} = a_{1j}, \quad j = 1, 2, \cdots, n$$

$$l_{i1} = a_{i1}/u_{11}, \quad i = 2, \cdots, n$$

③ 一般地, 令

$$S_i = a_{ik} - \sum_{j=1}^{k-1} l_{ij}u_{jk}, \quad i = k, \cdots, n$$

$$|S_{i_k/a_{i_k i_k}}| = \max_{k \leq i_k \leq n} |S_{i_k/a_{i_k i_k}}|$$

由于 A 非奇异, 所以 $S_{i_k} \neq 0$. 若 $i_k \neq k$, 则交换 A 与所得 L 的第 k 行与第 i_k 行, 于是 $u_{kk} = S_{i_k} \neq 0$, 且 $|l_{ij}| \leq 1 (i > k)$.

$$\begin{aligned} u_{kj} &= a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj}, \quad k = 2, \dots, n; \quad j = k, \dots, n \\ l_{ik} &= \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}}{u_{kk}}, \quad k = 2, \dots, n-1; \quad i = k+1, \dots, n \\ u_{kj} &= 0, \quad k > j, \quad l_{ik} = 0, \quad i < k \end{aligned}$$

(3) 解 $Ax=b$ 等价于解

$$P_n \cdots P_1 Ax = P_n \cdots P_1 b$$

即

$$LUx = \tilde{b} = P_n \cdots P_1 b$$

此式等价于求解 $Ly = \tilde{b}$, $Ux = y$. 计算公式为

$$y_1 = \tilde{b}_1, \quad y_i = \tilde{b}_i - \sum_{j=1}^{i-1} l_{ij} y_j, \quad i = 2, \dots, n$$

$$x_n = y_n / u_{nn}$$

$$x_i = \frac{y_i - \sum_{j=i+1}^n u_{ij} x_j}{u_{ii}}, \quad i = n-1, \dots, 1$$

(4) 一般地说, 优先推荐解线性方程组 $Ax=b$ 的方法是:

CALL LUDCMP (A,N,NP,INDX,D)

CALL LUBKSB (A,N,NP,INDX,B)

解 X 将存储在 B 中. 原始矩阵 A 已经被存储.

如果要连续解具有相同 A 而不同 b 的线性方程组时, 只需重复

CALL LUBKSB (A,N,NP,INDX,B)

因为 LUBKSB 所需要的输入 A 和 $INDX$ 可以从 LUDCMP 中得到.

3. 使用说明

LUDCMP (A,N,NP,INDX,D)

LUBKSB (A,N,NP,INDX,B)

N 整型变量, 输入参数, A 的阶数

- NP 整型变量,输入参数, A 的物理维数
- A 实型数组,输入、输出参数. 在 LUDCMP 中,输入时按列存放实方阵 A ,输出时,对角线以下部分存放单位下三角矩阵 L ,对角线及其以上部分存放上三角矩阵 U . 在 LUBKSB 中,将 LUDCMP 中输出结果 A 作为输入
- INDX 整型数组,在子过程 LUDCMP 中为输出参数,用于记录置换矩阵,称为置换向量,在子过程 LUBKSB 中为输入参数,输入子过程 LUDCMP 的输出结果
- D ± 1 ,输出参数,依赖于行交换次数为偶(+1)还是奇(-1)
- B 实型数组,输入、输出参数,输入实向量 b . 输出时,方程组的解 x 存储在数组 B 中

4. 过程

(1) 子过程 LUDCMP.

```

SUBROUTINE ludcmp(a,n,np,indx,d)
PARAMETER (nmax=100,tiny=1.0e-20)
DIMENSION a(np,np),indx(n),vv(nmax)
d=1.
do i=1,n
    aamax=0.
    do j=1,n
        if (abs(a(i,j))>aamax) aamax=abs(a(i,j))
    end do
    if (aamax==0.) pause 'singular matrix.'
    vv(i)=1./aamax
end do
do j=1,n
    if (j>1) then
        do i=1,j-1
            sum=a(i,j)
            if (i>1) then
                do k=1,i-1
                    sum=sum-a(i,k)*a(k,j)
                end do
            end do

```

```

        a(i,j)=sum
    endif
end do
endif
aamax=0.
do i=j,n
    sum=a(i,j)
    if (j>1) then
        do k=1,j-1
            sum=sum-a(i,k)*a(k,j)
        end do
        a(i,j)=sum
    endif
    dum=vv(i)*abs(sum)
    if (dum>=aamax) then
        imax=i
        aamax=dum
    endif
end do
if (j/=imax) then
    do k=1,n
        dum=a(imax,k)
        a(imax,k)=a(j,k)
        a(j,k)=dum
    end do
    d=-d
    vv(imax)=vv(j)
endif
indx(j)=imax
if (j/=n) then
    if (a(j,j)==0.) a(j,j)=tiny
    dum=1./a(j,j)
    do i=j+1,n
        a(i,j)=a(i,j)*dum
    end do
endif
end do

```

```

if(a(n,n) /= 0.) a(n,n)=tiny
END SUBROUTINE ludcmp

```

(2) 子过程 LUBKSB.

```

SUBROUTINE lubksb(a,n,np,indx,b)
DIMENSION a(np,np),indx(n),b(n)
ii=0
do i=1,n
    ll=indx(i)
    sum=b(ll)
    b(ll)=b(i)
    if (ii/=0) then
        do j=ii,i-1
            sum=sum-a(i,j)*b(j)
        end do
    else if (sum/=0.) then
        ii=i
    endif
    b(i)=sum
end do
do i=n,1,-1
    sum=b(i)
    if(i<n) then
        do j=i+1,n
            sum=sum-a(i,j)*b(j)
        end do
    endif
    b(i)=sum/a(i,i)
end do
END SUBROUTINE lubksb

```

5. 例子

在验证程序 DIR2 中,为了解线性方程组,还需调用 LUDCMP. 为了验证程序的正确性,将解与原系数矩阵相乘,以便与给定的右端向量相比较. LUBKSB 不能单独使用,必须和 LUDCMP 联合使用. 验证程序 DIR2 如下:

```

PROGRAM DIR2
! Driver program for routine LUBKSB,LUDCMP
PARAMETER(N=3,NP=20)
DIMENSION A(NP,NP),B(NP),A1(NP,NP),X(NP),INDX(NP)
DATA B/1.,2.,3./
A(1,1)=1.;A(1,2)=2.;A(1,3)=3.
A(2,1)=2.;A(2,2)=2.;A(2,3)=3.
A(3,1)=3.;A(3,2)=3.;A(3,3)=3.
Print *, '已知的方程组的右端向量'
DO I=1,N
    WRITE(*, '(1X,3F12.6)') B(I)
END DO
DO I=1,N
    DO J=1,N
        A1(I,J)=A(I,J)
    END DO
END DO
Call LUDCMP(A1,N,NP,INDX,P)
DO L=1,N
    X(L)=B(L)
END DO
CALL LUBKSB(A1,N,NP,INDX,X)
WRITE(*,*)
Print *, '计算出的方程组的解'
DO I=1,N
    WRITE(*, '(1X,3F12.6)') X(I)
END DO
! 将计算出的解 B 乘以系数矩阵,以验证计算结果正确
DO L=1,N
    B(L)=0.
    DO J=1,N
        B(L)=B(L)+A(L,J)*X(J)
    END DO
END DO
WRITE(*,*)
Print *, '计算出的解乘以系数矩阵的结果'
DO I=1,N

```

```

WRITE(*,'(IX,3F12.6)') B(I)
END DO
END

```

计算结果如下:

已知的方程组的右端向量

```

1.000000
2.000000
3.000000

```

计算出的方程组的解

```

1.000000
0.000000
0.000000

```

计算出的解乘以系数矩阵的结果

```

1.000000
2.000000
3.000000

```

1.3 追 赶 法

1. 功能

求解三对角方程组

$$\begin{bmatrix}
 b_1 & c_1 & & & \\
 a_2 & b_2 & c_2 & & \\
 & \dots & \dots & \dots & \\
 & & \dots & \dots & \dots \\
 & & & a_{n-1} & b_{n-1} & c_{n-1} \\
 & & & & a_n & b_n
 \end{bmatrix} \mathbf{x} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-1} \\ r_n \end{bmatrix}$$

2. 方法

(1) 把三对角矩阵作 LU 分解:

$$A = LU = \begin{bmatrix} \beta_1 & & & \\ a_2 & \beta_2 & & \\ & \cdots & \cdots & \\ & & \cdots & \cdots \\ & & & a_n & \beta_n \end{bmatrix} \begin{bmatrix} 1 & \delta_1 & & & \\ & 1 & \delta_2 & & \\ & & \cdots & \cdots & \\ & & & \cdots & \cdots \\ & & & & 1 & \delta_{n-1} \\ & & & & & 1 \end{bmatrix}$$

其中 β_i, δ_i 如下计算:

$$\beta_1 = b_1, \quad \delta_k = G_k / \beta_k$$

$$\beta_{k+1} = b_{k+1} - a_{k+1} \delta_k, \quad k = 1, 2, \cdots, n$$

(2) 原方程等价于 $Ly=r, Ux=y$.

计算公式:

$$\text{追过程} \quad y_1 = r_1 / b_1, y_i = (r_i - a_i y_{i-1}) / \beta_i, \quad i = 2, \cdots, n$$

$$\text{赶过程} \quad x_n = y_n, x_j = y_j - \delta_j x_{j+1}, \quad j = n-1, \cdots, 1$$

3. 使用说明

TRIDAG (A,B,C,R,U,N)

N 整型变量,输入参数,方程的阶数

A 实型数组,输入参数,存放 (a_2, \cdots, a_n)

B 实型数组,输入参数,存放 (b_1, \cdots, b_n)

C 实型数组,输入参数,存放 (c_1, \cdots, c_{n-1})

R 实型数组,输入参数,存放 (r_1, \cdots, r_n)

U 实型数组,输出参数,输出解向量

4. 过程

子过程 TRIDAG.

```
SUBROUTINE tridag(a,b,c,r,u,n)
```

```
PARAMETER (nmax=100)
```

```
REAL gam(nmax),a(n),b(n),c(n),u(n),r(n)
```

```
INTEGER j,n
```

```
if (b(1) == 0.) pause 'b(1)=0 in tridag'
```

```
bct=b(1)
```

```
u(1)=r(1)/bct
```



```

do j=2,n
    gam(j)=c(j-1)/bet
    bet=b(j)-a(j)*gam(j)
    if (bet==0.) pause 'bet=0 in tridag'
    u(j)=(r(j)-a(j)*u(j-1))/bet
end do
do j=n-1,1,-1
    u(j)=u(j)-gam(j+1)*u(j+1)
end do
END SUBROUTINE tridag

```

5. 例子

为了验证子过程 TRIDAG, 将得到的解和系数矩阵相乘, 以验证是否与方程组右端的向量相等, 验证程序 DIR3 如下:

```

PROGRAM DIR3
! Driver program for routine TRIDAG
PARAMETER(N=3)
DIMENSION A1(N,N),A(N),B(N),C(N),R(N),U(N),X(N)
DATA A1/1.,2.,0.,2.,2.,3.,0.,3.,3./
DATA R/1.,2.,3./
Print *, '已知的方程组的右端向量'
DO I=1,N
    WRITE(*,'(1X,3F12.6)') R(I)
END DO
DO I=2,N
    A(I)=A1(I,I-1)
END DO
DO I=1,N-1
    C(I)=A1(I+1,I)
END DO
DO I=1,N
    B(I)=A1(I,I)
END DO
Call TRIDAG(A,B,C,R,U,N)
WRITE(*,*)
Print *, '计算出的方程组的解'

```

```

DO I=1,N
    WRITE(*,'(1X,3F12.6)') U(I)
END DO
! 将计算出的解 B 乘以系数矩阵,以验证计算结果正确
DO L=1,N
    x(L)=0.
    DO J=1,N
        X(L)=X(L)+A1(L,J)*U(J)
    END DO
END DO
WRITE(*,*)
Print*,'计算出的解乘以系数矩阵的结果'
DO I=1,N
    WRITE(*,'(1X,3F12.6)') X(I)
END DO
END

```

计算结果如下:

已知的方程组的右端向量

```

1.000000
2.000000
3.000000

```

计算出的方程组的解

```

-0.200000
0.600000
0.400000

```

计算出的解乘以系数矩阵的结果

```

1.000000
2.000000
3.000000

```

1.4 五对角线性方程组解法

1. 功能

用追赶法求解线性方程组 $Ax=r$, 其中 A 为五对角方阵.

2. 方法

设

$$A = \begin{bmatrix} c_1 & d_1 & e_1 & & & & \\ b_2 & c_2 & d_2 & e_2 & & & \\ a_3 & b_3 & c_3 & d_3 & e_3 & & \\ & a_4 & b_4 & c_4 & d_4 & e_4 & \\ & & \dots & \dots & \dots & \dots & \dots \\ & & & \dots & \dots & \dots & \dots \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} & \\ & & & & & a_n & b_n & c_n & \end{bmatrix} = LU$$

$$L = \begin{bmatrix} w_1 & & & & & & \\ \sigma_2 & w_2 & & & & & \\ a_3 & \sigma_3 & w_3 & & & & \\ & a_4 & \sigma_4 & w_4 & & & \\ & & \dots & \dots & \dots & & \\ & & & \dots & \dots & \dots & \\ & & & & a_n & \sigma_n & w_n \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & \beta_1 & \alpha_1 & & & & \\ & 1 & \beta_2 & \alpha_2 & & & \\ & & 1 & \beta_3 & \alpha_3 & & \\ & & & 1 & \beta_4 & \alpha_4 & \\ & & & & \dots & \dots & \dots \\ & & & & & 1 & \beta_{n-2} & \alpha_{n-2} \\ & & & & & & 1 & \beta_{n-1} \\ & & & & & & & 1 \end{bmatrix}$$

即对 A 进行 LU 分解, 则

$$\begin{aligned} w_1 &= c_1, \quad \beta_1 = d_1/w_1, \quad \alpha_1 = e_1/w_1 \\ \sigma_2 &= b_2, \quad w_2 = c_2 - \sigma_2\beta_1 \\ \beta_2 &= (d_2 - \sigma_2\alpha_1)/w_2, \quad \alpha_2 = e_2/w_2 \\ \sigma_k &= b_k - a_k\beta_{k-2}, \quad k = 3, 4, \dots, n \\ w_k &= c_k - a_k\alpha_{k-2} - \sigma_k\beta_{k-1}, \quad k = 3, 4, \dots, n \end{aligned}$$

$$\beta_k = (d_k - \sigma_k a_{k-1})/w_k, \quad k = 3, 4, \dots, n-1$$

$$a_k = e_k/w_k, \quad k = 3, 4, \dots, n-2$$

于是求解 $Ax=r$ 即为求解两个三角方程

$$Ly = r, \quad Ux = y$$

追过程 $y_1 = r_1/w_1, \quad y_2 = (r_2 - \sigma_2 y_1)/w_2$

$$y_k = (r_k - a_k y_{k-2} - \sigma_k y_{k-1}), \quad k = 3, \dots, n$$

赶过程 $x_n = y_n, \quad x_{n-1} = y_{n-1} - \beta_{n-1} x_n$

$$x_k = y_k - \beta_k x_{k+1} - \alpha_k x_{k+2}, \quad k = n-2, n-3, \dots, 1$$

3. 使用说明

PENDAG (A,B,C,D,E,R,U,N)

N 整型变量,输入参数,方程阶数

A N 个元素的一维实型数组,输入参数,存放系数矩阵的下次对角元素下面的元素 $(a_1, a_2, a_3, \dots, a_n)$, 其中 a_1, a_2 任意

B N 个元素的一维实型数组,输入参数,存放系数矩阵的下次对角元素 (b_1, b_2, \dots, b_n) , 其中 b_1 任意

C N 个元素的一维实型数组,输入参数,存放系数矩阵的对角元素

D N 个元素的一维实型数组,输入参数,存放系数矩阵的上次对角元素 (d_1, d_2, \dots, d_n) , 其中 d_n 任意

E N 个元素的一维实型数组,输入参数,存放系数矩阵的上次对角元素上面的元素 $(e_1, e_2, \dots, e_{n-2}, e_{n-1}, e_n)$, 其中 e_{n-1}, e_n 任意

R N 个元素的一维实型数组,输入参数,存放方程的右端向量

U N 个元素的一维实型数组,输出参数,输出方程的解向量

4. 过程

子过程 PENDAG.

```
SUBROUTINE pendag(a,b,c,d,e,r,u,n)
```

```
PARAMETER (nmax=100)
```

```
REAL a(n),b(n),c(n),d(n),e(n),r(n),u(n),w(nmax),&  
      beta(nmax),alpha(nmax),cg(nmax),h(nmax)
```

```
INTEGER k,n
```

```
w(1)=c(1)
```

```
beta(1)=0.0
```

```
beta(2)=d(1)/w(1)
```

```

alpha(1) := 0.0
alpha(2) := e(1)/w(1)
alpha(n) = 0.0
alpha(n+1) = 0.0
do k=2,n
    cg(k) = b(k) - a(k) * beta(k-1)
    w(k) = c(k) - a(k) * alpha(k-1) - cg(k) * beta(k)
    if (w(k).eq. 0.0) pause ' w(k)=0.0 in pendag'
    beta(k+1) := (d(k) - cg(k) * alpha(k))/w(k)
    alpha(k+1) = e(k)/w(k)
end do
h(1) = 0.0
h(2) = r(1)/w(1)
do k=2,n
    h(k+1) = (r(k) - a(k) * h(k-1) - cg(k) * h(k))/w(k)
end do
u(n) = h(n+1)
u(n-1) = h(n) - beta(n) * u(n)
do k=n-2,1,-1
    u(k) = h(k+1) - beta(k+1) * u(k+1) - alpha(k+1) * u(k+2)
end do
END SUBROUTINE pendag

```

5. 例子

为了验证子过程 PENDAG, 在我们的例子中将数值解法的解和系数矩阵相乘, 和方程组右端的向量相比较. 验证程序 DIR4 如下:

```

PROGRAM DIR4!
Driver program for routine PENDAG
PARAMETER(N=7)
DIMENSION A1(N,N),A(N),B(N),C(N),D(N),E(N),&
    R(N),U(N),X(N)
DATA A1/4.,1.,1.,0.,0.,0.,0.,1.,5.,2.,2.,0.,0.,0.,&
    1.,2.,6.,3.,3.,0.,0.,0.,2.,3.,7.,4.,4.,0.,&
    0.,0.,3.,4.,8.,5.,5.,0.,0.,0.,4.,5.,9.,6.,&
    0.,0.,0.,0.,5.,6.,10./
DATA R/1.,2.,3.,4.,5.,6.,7./

```

```

Print *, '已知的方程组的右端向量'
DO I=1,N
    WRITE(*, '(1X,3F12.6)') R(I)
END DO
DO I=3,N
    A(I)=A1(I,I-2)
END DO
DO I=2,N
    B(I)=A1(I,I-1)
END DO
DO I=1,N-1
    D(I)=A1(I,I+1)
END DO
DO I=1,N-2
    E(I)=A1(I,I+2)
END DO
DO I=1,N
    C(I)=A1(I,I)
END DO
Call PENDAG(A,B,C,D,E,R,U,N)
WRITE(*,*)
Print *, '计算出的方程组的解'
DO I=1,N
    WRITE(*, '(1X,3F12.6)') U(I)
END DO
1 将计算出的解 B 乘以系数矩阵,以验证计算结果正确
DO L=1,N
    x(L)=0.
    DO J=1,N
        X(L)=X(L)+A1(L,J)*U(J)
    END DO
END DO
WRITE(*,*)
Print *, '计算出的解乘以系数矩阵的结果'
DO I=1,N
    WRITE(*, '(1X,3F12.6)') X(I)
END DO

```

END

计算结果如下:

已知的方程组的右端向量

1.000000

2.000000

3.000000

4.000000

5.000000

6.000000

7.000000

计算出的方程组的解

0.173383

-0.043744

0.350211

0.672457

-0.401511

0.016075

0.910401

计算出的解乘以系数矩阵的结果

1.000000

2.000000

3.000000

4.000000

5.000000

6.000000

7.000000

1.5 线性方程组解的迭代改善

1. 功能

改进已知近似解的精度,既可用于求一般线性方程组的高精度解,也可用于病态方程组的求解,它是一个 $O(n^2)$ 过程.

2. 方法

设 $Ax=b$ 的精确解为 x^* ,已知一近似解 $x^*+\delta x$,即 $x^*-\delta x$ 满足

$$A(x^* + \delta x) = b + \delta b \neq b$$

$$\delta b = A(x^* + \delta x) - b$$

作 A 的 LU 分解, 求解

$$A\delta x = \delta b = A(x^* + \delta x) - b \quad (\text{右端项 } \delta b \text{ 用双精度计算})$$

则

$$x^* = (x^* + \delta x) - \delta x$$

3. 使用说明

MPROVE (A,ALUD,N,INDX,B,X)

N 整型数组, 输入参数, 方程的阶数

A 二维实型数组, 输入参数, 方程的系数矩阵

B 实型数组, 输入参数, 输入向量 b

X 实型数组, 输入、输出参数, 输入时存放近似解 $x^* + \delta x$, 输出时存放改善解

ALUD 实型数组, 输入参数, 输入由子过程 LUDCMP 得到的 A 的 LU 分解

INDX 同 1.2 节中使用说明

4. 过程

子过程 MPROVE.

```
SUBROUTINE mprove(a,alud,n,np,indx,b,x)
```

```
PARAMETER (nmax=100)
```

```
! USES lubksb
```

```
REAL a(np,np),alud(np,np),b(n),x(n),r(nmax)
```

```
INTEGER i,n,indx(np)
```

```
REAL * 8 sdp
```

```
do i=1,n
```

```
    sdp=-b(i)
```

```
    do j=1,n
```

```
        sdp=sdp+dblc(a(i,j))*dblc(x(j))
```

```
    end do
```

```
    r(i)=sdp
```

```
end do
```

```
call lubksb(alud,n,np,indx,r)
```



```

do i=1,n
    x(i)=x(i)-r(i)
end do
END SUBROUTINE mprove

```

5. 例子

首先利用 LU 分解求得线性方程组的解(即利用子过程 LUDCMP 和 LUBKSB),然后利用给每个解加上一个随机数的办法,使解变得不准确.最后,再利用子过程 MPROVE 求出线性方程组的改善解.验证程序 D1R5 如下:

```

PROGRAM D1R5
! Driver program for routine MPROVE
PARAMETER(N=5,NP=5)
! USES RAN3,LUDCMP,LUBKSB
DIMENSION A(N,N),INDX(N),B(N),X(N),AA(N,N)
DATA A/1.0,2.0,1.0,4.0,5.0,2.0,3.0,1.0,5.0,1.0,&
      3.0,4.0,1.0,1.0,2.0,4.0,5.0,1.0,2.0,3.0,&
      5.0,1.0,1.0,3.0,4.0/
DATA B/1.0,1.0,1.0,1.0,1.0/
DO I=1,N
    X(I)=B(I)
    DO J=1,N
        AA(I,J)=A(I,J)
    END DO
END DO
CALL LUDCMP(AA,N,NP,INDX,D)
CALL LUBKSB(AA,N,NP,INDX,X)
WRITE(*, '(1X,A)') '输出方程组的解'
WRITE(*, '(1X,5F12.6)') (X(I), I=1,N)
! Now phoney up X and let MPROVE fit it
IDUM=-13
DO I=1,N
    X(I)=X(I)*(1.0+0.2*RAN3(IDUM))
END DO
WRITE(*, '(1X,A)') '输出干扰后的解'
WRITE(*, '(1X,5F12.6)') (X(I), I=1,N)
CALL MPROVE(A,AA,N,NP,INDX,B,X)

```

```
WRITE(*, '(1X,A)') '输出改善后的解'
WRITE(*, '(1X,5F12.6)') (X(I), I=1,N)
END
```

计算结果如下:

输出方程组的解

```
0.200000  0.200000  2.200000  -1.800000  0.200000
```

输出干扰后的解

```
0.218795  0.216818  2.471051  -2.108161  0.210452
```

输出改善后的解

```
0.200000  0.200000  2.200000  -1.800000  0.200000
```

1.6 范德蒙(Vandermonde)方程组解法

1. 功能

求解范德蒙线性方程组 $VW=q$, 即

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \cdots & \cdots & \cdots & \cdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_n \end{bmatrix}$$

2. 方法

(1) 构造多项式

$$p_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^n \frac{x - x_k}{x_j - x_k} = \sum_{k=1}^n A_{jk} x^{k-1}, \quad j = 1, \cdots, n$$

记

$$A = (A_{jk}) = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \cdots & \cdots & \cdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix}$$

则

$$P_j(x_i) = \delta_{ij} = \sum_{k=1}^n A_{jk} x_i^{k-1}$$

即 $AV = I$ (I 为单位矩阵)
所以

$$W = Aq$$

(2) 设

$$p(x) = \prod_{k=1}^n (x - x_k) = \sum_{k=1}^n c_k x^{k-1} + x^n$$

$$p_{1j}(x) = p(x)/(x - x_j) = \sum_{k=1}^n b_k x^{k-1}$$

$$T_j = P_{1j}(x_j)$$

则 $P_j(x) = P_{1j}(x)/T_j = \sum_{k=1}^n (b_k/T_j)x^{k-1}$

由综合除法

$$b_n = 1$$

$$b_k = \sum_{i=k+1}^n c_i x_j^{-(i-k+1)} + x_j^{n-k} = b_{k+1}x_j + c_{k+1}, \quad j = n-1, \dots, 1$$

3. 使用说明

VANDER (X,W,Q,N)

N 整型变量, 输入参数, 方程的阶数

Q 实型数组, 输入参数, 存放实向量 (q_1, \dots, q_n)

W 实型数组, 输出解向量

X 实型数组, 输入参数, 存放 (x_1, \dots, x_n)

4. 过程

子过程 VANDER.

SUBROUTINE vander(x,w,q,n)

PARAMETER (nmax=100,zero=0.0,one=1.0)

REAL x(n),w(n),q(n),c(nmax)

REAL xx,t,b,s

INTEGER i,n,j,k,k1

if(n==1) then

w(1)=q(1)

else

do i=1,n

```

        c(i)=zero
    end do
    c(n)=-x(1)
    do i=2,n
        xx=-x(i)
        do j=n+1-i,n-1
            c(j)=c(j)+xx*c(j+1)
        end do
        c(n)=c(n)+xx
    end do
    do i=1,n
        xx=x(i)
        t=one
        b=one
        s=q(n)
        k=n
        do j=2,n
            k1=k-1
            b=c(k)+xx*b
            s=s+q(k1)*b
            t=xx*t+b
            k=k1
        end do
        w(i)=s/t
    end do
end if
END SUBROUTINE vander

```

5. 例子

我们所提供例子的数据记录在验证程序 D1R6 中. 方程组的右端向量在 Q 中, 求得的解存放在 W 中, 并将解与系数矩阵相乘, 使其与方程组右端的向量相比较. 验证程序 D1R6 如下:

```

PROGRAM D1R6
! Driver program for routine VANDER
PARAMETER(N=5)
DIMENSION X(N),Q(N),W(N),TERM(N)

```

```

DATA X/1.0,1.5,2.0,2.5,3.0/
DATA Q/1.0,1.5,2.0,2.5,3.0/
write(*,*)'已知方程组的右端向量'
DO I=1,N
    WRITE(*, '(5X,F12.6)') Q(I)
END DO
CALL VANDER(X,W,Q,N)
WRITE(*,*)
WRITE(*,*)'计算出的方程组的解'
DO I=1,N
    WRITE(*, '(5X,F12.6)') W(I)
END DO
WRITE(*,*)
WRITE(*,*)'计算出的方程组的解乘以系数矩阵,以验证计算结果正确'
WRITE(*, '(1X,T6,A,T25,A)') '解乘以系数矩阵','方程组的右端向量'
SUM=0.0
DO I=1,N
    TERM(I)=W(I)
    SUM=SUM+W(I)
END DO
WRITE(*, '(1X,F14.4,8X,F12.4)') SUM,Q(1)
DO I=2,N
    SUM=0.0
    DO J=1,N
        TERM(J)=TERM(J)*X(J)
        SUM=SUM+TERM(J)
    END DO
    WRITE(*, '(1X,F14.4,8X,F12.4)') SUM,Q(I)
END DO
END

```

计算结果如下:

已知方程组的右端向量

```

1.000000
1.500000
2.000000
2.500000

```

3.000000

计算出的方程组的解

--1.083333

4.000000

-3.000000

1.333333

-0.250000

计算出的方程组的解乘以系数矩阵,以验证计算结果正确

解乘以系数矩阵 方程组的右端向量

1.0000

1.0000

1.5000

1.5000

2.0000

2.0000

2.5000

2.5000

3.0000

3.0000

1.7 托伯利兹(Toeplitz)方程组解法

1. 功能

求解 $Rx=y$, 即给定 $2n-1$ 个数 $R_k, k=-n+1, \dots, -1, 0, 1, \dots, n-1$, 求解

$$\begin{bmatrix} R_0 & R_{-1} & R_{-2} & \cdots & R_{-n+2} & R_{-n+1} \\ R_1 & R_0 & R_{-1} & \cdots & R_{-n+3} & R_{-n+2} \\ & & \cdots & & & \\ R_{n-1} & R_{n-2} & R_{n-3} & \cdots & R_1 & R_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

2. 方法

(1) $Rx=y$, 即

$$\sum_{j=1}^n R_{i-j} x_j = y_i, \quad i=1, \dots, n \quad (1-1)$$

对 $m=1, 2, \dots, n$ 依次求解.

$$\sum_{j=1}^m R_{i-j} x_j^{(m)} = y_i, \quad i=1, \dots, m \quad (1-2)$$

当 $m=n$ 时即得所求解.

(2) ①设对 m 已知解 $\{x_j^{(m)}\}_{j=1}^m$, 求 $\{x_j^{(m+1)}\}_{j=1}^{m+1}$. 满足

$$\sum_{j=1}^m R_{i-j} x_j^{(m+1)} + R_{i-(m+1)} x_{m+1}^{(m+1)} = y_i, \quad i=1, \dots, m+1 \quad (1-3)$$

消去 $\{y_i\}_1^m$ 得

$$\sum_{j=1}^m R_{i-j} \frac{x_j^{(m)} - x_j^{(m+1)}}{x_{m+1}^{(m+1)}} = R_{i-(m+1)}$$

即

$$\sum_{j=1}^m R_{j-i} G_j^{(m)} = R_{-i}, \quad i = 1, \dots, m \quad (1-4)$$

其中

$$G_j^{(m)} = \frac{x_{m+1-j}^{(m)} - x_{m+1-j}^{(m+1)}}{x_{m+1}^{(m+1)}} \quad (1-5)$$

即

$$x_{m+1-j}^{(m+1)} = x_{m+1-j}^{(m)} - x_{m+1}^{(m+1)} G_j^{(m)}, \quad j = 1, \dots, m \quad (1-6)$$

或

$$x_j^{(m+1)} = x_j^{(m)} - x_{m+1}^{(m+1)} G_{m+1-j}^{(m)}, \quad j = 1, \dots, m \quad (1-7)$$

于是为求 $\{x_j^{(m+1)}\}_1^{m+1}$, 只需求得 $x_{m+1}^{(m+1)}$ 及 $\{G_j^{(m)}\}_1^m$.

② 把式(1-7)代入式(1-3)得

$$x_{m+1}^{(m+1)} = \frac{\sum_{j=1}^m R_{m+1-j} x_j^{(m)} - y_{m+1}}{\sum_{j=1}^m R_{m+1-j} G_{m+1-j}^{(m)} - R_0} \quad (1-8)$$

下面找出 $G_j^{(m)}$ 的递归关系.

(3) 考虑左解 Z , 即 Z 满足

$$Z^T R = y^T \text{ 等价于 } R^T Z = y$$

即

$$\sum_{j=1}^n R_{j-i} z_j = y_i, \quad i = 1, \dots, m \quad (1-9)$$

同样对 $m=1, 2, \dots$, 依次求解

$$\sum_{j=1}^m R_{j-i} z_j^{(m)} = y_i, \quad i = 1, \dots, m \quad (1-10)$$

直到 $m=n$, 类似于式(1-2)中有

$$\sum_{j=1}^m R_{j-i} \frac{z_j^{(m)} - z_j^{(m+1)}}{z_{m+1}^{(m+1)}} = R_{m+1-i}, \quad i = 1, \dots, m$$

即

$$\sum_{j=1}^m R_{i-j} \frac{z_{m+1-j}^{(m)} - z_{m+1-j}^{(m+1)}}{z_{m+1}^{(m+1)}} = R_i, \quad i = 1, \dots, m$$

或

$$\sum_{j=1}^m R_{j-j} H_j^{(m)} = R, \quad (1-11)$$

其中

$$H_j^{(m)} = \frac{z_{m+1-j}^{(m)} - z_{m+1-j}^{(m+1)}}{z_{m+1}^{(m+1)}} \quad (1-12)$$

即

$$z_{m+1-j}^{(m+1)} = z_{m+1-j}^{(m)} - z_{m+1}^{(m+1)} H_j^{(m)} \quad (1-13)$$

或

$$z_j^{(m+1)} = z_j^{(m)} - z_{m+1}^{(m+1)} H_{m+1-j}^{(m)} \quad (1-14)$$

$$z_{m+1}^{(m+1)} = \frac{\sum_{j=1}^m R_{j-m-1} z_j^{(m)} - y_{m+1}}{\sum_{j=1}^m R_{j-m-1} H_{m+1-j}^{(m)} - R_0} \quad (1-15)$$

(4) 比较式(1-4)与式(1-10),由(3)中式(1-14)和(1-15)有

$$G_j^{(m+1)} = G_j^{(m)} - G_{m+1}^{(m+1)} H_{m+1-j}^{(m)}, \quad j = 1, \dots, m \quad (1-16)$$

$$G_{m+1}^{(m+1)} = \frac{\sum_{j=1}^m R_{j-m-1} G_j^{(m)} - R_{-m+1}}{\sum_{j=1}^m R_{j-m-1} H_{m+1-j}^{(m)} - R_0} \quad (1-17)$$

比较式(1-2)与式(1-11),由式(1-7)和(1-8)有

$$H_j^{(m+1)} = H_j^{(m)} - H_{m+1}^{(m+1)} G_{m+1-j}^{(m)}, \quad j = 1, \dots, m \quad (1-18)$$

$$H_{m+1}^{(m+1)} = \frac{\sum_{j=1}^m R_{m+1-j} H_j^{(m)} - R_{m+1}}{\sum_{j=1}^m R_{m+1-j} G_{m+1-j}^{(m)} - R_0} \quad (1-19)$$

(5) 算法

① 当 $m=1$ 时,易见

$$x_1^{(1)} = y_1/R_0, \quad G_1^{(1)} = R_{-1}/R_0, \quad H_1^{(1)} = R_1/R_0$$

② 设对 m 已知解 $\{x_j^{(m)}\}_1^m$ 及 $G_j^{(m)}, H_j^{(m)}, j=1, \dots, m$.

由式(1-17)、(1-19)、(1-16)、(1-18)计算出 $\{G_j^{(m+1)}, H_j^{(m+1)}\}_{j=1}^{m+1}$.

③ 由式(1-7)、式(1-8)即得 $\{x_j^{(m+1)}\}_1^{m+1}$.

3. 使用说明

TOEPLZ (R,X,Y,N)

N 整型变量,输入参数,方程阶数

Y 实型数组,输入参数,存放 (y_1, \dots, y_n)

R 一维实型数组,输入参数,存放 (R_1, \dots, R_{2n-1})
 X 实型数组,输出解向量

4. 过程

子过程 TOEPLZ.

```

SUBROUTINE toeplz(r,x,y,n)
INTEGER n,nmax
REAL r(2*n-1),x(n),y(n)
PARAMETER (nmax=100)
INTEGER j,k,m,m1,m2
REAL pp,pt1,pt2,qq,qt1,qt2,sd,sgd,sgn,shn,sxn,&
      g(nmax),h(nmax)
if (r(n) == 0.) then
      pause 'levinson method fails; singular &
              principal minor'

      return
end if
x(1) = y(1)/r(n)
if(n == 1) return
g(1) = r(n-1)/r(n)
h(1) = r(n+1)/r(n)
do m = 1, n
      m1 = m + 1
      sxn = -y(m1)
      sd = -r(n)
      do j = 1, m
            sxn = sxn + r(n+m1-j) * x(j)
            sd = sd + r(n+m1-j) * g(m-j+1)
      end do
      if (sd == 0.) then
            pause 'levinson method fails; singular &
                    principal minor'

            return
      end if
      x(m1) = sxn/sd
      do j = 1, m

```

```

      x(j)=x(j)-x(m1)*g(m-j+1)
    end do
    if (m1==n) return
    sgn=-r(n-m1)
    shn=-r(n+m1)
    sgd=-r(n)
    do j=1,m
      sgn=sgn+r(n+j-m1)*g(j)
      shn=shn+r(n+m1-j)*h(j)
      sgd=sgd+r(n+j-m1)*h(m-j+1)
    end do
    if (sd==0. .or. sgd==0. ) then
      pause 'levinson method fails: singular &
              principal minor'

      return
    end if
    g(m1)=sgn/sgd
    h(m1)=shn/sd
    k=m
    m2=(m+1)/2
    pp=g(m1)
    qq=h(m1)
    do j=1 ,m2
      pt1=g(j)
      pt2=g(k)
      qt1=h(j)
      qt2=h(k)
      g(j)=pt1-pp*qt2
      g(k)=pt2-pp*qt1
      h(j)=qt1-qq*pt2
      h(k)=qt2-qq*pt1
      k=k-1
    end do
  end do
  pause 'never get here'
END SUBROUTINE toeplz

```

5. 例子

我们举一个很简单的例子,在验证程序中我们将 $2N-1$ 个数 R_i 放在数组 R 中,方程组右端的向量放在数组 Y 中.利用子过程 TOEPLZ 求得解 X ,并将其与系数矩阵相乘,以验证是否等于方程组右端的向量.验证程序 DIR7 如下:

```

PROGRAM DIR7
1 Driver program for routine TOEPLZ
PARAMETER(N=5,N2=2 * N)
DIMENSION X(N),Y(N),R(N2)
DO I=1,N
Y(I)=0.1 * I
END DO
DO I=1,2 * N -1
R(I)=1. /I
END DO
CALL TOEPLZ(R,X,Y,N)
WRITE(*,*) '计算出的方程组的解'
DO I=1,N
WRITE(*, '(5X,A2,I1,A2,E13.6)') 'X(',I,')= ',X(I)
END DO
WRITE(*, '(/1X,A)') '将计算出的解乘以系数矩阵,以验证计算结果正确'
WRITE(*, '(1X,T6,A,5x,T25,A)') '解乘以系数矩阵','方程组的右端向量'
DO I=1,N
SUM=0.0
DO J=1,N
SUM=SUM+ R(N+I-J) * X(J)
END DO
WRITE(*, '(1X,F13.4,9x,f12.4)') SUM,Y(I)
END DO
END

```

计算结果如下:

计算出的方程组的解

$$X(1) = 0.132098E+04$$

$$X(2) = -0.245993E+04$$

$$X(3) = 0.144635E-04$$

$$X(4) = -0.287100E-03$$

$$X(5) = 0.124692E+02$$

将计算出的解乘以系数矩阵,以验证计算结果正确

解乘以系数矩阵 方程组的右端向量

$$0.1000 \qquad 0.1000$$

$$0.2000 \qquad 0.2000$$

$$0.3000 \qquad 0.3000$$

$$0.4000 \qquad 0.4000$$

$$0.5000 \qquad 0.5000$$

1.8 奇异值分解

1. 功能

对任一 $m \times n$ 实矩阵 A , 求正交矩阵 U 和 V 使 $A = UWV^T$, 其中 $W = \text{diag}(w_1, \dots, w_n)$, $w_1 \geq w_2 \geq \dots \geq w_n \geq 0$ 是矩阵 $A^T A$ 的 n 个特征值的非负平方根. 子过程 SVDCMP 的作用是求得 U, W 和 V^T , 子过程 SVBKSJ 的作用是利用上述的 U, W 和 V^T 求得线性方程组 $Ax = b$ 的最小二乘解.

2. 方法

不失一般性, 设 $m \geq n$, 否则考虑 A^T 或矩阵 $\begin{bmatrix} A \\ O_{(n-m) \times n} \end{bmatrix}$.

(1) 首先用 Householder 变换矩阵化 A 为双对角矩阵 $J^{(0)}$:

$$J^{(0)} = P_n \cdots P_1 A Q_1 \cdots Q_{n-2} = \begin{bmatrix} w_1 & r_2 & & & & & \\ & w_2 & r_3 & & & & \\ & & w_3 & r_4 & & & \\ & & & \dots & \dots & & \\ & & & & w_{n-1} & r_n & \\ & & & & & w_n & \\ & & & & & & O_{(m-n) \times n} \end{bmatrix}$$

其中 P_k, Q_k 均为镜像矩阵或单位矩阵, 从而 $J^{(0)}$ 与 A 有相同的奇异值.

记 $A^{(1)} = A$. 设对 A 已进行了 $k-1$ 步 Householder 变换, 即有形式

$$A^{(k)} = P_{k-1} A^{(k-1)} Q_{k-1} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & A_{22}^{(k)} & A_{23}^{(k)} \end{bmatrix}$$

其中 $A_{11}^{(k)} \in R^{(k-1) \times (k-1)}$ 为双对角矩阵, $A_{12}^{(k)} \in R^{(k-1) \times 1}$, $A_{22}^{(k)} \in R^{(m-k+1) \times 1}$, $A_{23}^{(k)} \in R^{(m-k+1) \times (n-k)}$.

第 k 步的计算式为:

① 选取 $u^{(k)}$.

② $h_k = \frac{1}{2} \|u^{(k)}\|_2^2$.

③ $x^{(k)} = [A^{(k)}]^T \cdot u^{(k)} / h_k$.

④ $A^{(k+\frac{1}{2})} = P_{(k)} A^{(k)} = A^{(k)} - u^{(k)} [x^{(k)}]^T$.

⑤ 选取 $v^{(k)}$.

⑥ $q_k = \frac{1}{2} \|v^{(k)}\|_2^2$.

⑦ $y^{(k)} = A^{(k+\frac{1}{2})} v^{(k)} / q_k$.

⑧ $A^{(k+1)} = A^{(k+\frac{1}{2})} - y^{(k)} [v^{(k)}]^T$.

$u^{(k)}, v^{(k)}$ 如下选取: 不妨设 $A_{22}^{(k)} \neq 0$, 则 $A_{22}^{(k)} = \|A_{22}^{(k)}\|_1 \cdot \tilde{A}_{22}^{(k)}$, 其中 $\tilde{A}_{22}^{(k)} = A_{22}^{(k)} / \|A_{22}^{(k)}\|_1$. 于是取

$$u^{(k)} = \|A_{22}^{(k)}\|_1 (0, \dots, 0, \tilde{u}^{(k)T})^T$$

其中

$$\begin{aligned} W_k &= -\text{sign}(a_{kk}^{(k)}) \|\tilde{A}_{22}^{(k)}\|_2 \\ \tilde{e}_1^{(k)} &= (1, 0, \dots, 0)^T \in R^{m-k+1} \\ \tilde{u}^{(k)} &= \tilde{A}_{22}^{(k)} - W_k \tilde{e}_1^{(k)} \end{aligned}$$

同样, 设

$$A^{(k+\frac{1}{2})} = \begin{bmatrix} A_{11}^{(k+\frac{1}{2})} & & \\ A_{21}^{(k+\frac{1}{2})} & A_{22}^{(k+\frac{1}{2})} & \\ & & A_{33}^{(K+\frac{1}{2})} \end{bmatrix}$$

其中

$$\begin{aligned} A_{11}^{(k+\frac{1}{2})} &\in R^{(k-1) \times k}, \quad A_{21}^{(k+\frac{1}{2})} \in R^{1 \times k} \\ A_{22}^{(k+\frac{1}{2})} &\in R^{1 \times (n-k)}, \quad A_{33}^{(k+\frac{1}{2})} \in R^{(m-k) \times (n-k)} \end{aligned}$$

设 $A_{22}^{(k+\frac{1}{2})} \neq 0$, 取 $\hat{A}_{22}^{(k+\frac{1}{2})} = A_{22}^{(k+\frac{1}{2})} / \|A_{22}^{(k+\frac{1}{2})}\|_1$, $v^{(k)} = \|A_{22}^{(k+\frac{1}{2})}\| (0, \dots, 0, \hat{v}^{(k)T})^T$. 其中

$$\begin{aligned} r_{k+1} &= -\text{sign}(a_{k,k+\frac{1}{2}}^{(k+\frac{1}{2})}) \|\hat{A}_{22}^{(k+\frac{1}{2})}\|_2 \\ \hat{e}_1^{(k)} &= (1, 0, \dots, 0)^T \in R^{n-k} \\ \hat{v}^{(k)} &= \hat{A}_{22}^{(k+\frac{1}{2})} - r_{k+1} \hat{e}_1^{(k)} \end{aligned}$$

(2) $Q=Q_1\cdots Q_{n-2}, P=P_1\cdots P_n$ 的迭代计算.

① Q 的迭代计算.

$$Q'_{n-2} = Q_{n-2}$$

$$Q'_j = Q_j Q'_{j+1}, \quad j = n-3, \cdots, 1$$

$$Q = Q'_1$$

② P 的迭代计算与 Q 的相同.

(3) 计算 $J^{(0)}$ 的奇异值.

① $J^{(0)}$ 的奇异值为三对角矩阵 $M \equiv J^{(0)T} J^{(0)}$ 的特征值. 为此作迭代 $J^{(i+1)} = S^{(i)} J^{(i)} T^{(i)}$. 选择正交矩阵 $S^{(i)}$ 和 $T^{(i)}$ 使 $J^{(i)}$ 对角化. 这里取 $S^{(i)}$ 和 $T^{(i)}$ 为 Givens 反射矩阵之积.

② 为简单起见, 记

$$J \equiv J^{(i)}, \quad \bar{J} \equiv J^{(i+1)}, \quad S \equiv S^{(i)}, \quad T \equiv T^{(i)}, \quad M \equiv J^T J, \quad \bar{M} \equiv \bar{J}^T \bar{J}$$

对 J 左右交替地施行 Givens 反射变换, 使得

$$\bar{J} = S_{n-1,n} S_{n-2,n-1} \cdots S_{12} J T_{12} T_{23} \cdots T_{n-1,n}$$

其中 $S_{i,i+1}, T_{i,i+1}$ 的形式为

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & c & s & \\ & & & s & -c & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \quad \begin{matrix} (i) \\ (i+1) \end{matrix}$$

$$c^2 + s^2 = 1$$

(a) 用 QR 方法首先确定实位移 k_i .

(b) 在 T_{12} 中取

$$\begin{bmatrix} c \\ s \end{bmatrix} = \alpha \begin{bmatrix} w_1^2 - k_i \\ r_2 w_1 \end{bmatrix}, \quad c^2 + s^2 = 1$$

(c) 则 T_{12} 使 J 生成元素 $\{J\}_{21}$, 一般地取 $T_{i,i+1}$ 消去 $\{J\}_{i-1,i+1}$, 则会生成元素 $\{J\}_{i+1,i}$, 取 $S_{i,i+1}$ 消去 $\{J\}_{i+1,i}$, 则生成元素 $\{J\}_{i,i+2}$.

(d) 最终得到 \bar{J} 也是双对角矩阵且 M 到 \bar{M} 是一个带有位移 k_i 的 QR 变换.

对于线性方程组 $Ax=b$, 设 $A=UWV^T$ 是 A 的奇异值分解, 则线性方程组 $Ax=b$ 的最小二乘解是

$$x = A^+ b = VW^+ U^T b$$

其中 A^+ 表示 A 的广义逆, 这里

$$A^+ = VW^+ U^T$$

$$W^+ = \text{diag}(w_1^*, w_2^*, \dots, w_n^*)$$

$$W_i^* = \begin{cases} 0, & w_i = 0 \\ 1/w_i, & w_i \neq 0, \end{cases} \quad i = 1, 2, \dots, n$$

若线性方程组有解, 则最小二乘解即是线性方程组的解.

3. 使用说明

(1) SVDCMP (A, M, N, W, V)

N 整型变量, 输入参数, A 的列数

M 整型变量, 输入参数, A 的行数

A 实型数组, 输入、输出参数, 输入时按列存放实矩阵 A , 输出矩阵 U

W 实型数组, 输出参数, 输出奇异值

V 实型数组, 输出参数, 输出变换矩阵 V

(2) SVBKSB (U, W, V, M, N, B, X)

U, W, V 输入参数, 由子过程 SVDCMP 所输出

M 整型变量, 输入参数, 系数矩阵的行数

N 整型变量, 输入参数, 系数矩阵的列数

B 实型数组, 输入参数, 输入方程组右端的向量

X 实型数组, 输出参数, 输出方程组的最小二乘解

4. 过程

(1) 子过程 SVDCMP.

```
SUBROUTINE svdcmp(a,m,n,w,v)
```

```
PARAMETER(nmax=100)
```

```
REAL a(m,n),w(n),v(n,n),rvl(nmax)
```

```
INTEGER i,l,k,nm
```

```
REAL g,s,scale,f,h,c,x,y,z
```

```
if(m<n) pause 'you must augment a with extra zero rows.'
```

```
g=0.0
```

```
scale=0.0
```

```
anorm=0.0
```

```
do i=1,n
```

```
l=i+1
rv1(i)=scale*g
g=0.0
s=0.0
scale=0.0
if(i<=m) then
  do k=i,m
    scale=scale+abs(a(k,i))
  end do
  if(scale/=0.0) then
    do k=i,m
      a(k,i)=a(k,i)/scale
      s=s+a(k,i)*a(k,i)
    end do
    f=a(l,i)
    g=-sign(sqrt(s),f)
    h=f*g-s
    a(i,i)=f-g
    if (i/=n) then
      do j=l,n
        s=0.0
        do k=i,m
          s=s+a(k,i)*a(k,j)
        end do
        f=s/h
        do k=i,m
          a(k,j)=a(k,j)+f*a(k,i)
        end do
      end do
    endif
    do k=i,m
      a(k,i)=scale*a(k,i)
    end do
  endif
endif
w(i)=scale*g
g=0.0
```



```

s=0.0
scale=0.0
if ((i<=m).and.(i/=n)) then
  do k=1,n
    scale=scale+abs(a(i,k))
  end do
  if (scale/=0.0) then
    do k=1,n
      a(i,k)=a(i,k)/scale
      s=s+a(i,k)*a(i,k)
    end do
    f=a(i,1)
    g=-sign(sqrt(s),f)
    h=f*g-s
    a(i,1)=f-g
    do k=1,n
      rv1(k)=a(i,k)/h
    end do
    if (i/=m) then
      do j=1,m
        s=0.0
        do k=1,n
          s=s+a(j,k)*a(i,k)
        end do
        do k=1,n
          a(j,k)=a(j,k)+s*rv1(k)
        end do
      end do
    endif
    do k=1,n
      a(i,k)=scale*a(i,k)
    end do
  endif
endif
anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
end do
do i=n,1,-1

```

```

if (i<n) then
  if (g/=0.0) then
    do j=1,n
      v(j,i)=(a(i,j)/a(i,1))/g
    end do
    do j=1,n
      s=0.0
      do k=1,n
        s=s+a(i,k)*v(k,j)
      end do
      do k=1,n
        v(k,j)=v(k,j)+s*v(k,i)
      end do
    end do
  endif
  do j=1,n
    v(i,j)=0.0
    v(j,i)=0.0
  end do
endif
v(i,i)=1.0
g=rv1(i)
l=j
end do
do i=n,1,-1
  l=i+1
  g=w(i)
  if (i<n) then
    do j=1,n
      a(i,j)=0.0
    end do
  endif
  if (g/=0.0) then
    g=1.0/g
    if (i/=n) then
      do j=1,n
        s=0.0

```

```

do k=1,m
    s=s+a(k,i)*a(k,j)
end do
f=(s/a(i,i))*g
do k=i,m
    a(k,j)=a(k,j)+f*a(k,i)
end do
end do
endif
do j=i,m
    a(j,i)=a(j,i)*g
end do
else
do j=i,m
    a(j,i)=0.0
end do
endif
a(i,i)=a(i,i)+1.0
end do
do k=n,1,-1
do its=1,30
do l=k,1,-1
nm=l-1
if((abs(rv1(l))+anorm)==anorm) exit
if((abs(w(nm))+anorm)==anorm) exit
end do
if (abs(rv1(l))+anorm/=anorm) then
c=0.0
s=1.0
do i=1,k
f=s*rv1(i)
if((abs(f)+anorm)/=anorm) then
g=w(i)
h=sqrt(f*f+g*g)
w(i)=h
h=1.0/h
c=(g*h)

```

```

      s = -(f * h)
      do j = 1, m
        y = a(j, nm)
        z = a(j, i)
        a(j, nm) = (y * c) + (z * s)
        a(j, i) = -(y * s) + (z * c)
      end do
    endif
  end do
end if
z = w(k)
if(l == k) then
  if(z < 0.0) then
    w(k) = -z
    do j = 1, n
      v(j, k) = -v(j, k)
    end do
  endif
  exit
endif
if(its == 30) pause 'no convergence in 30 iterations'
x = w(l)
nm = k - 1
y = w(nm)
g = rv1(nm)
h = rv1(k)
f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0 * h * y)
g = sqrt(f * f + 1.0)
f = ((x - z) * (x + z) + h * ((y / (f + sign(g, f))) - h)) / x
c = 1.0
s = 1.0
do j = 1, nm
  i = j + 1
  g = rv1(i)
  y = w(i)
  h = s * g
  g = g * c

```

```

z=sqrt(f*f+h*h)
rvl(j)=z
c=f/z
s=h/z
f=(x*c)+(g*s)
g=-(x*s)+(g*c)
h=y*s
y=y*c
do nm=1,n
    x=v(nm,j)
    z=v(nm,i)
    v(nm,j)=(x*c)+(z*s)
    v(nm,i)=-(x*s)+(z*c)
end do
z=sqrt(f*f+h*h)
w(j)=z
if(z/=0.0) then
    z=1.0/z
    c=f*z
    s=h*z
endif
f=(c*g)+(s*y)
x=-(s*g)+(c*y)
do nm=1,m
    y=a(nm,j)
    z=a(nm,i)
    a(nm,j)=(y*c)+(z*s)
    a(nm,i)=-(y*s)+(z*c)
end do
end do
rvl(l)=0.0
rvl(k)=f
w(k)=x
end do
end do
END SUBROUTINE svdcmp

```

(2) 子过程 SVBKSB.

```

SUBROUTINE svbksb(u,w,v,m,n,b,x)
PARAMETER (nmax=100)
REAL u(m,n),w(n),v(n,n),b(n),x(n),tnmax(nmax),s
INTEGER j,jj
do j=1,n
    s=0.0
    if (w(j)/=0.0) then
        do i=1,m
            s=s+u(i,j)*b(i)
        end do
        s=s/w(j)
    end if
    tnmax(j)=s
end do
do j=1,n
    s=0.0
    do jj=1,n
        s=s+v(j,jj)*tnmax(jj)
    end do
    x(j)=s
end do
END SUBROUTINE svbksb

```

5. 例子

子过程 SVDCMP 和 SVBKSB 与 1.2 节中子过程 LUDCMP 和 LUBKSB 的情况类似. 两者必须联合使用, 即先单独使用 SVDCMP 看能否产生适当的分解, 然后再联合使用对线性方程组求解.

在验证 SVBKSB 的例子中, 调用 SVBKSB 解线性方程组必须先调用子过程 SVDCMP, 将所得到的 U , W 和 V 输入子过程 SVBKSB. 为了验证解的正确性, 我们在验证程序 D1R8 中将解与原系数矩阵相乘, 与方程组右端的向量相比较. 验证程序 D1R8 如下:

```

PROGRAM D1R8
! Driver program for routine SVBKSB,SVDCMP
PARAMETER(N=5)

```

```

DIMENSION A(N,N),B(N),U(N,N),W(N)
DIMENSION V(N,N),X(N)
DATA A/1.4,1.6,3.8,4.6,2.6,2.1,1.5,8.,8.2,2.9,&
      2.1,1.1,9.6,8.4,0.1,7.4,0.7,5.4,0.4,9.9,&
      9.6,5.0,8.8,8.0,7.7/
DATA B/1.1,1.6,4.7,9.1,.1/
! Copy A into U
DO K=1,N
  DO L=1,N
    U(K,L)=A(K,L)
  END DO
END DO
! Decompose matrix A
CALL SVDCMP(U,N,N,W,V)
! Find maximum singular value
WMAX=0.0
DO K=1,N
  IF(W(K)>WMAX) WMAX=W(K)
END DO
! Define "small"
WMIN=WMAX*(1.0E-06)
! Zero the "small" singular values
DO K=1,N
  IF(W(K)<WMIN) W(K)=0.0
END DO
CALL SVBKSB(U,W,V,N,N,B,X)
WRITE(*,*) '    Solution vector is;'
WRITE(*,'(1X,6F12.6)')(X(K),K=1,N)
WRITE(*,*)
WRITE(*,*) '    Original right-hand side vector;'
WRITE(*,'(1X,6F12.6)')(B(K),K=1,N)
WRITE(*,*)
WRITE(*,*) '    Result of (matrix)*(sol''n vector);'
DO K=1,N
  B(K)=0.0
  DO J=1,N
    B(K)=B(K)+A(K,J)*X(J)
  
```

```

      END DO
END DO
WRITE(*, '(1X,6F12.6)') (B(K), K=1,N)
END

```

计算结果如下:

Solution vector is:

-5.208555 5.736682 -2.537465 -1.029813 0.968148

Original right-hand side vector:

1.100000 1.600000 4.700000 9.100000 0.100000

Result of (matrix) * (solution vector):

1.099984 1.599994 4.699993 9.099991 0.099982

1.9 线性方程组的共轭梯度法

1. 功能

用共轭梯度法求解高阶线性方程组 $Ax=b$, 由于该方法对矩阵的元素结构没有特殊要求, 所以, 它常用于求解一般大型稀疏矩阵的线性方程组.

2. 方法

考虑函数

$$f(x) = \frac{1}{2} \|Ax - b\|_2^2$$

极小化 $f(x)$ 即得解.

(1) 首先选取 $x^{(0)} \in R^n$, 并计算

$$S^{(0)} = b - Ax^{(0)}, \quad P^{(0)} = r^{(0)} = A^T S^{(0)}$$

(2) 对 $k=0, 1, \dots$

① 若 $P^{(k)}=0$, 则停止, 这时 $x^{(k)}$ 即为所求解, 否则转②.

② 计算

$$q^{(k)} = AP^{(k)}, \quad \alpha_k = \frac{r^{(k)T} r^{(k)}}{q^{(k)T} q^{(k)}}$$

$$x^{(k+1)} = x^{(k)} + \alpha_k P^{(k)}$$

$$S^{(k+1)} = S^{(k)} - \alpha_k q^{(k)}$$

$$r^{(k+1)} = A^T S^{(k+1)}, \quad \beta_k = \frac{r^{(k+1)T} r^{(k+1)}}{r^{(k)T} r^{(k)}}$$

$$P^{(k+1)} = r^{(k+1)} + \beta_k P^{(k)}$$

3. 使用说明

SPARSE (B,N,ASUB,ATSUB,X,RSQ)

N 整型变量,输入参数,方程组阶数

B 实型数组,输入参数,存放方程组的右端向量 b

X 实型数组,输入、输出参数,输入初始向量 $x^{(0)}$,输出解向量

RSQ 实型变量,输出参数,输出 $\|Ax - b\|_2$. 如果 RSQ 较大,即不太小,则说明矩阵是数值奇异的,这时相应的解为最小二乘逼近

ASUB 子过程名,用于计算 Ax ,在 ASUB(XIN,XOUT)中,XIN 为一维数组,输入 x 的值,XOUT 为一维数组,输出 Ax 的结果,该子过程由用户根据方程组的系数矩阵自编

ATSUB 子过程名,用于计算 $A^T x$,在 ATSUB(XIN,XOUT)中,XIN 为一维数组,输入 x 的值,XOUT 为一维数组,输出 $A^T x$ 的结果,该子过程由用户根据方程组的系数矩阵自编

4. 过程

子过程 SPARSE.

```
SUBROUTINE sparse(b,n,asub,atsub,x,rsq)
```

```
PARAMETER (nmax=500,eps=1.e-6)
```

```
DIMENSION b(n),x(n),g(nmax),h(nmax),xi(nmax),xj(nmax)
```

```
LOGICAL done
```

```
eps2=n*eps**2
```

```
irst=0
```

```
do
```

```
done=-1
```

```
irst=irst+1
```

```
call asub(x,xi)
```

```
rp=0.0
```

```
bsq=0.0
```

```
do j=1,n
```

```
bsq=bsq+b(j)**2
```

```
xi(j)=xi(j)-b(j)
```

```

      rp=rp+xi(j) * * 2
end do
call atsub(xi,g)
do j=1,n
  g(j)=-g(j)
  h(j)=g(j)
end do
do iter=1,10 * n
  call asub(h,xi)
  anum=0.
  aden=0.
  do j=1,n
    anum=anum+g(j) * h(j)
    aden=aden+xi(j) * * 2
  end do
  if(aden==0.0 ) pause 'very singular matrix'
  anum=anum/aden
  do j=1,n
    xi(j)=x(j)
    x(j)=x(j)+anum * h(j)
  end do
  call asub(x,xj)
  rsq=0.
  do j=1,n
    xj(j)=xj(j)-b(j)
    rsq=rsq + xj(j) * * 2
  end do
  if(rsq==rp. or. rsq<=bsq * eps2) return
  if(rsq>rp) then
    do j=1,n
      x(j)=xi(j)
    end do
    if(first>=3) return
    done=0
  end if
  if(.not. done) exit
  rp=rsq

```

```

      call atsub(xj,xi)
      gg=0.0
      dgg=0.0
      do j=1,n
        gg=gg+g(j)* * 2
        dgg=dgg+(xi(j)+g(j))* xi(j)
      end do
      if(gg==0.) return
      gam=dgg/gg
      do j=1,n
        g(j)=-xi(j)
        h(j)=g(j)+gam * h(j)
      end do
    end do
    if(.not. done) exit
  end do
  PAUSE 'too many iterations'
END SUBROUTINE sparse

```

5. 例子

我们所举的例子,其系数矩阵 $A(20 \times 20)$ 为

$$A = \begin{bmatrix} 1.0 & 2.0 & 0.0 & 0.0 & \cdots \\ -2.0 & 1.0 & 2.0 & 0.0 & \cdots \\ 0.0 & -2.0 & 1.0 & 2.0 & \cdots \\ 0.0 & 0.0 & -2.0 & 1.0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

方程组的右端向量 b 为: $(3.0, 1.0, 1.0, \dots, -1.0)$. 在验证程序 D1R9 中, 让 x 的每个分量的初值均为零. 读者可以自己任意指定初值, 这对结果通常没有影响. 在计算结果中我们对解进行了核查, 即让计算出的 x 与系数矩阵 A 相乘, 与方程组右端向量相比较, 程序 D1R9 如下:

```

PROGRAM D1R9
! Driver program for routine SPARSE
PARAMETER(N=20)
COMMON M
DIMENSION B(N),X(N),BCMP(N)

```

```

EXTERNAL ASUB,ATSUB
M=N
DO I=1,N
    X(I)=0.0
    B(I)=1.0
END DO
B(1)=3.0
B(N)=-1.0
CALL SPARSE(B,N,ASUB,ATSUB,X,RSQ)
WRITE(*, '(1X,A,E15.6)') 'Sum-squared residual:',RSQ
WRITE(*, '(1X,A)') 'Solution vector:'
WRITE(*, '(1X,5F12.6)') (X(I), I=1,N)
CALL ASUB(X,BCMP)
WRITE(*, '(1X,A)') 'press RETURN to END DO...'
READ(*,*)
WRITE(*, '(1X,A/T8,A,T22,A)') &
    'Test of solution vector:', 'a * x', 'b'
DO I=1,N
    WRITE(*, '(1X,2F12.6)') .BCMP(I),B(I)
END DO
END PROGRAM

SUBROUTINE ASUB(XIN,XOUT)
COMMON N
DIMENSION XIN(N),XOUT(N)
XOUT(1)=XIN(1)+2.0*XIN(2)
XOUT(N)=-2.0*XIN(N-1)+XIN(N)
DO I=2,N-1
    XOUT(I)=-2.0*XIN(I-1)+XIN(I)+2.0*XIN(I+1)
END DO
END SUBROUTINE ASUB

SUBROUTINE ATSUB(XIN,XOUT)
COMMON N
DIMENSION XIN(N),XOUT(N)
XOUT(1)=XIN(1)-2.0*XIN(2)
XOUT(N)=2.0*XIN(N-1)+XIN(N)
DO I=2,N-1
    XOUT(I)=2.0*XIN(I-1)+XIN(I)-2.0*XIN(I+1)

```

END DO

END SUBROUTINE ATSUB

计算结果如下:

Sum-squared residual: 0.401457E-12

Solution vector:

1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	1.000000	1.000000	1.000000	1.000000
1.000000	1.000000	1.000000	1.000000	1.000000

press RETURN to END DO...

Test of solution vector:

a * x	b
3.000000	3.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
1.000000	1.000000
-1.000000	-1.000000

1.10 对称方程组的乔列斯基(Cholesky)分解法

1. 功能

用不带平方根的乔列斯基方法解线性方程组 $Ax=b$, 其中 A 为 $n \times n$ 非奇异对称矩阵, 它不用开方运算, 精度较好, 且避免了开方中对纯虚数的处理工作.

子过程 CHODCM 实现矩阵的乔列斯基分解 $A=LDL^T$, 其中 L 为单位下三角矩阵, D 为对角矩阵; 子过程 CHOBSB 用子过程 CHODCM 的分解结果求解线性方程组.

2. 方法

(1) 设 $A=LDL^T$,

$$L = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ & \cdots & \cdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix}, \quad D = \text{diag}(d_1, \cdots, d_n)$$

则

$$d_1 = a_{11}, \quad l_{i1} = a_{i1}/d_1, \quad i = 2, \cdots, n$$

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} d_k) / d_j, \quad i = 3, \cdots, n$$

$$d_j = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_k, \quad j = 2, \cdots, n$$

为减少重复运算, 令 $t_{ik} = l_{ik} d_k$, 则

$$d_1 = a_{11}, \quad t_{i1} = a_{i1} = a_{1i}$$

$$l_{i1} = t_{i1}/d_1, \quad i = 2, \cdots, n$$

$$t_{ij} = a_{ij} - \sum_{k=1}^{j-1} t_{ik} l_{jk}$$

$$l_{ij} = t_{ij}/d_j$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} t_{ik} l_{ik}, \quad j = 2, \cdots, i-1; \quad i = 2, \cdots, n$$

(2) 求解 $Ax=b$ 等价于求解两个线性方程组 $Ly=b$ 与 $DL^T x=y$.

计算公式是

$$y_1 = b_1$$

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, \quad i = 2, \dots, n$$

$$x_n = y_n / d_n$$

$$x_i = y_i / d_i - \sum_{j=i+1}^n l_{ji} x_j, \quad i = n-1, \dots, 1$$

3. 使用说明

(1) CHODCM (A,N,D,T)

N 整型变量,输入参数,矩阵 A 的阶数

A $N \times N$ 个元素的二维实型数组,输入、输出参数,输入时存放 A 的上三角部分,严格下三角部分可置为任意数,输出时严格下三角部分存放分解 L 的严格下三角部分,而上三角部分不变

D N 个元素的一维实型数组,输出参数,存放分解中对角矩阵 D 的对角元素

T N 个元素的一维实型数组,工作单元

(2) CHOBSB (A,N,D,B)

N 整型变量,输入参数,方程阶数

A $N \times N$ 个元素的二维实型数组,输入参数,存放子过程 CHODCM 输出的 A

D N 个元素的一维实型数组,输入参数,存放子过程 CHODCM 输出的 D

B N 个元素的一维实型数组,输入、输出参数,输入时存放方程组的右端向量,输出时存放解向量

4. 过程

(1) 子过程 CHODCM.

```
SUBROUTINE chodcm(a,n,d,t)
```

```
REAL a(n,n),d(n),t(n)
```

```
INTEGER i,j
```

```
do i=1,n
```

```
    sum=a(i,i)
```

```
    do j=1,i-1
```

```
        t(j)=a(j,i)
```

```

do k=1,j-1
  t(j)=t(j)-t(k)*a(j,k)
end do
if (d(j)==0.) then
  if (t(j)/=0.) then
    pause 'no cholesky decomposition'
  else
    a(i,j)=1.0
  endif
else
  a(i,j)=t(j)/d(j)
endif
sum=sum-t(j)*a(i,j)
end do
d(i)=sum
end do
END SUBROUTINE chodcm

```

(2) 子过程 CHOBSB.

```

SUBROUTINE chobsb(a,n,d,b)
REAL a(n,n),d(n),b(n)
INTEGER i,j
do i=1,n
  sum=b(i)
  do j=1,i-1
    sum=sum-a(i,j)*b(j)
  end do
  b(i)=sum
end do
do i=n,1,-1
  if (d(i)==0.) then
    pause 'singular matrix'
    return
  else
    sum=b(i)/d(i)
  endif
  do j=i+1,n

```



```

        sum=sum-a(j,i)*b(j)
    end do
    b(i)=sum
end do
END SUBROUTINE chobsb

```

5. 例子

在验证子过程 CHOBSB 的例子中,子过程 CHOBSB 必须与子过程 CHODCM 联合使用. 验证子过程 CHOBSB 的验证程序 D1R10 中首先调用 CHODCM,再调用 CHOBSB 得到方程组的解,然后把原矩阵与由 CHOBSB 得到的解向量相乘和方程组的右端向量比较. 注意,这里仅用原矩阵的上三角部分的元素,对原矩阵作为对称矩阵处理. 验证程序 D1R10 如下:

```

PROGRAM D1R10
! Driver program for routine CHOBSB,CHODCM
PARAMETER(N=5)
DIMENSION A(N,N),B(N),D(N),T(N)
DIMENSION C(N,N),X(N)
DATA A/1.4,1.6,3.8,4.6,2.6,2.1,1.5,8.,8.2,2.9,&
      2.1,1.1,9.6,8.4,0.1,7.4,0.7,5.4,0.4,9.9,&
      9.6,5.0,8.8,8.0,7.7/
DATA B/4.0,9.3,8.4,0.4,4.1/
! Save matrix a for later testing
DO L=1,N
    DO K=1,N
        C(K,L)=A(K,L)
    END DO
END DO
! Do Cholesky decomposition
CALL CHODCM(C,N,D,T)
! Solve equations for each right-hand vector
DO K=1,N
    X(K)=B(K)
END DO
CALL CHOBSB(C,N,D,X)
! Test results with original matrix
WRITE(*,*) 'Right-hand side vector'

```

```

WRITE(*,'(1X,5F12.6)')(B(L),L=1,N)
WRITE(*,*)
WRITE(*,*)'Solution vector'
WRITE(*,'(1X,5F12.6)')(X(L),L=1,N)
WRITE(*,*)
WRITE(*,*)'Result of matrix applied to sol''n vector'
DO L=1,N
    B(L)=0.0
    DO J=1,L
        B(L)=B(L)+A(J,L)*X(J)
    END DO
    DO J=L+1,N
        B(L)=B(L)+A(L,J)*X(J)
    END DO
END DO
WRITE(*,'(1X,5F12.6)')(B(L),L=1,N)
END

```

计算结果如下:

Right-hand side vector

4.000000 9.300000 8.400000 0.400000 4.100000

Solution vector

-2.484251 5.972712 1.055125 -2.596297 1.242926

Result of matrix applied to sol'n vector

4.000000 9.299999 8.400000 0.400002 4.100000

1.11 矩阵的 QR 分解

1. 功能

用镜像矩阵求出矩阵的 QR 分解(正交三角分解),并求解线性方程组,该方法不必选主元,但其计算过程非常稳定.也可用于求矩阵的广义逆和求解线性最小二乘问题.

子过程 QRDCMP 用镜像矩阵求 $m \times n$ 矩阵 A 的 QR 分解 $A=Q^T R$,其中 Q 是 $m \times m$ 正交矩阵, R 为 $m \times n$ 上三角矩阵;子过程 QRBKSB 用矩阵的 QR 分解求解线性方程组 $Ax=b$,其中 A 为 n 阶非奇异方阵.

2. 方法

(1) $m \times n$ 矩阵 A 的 QR 分解.

① 求 n 个 m 阶正交阵 Q , $k=1, \dots, n$, 使

$$R = A^{(m)} = Q_n Q_{n-1} \cdots Q_1 A$$

为上三角矩阵.

② 设 $A^{(1)} = A$, 对 A 已进行了 $k-1$ 步变换, 使

$$A^{(k)} = Q_{k-1} A^{(k-1)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & A_{13}^{(k)} \\ & A_{22}^{(k)} & A_{23}^{(k)} \\ & & & \end{bmatrix}$$

$A_{11}^{(k)} \in R^{(k-1) \times (k-1)}$ 为上三角矩阵.

其中

$$\begin{aligned} A_{12}^{(k)} &\in R^{(k-1) \times 1}, & A_{13}^{(k)} &\in R^{(k-1) \times (n-k)} \\ A_{22}^{(k)} &\in R^{(m-k+1) \times 1}, & A_{23}^{(k)} &\in R^{(m-k+1) \times (n-k)} \end{aligned}$$

则第 k 步的计算公式是:

(a) $\alpha = -\operatorname{sgn}(a_{kk}^{(k)}) \|A_{22}^{(k)}\|_2$;

(b) $\tilde{u}^{(k)} = A_{22}^{(k)} - \alpha e_1^{(k)}$, $e_1^{(k)} = (1, 0, \dots, 0)^T \in R^{m-k+1}$;

$u^{(k)} = (0, \dots, 0, \tilde{u}^{(k)}) \in R^m$;

(c) $h_k = \frac{1}{2} \|u^{(k)}\|_2$;

(d) $x^{(k)} = [A^{(k)}]^T u^{(k)} / h_k$;

(e) $A^{(k+1)} = Q_k A^{(k)} = \left(I - \frac{1}{h_k} u^{(k)} u^{(k)T} \right) A^{(k)} = A^{(k)} - u^{(k)} [x^{(k)}]^T$.

(2) 方程组 $Ax=b$ 等价于

$$y = Qb, \quad Rx = y$$

其解的计算公式是

$$y_i = \sum_{j=1}^n q_{ij} b_j, \quad i = 1, \dots, n$$

$$x_n = y_n / r_{nn}$$

$$x_i = y_i - \sum_{j=i+1}^n r_{ij} x_j, \quad i = n-1, \dots, 1$$

其中

$$Q = (q_{ij})_{n \times n}, \quad R = (r_{ij})_{n \times n}$$

3. 使用说明

(1) QRDCMP (A, M, N, Q)

- M,N 整型变量,输入参数,分别为矩阵 A 的行数(M)和列数(N)
- A $M \times N$ 个元素的二维实型数组,输入、输出参数,输入时存放要分解的矩阵 A ,输出时存放 QR 分解的上三角矩阵 R
- Q $M \times M$ 个元素的二维实型数组,输出参数,存放分解 $A = Q^T R$ 中的 M 阶正交矩阵 Q
- (2) QRBKSB (A,N,Q,B,X)
- N 整型变量,输入参数,方程阶数
- A $N \times N$ 个元素的二维实型数组,输入参数,输入由子过程 QRDCMP 输出的 A
- Q $N \times N$ 个元素的二维实型数组,输入参数,存放由子过程 QRDCMP 输出的 Q
- B N 个元素的一维实型数组,输入参数,存放方程组的右端向量
- X N 个元素的一维实型数组,输出参数,存放方程组的解向量

4. 过程

(1) 子过程 QRDCMP.

```

SUBROUTINE qrdcmp(a,m,n,q)
REAL a(n,n),q(m,m)
INTEGER i,k,j
REAL s,t,f,h
do i=1,m
  do j=1,m
    q(i,j)=0.
  end do
  q(i,i)=1.
end do
do k=1,m-1
  s=0.
  do i=k,m
    s=s+abs(a(i,k))
  end do
  if (s/=0.) then
    r=0.
    do i=k,m
      a(i,k)=a(i,k)/s
    end do
  end if
end do

```

```

        t=t+a(i,k)*a(i,k)
    end do
    t=-sign(sqrt(t),a(k,k))
    a(k,k)=a(k,k)-t
    h=-t*a(k,k)
    do j=k+1,n
        f=0.
        do i=k,m
            f=f+a(i,k)*a(i,j)
        end do
        f=f/h
        do i=k,m
            a(i,j)=a(i,j)-a(i,k)*f
        end do
    end do
    do j=1,m
        f=0.
        do i=k,m
            f=f+a(i,k)*q(i,j)
        end do
        f=f/h
        do i=k,m
            q(i,j)=q(i,j)-a(i,k)*f
        end do
    end do
    a(k,k)=t*s
    do i=k+1,m
        a(i,k)=0.
    end do
endif
end do
END SUBROUTINE qrdcmp

```

(2) 子过程 QRBKSB.

```

SUBROUTINE qrbksb(a,n,q,b,x)
REAL a(n,n),q(n,n),b(n),x(n)
INTEGER i,j

```

```

REAL sum
do i=1,n
  sum=0.
  do j=1,n
    sum=sum+q(i,j)*b(j)
  end do
  x(i)=sum
end do
do i=n,1,-1
  sum=x(i)
  do j=i+1,n
    sum=sum-a(i,j)*x(j)
  end do
  if (a(i,i)==0.0) pause 'a is singular matrix.'
  x(i)=sum/a(i,i)
end do
END SUBROUTINE qrbksb

```

5. 例子

子过程 QRBKSB 是用矩阵的 QR 分解求解线性方程组,它必须与子过程 QRDCMP 联合使用. 验证 QRBKSB 的验证程序 D1R11 中首先调用 QRDCMP, 然后再调用 QRBKSB 求得线性方程组的解,并把原系数矩阵与所求得解向量相乘后与方程组的右端向量相比较,验证程序 D1R11 如下:

```

PROGRAM D1R11
! Driver program for routine QRBKSB,QRDCMP
PARAMETER(N=5)
DIMENSION A(N,N),B(N),Q(N,N)
DIMENSION C(N,N),X(N),R(N)
DATA A/1.4,1.6,3.8,4.6,2.6,2.1,1.5,8.,8.2,2.9,&
      2.1,1.1,9.6,8.4,0.1,7.4,0.7,5.4,0.4,9.9,&
      9.6,5.0,8.8,8.0,7.7/
DATA B/1.1,1.6,4.7,9.1,.1/
! Save matrix a for later testing
DO L=1,N
  DO K=1,N
    C(K,L)=A(K,L)

```

```

      END DO
END DO
! Do QR decomposition
CALL QRDCMP(C,N,N,Q)
! Solve equations for each right-hand vector
DO L=1,N
      R(L)=B(L)
END DO
CALL QRBKSB(C,N,Q,R,X)
WRITE(*,*)
WRITE(*,*) 'Solution vector'
WRITE(*,'(1X,5F12.6)') (X(L), L=1,N)
! Test results with original matrix
WRITE(*,*)
WRITE(*,*) 'Right-hand side vector'
WRITE(*,'(1X,5F12.6)') (B(L), L=1,N)
WRITE(*,*)
WRITE(*,*) 'Result of matrix applied to sol'n vector'
DO L=1,N
      B(L)=0.0
      DO J=1,N
            B(L)=B(L)+A(L,J)*X(J)
      END DO
END DO
WRITE(*,'(1X,5F12.6)') (B(L), L=1,N)
END

```

计算结果如下:

Solution vector

-5.208570 5.736697 -2.537474 1.029814 0.968152

Right-hand side vector

1.100000 1.600000 4.700000 9.100000 0.100000

Result of matrix applied to sol'n vector

1.100002 1.600001 4.699999 9.100002 0.100001

1.12 松弛迭代法

1. 功能

用松弛法求解线性代数方程组 $Ax=b$, 其中 A 为 $n \times n$ 阶方阵, b 为 n 维向量.

2. 方法

令 $A=D-L-U$, 其中 D 为 A 的对角线部分, $-L, -U$ 分别为 A 的严格下、上三角部分, 则松弛迭代法求解线性方程组 $Ax=b$ 的迭代格式是

$$\begin{aligned} x^{(k+1)} &= (1-\omega)x^{(k)} + \omega D^{-1}(b + Lx^{(k+1)} + Ux^{(k)}) \\ k &= 0, 1, 2, \dots \end{aligned}$$

即

$$\begin{aligned} x_i^{(k+1)} &= (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) \\ &= x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) \\ i &= 1, 2, \dots, n; \quad k = 0, 1, 2, \dots \end{aligned} \quad (1-20)$$

其中 $\omega \in (0, 2)$ 称为松弛因子. $\omega=1$ 时即为高斯-塞德尔(Gauss-Seidel)迭代法.

若 A 具有相容次序, 则最佳松弛因子为

$$\omega_b = 2/[1 + \sqrt{1 - \rho(B_J)^2}] = 2(1 - \sqrt{1 - \rho(B_J)^2})/\rho(B_J)^2 \quad (1-21)$$

其中 $B_J = D^{-1}(L+U)$, $\rho(B_J)$ 表示 B_J 的谱半径, 即按模最大的特征值的模.

松弛法的迭代矩阵是

$$B_\omega = (I - \omega L)^{-1}[(1-\omega)I + \omega U]$$

松弛法收敛的充分必要条件是 $\rho(B_\omega) < 1$. 若系数矩阵 A 对称正定, 当 $0 < \omega < 2$ 时松弛迭代法必定收敛.

在实际计算中, 常取 $\omega=1$ 或 $\omega=1.5$ 先迭代若干次(比如为 m 次), 由

$$\rho(B_{SOR}) \approx \frac{\|x^{(m)} - x^{(m-1)}\|_\infty}{\|x^{(m-1)} - x^{(m-2)}\|_\infty} = \bar{\lambda} \quad (1-22)$$

使 $\bar{\lambda}$ 有相当位有效数字后, 再由

$$\rho(B_J) = \frac{\rho(B_{SOR}) + \omega - 1}{\omega \sqrt{\rho(B_{SOR})}}$$

得 $\rho(B_J)$ 的近似值, 代入式(1-21)得 ω_b 的近似值, 这里 B_{SOR} 为松弛迭代矩阵

$$B_{SOR} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]$$

在实用中, 通常也采用试算的办法选取最佳松弛因子, 即从同一个初始向量出发, 取不同的松弛因子(可将其中一个取为 $\omega=1$ 或 1.5)迭代相同次数(不宜次数太少), 比较残余向量

$$r^{(k)} = b - Ax^{(k)}$$

弃去使 $\|r^{(k)}\|$ 较大的松弛因子. 这一方法虽然简单, 但往往是有效的, 特别是需要求解多个具有相同系数矩阵的方程组时, 更为有效.

3. 使用说明

SSOR (A,N,B,X,EPS,OM,II)

N 整型变量, 输入参数, 方程组的阶数

A $N \times N$ 个元素的二维实型数组, 输入参数, 存放方程组的系数矩阵 A

B N 个元素的一维实型数组, 输入参数, 存放方程组的右端向量 b

X N 个元素的一维实型数组, 输入、输出参数, 输入时存放迭代的初始近似解, 输出时存放所求得的近似解

EPS 实型变量, 输入参数, 解的精度

OM 实型变量, 输入参数, 存放松弛因子

II 整型变量, 输出参数, 最终迭代次数

4. 过程

子过程 SSOR.

```
SUBROUTINE ssor(a,n,b,x,eps,om,ii)
```

```
PARAMETER (imax=200)
```

```
REAL a(n,n),b(n),x(n)
```

```
INTEGER i,j,ii
```

```
REAL r,rx
```

```
do i=1,n
```

```
    r=1./a(i,i)
```

```
    b(i)=b(i)*r
```

```
    do j=1,n
```

```
        a(i,j)=a(i,j)*r
```

```
    end do
```

```
end do
```

```

do ii=1,imax
  rx=0.0
  do i=1,n
    r=b(i)
    do j=1,n
      r=r-a(i,j)*x(j)
    end do
    if (abs(r)>rx) rx=abs(r)
    x(i)=x(i)+om*r
  end do
  if (om*rx<=eps) return
end do
PAUSE 'Too many iterations'
END SUBROUTINE ssor

```

5. 例子

在过程中取松弛因子 $\omega=1.3$, 验证程序 D1R12 把求得的数值解与系数矩阵相乘后和方程组的右端向量相比较. 验证程序 D1R12 如下:

```

PROGRAM D1R12
! Driver program for routine SSOR
PARAMETER(N=5,eps=1.E-7,OM=1.3)
DIMENSION A(N,N),C(N,N),R(N),B(N),X(N)
DATA A/1.,1.,0.,0.,0.,1.,2.,1.,0.,0.,&
0.,1.,3.,1.,0.,0.,0.,1.,4.,1.,&
0.,0.,0.,1.,5./
DATA B/2.,4.,5.,6.,6./
DO I=1,N
DO J=1,N
C(I,J)=A(I,J)
END DO
END DO
DO L=1,N
  R(L)=B(L)
  X(L)=0.0
END DO
CALL SSOR(C,N,R,X,EPS,OM,II)

```

```
WRITE(*,*) 'Solution vector'
WRITE(*, '(1X,5F12.6)') (X(L), L=1,N)
! Test results with original matrix
WRITE(*,*) 'Right-hand side vector'
WRITE(*, '(1X,5F12.6)') (B(L), L=1,N)
WRITE(*,*) 'Result of matrix applied to sol''n vector'
DO L=1,N
    B(L)=0.0
    DO J=1,N
        B(L)=B(L)+A(L,J)*X(J)
    END DO
END DO
WRITE(*, '(1X,5F12.6)') (B(L), L=1,N)
WRITE(*,*) 'Iteration times =',II
END
```

计算结果如下:

```
Solution vector
    1.000000    1.000000    1.000000    1.000000    1.000000
Right-hand side vector
    2.000000    4.000000    5.000000    6.000000    6.000000
Result of matrix applied to sol'n vector
    2.000000    4.000000    5.000000    6.000000    6.000000
Iteration times =17
```

第 2 章 插 值

本章介绍代数插值法、有理函数插值法、样条函数插值法和多维插值. 它们既可用于等距节点的插值, 也可用于不等距节点的插值; 既可用于内插, 也可用于外插. 三角几何函数的插值与傅里叶方法有关, 请参看第 12 章.

拉格朗日插值(见 2.1 节)是最基本、最简单的插值方法, 也是数值积分和求常微分方程数值解的重要工具. 该方法是用一个 n 次多项式去拟合一个函数 $f(x)$, 随着多项式方次 n 的增加, 不仅增大了计算工作量, 而且有时会带来很大的误差, 如“龙格”(Runge)现象. 因此 n 不宜太大, 使用时通常以不超过五六次为宜. 为了避免出现这种振荡误差, 方法之一是采用分段低次插值, 如线性插值或抛物插值. 它们不仅简便, 而且逼近效果往往优于高次拉格朗日插值, 不足之处是在各小区间的连接处其导数不连续, 这会引起失真, 不便于理论分析和工程设计. 方法之二是采用分段光滑插值. 在每两个插值节点的小区间内用低次插值, 但又保证在插值节点处导数连续. 也就是在整个插值区间内, 插值函数的曲线是光滑的, 这就是所谓的样条函数插值. 三次样条插值(见 2.3 节)是常见的样条插值法, 其插值函数二阶导数连续. 方法之三是采用有理逼近(见 2.2 节). 一般来说, 在插值节点相同的条件下, 利用有理分式逼近 $f(x)$ 要比插值多项式逼近的精度高一些.

在分段插值中, 对给定的某个特定的插值变量 x , 要求出该点的函数值时, 需事先快速确定 x 的位置. 这时可采用有序表的检索法(见 2.4 节)中的子过程, 其中子过程 LOCATE 用于一般情形的搜寻, 而子过程 HUNT 用于多个相近的插值变量连续地在同一有序表中进行多次搜寻. 用这两个子过程可进行各种插值的分段插值.

有时我们希望知道的不是通过若干个点上的插值多项式的函数值, 而是插值多项式的系数, 这时可采用插值多项式(见 2.5 节)中介绍的方法.

2.6 节和 2.7 节中介绍了二维插值, 实际上多维插值一般由一系列一维插值来完成.

2.1 拉格朗日插值

1. 功能

给定函数 $y=f(x)$ 在 n 个不同插值节点 $x_i (i=1, \dots, n)$ 的函数值 $y_i=f(x_i)$ ($i=1, \dots, n$), 用拉格朗日 (Lagrange) 插值多项式求函数在 x 处的函数值 y .

2. 方法

(1) 拉格朗日的经典公式

$$P(x) = \sum_{k=1}^n f(x_k) l_k(x)$$

其中

$$l_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^n \frac{x - x_j}{x_k - x_j}$$

(2) 递推关系 (Neville 算法)

记 $P_{j,k+1}(x)$ 为 $f(x)$ 关于节点 $x_j, x_{j+1}, \dots, x_{j+k}$ 的 k 次拉格朗日插值多项式, 则

$$P_{j,k+1}(x) = \frac{(x - x_j)P_{j+1,k}(x) - (x - x_{j+k})P_{j,k}(x)}{x_{j+k} - x_j}$$

其中

$$P_{j,1} = y_j, \quad j = 1, \dots, n-k; \quad k = 1, \dots, n-1$$

为改善精度, 保持小的误差, 定义

$$C_{j,k} \equiv P_{j,k} - P_{j,k-1}$$

$$D_{j,k} \equiv P_{j,k} - P_{j+1,k}, \quad j = 2, \dots, n-k+1$$

$$C_{j,1} \equiv D_{j,1} \equiv P_{j,1} = y_j, \quad k = 1, \dots, n$$

则

$$C_{j,k+1} = \frac{(x - x_j)(C_{j+1,k} - D_{j,k})}{x_{j+k} - x_j}$$

$$D_{j,k+1} = \frac{(x - x_{j+k})(C_{j+1,k} - D_{j,k})}{x_{j+k} - x_j}$$

$$j = 1, \dots, n-k; \quad k = 1, \dots, n-1$$

最终结果 $P_{1,n}(x)$ 由 $P_{j,k} = P_{j,k-1} + C_{j,k} = P_{j+1,k-1} + D_{j,k}$ 计算.

3. 使用说明

POLINT (XA,YA,N,X,Y,DY)

N 整型变量,输入参数,节点个数

XA N 个元素的一维实型数组,输入参数,存放插值节点

YA N 个元素的一维实型数组,输入参数,存放函数值 $(y_1, \dots, y_n)^T$

X 实型变量,输入参数,插值自变量

Y 实型变量,输出参数,所求值

DY 实型变量,输出参数,误差估计

4. 过程

子过程 POLINT.

```
SUBROUTINE polint(xa,ya,n,x,y,dy)
```

```
INTEGER n,NMAX
```

```
REAL dy,x,y,xa(n),ya(n)
```

```
PARAMETER (NMAX=10)
```

```
INTEGER i,m,ns
```

```
REAL den,dif,dift,ho,hp,w,c(NMAX),d(NMAX)
```

```
ns=1
```

```
dif=abs(x-xa(1))
```

```
do i=1,n
```

```
    dift=abs(x-xa(i))
```

```
    if (dift<dif) then
```

```
        ns=i
```

```
        dif=dift
```

```
    endif
```

```
    c(i)=ya(i)
```

```
    d(i)=ya(i)
```

```
end do
```

```
y=ya(ns)
```

```
ns=ns-1
```

```
do m=1,n-1
```

```
    do i=1,n-m
```

```
        ho=xa(i)-x
```

```
        hp=xa(i+m)-x
```

```

      w=c(i+1)-d(i)
      den=ho-hp
      if(den==0.) pause 'failure in polint'
      den=w/den
      d(i)=hp*den
      c(i)=ho*den
    end do
    if (2*ns<n-m) then
      dy=c(ns+1)
    else
      dy=d(ns)
      ns=ns-1
    endif
    y=y+dy
  end do
END SUBROUTINE polint

```

5. 例子

我们所取的例子是 $f(x)=\sin x$ 和 $f(x)=e^x$. $\sin x$ 的区间是 $(0, \pi)$, e^x 的区间是 $(0, 1.0)$. 验证程序中的 N 是任意给定的 ($N < 10$), 当 N 增长时, 我们可以观察到结果在不断地改进. 验证程序 D2R1 如下:

```

PROGRAM D2R1
! Driver for routine POLINT
PARAMETER(NP=10,PI=3.1415926)
DIMENSION XA(NP),YA(NP)
WRITE(*,*)'Generation of interpolation tables'
WRITE(*,*)'... sin(x)    0<x<PI'
WRITE(*,*)'... exp(x)    0<x<1'
WRITE(*,*)'How many entries go in these tables? (Note: N<10)'
READ(*,*) N
DO NFUNC=1,2
  IF (NFUNC==1) THEN
    WRITE(*,*)'sine function from 0 to PI'
    DO I=1,N
      XA(I)=I*PI/N
      YA(I)=SIN(XA(I))
    
```

```

      END DO
    ELSE IF (NFUNC==2) THEN
      WRITE(*,*) 'exponential function from 0 to 1'
      DO I=1,N
        XA(I)=I*1.0/N
        YA(I)=EXP(XA(I))
      END DO
    ELSE
      STOP
    ENDIF
    WRITE(*, '(T10,A1,T20,A4,T28,A12,T46,A5)') &
      'x', 'f(x)', 'interpolated', 'error'
    DO I=1,10
      IF(NFUNC==1) THEN
        X=(-0.05+I/10.0)*PI
        F=SIN(X)
      ELSE IF(NFUNC==2) THEN
        X=(-0.05+I/10.)
        F=EXP(X)
      ENDIF
      CALL POLINT(XA,YA,N,X,Y,DY)
      WRITE(*, '(1X,3F12.6,E15.4)') X,F,Y,DY
    END DO
    WRITE(*,*) '*****'
    WRITE(*,*) 'Press RETURN'
    READ(*,*)
  END DO
END

```

计算结果如下：

Generation of interpolation tables

... sin(x) $0 < x < \text{PI}$

... exp(x) $0 < x < 1$

How many entries go in these tables? (Note: $N < 10$)

9

sine function from 0 to PI

x	f(x)	interpolated	error
0.157080	0.156434	0.156436	0.5028E-04
0.471239	0.453990	0.453990	- 0.3315E-05
0.785398	0.707107	0.707107	0.6029E-06
1.099557	0.891007	0.891006	-0.1623E-06
1.413717	0.987688	0.987688	0.3706E-07
1.727876	0.987688	0.987688	0.3704E-07
2.042035	0.891007	0.891007	-0.1625E-06
2.356194	0.707107	0.707107	0.6047E-06
2.670354	0.453991	0.453991	-0.3320E-05
2.984513	0.156435	0.156434	-0.2999E-05

Press RETURN

exponential function from 0 to 1

x	f(x)	interpolated	error
0.050000	1.051271	1.051273	0.7595E-06
0.150000	1.161834	1.161834	0.4368E-07
0.250000	1.284025	1.284026	0.8988E-08
0.350000	1.419068	1.419067	-0.3323E-08
0.450000	1.568312	1.568312	0.5615E-09
0.550000	1.733253	1.733253	0.5791E-09
0.650000	1.915541	1.915541	-0.2417E-08
0.750000	2.117000	2.117000	0.1004E-07
0.850000	2.339647	2.339647	-0.3210E-07
0.950000	2.585710	2.585710	-0.1088E-06

Press RETURN

2.2 有理函数插值

1. 功能

给定函数 $f(x)$ 在 $n+1$ 个互异的节点 $x_i (i=0, 1, \dots, n)$ 处的函数值 $y_i =$

$f(x_i)$, 用满足

$$R(x_i) = y_i \quad (i = 0, 1, \dots, n)$$

的有理分式 $R(x)$ 求函数在 x 点的函数值 y .

2. 方法

我们采用 Bulirsch 和 Stoer 的 Neville 型算法.

记 $R_{i(i+1)\dots(i+m)}$ 为通过 $m+1$ 个点 (x_{i+j}, y_{i+j}) , $j=0, 1, \dots, m$ 的有理函数, 即设

$$R_{i(i+1)\dots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1x + \dots + p_\mu x^\mu}{q_0 + q_1x + \dots + q_\nu x^\nu}$$

其中

$$m+1 = \mu + \nu + 1$$

$$R_{i(i+1)\dots(i+m)} = \begin{cases} 0, & \text{当 } m = -1 \text{ 时} \\ y_i, & \text{当 } m = 0 \text{ 时} \end{cases}$$

则

$$R_{i(i+1)\dots(i+m)} =$$

$$R_{(i+1)\dots(i+m)} + \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{\left(\frac{x - x_i}{x - x_{i+m}}\right) \left(1 - \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{R_{(i+1)\dots(i+m)} - R_{(i+1)\dots(i+m-1)}}\right)} - 1$$

定义

$$C_{m,i} \equiv R_{i\dots(i+m)} - R_{i\dots(i+m-1)}$$

$$D_{m,i} \equiv R_{i\dots(i+m)} - R_{(i+1)\dots(i+m)}$$

则

$$C_{m+1,i} - D_{m+1,i} = C_{m,i+1} - D_{m,i}$$

$$D_{m+1,i} = \frac{C_{m,i+1}(C_{m,i+1} - D_{m,i})}{\left(\frac{x - x_i}{x - x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}}$$

$$C_{m+1,i} = \frac{\left(\frac{x - x_i}{x - x_{i+m+1}}\right) D_{m,i} (C_{m,i+1} - D_{m,i})}{\left(\frac{x - x_i}{x - x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}}$$

3. 使用说明

RATINT (XA,YA,N,X,Y,DY)

N 整型变量, 输入参数, 插值节点数

XA N 个元素的一维实型数组,输入参数,存放节点值
 YA N 个元素的一维实型数组,输入参数,存放节点的相应函数值
 X 实型变量,输入参数,插值自变量
 Y 实型变量,输出参数,插值结果
 DY 实型变量,输出参数,精度估计

4. 过程

子过程 RATINT.

```

SUBROUTINE ratint(xa,ya,n,x,y,dy)
INTEGER n,NMAX
REAL dy,x,y,xa(n),ya(n),TINY
PARAMETER (NMAX=10,TINY=1.e-25)
INTEGER i,m,ns
REAL dd,h,hh,l,w,c(NMAX),d(NMAX)
ns=1
hh=abs(x-xa(1))
do i=1,n
    h=abs(x-xa(i))
    if (h==0.)then
        y=ya(i)
        dy=0.0
        return
    else if (h<hh) then
        ns=i
        hh=h
    endif
    c(i)=ya(i)
    d(i)=ya(i)-TINY
end do
y=ya(ns)
ns=ns-1
do m=1,n-1
    do i=1,n-m
        w=c(i+1)-d(i)
        h=xa(i+m)-x
        t=(xa(i)-x)*d(i)/h
    
```

```

      dd=t-c(i+1)
      if(dd==0.) pause 'failure in ratint'
      dd=w/dd
      d(i)=c(i-1)*dd
      c(i)=t*dd
    end do
    if (2*ns<=n-m)then
      dy=c(ns+1)
    else
      dy=d(ns)
      ns=ns-1
    endif
    y=y+dy
  end do
END SUBROUTINE ratint

```

5. 例子

我们采用以下已知函数 $F(x)$ 作为例子：

$$F(x) = \frac{xe^{-x}}{(x-1)^2 + 1}$$

仔细观察可以发现,有两个试验点和给定的参考点相重合,因而给出精确结果. 其余的点是插值结果. 计算结果给出的估计误差为 DYY,读者可以和实际误差 $|YY-YEXP|$ 相比较,实际误差均低于估计误差. 验证程序 D2R2 如下:

```

PROGRAM D2R2
  ! Driver for routine RATINT
  PARAMETER(NPT=6,EPSSQ=1.0)
  DIMENSION X(NPT),Y(NPT)
  F(X)=X*EXP(-X)/((X-1.0)**2+EPSSQ)
  DO I=1,NPT
    X(I)=I*2.0/NPT
    Y(I)=F(X(I))
  END DO
  WRITE(*, '(1X,A/)' )&
    'Diagonal rational function interpolation'
  WRITE(*, '(1X,T6,A,T13,A,T26,A,T40,A)' )&
    'x','interap.','accuracy','actual'

```

```

DO I=1,10
  XX=0.2 * I
  CALL RATINT(X,Y,NPT,XX,YY,DYY)
  YEXP=F(XX)
  WRITE( *, '(1X,F6.2,F12.6,E15.4,F12.6)') XX,YY,DYY,YEXP
END DO
END

```

计算结果如下:

Diagonal rational function interpolation

x	interap.	accuracy	actual
0.20	0.093921	-0.1208E-01	0.099845
0.40	0.197969	0.2039E-02	0.197153
0.60	0.284187	0.9732E-03	0.283868
0.80	0.345369	-0.9988E-03	0.345638
1.00	0.367879	0.0000E+00	0.367879
1.20	0.347641	0.6608E-03	0.347532
1.40	0.297566	-0.4332E-03	0.297617
1.60	0.237480	-0.6207E-03	0.237525
1.80	0.181508	0.2497E-02	0.181426
2.00	0.135335	0.0000E+00	0.135335

2.3 三次样条插值

1. 功能

给定函数 $y=f(x)$ 的样条节点 $x_1 < x_2 < \cdots < x_n$ 及对应函数值 $y_i (i=1, \cdots, n)$, 用三次样条函数求函数在插值点 x 的值 y .

这里给出两个子过程: SPLINE 和 SPLINT. 前者用于计算插值函数在节点处的二阶导数值, 后者计算插值函数在插值点的函数值.

2. 方法

(1) 在每个区间 $[x_{j-1}, x_j]$ 上, 三次样条函数可表示为:

$$S(x) = M_{j-1} \frac{(x_j - x)^3}{6h_{j-1}} + M_j \frac{(x - x_{j-1})^3}{6h_{j-1}} + \left(y_{j-1} - \frac{M_{j-1}h_{j-1}^2}{6} \right) \frac{x_j - x}{h_{j-1}} \\ + \left(y_j - \frac{M_j h_{j-1}^2}{6} \right) \frac{x - x_{j-1}}{h_{j-1}}, \quad x \in [x_{j-1}, x_j]$$

其中

$$h_{j-1} = x_j - x_{j-1}, \quad M_j = S'(x_j)$$

(2) M_j 所满足的方程

$$\mu_j M_{j-1} + 2M_j + \lambda_j M_{j+1} = d_j$$

其中

$$\mu_j = \frac{h_{j-1}}{h_{j-1} + h_j}, \quad \lambda_j = 1 - \mu_j \\ d_j = 6 \left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}} \right) (h_{j-1} + h_j)^{-1} \\ j = 2, \dots, n-1$$

边界条件:

① 第一型插值条件

$$S'(x_1) = y'_1, \quad S'(x_n) = y'_n$$

由此导出

$$2M_1 + M_2 = d_1 \\ M_{n-1} + 2M_n = d_n \\ d_1 = \frac{6}{h_1} \left(\frac{y_2 - y_1}{h_1} - y'_1 \right) \\ d_n = \frac{6}{h_{n-1}} \left(y'_n - \frac{y_n - y_{n-1}}{h_{n-1}} \right).$$

② 自然样条

$$S''(x_1) = M_1 = 0 = M_n = S''(x_n)$$

由此得

$$M_1 = 0, \quad M_n = 0$$

从而 M_j 满足三对角方程.

3. 使用说明

(1) SPLINE (X,Y,N,YP1,YPN,Y2)

N 整型变量, 输入参数, 节点个数

X N 个元素的一维实型数组, 输入参数, 存放样条节点: $x_1 < x_2 < \dots < x_N$

Y N 个元素的一维实型数组, 输入参数, 存放节点的函数值

YP1, YPN 实型变量, 输入参数, 分别输入插值函数在 x_1 和 x_n 点的一阶导数值, 当这两个量或其中之一大于等于 10^{30} 时, 子过程自动用自然样条的条件

Y2 N 个元素的一维实型数组, 输出参数, 输出插值函数在节点的二阶导数值

(2) SPLINT (XA, YA, Y2A, N, X, Y)

N 整型变量, 输入参数, 节点个数

XA N 个元素的一维实型数组, 输入参数, 存放样条节点: $x_1 < x_2 < \dots < x_N$

YA N 个元素的一维实型数组, 输入参数, 存放样条节点上的函数值

Y2A N 个元素的一维实型数组, 输入参数, 存放由子过程 SPLINE 输出的二阶导数值

X 实型变量, 输入参数, 插值点

Y 实型变量, 输出参数, 插值结果

4. 过程

(1) 子过程 SPLINE.

```
SUBROUTINE spline(x,y,n,yp1,ypn,y2)
INTEGER n,NMAX
REAL yp1,ypn,x(n),y(n),y2(n)
PARAMETER (NMAX=500)
INTEGER i,k
REAL p,qn,sig,un,u(NMAX)
if (yp1>.99e30) then
    y2(1)=0.
    u(1)=0.
else
    y2(1)=-0.5
    u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
endif
do i=2,n-1
    sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
    p=sig*y2(i-1)+2.
    y2(i)=(sig-1.)/p
    u(i)=(6.*((y(i+1)-y(i))/(x(i+1)-x(i))-(y(i)-y(i-1))/8.
```

```

      (x(i)-x(i-1)))/(x(i-1)-x(i-1))-sig*u(i-1))/p
end do
if (ypn>.99e30) then
  qn=0.
  un=0.
else
  qn=0.5
  un=(3./(x(n)-x(n-1)))*(ypn-(y(n)-y(n-1))/(x(n)-x(n-1)))
endif
y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)
do k=n-1,1,-1
  y2(k)=y2(k)*y2(k+1)+u(k)
end do
END SUBROUTINE spline

```

(2) 子过程 SPLINT.

```

SUBROUTINE splint(xa,ya,y2a,n,x,y)
INTEGER n
REAL aa,x,y,xa(n),y2a(n),ya(n)
INTEGER k,khi,klo
REAL a,b,h
klo=1
khi=n
do
  aa=0
  if (khi-klo>1) then
    aa=2.
    k=(khi+klo)/2
    if(xa(k)>x) then
      khi=k
    else
      klo=k
    endif
    if(.not.aa>1.) exit
  endif
  if(.not.aa>1.) exit
end do

```



```

h=xa(khi)-xa(klo)
if (h==0.) pause 'bad xa input in splint'
a=(xa(khi)-x)/h
b=(x-xa(klo))/h
y=a*ya(klo)+b*ya(khi)+((a**3-a)*y2a(klo)+(b**3-b)*&
    y2a(khi))*(h**2)/6.
END SUBROUTINE splint

```

5. 例子

(1) 为了验证子过程 SPLINE, 我们取 $\sin(x)$ 为计算实例. 所要计算的自变量 x 的点存放在数组 X 中, 区间两端点的一阶导数值为 $YP1 = \cos x_1$, $YPN = \cos x_N$. 输出的二阶导数值存放在 $Y2$ 中. 精确解为 $-\sin x_i$. 验证程序 D2R3 如下:

```

PROGRAM D2R3
! Driver for routine SPLINE
PARAMETER(N=20,PI=3.141593)
DIMENSION X(N),Y(N),Y2(N)
WRITE(*,*) 'Second--derivative for sin(x) from 0 to PI'
! Generate array for interpolation
DO I=1,20
    X(I)=I*PI/N
    Y(I)=SIN(X(I))
END DO
! Calculate 2nd derivative with SPLINE
YP1=COS(X(1))
YPN=COS(X(N))
CALL SPLINE(X,Y,N,YP1,YPN,Y2)
! Test result
WRITE(*, '(T19,A,T35,A)') 'spline','actual'
WRITE(*, '(T6,A,T17,A,T33,A)') 'angle','2nd deriv',&
    '2nd deriv'
DO I=1,N
    WRITE(*, '(IX,F8.2,2F16.6)') X(I),Y2(I),-SIN(X(I))
END DO
END

```

计算结果如下:

Second-derivative for $\sin(x)$ from 0 to π

angle	spline	actual
	2nd deriv	2nd deriv
0.16	-0.156833	-0.156434
0.31	-0.309631	-0.309017
0.47	-0.454933	-0.453991
0.63	-0.588992	-0.587785
0.79	-0.708561	-0.707107
0.94	-0.810682	-0.809017
1.10	-0.892845	-0.891007
1.26	-0.953007	-0.951057
1.41	-0.989725	-0.987688
1.57	-1.002057	-1.000000
1.73	-0.989719	-0.987688
1.88	-0.953017	-0.951056
2.04	-0.892838	-0.891006
2.20	-0.810684	-0.809017
2.36	-0.708558	-0.707107
2.51	-0.588998	-0.587785
2.67	-0.454922	-0.453990
2.83	-0.309657	-0.309017
2.98	-0.156738	-0.156434
3.14	-0.000073	0.000000

(2) 验证子过程 SPLINT 的主程序 D2R4 所采用的实例是 $\sin x$ 和 e^x , 子过程 SPLINT 中所需要的二阶导数是调用子过程 SPLINE 所得到的. 函数在插值点 x_i 上的值存放在数组 Y2 中, 并与精确解进行了比较. 验证程序 D2R4 如下:

```

PROGRAM D2R4
! Driver for routine SPLINT, which calls SPLINE
PARAMETER(NP=10,PI=3.141593)
DIMENSION XA(NP),YA(NP),Y2(NP)
DO NFUNC=1,2
  IF(NFUNC==1) THEN
    WRITE(*,*) 'Sine function from 0 to PI'
    DO I=1,NP
      XA(I)=I*PI/NP
      YA(I)=SIN(XA(I))
    
```

```

END DO
YP1=COS(XA(1))
YPN=COS(XA(NP))
ELSE IF(NFUNC==2) THEN
WRITE(*,*) 'Exponential function from 0 to 1'
DO I=1,NP
XA(I)=1.0*I/NP
YA(I)=EXP(XA(I))
END DO
YP1=EXP(XA(1))
YPN=EXP(XA(NP))
ELSE
STOP
ENDIF
! Call SPLINE to get second derivatives
CALL SPLINE(XA,YA,NP,YP1,YPN,Y2)
! Call SPLINT for interpolations
WRITE(*, '(1X,T10,A1,T20,A4,T28,A13)') 'x','f(x)',&
'interpolation'
DO I=1,10
IF(NFUNC==1) THEN
X=(-0.05+I/10.0)*PI
F=SIN(X)
ELSE IF(NFUNC==2) THEN
X=(-0.05+I/10.0)
F=EXP(X)
ENDIF
CALL SPLINT(XA,YA,Y2,NP,X,Y)
WRITE(*, '(1X,3F12.6)') X,F,Y
END DO
WRITE(*,*) '*****'
*****
WRITE(*,*) 'Press RETURN'
READ(*,*)
END DO
END

```

计算结果如下:

Sine function from 0 to PI

x	f(x)	interpolation
0.157080	0.156434	0.156351
0.471239	0.453991	0.453981
0.785398	0.707107	0.707088
1.099558	0.891007	0.890984
1.413717	0.987688	0.987663
1.727876	0.987688	0.987663
2.042035	0.891006	0.890983
2.356195	0.707107	0.707088
2.670354	0.453990	0.453978
2.984513	0.156434	0.156433

Press RETURN

Exponential function from 0 to 1

x	f(x)	interpolation
0.050000	1.051271	1.051268
0.150000	1.161834	1.161834
0.250000	1.284025	1.284025
0.350000	1.419068	1.419067
0.450000	1.568312	1.568312
0.550000	1.733253	1.733253
0.650000	1.915541	1.915540
0.750000	2.117000	2.116999
0.850000	2.339647	2.339646
0.950000	2.585710	2.585709

Press RETURN

2.4 有序表的检索法

1. 功能

给定一单调数组 $x_i (i=1, \dots, n)$ 和一个数 x , 寻找一整数 $j: 1 \leq j \leq n$, 使 x 在 x_j 和 x_{j+1} 之间. 这种检索法可以应用于相关的一些内插方法中.

子过程 LOCATE 是二分法检索,子过程 HUNT 用于具关联值的检索.

2. 方法

(1) 二分法.

设数组为单调增加的,则算法为:

① $l=0, u=n+1$.

② 当 $u-l=1$ 时, $j=l$, 算法结束, 否则反复执行下列语句:

(a) $m=(u+l)/2$.

(b) 若 $x > x_m$, 则 $l=m$, 否则 $u=m$.

(2) 如果数组中有某些是相互关联的, 即有一些值是非常接近的, 我们采用如下算法:

① 首先给一个猜测值.

② 然后依次以增量 1, 2, 4, 8, ..., 搜寻出一个包含要求的区间.

③ 再用二分法检索该区间.

3. 使用说明

(1) LOCATE (XX, N, X, J)

N 整型变量, 输入参数, 数组元素的个数

XX N 个元素的一维实型数组, 输入参数, 存放给定的数组 (x_1, \dots, x_n)

X 实型变量, 输入参数, 输入给定值 x

J 整型变量, 输出参数, 所求值, 如果 $J=0$ 或 $J=N$, 则说明输入值 x 不在 x_1 与 x_n 之间

(2) HUNT (XX, N, X, JLO)

N, XX, X 同以上子过程 LOCATE

JLO 整型变量, 输入、输出参数, 输入时输入初始的猜测, 输出时存放结果, 若 $JLO=0$ 或 $JLO=N$, 则表示输入值 x 超出范围

4. 过程

(1) 子过程 LOCATE.

```
SUBROUTINE locate(xx,n,x,j)
```

```
INTEGER j,n
```

```
REAL aaa,x,xx(n)
```

```
INTEGER jl,jm,ju
```

```
jl=0
```

```

ju=n+1
do
  aaa=0.
  if(ju-jl>1) then
    aaa=2.
    jm=(ju+jl)/2
    if((xx(n)>=xx(1)).eqv.(x>=xx(jm))) then
      jl=jm
    else
      ju=jm
    endif
    if(.not.aaa>1.) exit
  endif
  if(.not.aaa>1.) exit
end do
if(x==xx(1)) then
  j=1
else if(x==xx(n)) then
  j=n-1
else
  j=jl
endif
END SUBROUTINE locate

```

(2) 子过程 HUNT.

```

SUBROUTINE hunt(xx,n,x,jlo)
INTEGER jlo,n
REAL x,xx(n)
INTEGER inc,jhi,jm
LOGICAL ascnd
ascnd=xx(n).ge.xx(1)
if(jlo<=0.or.jlo>n)then
  jlo=0
  jhi=n+1
do
  aaa=0.
  if(jhi-jlo==1) then

```

```

        if(x==xx(n)) jlo=n-1
        if(x==xx(1)) jlo=1
        return
    endif
    aaa=2.
    jm=(jhi+jlo)/2
    if(x>=xx(jm).eqv.ascnd) then
        jlo=jm
    else
        jhi=jm
    endif
    if(.not.aaa>1.) exit
end do
return
endif
inc=1
if(x>=xx(jlo).eqv.ascnd)then
    do
        aaa=0.
        jhi=jlo+inc
        if(jhi>n)then
            jhi=n+1
        else if(x>=xx(jhi).eqv.ascnd)then
            aaa=2.
            jlo=jhi
            inc=inc+inc
            if(.not.aaa>1.) exit
        endif
        if(.not.aaa>1.) exit
    end do
else
    jhi=jlo
    do
        aaa=0.
        jlo=jhi-inc
        if(jlo<1)then
            jlo=0

```

```

      else if (x < xx(jlo), eqv, ascnd) then
        aaa = 2.
        jhi = jlo
        inc = inc + inc
        if (.not. aaa > 1.) exit
      endif
      if (.not. aaa > 1.) exit
    end do
  endif
do
  aaa = 0.
  if (jhi - jlo == 1) then
    if (x == xx(n)) jlo = n - 1
    if (x == xx(1)) jlo = 1
    return
  endif
  aaa = 2.
  jm = (jhi + jlo) / 2
  if (x >= xx(jm), eqv, ascnd) then
    jlo = jm
  else
    jhi = jm
  endif
  if (.not. aaa > 1.) exit
end do
END SUBROUTINE hunt

```

5. 例子

(1) 为了验证 LOCATE, 我们取数组 x_i 为非均匀的, 它随 i 指数变化. 取 X 为一均匀变化的数列, 利用子过程 LOCATE 寻找 X 的位置. 对每一个 X , LOCATE 将找到一个 J , 此时 X 在 $XX(J)$ 和 $XX(J+1)$ 之间. 如果 J 是 0 或是 N (实例中 $N=100$), 表示 X 不在有序表 X_i 的范围之内. 若 X 小于 $X(1)$, 则打印出 “lower lim”; 若 X 大于 $X(N)$, 则打印出 “upper lim”. 验证程序如下:

```

PROGRAM D2R5
! Driver for routine LOCATE
PARAMETER(N=100)

```



```

DIMENSION XX(N)
! Create array to be searched
DO I=1,N
    XX(I)=EXP(I/20.0)-74.0
END DO
WRITE(*,*) 'Result of, j=0 indicates x too small'
WRITE(*,*) 'j=100 indicates x too large'
WRITE(*, '(T5,A7,T17,A1,T24,A5,T34,A7)') 'locate ', 'j', &
    'xx(j)', 'xx(j+1)'
! Do test
DO I=1,19
    X=-100.0+200.0*I/20.0
    CALL LOCATE(XX,N,X,J)
    IF(J==0) THEN
        WRITE(*, '(1X,F10.4,I6,A12,F12.6)') &
            X,J,'lower lim',XX(J+1)
    ELSE IF(J==N) THEN
        WRITE(*, '(1X,F10.4,I6,F12.6,A12)') &
            X,J,XX(J),'upper lim'
    ELSE
        WRITE(*, '(1X,F10.4,I6,2F12.6)') &
            X,J,XX(J),XX(J+1)
    ENDIF
END DO
END

```

计算结果如下:

Result of: j=0 indicates x too small
 j=100 indicates x too large

locate	j	xx(j)	xx(j+1)
-90.0000	0	lower lim	-72.948730
-80.0000	0	lower lim	-72.948730
-70.0000	27	-70.142578	-69.944801
-60.0000	52	-60.536263	-59.845963
-50.0000	63	-50.663937	-49.467468

-40.0000	70	-40.884548	-39.186684
-30.0000	75	-31.478918	-29.298815
-20.0000	79	-22.064632	-19.401850
-10.0000	83	-10.566000	-7.313669
0.0000	86	-0.300206	3.478463
10.0000	88	7.450869	11.626944
20.0000	90	16.017132	20.632408
30.0000	92	25.484316	30.584986
40.0000	94	35.947174	41.584286
50.0000	96	47.510418	53.740391
60.0000	97	53.740391	60.289780
70.0000	99	67.174965	74.413162
80.0000	100	74.413162	upper lim
90.0000	100	74.413162	upper lim

(2) 子过程 HUNT 和 LOCATE 功能一样, 只是 HUNT 经常用于有序表多次被检索且每次检索时其值非常接近上次检索的值. 验证程序 D2R6 中建立了数组 $XX(I)$, 且一系列的 X 点被给定位置. HUNT 返回的 J 使得 X 位于 $XX(J)$ 和 $XX(J+1)$ 之间. $J=0$ 和 $J=N$ 和 D2R5 中的意义相同. 验证程序 D2R6 如下:

```

PROGRAM D2R6
  ! Driver for routine HUNT
  PARAMETER(N=100)
  DIMENSION XX(N)
  ! Create array to be searched
  DO I=1,N
    XX(I)=EXP(I/20.0)-74.0
  END DO
  WRITE(*,*) 'Result of: j=0 indicates x too small'
  WRITE(*,*) 'j=100 indicates x too large'
  WRITE(*, '(T7,A7,T17,A5,T25,A1,T32,A5,T42,A7)') &
    'locate:', 'guess', 'j', 'xx(j)', 'xx(j+1)'
  ! Do test
  DO I=1,19

```

```

      X = -100.0 + 200.0 * I/20.0
! Trial parameter
      JI = 5 * I
      J = JI
! Begin search
      CALL HUNT(XX,N,X,J)
      IF(J == 0) THEN
          WRITE(*,'(1X,F12.6,2I6,A12,F12.6)') X,JI,J,&
              'lower lim',XX(J+1)
      ELSE IF(J.EQ.N) THEN
          WRITE(*,'(1X,F12.6,2I6,F12.6,A12)') X,JI,J,&
              XX(J),'upper lim'
      ELSE
          WRITE(*,'(1X,F12.6,2I6,2F12.6)')&
              X,JI,J,XX(J),XX(J+1)
      ENDIF
  END DO
END

```

计算结果如下:

Result of; j=0 indicates x too small
 j=100 indicates x too large

locate:	guess	j	xx(j)	xx(j+1)
-90.000000	5	0	lower lim	-72.948730
-80.000000	10	0	lower lim	-72.948730
-70.000000	15	27	-70.142578	-69.944801
-60.000000	20	52	-60.536263	-59.845963
-50.000000	25	63	-50.663937	-49.467468
-40.000000	30	70	-40.884548	-39.186684
-30.000000	35	75	-31.478918	-29.298815
-20.000000	40	79	-22.064632	-19.401850
-10.000000	45	83	-10.566000	-7.313669
0.000000	50	86	-0.300206	3.478463
10.000000	55	88	7.450869	11.626944

20.000000	60	90	16.017132	20.632408
30.000000	65	92	25.484316	30.584986
40.000000	70	94	35.947174	41.584286
50.000000	75	96	47.510418	53.740391
60.000000	80	97	53.740391	60.289780
70.000000	85	99	67.174965	74.413162
80.000000	90	100	74.413162	upper lim
90.000000	95	100	74.413162	upper lim

2.5 插值多项式

1. 功能

给定 n 个互异节点 x_i 及对应的函数值 $y_i = f(x_i)$ ($i=1, \dots, n$), 若以 x_i ($i=1, \dots, n$) 为节点的插值多项式为:

$$y = c_1 + c_2x + \dots + c_nx^{n-1}$$

求该多项式的系数 c_i ($i=1, \dots, n$).

这里给出两个子过程: POLCOE 和 POLCOF, 前者是 n^2 阶的, 后者是 n^3 阶的, 二者对较大的 n 都不很好, 一般以单精度计算时, $n \leq 10$ 时是满意的, 而以双精度计算时, $n \leq 20$ 时是满意的.

2. 方法

(1) 应用 Vandermonde 矩阵的方法.

① 求解下列线性方程组即得所求:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

② 记
$$P_j(x) = \prod_{\substack{k=1 \\ j \neq k}}^n \frac{x - x_k}{x_j - x_k} = \sum_{k=1}^n A_{jk} x^{k-1}$$

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ \vdots & \vdots & \cdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

则

$$P_j(x_i) = \delta_{ij} = \sum_{k=1}^n A_{jk} x_i^{k-1}$$

所以

$$(c_1, c_2, \dots, c_n) = (y_1, \dots, y_n)A$$

③ 设

$$P(x) = \prod_{k=1}^n (x - x_k) = \sum_{k=1}^n s_k x^{k-1} + x^n$$

$$q_j(x) = P(x)/(x - x_j) = \sum_{k=1}^n b_k x^{k-1}$$

$$Ph_j = q_j(x_j)$$

则

$$P_j(x) - \frac{q_j(x)}{Ph_j} = \sum_{k=1}^n b_k \frac{1}{Ph_j} x^{k-1}$$

故

$$c_j = \sum_{k=1}^n A_{kj} y_k = \sum_{k=1}^n b_k \frac{y_k}{Ph_j}$$

④ $Ph_j = q_j(x_j) = P'(x_j) = \sum_{k=2}^n (k-1)s_k x_j^{k-2} + n x_j^{n-1}$. 由综合除法 $b_n = 1$,
 $b_k = b_{k+1}x_j + s_{k+1}$, $k = n-1, \dots, 1$.

(2) 应用多项式插值的方法.

① 以 $x_i (i=1, \dots, n)$ 为节点作多项式插值, 其 Lagrange 多项式记为 $L_n(x)$,
 则 $c_1 = L_n(0)$.

② 记

$$\varphi_0(x, y) = y$$

$$\varphi_k(x, y) = (\varphi_{k-1} - c_k)/x, \quad k = 1, \dots, n-1$$

则

$$\varphi_k(x, y) = c_{k+1} + c_{k+2}x + \dots + c_n x^{n-k-1}$$

$$\varphi_k(x, y_i) = \sum_{j=k+1}^n c_j x_i^{j-k-1}, \quad k = 1, \dots, n-1$$

以 $x_i (i=k+1, \dots, n)$ 为节点作多项式插值, 其 Lagrange 多项式记为 $L_{n-k}(x)$, 则
 $c_{k+1} = L_{n-k}(0)$.

3. 使用说明

(1) POLCOE (X,Y,N,COF)

N 整型变量,输入参数,插值节点数
 X N 个元素的一维实型数组,输入参数,存放插值节点
 Y N 个元素的一维实型数组,输入参数,存放节点的函数值
 COF N 个元素的一维实型数组,输出参数,输出所求多项式的系数

(2) POLCOF (XA,YA,N,COF)

N 整型变量,输入参数,插值节点数
 XA N 个元素的一维实型数组,输入参数,存放插值节点
 YA N 个元素的一维实型数组,输入参数,存放节点的函数值
 COF N 个元素的一维实型数组,输出参数,输出所求多项式的系数

4. 过程

(1) 子过程 POLCOE.

```
SUBROUTINE polcoe(x,y,n,cof)
  INTEGER n,NMAX
  REAL cof(n),x(n),y(n)
  PARAMETER (NMAX=15)
  INTEGER i,j,k
  REAL b,ff,phi,s(NMAX)
  do i=1,n
    s(i)=0.
    cof(i)=0.
  end do
  s(n)=-x(1)
  do i=2,n
    do j=n+1-i,n-1
      s(j)=s(j)-x(i)*s(j+1)
    end do
    s(n)=s(n)-x(i)
  end do
  do j=1,n
    phi=n
    do k=n-1,1,-1
```

```

    phi=k * s(k+1)+x(j) * phi
end do
ff=y(j)/phi
b=1.
do k=n,1,-1
    cof(k)=cof(k)+b * ff
    b=s(k)+x(j) * b
end do
end do
END SUBROUTINE polcoe

```

(2) 子过程 POLCOF.

```

SUBROUTINE polcof(xa,ya,n,cof)
INTEGER n,NMAX
REAL cof(n),xa(n),ya(n)
PARAMETER (NMAX=15)
! USES polint
INTEGER i,j,k
REAL dy,xmin,x(NMAX),y(NMAX)
do j=1,n
    x(j)=xa(j)
    y(j)=ya(j)
end do
do j=1,n
    call polint(x,y,n+1-j,0.,cof(j),dy)
    xmin=1.e38
    k=0
    do i=1,n+1-j
        if (abs(x(i))<xmin) then
            xmin=abs(x(i))
            k=i
        endif
        if(x(i)/=0.) y(i)=(y(i)-cof(j))/x(i)
    end do
    do i=k+1,n+1-j
        y(i-1)=y(i)
        x(i-1)=x(i)
    end do
end do

```

```

    end do
  end do
END SUBROUTINE polcof

```

5. 例子

验证子过程 POLCOE 的主程序为 D2R7, 验证子过程 POLCOF 的主程序为 D2R8. 两者非常相似. 我们所取的例子是正弦函数 $\sin x$ 和指数函数 e^x . 给定的均匀划分的节点值存放在 $XA(I)$ 中, 其函数值存放在 $YA(I)$ 中. 通过子过程求得的系数存放在数组 COEFF. 利用多项式算得的试验点上的函数值存放在 SUM 中, 并与精确解 F 进行了比较.

(1) 验证子过程 POLCOE 的主程序 D2R7 如下:

```

PROGRAM D2R7
! Driver for routine POLCOE
PARAMETER(NP=5,PI=3.141593)
DIMENSION XA(NP),YA(NP),COEFF(NP)
DO NFUNC=1,2
  IF(NFUNC==1) THEN
    WRITE(*,*) 'Sine function from 0 to PI'
    DO I=1,NP
      XA(I)=I*PI/NP
      YA(I)=SIN(XA(I))
    END DO
  ELSE IF(NFUNC==2) THEN
    WRITE(*,*) 'Exponential function from 0 to 1'
    DO I=1,NP
      XA(I)=1.0*I/NP
      YA(I)=EXP(XA(I))
    END DO
  ELSE
    STOP
  ENDIF
  CALL POLCOE(XA,YA,NP,COEFF)
  WRITE(*,*) 'coefficients'
  WRITE(*,'(1X,6F12.6)') (COEFF(I),I=1,NP)
  WRITE(*,'(1X,T10,A1,T20,A4,T29,A10)') 'x','f(x)',&
    'polynomial'

```



```

DO I=1,10
  IF(NFUNC==1) THEN
    X=(-0.05+I/10.0)*PI
    F=SIN(X)
  ELSE IF(NFUNC==2) THEN
    X=(-0.05+I/10.0)
    F=EXP(X)
  ENDIF
  SUM=COEFF(NP)
  DO J=NP-1,1,-1
    SUM=COEFF(J)+SUM*X
  END DO
  WRITE(*, '(1X,3F12.6)') X,F,SUM
END DO
WRITE(*,*) '*****'
WRITE(*,*) 'Press RETURN'
READ(*,*)
END DO
END

```

计算结果如下:

Sine function from 0 to PI

coefficients

-0.000231 0.986063 0.051750 -0.232789 0.037056

x f(x) polynomial

0.157080 0.156434 0.155057

0.471239 0.453991 0.453399

0.785398 0.707107 0.707464

1.099558 0.891007 0.891268

1.413717 0.987688 0.987495

1.727876 0.987688 0.987489

2.042035 0.891006 0.891258

2.356195 0.707107 0.707473

2.670354 0.453990 0.453469

2.984513 0.156434 0.155242

Press RETURN

Exponential function from 0 to 1

coefficients

1.001029 0.989333 0.538610 0.105194 0.084129

x f(x) polynomial

0.050000 1.051271 1.051856

0.150000 1.161834 1.161945

0.250000 1.284025 1.283998

0.350000 1.419068 1.419048

0.450000 1.568312 1.568333

0.550000 1.733253 1.733292

0.650000 1.915541 1.915564

0.750000 2.117000 2.116994

0.850000 2.339647 2.339626

0.950000 2.585710 2.585705

Press RETURN

(2) 验证子过程 POLCOF 的主程序 D2R8 如下:

```
PROGRAM D2R8
```

```
! Driver for routine POLCOF
```

```
PARAMETER(NP=5,PI=3.141593)
```

```
DIMENSION XA(NP),YA(NP),COEFF(NP)
```

```
DO NFUNC=1,2
```

```
  IF(NFUNC==1) THEN
```

```
    WRITE(*,*) 'Sine function from 0 to PI'
```

```
    DO I=1,NP
```

```
      XA(I)=I*PI/NP
```

```
      YA(I)=SIN(XA(I))
```

```
    END DO
```

```
  ELSE IF(NFUNC==2) THEN
```

```
    WRITE(*,*) 'Exponential function from 0 to 1'
```

```
    DO I=1,NP
```

```
      XA(I)=1.0*I/NP
```

```

        YA(I)=EXP(XA(I))
    END DO
ELSE
    STOP
ENDIF
CALL POLCOF(XA,YA,NP,COEFF)
WRITE(*,*) '    coefficients'
WRITE(*,'(1X,6F12.6)') (COEFF(I),I=1,NP)
WRITE(*,'(1X,T10,A1,T20,A4,T29,A10)') 'x','f(x)',&
    'polynomial'
DO I=1,10
    IF(NFUNC==1) THEN
        X=(-0.05+I/10.0)*PI
        F=SIN(X)
    ELSE IF(NFUNC==2) THEN
        X=(-0.05+I/10.0)
        F=EXP(X)
    ENDIF
    SUM=COEFF(NP)
    DO J=NP-1,1,-1
        SUM=COEFF(J)+SUM*X
    END DO
    WRITE(*,'(1X,3F12.6)') X,F,SUM
END DO
WRITE(*,*) '*****'
WRITE(*,*) 'Press RETURN'
READ(*,*)
END DO
END

```

计算结果如下:

Sine function from 0 to PI

coefficients

0.000000	0.985328	0.052482	-0.233080	0.037096
----------	----------	----------	-----------	----------

x	f(x)	polynomial
---	------	------------

0.157080	0.156434	0.155189
----------	----------	----------

0.471239	0.453991	0.453418
0.785398	0.707107	0.707443
1.099558	0.891007	0.891246
1.413717	0.987688	0.987484
1.727876	0.987688	0.987485
2.042035	0.891006	0.891246
2.356195	0.707107	0.707443
2.670354	0.453990	0.453418
2.984513	0.156434	0.155189

Press RETURN

Exponential function from 0 to 1

coefficients

1.000532	0.994006	0.524018	0.123288	0.076437
----------	----------	----------	----------	----------

x	f(x)	polynomial
---	------	------------

0.050000	1.051271	1.051558
0.150000	1.161834	1.161878
0.250000	1.284025	1.284010
0.350000	1.419068	1.419060
0.450000	1.568312	1.568318
0.550000	1.733253	1.733258
0.650000	1.915541	1.915536
0.750000	2.117000	2.116994
0.850000	2.339647	2.339656
0.950000	2.585710	2.585727

Press RETURN

2.6 二元拉格朗日插值

1. 功能

设已知插值节点 $(x_i, y_i) (i=1, \dots, m; j=1, \dots, n)$ 及对应函数值 $z_{ij}=f(x_i, y_i) (i=1, \dots, m; j=1, \dots, n)$, 用拉格朗日插值法求函数在给定点 (x, y) 处的对

应函数值 z .

2. 方法

(1) 对每个 $i(i=1, \dots, m)$, 以 $y_j(j=1, \dots, n)$ 为插值节点, $z_{ij}(j=1, \dots, n)$ 为对应函数值, y 为插值变量, 作一元函数插值得 $u_i(i=1, \dots, m)$.

(2) 以 $x_i(i=1, \dots, m)$ 为插值节点, u_i 为对应函数值, x 为插值变量, 作一元函数插值得所求值 z .

3. 使用说明

POLIN2 (X1A,X2A,YA,M,N,X1,X2,Y,DY)

M 整型变量, 输入参数, x 轴上的分点个数

N 整型变量, 输入参数, y 轴上的分点个数

X1A M 个元素的一维实型数组, 输入参数, 存放 x 轴上的 M 个分点

X2A N 个元素的一维实型数组, 输入参数, 存放 y 轴上的 N 个分点

X1 实型变量, 输入参数, 插值变量的第一个分量

X2 实型变量, 输入参数, 插值变量的第二个分量

YA $M \times N$ 个元素的二维实型数组, 输入参数, 函数在插值节点上的函数值

Y 实型变量, 输出参数, 插值结果

DY 实型变量, 输出参数, 精度估计

4. 过程

子过程 POLIN2.

```
SUBROUTINE polin2(x1a,x2a,ya,m,n,x1,x2,y,dy)
```

```
INTEGER m,n,NMAX,MMAX
```

```
REAL dy,x1,x2,y,x1a(m),x2a(n),ya(m,n)
```

```
PARAMETER (NMAX=20,MMAX=20)
```

```
! USES polint
```

```
INTEGER j,k
```

```
REAL ymtmp(MMAX),yntmp(NMAX)
```

```
do j=1,m
```

```
  do k=1,n
```

```
    yntmp(k)=ya(j,k)
```

```
  end do
```

```
  call polint(x2a,yntmp,n,x2,ymtmp(j),dy)
```

```

end do
call polint(x1a,yintmp,m,x1,y,dy)
END SUBROUTINE polin2

```

5. 例子

我们所取的例子为 $f(x,y)=e^y\sin x$. 验证程序 D2R9 如下:

```

PROGRAM D2R9
! Driver for routine POLIN2
PARAMETER(N=5,PI=3.141593)
DIMENSION X1A(N),X2A(N),YA(N,N)
DO I=1,N
  X1A(I)=I * PI/N
  DO J=1,N
    X2A(J)=1.0 * J/N
    YA(I,J)=SIN(X1A(I)) * EXP(X2A(J))
  END DO
END DO
! Test 2—dimensional interpolation
WRITE(*, '(T9,A,T21,A,T32,A,T40,A,T58,A)') 'x1','x2', &
  'f(x)', 'interpolated', 'error'
DO I=1,4
  X1=(-0.1+I/5.0) * PI
  DO J=1,4
    X2=-0.1+J/5.0
    F=SIN(X1) * EXP(X2)
    CALL POLIN2(X1A,X2A,YA,N,N,X1,X2,Y,DY)
    WRITE(*, '(1X,4F12.6,F14.6)') X1,X2,F,Y,DY
  END DO
  WRITE(*,*) '*****'
END DO
END

```

计算结果如下:

x1	x2	f(x)	interpolated	error
0.314159	0.100000	0.341517	0.340283	0.041937

0.314159	0.300000	0.417129	0.415570	0.051215
0.314159	0.500000	0.509483	0.507585	0.062555
0.314159	0.700000	0.622284	0.619961	0.076404

0.942478	0.100000	0.894102	0.894644	-0.005991
0.942478	0.300000	1.092059	1.092576	-0.007316
0.942478	0.500000	1.333844	1.334496	-0.008936
0.942478	0.700000	1.629160	1.629945	-0.010915

1.570796	0.100000	1.105171	1.104992	0.003595
1.570796	0.300000	1.349859	1.349462	0.004390
1.570796	0.500000	1.648721	1.648263	0.005362
1.570796	0.700000	2.013753	2.013177	0.006549

2.199115	0.100000	0.894102	0.894644	-0.005991
2.199115	0.300000	1.092059	1.092576	-0.007316
2.199115	0.500000	1.333843	1.334496	-0.008936
2.199115	0.700000	1.629160	1.629945	-0.010915

2.7 双三次样条插值

1. 功能

给定插值节点 (x_i, y_i) 及对应的函数值 $z_{ij} = f(x_i, y_i) (i=1, \dots, m; j=1, \dots, n)$. 用双三次样条函数求函数在给定点 (x, y) 处的值.

这里有两个子过程: SPLIE2 和 SPLIN2, 前者计算函数在节点处关于 y 的二阶偏导数 $f_{yy}(x_i, y_i) (i=1, \dots, m; j=1, \dots, n)$, 后者用于计算插值结果.

2. 方法

(1) 对每个 $i (i=1, \dots, m)$, 以 $y_j (j=1, \dots, n)$ 为插值节点, $z_{ij} (j=1, \dots, n)$ 为对应函数值, y 为插值变量, 作一元三次自然样条插值得 $s_i (i=1, \dots, m)$.

(2) 以 $x_i (i=1, \dots, m)$ 为节点, s_i 为对应函数值, x 为插值变元, 作一元三次

自然样条插值则得所求结果.

3. 使用说明

(1) SPLIE2 (X1A,X2A,YA,M,N,Y2A)

M 整型变量,输入参数, x 方向上的节点个数

N 整型变量,输入参数, y 方向上的节点个数

X1A M 个元素的一维实型数组,输入参数,存放 x 方向上的节点 $x_1 < x_2 < \cdots < x_m$

X2A N 个元素的一维实型数组,输入参数,存放 y 方向上的节点 $y_1 < y_2 < \cdots < y_n$

YA $M \times N$ 个元素的二维实型数组,输入参数,存放函数值,其元素为 $(YA)_{ij} = z_{ij}$,存放顺序为: $z_{11}, z_{21}, \cdots, z_{m1}, z_{12}, \cdots, z_{m2}, \cdots, z_{1n}, \cdots, z_{mn}$

Y2A $M \times N$ 个元素的二维实型数组,输出参数,输出插值函数在插值节点处关于 y 的二阶偏导数,其存放顺序同 YA

(2) SPLIN2 (X1A,X2A,YA,Y2A,M,N,X1,X2,Y)

M,N,X1A,X2A,YA 均同子过程 SPLIE2

Y2A $M \times N$ 个元素的二维实型数组,输入参数,输入子过程 SPLIE2 的输出结果

X1 实型变量,输入参数,插值变量在 x 方向上的分量

X2 实型变量,输入参数,插值变量在 y 方向上的分量

Y 实型变量,输出参数,函数插值结果

4. 过程

(1) 子过程 SPLIE2.

```
SUBROUTINE splie2(x1a,x2a,ya,m,n,y2a)
```

```
INTEGER m,n,NN
```

```
REAL x1a(m),x2a(n),y2a(m,n),ya(m,n)
```

```
PARAMETER (NN=100)
```

```
! USES spline
```

```
INTEGER j,k
```

```
REAL y2tmp(NN),ytmp(NN)
```

```
do j=1,m
```

```
do k=1,n
```

```
    ytmp(k) = ya(j,k)
```



```

    end do
    call spline(x2a,ytmp,n,1.e30,1.e30,y2tmp)
    do k=1,n
        y2a(j,k)=y2tmp(k)
    end do
end do
END SUBROUTINE splie2

```

(2) 子过程 SPLIN2.

```

SUBROUTINE splin2(x1a,x2a,ya,y2a,m,n,x1,x2,y)
INTEGER m,n,NN
REAL x1,x2,y,x1a(m),x2a(n),y2a(m,n),ya(m,n)
PARAMETER (NN=100)
! USES spline,splint
INTEGER j,k
REAL y2tmp(NN),ytmp(NN),yytmp(NN)
do j=1,m
    do k=1,n
        ytmp(k)=ya(j,k)
        y2tmp(k)=y2a(j,k)
    end do
    call splint(x2a,ytmp,y2tmp,n,x2,yytmp(j))
end do
call spline(x1a,yytmp,m,1.e30,1.e30,y2tmp)
call splint(x1a,yytmp,y2tmp,m,x1,y)
END SUBROUTINE splin2

```

5. 例子

(1) 为了验证子过程 SPLIE2, 我们所举的例子为 $y = (x_1 x_2)^2$. 给定的插值节点为 10×10 均匀网格. 所计算的结果与这些点上关于 x_2 的精确的二阶导数值 $2x_1^2$ 进行了比较. 由于采用了自然样条, 因此计算结果与精确解吻合不够理想. 这也证明了如果有较好的边界导数条件, 就不要采用自然样条. 也就是说, 在子过程 SPLIE2 中, 在调用子过程 SPLINE 前, 若能给出了较好的一阶导数边界条件, 那么, 结果将大大改善. 验证程序 D2R10 如下:

```
PROGRAM D2R10
```

```

1 Driver for routine SPLIE2
PARAMETER(M=10,N=10)
DIMENSION X1(M),X2(N),Y(M,N),Y2(M,N)
DO I=1,M
    X1(I)=0.2 * I
END DO
DO I=1,N
    X2(I)=0.2 * I
END DO
DO I=1,M
    DO J=1,N
        X1X2=X1(I) * X2(J)
        Y(I,J)=X1X2 * * 2
    END DO
END DO
CALL SPLIE2(X1,X2,Y,M,N,Y2)
WRITE(*, '(1X,A)') 'Second derivatives from SPLIE2'
WRITE(*, '(1X,A/)') 'Natural spline assumed'
DO I=1,5
    WRITE(*, '(1X,5F12.6)') (Y2(I,J),J=1,5)
END DO
WRITE(*, '(1X,A/)') 'Actual second derivatives'
DO I=1,5
    DO J=1,5
        Y2(I,J)=2.0 * (X1(I) * * 2)
    END DO
    WRITE(*, '(1X,5F12.6)') (Y2(I,J),J=1,5)
END DO
END

```

计算结果如下：

Second derivatives from SPLIE2

Natural spline assumed

0.000000	0.101434	0.074264	0.081509	0.079698
0.000000	0.405735	0.297057	0.326037	0.318791
0.000000	0.912905	0.668378	0.733586	0.717278

0.000000	1.622942	1.188229	1.304150	1.275165
0.000000	2.535847	1.856607	2.037733	1.992452
Actual second derivatives				
0.080000	0.080000	0.080000	0.080000	0.080000
0.320000	0.320000	0.320000	0.320000	0.320000
0.720000	0.720000	0.720000	0.720000	0.720000
1.280000	1.280000	1.280000	1.280000	1.280000
2.000000	2.000000	2.000000	2.000000	2.000000

(2) 为了验证子过程 SPLIN2, 我们所取的例子为 $y = x_1 x_2 \exp(-x_1 x_2)$. 计算结果与精确解进行了比较. 精确解存放在数组 FF 中. 计算结果中有些点与精确解吻合不好, 是由于子过程 SPLIE2 中采用了自然样条的缘故. 注意: 在子过程 SPLIN2 中还调用了子过程 SPLINE 和 SPLINT. 验证程序 D2R11 如下:

```

PROGRAM D2R11
! Driver for routine SPLIN2
PARAMETER(M=10,N=10)
! USES splie2
DIMENSION X1(M),X2(N),Y(M,N),Y2(M,N)
DO I=1,M
  X1(I)=0.2*I
END DO
DO I=1,N
  X2(I)=0.2*I
END DO
DO I=1,M
  DO J=1,N
    X1X2=X1(I)*X2(J)
    Y(I,J)=X1X2*EXP(-X1X2)
  END DO
END DO
CALL SPLIE2(X1,X2,Y,M,N,Y2)
WRITE(*, '(1X,T9,A,T21,A,T31,A,T43,A)') &
  'X1','X2','SPLIN2','ACTUAL'
DO I=1,10
  XX1=0.1*I

```

```

      XX2=XX1 * * 2
      CALL SPLIN2(X1,X2,Y,Y2,M,N,XX1,XX2,F)
      X1X2=XX1 * XX2
      FF=X1X2 * EXP(-X1X2)
      WRITE(*, '(1X,4F12.6)') XX1,XX2,F,FF
END DO
END

```

计算结果如下：

X1	X2	SPLIN2	ACTUAL
0.100000	0.010000	0.000280	0.000999
0.200000	0.040000	0.009923	0.007936
0.300000	0.090000	0.028606	0.026281
0.400000	0.160000	0.060933	0.060032
0.500000	0.250000	0.109613	0.110312
0.600000	0.360000	0.173581	0.174039
0.700000	0.490000	0.243760	0.243406
0.800000	0.640000	0.306728	0.306839
0.900000	0.810000	0.351715	0.351663
1.000000	1.000000	0.367879	0.367879

第3章 数值积分

数值积分也叫做求积. 本章中的积分方法基于积分的定义,即用被积函数在积分区域上的函数值的和来计算积分.

对于性能良好的函数,利用梯形求积法(见 3.1 节)与辛普森求积法(见 3.2 节),程序简单而又十分准确,是数值积分中使用最广泛的方法.但对于仅在一些离散点上有值的函数,这两个方法只适用于等间距的情况,而且子区间的个数必须为 2 的幂.还应注意,为了达到高精度,当步数太多时,积累误差会开始增加,用 3.1 节和 3.2 节中的子过程计算时会不收敛.因此,在子过程中设置的精度 EPS,不宜任意提高.

龙贝格求积法是在梯形求积法的基础上,采用理查逊外推来加速收敛的一种求积方法.该方法效率高(函数求值的次数较少)且可以达到任意希望的精确度.因此,对于充分光滑的被积函数,我们推荐这种方法.但对于以函数表形式给出的函数,本方法不能直接使用.

高斯求积公式是通过选择适当的求积节点和求积系数,使求积公式具有最大次数的代数精度.对于相同的精度,该方法所需要的函数求值次数和计算次数最少.它的另一优点是在有些情况下,因为它不用端点上的函数值,所以能对奇异点进行有效的处理.其缺点是不能用于以函数表形式给出的函数的求积问题.

3.4 节中给出了各种反常积分的算法.对于第 4 章中的某些特殊函数,也可用它们进行计算.

求积分 $I = \int_a^b f(x)dx$ 等价于求解微分方程

$$\frac{dy}{dx} = f(x), \quad y(a) = 0$$

的值 $I=y(b)$. 由于解微分方程更强调步长选取,因此对于有强峰的被积函数的求积问题,应该用求解微分方程来计算积分(第 14 章).

对于多维积分,一般用累次积分的方法.如果积分区域的形状比较复杂,被积函数是振荡的或不连续的,但被积函数不是强峰函数时,可采用蒙特卡罗方法.如果被积函数为强峰函数,则应把积分区域划分成 n 个较小的区域,使被积函数在每个较小的区域上是光滑的,然后再分别积分.

除了本章的数值积分和常微分方程外,还有另外的积分法,一个重要的类型是基于函数逼近(参考第 5 章的切比雪夫逼近).读者也可用样条函数的子过程

SPLINE(见 2.3 节)做三次样条求积.

3.1 梯形求积法

1. 功能

用复化梯形求积公式计算定积分

$$I = \int_a^b f(x) dx$$

的近似值,使迭代法中相邻两项值的相对误差小于给定的误差限 ϵ .

子过程 TRAPZD 用于计算每一步的迭代值,而子过程 QTRAP 以误差限 $\epsilon=10^{-6}$ 和最大迭代步数 20 计算定积分的近似值.

2. 方法

假定 2^{n-1} 等分区间 $[a, b]$,则相应的复化梯形求积公式是

$$S_n = h_n \left[\frac{1}{2} f(a) + \frac{1}{2} f(b) + \sum_{i=2}^{2^{n-1}} f(x_i^{(n)}) \right]$$

其中

$$h_n = (b - a) / 2^{n-1}$$

$$x_i^{(n)} = a + (i - 1) / h_n, \quad i = 1, \dots, 2^{n-1} + 1$$

$$n = 1, 2, \dots$$

而

$$S_{n+1} = h_{n+1} \left[\frac{1}{2} f(a) + \frac{1}{2} f(b) + \sum_{i=2}^{2^n} f(x_i^{(n+1)}) \right]$$

$$x_{2^{i-1}}^{(n+1)} = x_i^{(n)}, \quad i = 1, 2, \dots, 2^{n-1} + 1$$

因此

$$S_{n+1} = \frac{1}{2} [S_n + h_n \sum_{i=2}^{2^{n-1}} f(x_{2i}^{(n+1)})]$$

3. 使用说明

(1) TRAPZD (FUNC, A, B, S, N)

N 整型变量,输入参数,迭代的阶段

A 实型变量,输入参数,积分下限

B 实型变量,输入参数,积分上限

S 实型变量,输出参数,输出第 N 步的迭代值

FUNC 函数子过程,用户自编,调用被积函数

(2) QTRAP (FUNC,A,B,S)

A 实型变量,输入参数,积分下限

B 实型变量,输入参数,积分上限

S 实型变量,输出参数,输出积分近似值

FUNC 函数子过程,用户自编,调用被积函数

4. 过程

(1) 子过程 TRAPZD.

```
SUBROUTINE trapzd(func,a,b,s,n)
INTEGER n
REAL a,b,s,func
EXTERNAL func
INTEGER it,j
REAL del,sum,tnm,x
if (n==1) then
    s=0.5*(b-a)*(func(a)+func(b))
else
    it=2*(n-2)
    tnm=it
    del=(b-a)/tnm
    x=a+0.5*del
    sum=0.
    do j=1,it
        sum=sum+func(x)
        x=x+del
    end do
    s=0.5*(s+(b-a)*sum/tnm)
endif
END SUBROUTINE trapzd
```

(2) 子过程 QTRAP.

```
SUBROUTINE qtrap(func,a,b,s)
INTEGER JMAX
```

```

REAL a,b,func,s,EPS
EXTERNAL func
PARAMETER (EPS=1.e-6, JMAX=20)
! USES trapzd
INTEGER j
REAL olds
olds=-1.e30
do j=1,JMAX
  call trapzd(func,a,b,s,j)
  if (abs(s-olds)<EPS * abs(olds)) return
  if (s==0. .and. olds==0. .and. j>6) return
  olds=s
end do
pause 'too many steps in qtrap'
END SUBROUTINE qtrap

```

5. 例子

(1) 为了验证 TRAPZD, 我们所举的例子为对下列函数进行数值积分:

$$\text{FUNC} = x^2(x^2 - 2)\sin x$$

此函数积分的精确解为:

$$\text{FINT} = 4x(x^2 - 7)\sin x - (x^4 - 14x^2 + 28)\cos x$$

积分的区间为 $A=0.0$ 到 $B=\pi/2$. 为了证明精度随着连续调用而增加, 在验证程序中 TRAPZD 被调用了 14 次. 计算结果与精确解 $\text{FINT}(B) - \text{FINT}(A)$ 相比较. 验证程序 D3R1 如下:

```

PROGRAM D3R1
! Driver for routine TRAPZD
PARAMETER (NMAX=14,PIO2=1.5707963)
EXTERNAL FUNC,FINT
A=0.0
B=PIO2
WRITE(*,'(1X,A)')&
      'Integral of FUNC with 2^(n-1) points'
WRITE(*,'(1X,A,F10.6)') 'Actual value of integral is',&
      FINT(B)-FINT(A)
WRITE(*,'(1X,T7,A,T16,A)') 'n','Approx. Integral'
DO I=1,NMAX

```



```

      CALL TRAPZD(FUNC,A,B,S,I)
      WRITE(*,'(1X,I6,F20.6)') I,S
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC
FUNCTION FINT(X)
! Integral of FUNC
  FINT=4.0*X*((X**2)-7.0)*SIN(X)&
      -((X**4)-14.0*(X**2)+28.0)*COS(X)
END FUNCTION FINT

```

计算结果如下:

Integral of FUNC with $2^{(n-1)}$ points

Actual value of integral is -0.479158

n	Approx. Integral
1	0.905772
2	-0.020945
3	-0.361461
4	-0.449584
5	-0.471756
6	-0.477308
7	-0.478696
8	-0.479044
9	-0.479130
10	-0.479149
11	-0.479153
12	-0.479154
13	-0.479153
14	-0.479153

(2) QTRAP 采用了和 TRAPZD 同样的积分方法,但能达到希望的精度(在 QTRAP 中确定为 $\text{EPS}=1.0\text{E}-6$).在 QTRAP 中,连续调用 TRAPZD 直达到达期望的精度为止.在验证程序 D3R2 中,我们将计算结果和积分的精确解进行比较.验证程序 D3R2 如下:

```

PROGRAM D3R2
! Driver for routine QTRAP
PARAMETER (PIO2=1.5707963)
EXTERNAL FUNC, FINT
A=0.0
B=PIO2
WRITE(*,'(1X,A)')&
    'Integral of FUNC computed with QTRAP'
WRITE(*,'(1X,A,F10.6)') 'Actual value of integral is',&
    FINT(B)-FINT(A)
CALL QTRAP(FUNC,A,B,S)
WRITE(*,'(1X,A,F10.6)')&
    'Result from routine QTRAP is', S
END PROGRAM
FUNCTION FUNC(X)
    FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC
FUNCTION FINT(X)
! Integral of FUNC
    FINT=4.0*X*((X**2)-7.0)*SIN(X)&
        -((X**4)-14.0*(X**2)+28.0)*COS(X)
END FUNCTION FINT

```

计算结果如下:

Integral of FUNC computed with QTRAP

Actual value of integral is -0.479158

Result from routine QTRAP is -0.479153

3.2 辛普森(Simpson)求积法

1. 功能

采用复化辛普森求积公式计算定积分

$$I = \int_a^b f(x) dx$$

的近似值,使两个相邻近似值的相对误差小于给定误差限 $\epsilon=10^{-6}$,子过程 QSIMP

允许的最大步数是 20.

2. 方法

把区间 $[a, b]$ 逐次分半, 令

$$h_n = (b - a) / 2^{n-1}$$

计算

$$T_{n+1} = \frac{1}{2} \left[T_n + h_n \sum_{i=2}^{2^{n-1}} f \left(a + \left(i - \frac{1}{2} \right) h_n \right) \right]$$

其中

$$T_1 = \frac{b-a}{2} [f(a) + f(b)], \quad n = 1, 2, 3, \dots$$

则复化辛普森求积公式为

$$S_n = \frac{1}{3} (4T_{n+1} - T_n)$$

3. 使用说明

QSIMP (FUNC, A, B, S)

A 实型变量, 输入参数, 积分下限

B 实型变量, 输入参数, 积分上限

S 实型变量, 输出参数, 输出结果

FUNC 函数子过程, 用户自编, 调用被积函数

4. 过程

子过程 QSIMP.

```
SUBROUTINE qsimp(func,a,b,s)
INTEGER JMAX
REAL a,b,func,s,EPS
EXTERNAL func
PARAMETER (EPS=1.e-6, JMAX=20)
! USES trapzd
INTEGER j
REAL os,ost,st
ost=-1.e30
os=-1.e30
do j=1,JMAX
```

```

    call trapzd(func,a,b,st,j)
    s=(4. * st-ost)/3.
    if (abs(s-os)<EPS * abs(os)) return
    if (s==0., and, os==0., and, j>6) return
    os=s
    ost=st
end do
pause 'too many steps in qsimp'
END SUBROUTINE qsimp

```

5. 例子

积分可以用 QSIMP 进行计算,子过程 QSIMP 采用了辛普森公式. 验证程序 D3R3 计算的积分和上一节的积分一样,给出结果的方式也相同. 验证程序 D3R3 如下:

```

PROGRAM D3R3
! Driver for routine QSIMP
PARAMETER (PIO2=1.5707963)
EXTERNAL FUNC,FINT
A=0.0
B=PIO2
WRITE(*, '(1X,A)') &
    'Integral of FUNC computed with QSIMP'
WRITE(*, '(1X,A,F10.6)') 'Actual value of integral is' , &
    FINT(B)-FINT(A)
CALL QSIMP(FUNC,A,B,S)
WRITE(*, '(1X,A,F10.6)') &
    'Result from routine QSIMP is' , S
END PROGRAM
FUNCTION FUNC(X)
    FUNC=(X * * 2) * (X * * 2-2.0) * SIN(X)
END FUNCTION FUNC
FUNCTION FINT(X)
! Integral of FUNC
    FINT=4.0 * X * ((X * * 2)-7.0) * SIN(X) &
        -(X * * 4)-14.0 * (X * * 2)-28.0) * COS(X)
END FUNCTION FINT

```

计算结果如下:

Integral of FUNC computed with QSIMP

Actual value of integral is -0.479158

Result from routine QSIMP is -0.479158

3.3 龙贝格(Romberg)求积法

1. 功能

用 $2k$ 阶的龙贝格方法求定积分

$$I = \int_a^b f(x) dx$$

的近似值,使两个相邻近似值的相对误差小于给定误差限 $\varepsilon = 10^{-6}$. 子过程 QROMB 设置 $k=5$,允许的最大步数是 20.

2. 方法

(1) 欧拉-马克劳林(Euler-Maclaurin)求和公式

$$\begin{aligned} I = & h \left[\frac{1}{2} f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right] \\ & - \frac{B_2 h^2}{2!} (f'(b) - f'(a)) - \cdots - \frac{B_{2k} h^{2k}}{(2k)!} (f^{(2k-1)}(b) \\ & - f^{(2k-1)}(a)) - \alpha_{2(k+1)}(h) h^{2k+2} \end{aligned}$$

其中

$$h = \frac{b-a}{n}, \quad x_i = a + ih, \quad i = 2, \cdots, n$$

$$x_1 = a, \quad \alpha_{2(k+1)}(h) = \frac{B_{2k+2}}{(2k+2)!} (b-a) f^{(2k+2)}(\xi(h))$$

$$a < \xi = \xi(h) < b$$

B_{2k} 是伯努利(Bernoulli)数,由下式定义:

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}$$

(2) 记

$$T_n(h) = h \left[\frac{1}{2} f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right]$$

则由欧拉-马克劳林求和公式有

$$T_n(h) = I + c_2 h^2 + c_4 h^4 + \cdots + c_{2k} h^{2k} + \alpha_{2(k+1)}(h) h^{2(k+1)}$$

其中 $c_{2j} = \frac{B_{2j}}{(2j)!} (f^{(2j-1)}(b) - f^{(2j-1)}(a))$. 此式右边级数当 $h=0$ 时, 即为 I .

(3) 设

$$P(h) = c_0 + c_2 h^2 + c_4 h^4 + \cdots + c_{2k} h^{2k}$$

满足

$$P(h_i) = T_i(h_i), \quad i = 1, 2, \cdots, k$$

其中

$$h_1 = b - a, \quad h_2 = \frac{h_1}{n_1}, \quad h_3 = \frac{h_1}{n_2}, \quad \cdots, \quad h_k = \frac{h_1}{n_{k-1}}$$

$n_1, n_2, \cdots, n_{k-1}$ 是严格递增的正整数, 则 $P(0)$ 为 I 的近似值.

应用多项式插值的 Neville 算法, 记 $P_{j,m+1}(h)$ 为满足

$$P_{j,m+1}(h_i) = T_i(h_i), \quad i = j, j+1, \cdots, j+m$$

的 $2m$ 次偶次插值多项式, 则

$$P_{j,m+1}(h) = \frac{(h^2 - h_j^2)P_{j+1,m}(h) - (h^2 - h_{j+m}^2)P_{j,m}(h)}{h_{j+m}^2 - h_j^2}$$

其中

$$P_{j,1}(h) = T_j(h_j)$$

$$j = 1, \cdots, k - m; \quad m = 1, \cdots, k - 1$$

令 $R_{j,m} = P_{j,m}(0)$, 则

$$R_{j,m+1} = \frac{h_{j+m}^2 R_{j,m} - h_j^2 R_{j+1,m}}{h_{j+m}^2 - h_j^2}$$

3. 使用说明

QROMB (FUNC, A, B, SS)

A 实型变量, 输入参数, 积分下限

B 实型变量, 输入参数, 积分上限

SS 实型变量, 输出参数, 输出结果

FUNC 函数子过程, 用户自编, 调用被积函数

4. 过程

子过程 QROMB.

SUBROUTINE qromb(func, a, b, ss)

INTEGER JMAX, JMAXP, K, KM

```

REAL a,b,func,ss,EPS
EXTERNAL func
PARAMETER (EPS=1.e-6, JMAX=20, JMAXP=JMAX+1, K=5, KM=K-1)
! USES polint,trapzd
INTEGER j
REAL dss,h(JMAXP),s(JMAXP)
h(1)=1.
do j=1,JMAX
  call trapzd(func,a,b,s(j),j)
  if (j>=K) then
    call polint(h(j-KM),s(j-KM),K,0.,ss,dss)
    if (abs(dss)<=EPS*abs(ss)) return
  endif
  s(j+1)=s(j)
  h(j+1)=0.25*h(j)
end do
pause 'too many steps in qromb'
END SUBROUTINE qromb

```

5. 例子

子过程 QROMB 将辛普森求积法推广到高阶的精度. 它连续地调用 TRAPZD 且将结果存储起来. 然后, 利用 POLINT(多项式插值)来得到我们想要的积分的值. 验证程序 D3R4 实质上 and QTRAP 及 QSIMP 的验证程序相同. 验证程序 D3R4 如下:

```

PROGRAM D3R4
! Driver for routine QROMB
PARAMETER (PIO2=1.5707963)
EXTERNAL FUNC,FINT
A=0.0
B=PIO2
WRITE(*, '(1X,A)')&
      'Integral of FUNC computed with QROMB'
WRITE(*, '(1X,A,F10.6)') 'Actual value of integral is',&
      FINT(B)-FINT(A)
CALL QROMB(FUNC,A,B,S)
WRITE(*, '(1X,A,F10.6)')&

```

```

                                'Result from routine QROMB is', S
END
FUNCTION FUNC(X)
    FUNC = (X * * 2) * (X * * 2 - 2.0) * SIN(X)
END FUNCTION FUNC
FUNCTION FINT(X)
    ! Integral of FUNC
    FINT = 4.0 * X * ((X * * 2) - 7.0) * SIN(X) &
        - ((X * * 4) - 14.0 * (X * * 2) + 28.0) * COS(X)
END FUNCTION FINT

```

计算结果如下:

Integral of FUNC computed with QROMB

Actual value of integral is -0.479158

Result from routine QROMB is -0.479159

3.4 反常积分

1. 功能

计算反常积分,即计算被积函数在积分区间的某一点有奇性,或者积分区间是无限的积分的近似值.

(1) 子过程 MIDPNT, MIDINF, MIDSQL, MIDSQU, MIDEXP 均用推广的中点公式计算反常积分的第 $N(N=1,2,\dots)$ 阶段的迭代值,其中:

子过程 MIDPNT 用于被积函数 $f(x)$, $x \in [a, b]$, 在区间端点无定义,但有极限的情形;

子过程 MIDINF 用于 $a > 0$, $b \rightarrow \infty$ 或 $b < 0$, $a \rightarrow -\infty$ 两种情形,且假定被积函数 $f(x)$ 在无穷远处与 $1/x^2$ 同阶, $ab > 0$;

子过程 MIDSQL 和子过程 MIDSQU 分别用于被积函数 $f(x)$ 在 $x=a$ 和在 $x=b$ 处具有平方根奇性的情形;

子过程 MIDEXP 用于 $b \rightarrow \infty$ 且被积函数在无穷远处指数衰减的情形.

(2) 子过程 QROMO 用于由上述任一子过程及龙贝格方法求反常积分的近似值,使两个相邻近似值的相对误差小于误差限 $\epsilon = 10^{-6}$,其最大迭代数为 14.

2. 方法

(1) 若把区间 $[a, b]$ n 等分, $h = \frac{b-a}{n}$, $x_1 = a$, $x_{i+1} = a + ih$, $i = 1, \dots, n$. 记 $f_{\frac{2i+1}{2}} = f\left(\frac{x_i + x_{i+1}}{2}\right)$, $i = 1, \dots, n$, 则推广的中点公式为:

$$\int_a^b f(x) dx = h \left[f_{\frac{3}{2}} + f_{\frac{5}{2}} + f_{\frac{7}{2}} + \dots + f_{\frac{2n+1}{2}} \right] + O\left(\frac{1}{n^2}\right)$$

(2) 如果 3^{n-1} 等分区间 $[a, b]$, 则相应的中点公式为:

$$s_n = h_n \sum_{i=1}^{3^{n-1}} f(x_i^{(n)})$$

其中

$$h_n = (b - a) / 3^{n-1}$$

$$x_i^{(n)} = a + (i - 1)h_n, \quad i = 1, \dots, 3^{n-1} + 1$$

$$\tilde{x}_i^{(n)} = \frac{1}{2}(x_i^{(n)} + x_{i+1}^{(n)}), \quad i = 1, \dots, 3^{n-1}$$

而

$$\begin{aligned} s_{n+1} &= h_{n+1} \sum_{i=1}^{3^n} f(\tilde{x}_i^{(n+1)}) \\ &= \frac{1}{3} h_n \left(\sum_{i=1}^{3^{n-1}} f(\tilde{x}_{3i-1}^{(n+1)}) + \sum_{i=1}^{3^{n-1}} f(x_{3i}^{(n+1)}) + \sum_{i=1}^{3^{n-1}} f(\tilde{x}_{3i+2}^{(n+1)}) \right) \\ &= \frac{1}{3} \left[s_n + h_n \sum_{i=1}^{3^{n-1}} (f(\tilde{x}_{3i-2}^{(n+1)}) + f(\tilde{x}_{3i}^{(n+1)})) \right] \end{aligned}$$

(3) 若 $ab > 0$, 则当 $a > 0$, $b \rightarrow +\infty$, 或 $b < 0$, $a \rightarrow -\infty$ 时有恒等式

$$\int_a^b f(x) dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt$$

(4) 若 $x=a$ 为 $f(x)$ 的瑕点, 且 $f(x)$ 在 $x=a$ 处与 $(x-a)^{-r}$ ($0 \leq r < 1$) 同阶, 则应用恒等式

$$\int_a^b f(x) dx = \frac{1}{1-r} \int_0^{(b-a)^{1-r}} t^{\frac{r}{1-r}} f\left(a + t^{\frac{1}{1-r}}\right) dt, \quad b > a$$

若 $f(x)$ 在 $x=b$ 处与 $(b-x)^{-r}$ ($0 \leq r < 1$) 同阶, 则应用恒等式

$$\int_a^b f(x) dx = \frac{1}{1-r} \int_0^{(b-a)^{1-r}} t^{\frac{r}{1-r}} f\left(b - t^{\frac{1}{1-r}}\right) dt, \quad b > a$$

特别对 $r = \frac{1}{2}$ 有

$$\int_a^b f(x) dx = \int_0^{\sqrt{b-a}} 2t f(a+t^2) dt, \quad b > a$$

$$\int_a^b f(x) dx = \int_0^{\sqrt{b-a}} 2t f(b-t^2) dt, \quad b > a$$

$$(5) \int_a^\infty f(x) dx = \int_0^{e^{-a}} f(-\ln t) \frac{dt}{t}, \text{ 其中 } \ln t \text{ 为 } t \text{ 的自然对数.}$$

(6) 龙贝格方法.

应用第二欧拉-马克劳林求和公式

$$\begin{aligned} \int_a^b f(x) dx = & h \sum_{i=1}^n f\left(\frac{a+b}{2}\right) - \frac{B_2 h^2}{4} (f'(b) - f'(a)) - \dots \\ & - \frac{B_{2k} h^{2k}}{(2k)!} (1 - 2^{-2k+1}) (f^{(2k-1)}(b) - f^{(2k-1)}(a)) \end{aligned}$$

请参考 3.3 节.

3. 使用说明

(1) MIDPNT (FUNC, A, B, S, N)

N 整型变量, 输入参数, 迭代的阶段

A 实型变量, 输入参数, 积分下限

B 实型变量, 输入参数, 积分上限

S 实型变量, 输出参数, 输出第 N 阶段的迭代值

FUNC 函数子过程, 用户自编, 调用被积函数

(2) MIDINF (FUNK, AA, BB, S, N)

N 整型变量, 输入参数, 迭代阶段

AA 实型变量, 输入参数, 积分下限, 若积分下限为 $-\infty$, 则输入一个绝对值很大的负数, 比如为 -1×10^{20}

BB 实型变量, 输入参数, 积分上限, 若积分上限为 $+\infty$, 则输入一个很大的正数, 比如为 1×10^{20}

S 实型变量, 输出参数, 输出结果

FUNK 函数子过程, 用户自编, 调用被积函数

(3) MIDSQ (FUNK, AA, BB, S, N)

MIDSQU (FUNK, AA, BB, S, N)

N 整型变量, 输入参数, 迭代阶段

AA 实型变量, 输入参数, 积分下限

BB 实型变量, 输入参数, 积分上限

S 实型变量, 输出参数, 输出结果

FUNK 函数子过程, 用户自编, 调用被积函数

(4) MIDEXP (FUNK,AA,BB,S,N)

N 整型变量,输入参数,迭代阶段

AA 实型变量,输入参数,积分下限

BB 实型变量,输入参数,积分上限,输入一个很大的正数,比如为 1×10^{40}

S 实型变量,输出参数,输出结果

FUNK 函数子过程,用户自编,调用被积函数

(5) QROMO (FUNC,A,B,SS,CHOOSE)

A 实型变量,输入参数,积分下限

B 实型变量,输入参数,积分上限

SS 实型变量,输出参数,输出结果

FUNC 函数子过程,用户自编,调用被积函数

CHOOSE 字符串,可选为子过程 MIDPNT,MIDINF,MIDSQL,MID-SQU 中之一

4. 过程

(1) 子过程 MIDPNT.

SUBROUTINE midpnt(func,a,b,s,n)

INTEGER n

REAL a,b,s,func

EXTERNAL func

INTEGER it,j

REAL ddel,del,sum,tnm,x

if (n==1) then

s=(b-a)*func(0.5*(a+b))

else

it=3*(n-2)

tnm=it

del=(b-a)/(3.*tnm)

ddel=del+del

x=a+0.5*del

sum=0.

do j=1,it

sum=sum+func(x)

x=x+ddel

```

        sum=sum+func(x)
        x=x+del
    end do
    s=(s+(b-a)*sum/tnm)/3.
endif
END SUBROUTINE midpnt

```

(2) 子过程 MIDINF.

```

SUBROUTINE midinf(funk,aa,bb,s,n)
INTEGER n
REAL aa,bb,s,funk
EXTERNAL funk
INTEGER it,j
REAL a,b,ddel,del,sum,tnm,func,x
func(x)=funk(1./x)/x**2
b=1./aa
a=1./bb
if (n==1) then
    s=(b-a)*func(0.5*(a+b))
else
    it=3*(n-2)
    tnm=it
    del=(b-a)/(3.*tnm)
    ddel=del+del
    x=a+0.5*del
    sum=0.
    do j=1,it
        sum=sum+func(x)
        x=x+ddel
        sum=sum- func(x)
        x=x+del
    end do
    s=(s+(b-a)*sum/tnm)/3.
endif
END SUBROUTINE midinf

```

(3) 子过程 MIDSQI.

```

SUBROUTINE midsqi(funk,aa,bb,s,n)
INTEGER n
REAL aa,bb,s,funk
EXTERNAL funk
INTEGER it,j
REAL ddel,del,sum,tnm,x,func,a,b
func(x)=2. * x * funk(aa+x * * 2)
b=sqrt(bb-aa)
a=0.
if (n==1) then
    s=(b-a) * func(0.5 * (a+b))
else
    it=3 * * (n-2)
    tnm=it
    del=(b-a)/(3. * tnm)
    ddel=del+del
    x=a+0.5 * del
    sum=0.
    do j=1,it
        sum=sum+func(x)
        x=x+ddel
        sum=sum+func(x)
        x=x+del
    end do
    s=(s+(b-a) * sum/tnm)/3.
endif
END SUBROUTINE midsqi

```

(4) 子过程 MIDSQU.

```

SUBROUTINE midsqu(funk,aa,bb,s,n)
INTEGER n
REAL aa,bb,s,funk
EXTERNAL funk
INTEGER it,j
REAL ddel,del,sum,tnm,x,func,a,b
func(x)=2. * x * funk(bb-x * * 2)
b=sqrt(bb-aa)

```

```

a=0.
if (n==1) then
    s=(b-a)*func(0.5*(a+b))
else
    it=3*(n-2)
    tnm=it
    del=(b-a)/(3.*tnm)
    ddel=del+del
    x=a+0.5*del
    sum=0.
    do j=1,it
        sum=sum+func(x)
        x=x+ddel
        sum=sum+func(x)
        x=x-del
    end do
    s=(s+(b-a)*sum/tnm)/3.
endif
END SUBROUTINE midsqu

```

(5) 子过程 MIDEXP.

```

SUBROUTINE midexp(funk,aa,bb,s,n)
INTEGER n
REAL aa,bb,s,funk
EXTERNAL funk
INTEGER it,j
REAL ddel,del,sum,tnm,x,func,a,b
func(x)=funk(-log(x))/x
b=exp(-aa)
a=0.
if (n==1) then
    s=(b-a)*func(0.5*(a+b))
else
    it=3*(n-2)
    tnm=it
    del=(b-a)/(3.*tnm)
    ddel=del+del

```

```

x=a+0.5*del
sum=0.
do j=1,it
    sum=sum+func(x)
    x=x+del
    sum=sum+func(x)
    x=x-del
end do
s=(s+(b-a)*sum/tnm)/3.
endif
END SUBROUTINE midexp

```

(6) 子过程 QROMO.

```

SUBROUTINE qromo(func,a,b,ss,choose)
INTEGER JMAX,JMAXP,K,KM
REAL a,b,func,ss,EPS
EXTERNAL func,choose
PARAMETER (EPS=1.e-6, JMAX=14, JMAXP=JMAX+1, K=5, KM=K-1)
! USES polint
INTEGER j
REAL dss,h(JMAXP),s(JMAXP)
h(1)=1.
do j=1,JMAX
    call choose(func,a,b,s(j),j)
    if (j>=K) then
        call polint(h(j-KM),s(j-KM),K,0.,ss,dss)
        if (abs(dss)<=EPS*abs(ss)) return
    endif
    s(j+1)=s(j)
    h(j+1)=h(j)/9.
end do
pause 'too many steps in qromo'
END SUBROUTINE qromo

```

5. 例子

(1) 为了验证子过程 MIDPNT,我们在验证程序 D3R5 中采用被积函数为

$\text{FUNC2}=1/\sqrt{x}$, 该函数在原点是奇异的, 积分限为 $A=0.0$ 和 $B=1.0$. 函数 FUNC2 的积分为 $\text{FINT2}=2\sqrt{x}$. 计算结果和 $\text{FINT2}(B)-\text{FINT2}(A)$ 相比较. 验证程序 D3R5 如下:

```

PROGRAM D3R5
  1 Driver for routine MIDPNT
  PARAMETER(NMAX=10)
  EXTERNAL FUNC2,FINT2
  A=0.0
  B=1.0
  WRITE(*,*) 'Integral of FUNC2 computed with MIDPNT'
  WRITE(*,*) &
    'Actual value of integral is', FINT2(B)-FINT2(A)
  WRITE(*,'(1X,T7,A,T20,A)') 'n','Approx. Integral'
  DO I=1,NMAX
    CALL MIDPNT(FUNC2,A,B,S,I)
    WRITE(*,'(1X,I6,F24.6)') I,S
  END DO
END PROGRAM

FUNCTION FUNC2(X)
  FUNC2=1.0/SQRT(X)
END FUNCTION FUNC2

FUNCTION FINT2(X)
  1 Integral of FUNC2
  FINT2=2.0*SQRT(X)
END FUNCTION FINT2

```

计算结果如下:

Integral of FUNC2 computed with MIDPNT

Actual value of integral is 2.000000

n	Approx. Integral
1	1.414214
2	1.653049
3	1.798624
4	1.883616
5	1.932792

6	1.961197
7	1.977596
8	1.987067
9	1.992539
10	1.995702

(2) 在验证程序 D3R6 中验证了 MIDPNT 的各种特殊形式(即 MIDSQL, MIDSQU, MIDINF). 在这些试验中, 我们设无穷大为 $1.0\text{E}20$. 下列积分被执行.

① 计算 $\int_{0.0}^{\pi/2} \frac{\sqrt{x}}{\sin x} dx$ (在 $x=0$ 处具有 $\frac{1}{\sqrt{x}}$ 奇异性, 采用 MIDSQL).

② 计算 $\int_{\pi/2}^{\pi} \frac{\sqrt{\pi-x}}{\sin x} dx$ (在积分上限 $x = \pi$ 处具有 $\frac{1}{\sqrt{x}}$ 奇异性, 采用 MIDSQU).

③ 计算 $\int_{\pi/2}^{\infty} \frac{\sin x}{x^2} dx$ (积分区间延展到 ∞ , 采用 MIDINF. 像下一个积分一样, 收敛十分慢).

④ 计算 $\int_{-\infty}^{-\pi/2} \frac{\sin x}{x^2} dx$ (积分区间延展到 $-\infty$, 采用 MIDINF).

⑤ 计算 $\int_{0.0}^{\infty} \frac{e^{-x}}{\sqrt{x}} dx$ (被积函数在 $x=0.0$ 处有奇异性, 且积分区间延展到 ∞ , 现将积分区间分为两部分, 在 $(0.0, \pi/2)$ 和 $(\pi/2, \infty)$ 分别采用 MIDSQL 和 MIDINF 进行积分. 它们将分别给出结果 RES1 和 RES2, 两个结果相加给出完整积分的值).

验证程序 D3R6 如下:

```

PROGRAM D3R6
! Driver for routine QROMO
PARAMETER(X1=0.0,X2=1.5707963,X3=3.1415926,AINF=1.0E20)
EXTERNAL FUNCL, FUNCU, MIDSQL, MIDSQU, FNCINF, MIDINF, &
      FNCEND
WRITE(*, '(/1X,A)') 'Improper integrals:'
CALL QROMO(FUNCL,X1,X2,RESULT1,MIDSQL)
WRITE(*, '(/1X,A)') &
      'Function: SQRT(x)/SIN(x)           Interval: (0,PI/2)'
WRITE(*, '(1X,A,F8.4)') &
      'Using: MIDSQL                      Result: ', RESULT1
CALL QROMO(FUNCU,X2,X3,RESULT1,MIDSQU)

```

```

WRITE(*, '(/1X,A)')&
  'Function: SQRT(PI-x)/SIN(x)          Interval: (PI/2,PI)'
WRITE(*, '(1X,A,F8.4)')&
  'Using: MIDSQU                        Result:', RESULT1
CALL QROMO(FNCINF,X2,AINF,RESULT1,MIDINF)
WRITE(*, '(/1X,A)')&
  'Function: SIN(x)/x * * 2             Interval: (PI/2,infty)'
WRITE(*, '(1X,A,F8.4)')&
  'Using: MIDINF                        Result:', RESULT1
CALL QROMO(FNCINF,-AINF,-X2,RESULT1,MIDINF)
WRITE(*, '(/1X,A)')&
  'Function: SIN(x)/x * * 2             Interval: (-infty,-PI/2)'
WRITE(*, '(1X,A,F8.4)')&
  'Using: MIDINF                        Result:', RESULT1
CALL QROMO(FNCEND,X1,X2,RES1,MIDSQL)
CALL QROMO(FNCEND,X2,AINF,RES2,MIDINF)
WRITE(*, '(/1X,A)')&
  'Function: EXP(-x)/SQRT(x)           Interval: (0,infty)'
WRITE(*, '(1X,A,F8.4)')&
  'Using: MIDSQL,MIDINF                 Result:', RES1+RES2
END PROGRAM
FUNCTION FUNCL(X)
  FUNCL=SQRT(X)/SIN(X)
END FUNCTION FUNCL
FUNCTION FUNCU(X)
  PARAMETER(PI=3.1415926)
  FUNCU=SQRT(PI-X)/SIN(X)
END FUNCTION FUNCU
FUNCTION FNCINF(X)
  FNCINF=SIN(X)/(X * * 2)
END FUNCTION FNCINF
FUNCTION FNCEND(X)
  FNCEND=EXP(-X)/SQRT(X)
END FUNCTION FNCEND

```

计算结果如下:

Improper integrals:

Function: Sqrt(x)/sin(x)	Interval: (0,PI/2)
Using: MIDSQ	Result: 2.7531
Function: Sqrt(PI-x)/sin(x)	Interval: (PI/2,PI)
Using: MIDSQ	Result: 2.7530
Function: sin(x)/x * * 2	Interval: (PI/2,infty)
Using: MIDINF	Result: 0.1644
Function: sin(x)/x * * 2	Interval: (-infty,-PI/2)
Using: MIDINF	Result: -0.1646
Function: EXP(-x)/Sqrt(x)	Interval: (0,infty)
Using: MIDSQ,MIDINF	Result: 1.7725

3.5 高斯(Gauss)求积法

1. 功能

用高斯求积公式计算定积分

$$I = \int_a^b f(x)dx$$

的近似值.

子过程 QGAUS 用 10 点高斯-勒让德(Gauss-Legendre)积分法求定积分的近似值;子过程 GAULEG 用于计算高斯-勒让德 N 点求积公式中的 N 个权系数及 N 个节点.

2. 方法

(1) 区间 $[-1,1]$ 上的勒让德多项式:

$$p_0(x) \equiv 1, \quad p_1(x) = x$$

$$p_{i+1}(x) = \frac{1}{i+1}[(2i+1)xp_i(x) - ip_{i-1}(x)], \quad i = 1, 2, 3, \dots$$

(2) n 点高斯-勒让德求积公式:

$$\int_{-1}^1 f(x)dx = \sum_{i=1}^n w_i f(x_i) + R_n$$

其中 x_i 为 $p_n(x)$ 的第 i 个零点($i=1, \dots, n$),

$$w_i = 2/[(1-x_i^2)(p'_n(x_i))^2], \quad i = 1, \dots, n$$

$$R_n = \frac{2^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi), \quad \xi \in (-1, 1)$$

(3) 区间 $[a, b]$ 上的高斯-勒让德求积公式:

$$\int_a^b f(y)dy = \frac{b-a}{2} \sum_{i=1}^n w_i f(y_i) + R_n$$

其中

$$y_i = \frac{b-a}{2}x_i + \frac{b+a}{2}$$

$$x_i, w_i \text{ 同上, } i = 1, 2, \dots, n$$

$$R_n = \frac{(b-a)^{(2n+1)}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi), \quad \xi \in (a, b)$$

(4) 对每个 i , 取初值 $Z_i = \cos\left\{\frac{4i-1}{2(2n+1)}\pi\right\}$, 用牛顿法求 $x_i, i=1, 2, \dots, n$.

3. 使用说明

(1) QGAUS (FUNC, A, B, SS)

A 实型变量, 输入参数, 积分下限

B 实型变量, 输入参数, 积分上限

SS 实型变量, 输出参数, 输出结果

FUNC 函数子过程, 用户自编, 调用被积函数

(2) GAULEG (X1, X2, X, W, N)

N 整型变量, 输入参数, 节点数

X1 实型变量, 输入参数, 积分下限

X2 实型变量, 输入参数, 积分上限

X N 个变元的一维实型数组, 输出参数; 存放 N 个节点, 即勒让德多项式的零点

W N 个变元的一维实型数组, 输出参数; 存放求积公式的权系数

4. 过程

(1) 子过程 QGAUS.

```
SUBROUTINE qgaus(func,a,b,ss)
```

```
REAL a,b,ss,func
```

```
EXTERNAL func
```

```
INTEGER j
```

```
REAL dx,xm,xr,w(5),x(5)
```

```
SAVE w,x
```

```
DATA w/.2955242247,.2692667193,.2190863625,.1494513491,&
```

```

        .0666713443/
DATA x/.1488743389,.4333953941,.6794095682,.8650633666,&.
        .9739065285/
xm=0.5*(b+a)
xr=0.5*(b-a)
ss=0
do j=1,5
    dx=xr*x(j)
    ss=ss+w(j)*(func(xm+dx)+func(xm-dx))
end do
ss=xr*ss
END SUBROUTINE qgaus

```

(2) 子过程 GAULEG.

```

SUBROUTINE gauleg(x1,x2,x,w,n)
INTEGER n
REAL x1,x2,x(n),w(n)
DOUBLE PRECISION EPS
PARAMETER (EPS=3.d-14)
INTEGER i,j,m
DOUBLE PRECISION p1,p2,p3,pp,xl,xm,z,zl
m=(n+1)/2
xm=0.5d0*(x2+x1)
xl=0.5d0*(x2-x1)
do i=1,m
    z=cos(3.141592654d0*(i-.25d0)/(n+.5d0))
    do
        p1=1.d0
        p2=0.d0
        do j=1,n
            p3=p2
            p2=p1
            p1=((2.d0*j-1.d0)*z*p2-(j-1.d0)*p3)/j
        end do
        pp=n*(z*p1-p2)/(z*z-1.d0)
        zl=z
        z=zl-p1/pp
    end do

```

```

      if(.not. abs(z-z1)>EPS) exit
    end do
    x(i)=xm-xl*z
    x(n+1-i)=xm+xl*z
    w(i)=2.d0*xl/((1.d0-z*z)*pp*pp)
    w(n+1-i)=w(i)
  end do
END SUBROUTINE gauleg

```

5. 例子

(1) 为了验证程序 QGAUS, 我们采用的例子是函数 $x\exp(-x)$, 该函数在积分区间 x_1 到 x 上积分的精确值为 $(1+x_1)\exp(-x_1)-(1+x)\exp(-x)$. 子过程 QGAUS 中用变量 SS 返回积分值. 验证程序 D3R7 计算了 0.0 到 0.5 以及 0.0 到 5.00 一系列不同积分区间的积分, 并与精确解进行了比较. 验证程序 D3R7 如下:

```

PROGRAM D3R7
! Driver for routine QGAUS
EXTERNAL FUNC
PARAMETER(X1=0.0,X2=5.0,NVAL=10)
DX=(X2-X1)/NVAL
WRITE(*, '(1X,A,T12,A,T23,A/)' ) '0.0 to', 'QGAUS', &
      'Expected'
DO I=1,NVAL
  X=X1+I*DX
  CALL QGAUS(FUNC,X1,X,SS)
  WRITE(*, '(1X,F5.2,2F12.6)' ) X,SS,&
      -(1.0+X)*EXP(-X)+(1.0+X1)*EXP(-X1)
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC=X*EXP(-X)
END FUNCTION FUNC

```

计算结果如下:

0.0 to	QGAUS	Expected
--------	-------	----------

0.50	0.090204	0.090204
1.00	0.264241	0.264241
1.50	0.442175	0.442175
2.00	0.593994	0.593994
2.50	0.712703	0.712703
3.00	0.800852	0.800852
3.50	0.864112	0.864112
4.00	0.908422	0.908422
4.50	0.938901	0.938901
5.00	0.959572	0.959572

(2) 验证了过程 GAULEG 的程序 D3R8 对上述同样的函数执行了同样的积分方法. 但是, 它对高斯-勒让德计算公式中选用了自己的横坐标和权系数, 且不限制为 10 点公式, 它能对任意 N 做 N 点计算. 在验证程序 D3R8 中, 我们在积分区间 0.0 到 1.0 上所取的 $N=10$. 计算结果中我们罗列了这些横坐标和权系数, 并将计算出的积分值与精确解进行了比较. 验证程序 D3R8 如下:

```

PROGRAM D3R8
! Driver for routine GAULEG
PARAMETER (NPOINT=10,X1=0.0,X2=1.0,X3=10.0)
DIMENSION X(NPOINT),W(NPOINT)
CALL GAULEG(X1,X2,X,W,NPOINT)
WRITE(*, '( /1X,T3,A,T10,A,T22,A/ )' ) '#', 'X(I)', 'W(I)'
DO I=1,NPOINT
    WRITE(*, '(1X,I2,2F12.6)') I,X(I),W(I)
END DO
! Demonstrate the use of GAUEG for an integral
CALL GAULEG(X1,X3,X,W,NPOINT)
XX=0.0
DO I=1,NPOINT
    XX=XX+W(I)*FUNC(X(I))
END DO
WRITE(*, '( /1X,A,F12.6)') 'Integral from GAULEG:',XX
WRITE(*, '(1X,A,F12.6)')&
    'Actual value:',1.0-(1.0+X3)*EXP(-X3)
END PROGRAM
FUNCTION FUNC(X)

```

```

      FUNC=X*EXP(-X)
END FUNCTION FUNC

```

计算结果如下:

#	X(I)	W(I)
1	0.013047	0.033336
2	0.067468	0.074726
3	0.160295	0.109543
4	0.283302	0.134633
5	0.425563	0.147762
6	0.574437	0.147762
7	0.716698	0.134633
8	0.839705	0.109543
9	0.932532	0.074726
10	0.986953	0.033336

Integral from GAULEG: 0.999501

Actual value: 0.999501

3.6 三重积分

1. 功能

用高斯法计算三重积分

$$I = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z) dz$$

其中 $y_i(x), z_i(x,y) (i=1,2)$ 均为已知连续函数, $x_i (i=1,2)$ 为已知常数.

2. 方法

定义

$$G(x,y) = \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z) dz$$

$$H(x) = \int_{y_1(x)}^{y_2(x)} G(x,y) dy$$

则

$$I = \int_{x_1}^{x_2} H(x) dx$$

3. 使用说明

QUAD3D (XX1,XX2,SS)

XX1 实型变量,输入参数,输入最外层积分下限

XX2 实型变量,输入参数,输入最外层积分上限

SS 实型变量,输出参数,输出结果

用本子过程必须提供以下五个函数子过程(用户自编):

FUNCTION FUNC (X,Y,Z)被积函数;

FUNCTION Y1(X)第二层积分下限函数;

FUNCTION Y2(X)第二层积分上限函数;

FUNCTION Z1(X,Y)最内层积分下限函数;

FUNCTION Z2(X,Y)最内层积分上限函数.

注:本子过程调用的三个子过程 QGAUSX,QGAUSY,QGAUSZ 是子过程 QGAUS 的三个拷贝.

4. 过程

子过程 QUAD3D.

```
SUBROUTINE quad3d(x1,x2,ss)
```

```
REAL ss,x1,x2,h
```

```
EXTERNAL h
```

```
! USES h,qgausx
```

```
call qgausx(h,x1,x2,ss)
```

```
END SUBROUTINE quad3d
```

```
FUNCTION f(zz)
```

```
REAL f,zz,func,x,y,z
```

```
COMMON /xyz/ x,y,z
```

```
! USES func
```

```
z=zz
```

```
f=func(x,y,z)
```

```
END FUNCTION f
```

```
FUNCTION g(yy)
```

```
REAL g,yy,f,z1,z2,x,y,z
```

```
EXTERNAL f
```

```
COMMON /xyz/ x,y,z
```

```
! USES f,qgausz,z1,z2
```

```

REAL ss
y=yy
call qgausz(f,z1(x,y),z2(x,y),ss)
g=ss
END FUNCTION g
FUNCTION h(xx)
REAL h,xx,g,y1,y2,x,y,z
EXTERNAL g
COMMON /xyz/ x,y,z
! USES g,qgausy,y1,y2
REAL ss
x=xx
call qgausy(g,y1(x),y2(x),ss)
h=ss
END FUNCTION h

```

5. 例子

验证程序 D3R9 所取的例子为函数 $\text{FUNC} = x^2 + y^2 + z^2$ 在球体上的积分, 其半径 XMAX 取为 0.1, 0.2, ..., 1.0 等一系列值. 计算结果与精确解 $4\pi(\text{XMAX})^5/5$ 作了比较. 验证程序 D3R9 如下:

```

PROGRAM D3R9
! Driver for routine QUAD3D
COMMON XMAX
PARAMETER (PI=3.1415926,NVAL=10)
WRITE(*,*) 'Integral of r^2 over a spherical volume'
WRITE(*, '(/4X,A,T14,A,T24,A)') 'Radius', 'QUAD3D', &
    'Actual'
DO I=1,NVAL
    XMAX=0.1*I
    XMIN=-XMAX
    CALL QUAD3D(XMIN,XMAX,S)
    WRITE(*, '(1X,F8.2,2F10.4)') &
        XMAX,S,4.0*PI*(XMAX**5)/5.0
END DO
END PROGRAM
FUNCTION FUNC(X,Y,Z)

```

```

      FUNC=X * * 2 * Y * * 2 + Z * * 2
END FUNCTION FUNC
FUNCTION Z1(X,Y)
COMMON XMAX
      Z1= -SQRT(ABS(XMAX * * 2 - X * * 2 - Y * * 2))
END FUNCTION Z1
FUNCTION Z2(X,Y)
COMMON XMAX
      Z2=SQRT(ABS(XMAX * * 2 - X * * 2 - Y * * 2))
END FUNCTION Z2
FUNCTION Y1(X)
COMMON XMAX
      Y1= -SQRT(ABS(XMAX * * 2 - X * * 2))
END FUNCTION Y1
FUNCTION Y2(X)
COMMON XMAX
      Y2=SQRT(ABS(XMAX * * 2 - X * * 2))
END FUNCTION Y2
SUBROUTINE QGAUSX(FUNC,A,B,SS)
DIMENSION X(5),W(5)
DATA X/.1488743389,.4333953941,.6794095682,&
      .8650633666,.9739065285/
DATA W/.2955242247,.2692667193,.2190863625,&
      .1494513491,.0666713443/
XM=0.5 * (B+A)
XR=0.5 * (B-A)
SS=0.
DO J=1,5
      DX=XR * X(J)
      SS=SS+W(J) * (FUNC(XM+DX)+FUNC(XM-DX))
END DO
SS=XR * SS
END SUBROUTINE QGAUSX
SUBROUTINE QGAUSY(FUNC,A,B,SS)
DIMENSION X(5),W(5)
DATA X/.1488743389,.4333953941,.6794095682,&
      .8650633666,.9739065285/

```

```

DATA W/.2955242247,.2692667193,.2190863625,&
      .1494513491,.0666713443/
XM=0.5*(B+A)
XR=0.5*(B-A)
SS=0.
DO J=1,5
    DX=XR*X(J)
    SS=SS+W(J)*(FUNC(XM+DX)+FUNC(XM-DX))
END DO
SS=XR*SS
RETURN
END SUBROUTINE QGAUSY
SUBROUTINE QGAUSZ(FUNC,A,B,SS)
DIMENSION X(5),W(5)
DATA X/.1488743389,.4333953941,.6794095682,&
      .8650633666,.9739065285/
DATA W/.2955242247,.2692667193,.2190863625,&
      .1494513491,.0666713443/
XM=0.5*(B+A)
XR=0.5*(B-A)
SS=0.
DO J=1,5
    DX=XR*X(J)
    SS=SS+W(J)*(FUNC(XM+DX)+FUNC(XM-DX))
END DO
SS=XR*SS
END SUBROUTINE QGAUSZ

```

计算结果如下：

Integral of r^2 over a spherical volume

Radius	QUAD3D	Actual
0.10	0.0000	0.0000
0.20	0.0008	0.0008
0.30	0.0061	0.0061
0.40	0.0258	0.0257
0.50	0.0786	0.0785

0.60	0.1956	0.1954
0.70	0.4227	0.4224
0.80	0.8242	0.8235
0.90	1.4852	1.4841
1.00	2.5152	2.5133

第4章 特殊函数

本章共有 32 个过程,它们可以用于计算解析解中频繁出现的各种各样的函数.这些函数过程的验证程序基本上是相同的.为了验证过程的通用性,所取的自变量尽量采用一些极端值,以便只用少量的数据即可进行全面的验证.

4.1 Γ 函数、贝塔函数、阶乘及二项式系数

1. 功能

本节有五个函数过程,它们的功能是计算 Γ 函数 $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ 及贝塔函数 $B(z, w) = \int_0^1 t^{z-1} (1-t)^{w-1} dt$ 的函数值.

(1) 函数过程 GAMMLN,对实数 $x > 0$,计算 $\ln \Gamma(x)$.

(2) 函数过程 FACTRL,对给定的正整数 n ,用浮点数计算 n 的阶乘 $n!$.

(3) 函数过程 FACTLN,用浮点数计算 $\ln(n!)$.

(4) 函数过程 BICO,用浮点数计算二项式系数 $\binom{n}{k} = \frac{n!}{k! (n-k)!}$, $0 \leq k \leq n$.

(5) 函数过程 BETA,计算贝塔函数 $B(z, w)$ 的值.

2. 方法

(1) Γ 函数的计算.

由 Lanczos 逼近公式,可选择整数 n, q 及常数系数 $c_i (i=0, 1, \dots, n)$ 有

$$\Gamma(z+1) = \left(z + q + \frac{1}{2}\right)^{z+\frac{1}{2}} e^{-(z+q+\frac{1}{2})} \cdot \sqrt{2\pi} \left[\sum_{i=1}^n \frac{c_i}{z+i} + c_0 + \epsilon \right] \quad (z > 0)$$

其中 ϵ 表示误差.

取 $n=6, q=5, c_0=1, c_1=76.18009173, c_2=-86.50532033, c_3=24.01409822, c_4=-1.231739516, c_5=0.00120858003, c_6=0.536382 \times 10^{-5}$, 则 $|\epsilon| < 2 \times 10^{-10}$.

对 $\Gamma(x)$, 如果 $x > 1$, 记 $x = 1 + z$, $z > 0$, 则

$$\begin{aligned}\ln \Gamma(x) &= \ln \Gamma(1+z) \\ &= \left(z + \frac{1}{2}\right) \ln(z + 5.5) - (z + 5.5) \\ &\quad + \ln \left[\sqrt{2\pi} \left(\sum_{i=1}^6 \frac{c_i}{z+i} + c_0 + \epsilon \right) \right]\end{aligned}$$

如果 $0 < x < 1$, 由 $\Gamma(z+1) = z\Gamma(z)$, 则有 $\Gamma(x) = \Gamma(x+1)/x$, 或者由于

$$\Gamma(1-z) = \frac{\pi}{\Gamma(z)\sin(\pi z)} = \frac{\pi z}{\Gamma(1+z)\sin(\pi z)}$$

记 $x = 1 - z$, 则 $0 < z < 1$.

(2) $n!$ 的计算.

$n!$ 由 $n! = \Gamma(n+1)$ 计算, 但这只用于 n 很大时, 当 n 很小时这样做很浪费时间, 直接计算 $n!$ 则更可取.

(3) 计算贝塔函数

$$B(z, w) = B(w, z) = \int_0^1 t^{z-1} (1-t)^{w-1} dt = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)}$$

3. 使用说明

(1) GAMMLN (XX)

XX 大于 1 的实自变量, 输入参数

调用此函数过程后, 得到值 $\ln \Gamma(XX)$, $XX > 1$, 对 $0 < XX \leq 1$ 的 $\Gamma(XX)$ 或 $\ln \Gamma(XX)$, 由关系式 $\Gamma(1-z) = \frac{\pi z}{\Gamma(1+z)\sin(\pi z)}$ 或 $\Gamma(xx) = \Gamma(xx+1)/xx$ 计算.

(2) FACTRL (N)

N 正整数自变量, 输入参数

(3) FACTLN (N)

N 正整数自变量, 输入参数

用户调用此函数过程得到值 $\ln(N!)$, 它被(4)调用.

(4) BICO (N, K)

N, K 均为正整数自变量, 输入参数, 且满足 $0 \leq K \leq N$

本函数过程要调用函数过程 FACTLN(N). 调用本函数过程后可得出值

$$\binom{N}{K} = \frac{N!}{K! (N-K)!}.$$

(5) BETA (Z,W)

Z,W 均为正实数自变量,输入参数

本函数过程通过调用第一个函数过程 GAMMLN 得出值.

4. 过程

(1) 函数过程 GAMMLN.

```

FUNCTION gammln(xx)
REAL gammln,xx
INTEGER j
DOUBLE PRECISION ser,stp,tmp,x,y,cof(6)
SAVE cof,stp
DATA cof,stp/76.18009172947146d0,-86.50532032941677d0,&
    24.01409824083091d0,-1.231739572450155d0,&
    .1208650973866179d-2,-.5395239384953d-5,&
    2.5066282746310005d0/
x=xx
y=x
tmp=x+5.5d0
tmp=(x+0.5d0)*log(tmp)-tmp
ser=1.0000000000190015d0
do j=1,6
    y=y+1.d0
    ser=ser+cof(j)/y
end do
gammln=tmp-log(stp*ser/x)
END FUNCTION gammln

```

(2) 函数过程 FACTRL.

```

FUNCTION factrl(n)
INTEGER n
REAL factrl
! USES gammln
INTEGER j,ntop
REAL a(33),gammln
SAVE ntop,a

```



```

DATA ntop,a(1)/0,1. /
if (n<0) then
    pause 'negative factorial in factrl'
else if (n<=ntop) then
    factrl=a(n+1)
else if (n<=32) then
    do j=ntop+1,n
        a(j+1)=j*a(j)
    end do
    ntop=n
    factrl=a(n+1)
else
    factrl=exp(gammln(n+1.))
endif
END FUNCTION factrl

```

(3) 函数过程 FACTLN.

```

FUNCTION factln(n)
INTEGER n
REAL factln
! USES gammln
REAL a(100),gammln
SAVE a
DATA a/100* -1. /
if (n<0) pause 'negative factorial in factln'
if (n<=99) then
    if (a(n+1)<0.) a(n+1)=gammln(n+1.)
    factln=a(n+1)
else
    factln=gammln(n+1.)
endif
END FUNCTION factln

```

(4) 函数过程 BICO.

```

FUNCTION bico(n,k)
INTEGER k,n

```

```

REAL bico
! USES factln
REAL factln
bico=nint(exp(factln(n)-factln(k)-factln(n-k)))
END FUNCTION bico

```

(5) 函数过程 BETA.

```

FUNCTION beta(z,w)
REAL beta,w,z
! USES gammln
REAL gammln
beta=exp(gammln(z)+gammln(w)-gammln(z+w))
END FUNCTION beta

```

5. 例子

(1) 验证 GAMMLN 的程序 D4R1 如下:

```

PROGRAM D4R1
! Driver for routine GAMMLN
CHARACTER TEXT * 14
PARAMETER(P1=3.1415926)
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Gamma Function') exit
END DO
READ(5,*) NVAL
WRITE(*,*) 'Log of GAMMA function:'
WRITE(*, '(1X,T11,A1,T24,A6,T40,A10)') 'X', 'Actual', 'GAMMA(X)'
DO I=1,NVAL
  READ(5,*) X,ACTUAL
  IF (X>0.0) THEN
    IF (X>=1.0) THEN
      CALC=GAMMLN(X)
    ELSE
      CALC=GAMMLN(X+1.0)-LOG(X)
    ENDIF

```

```

WRITE(*, '(F12.2,2F18.6)') X, LOG(ACTUAL), CALC
ENDIF
END DO
CLOSE(5)
END

```

计算结果如下:

Log of GAMMA function:

X	Actual	GAMMA(X)
1.00	0.000000	0.000000
1.20	-0.085374	-0.085374
1.40	-0.119613	-0.119613
1.60	-0.112592	-0.112592
1.80	-0.071084	-0.071084
2.00	0.000000	0.000000
0.20	1.524064	1.524064
0.40	0.796678	0.796678
0.60	0.398234	0.398234
0.80	0.152060	0.152060
10.00	12.801827	12.801827
20.00	39.339886	39.339886
30.00	71.257042	71.257042

(2) 验证 FACTRL 的程序 D4R2 如下:

```

PROGRAM D4R2
! Driver for routine FACTRL
CHARACTER TEXT * 11
OPEN(5, FILE='d:\VF_SHU\DATA\FNCVAL.DAT', STATUS='OLD')
Do
  READ(5, '(A)') TEXT
  IF(.NOT. text/='N--factorial') EXIT
END Do
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T6,A1,T21,A6,T38,A9)') &

```

```

      'N','Actual','FACTRL(N)'
Do 1=1,NVAL
  READ(5,*) N,ACTUAL
  IF (ACTUAL.LT. (1.0E10)) THEN
    WRITE(*, '(16,2F20.0)') N,ACTUAL,FACTRL(N)
  ELSE
    WRITE(*, '(16,2E20.7)') N,ACTUAL,FACTRL(N)
  ENDIF
END Do
CLOSE(5)
END

```

计算结果如下:

N—factorial

N	Actual	FACTRL(N)
1	1.	1.
2	2.	2.
3	6.	6.
4	24.	24.
5	120.	120.
6	720.	720.
7	5040.	5040.
8	40320.	40320.
9	362880.	362880.
10	3628800.	3628800.
11	39916800.	39916800.
12	479001600.	479001600.
13	6227020800.	6227020800.
14	0.8717829E+11	0.8717829E+11
15	0.1307675E+13	0.1307674E+13
20	0.2432904E+19	0.2432902E+19
25	0.1551122E+26	0.1551121E+26
30	0.2652528E+33	0.2652529E+33

(3) 验证 BICO 的程序 D4R3 如下:

```

PROGRAM D4R3
! Driver for routine BICO
CHARACTER TEXT * 21
OPEN(5,FILE:='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT.TEXT/='Binomial Coefficients') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T6,A1,T12,A1,T15,A9,T30,A9)') &
  'N','K','Actual','BICO(N,K)'
DO I=1,NVAL
  READ(5,*) N,K,BINCO
  WRITE(*,'(2I6,2F12.0)') N,K,BINCO,BICO(N,K)
END DO
CLOSE(5)
END

```

计算结果如下:

Binomial Coefficients

N	K	Actual	BICO(N,K)
1	0	1.	1.
6	1	6.	6.
6	3	20.	20.
6	5	6.	6.
15	1	15.	15.
15	3	455.	455.
15	5	3003.	3003.
15	7	6435.	6435.
15	9	5005.	5005.
15	11	1365.	1365.
15	13	105.	105.

25	1	25.	25.
25	3	2300.	2300.
25	5	53130.	53130.
25	7	480700.	480699.
25	9	2042975.	2042977.
25	11	4457400.	4457398.
25	13	5200300.	5200284.
25	15	3268760.	3268759.
25	17	1081575.	1081575.

(4) 验证 FACTLN 的程序 D4R4 如下:

```

PROGRAM D4R4
! Driver for routine FACTLN
CHARACTER TEXT * 11
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='N-factorial') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) 'Log of N-factorial'
WRITE(*, '(1X,T6,A1,T18,A6,T34,A9)') &
  'N', 'Actual', 'FACTLN(N)'
DO I=1,NVAL
  READ(5,*) N,VALUE
  WRITE(*, '(16,2F18.6)') N,LOG(VALUE),FACTLN(N)
END DO
CLOSE(5)
END

```

计算结果如下:

Log of N-factorial

N	Actual	FACTLN(N)
1	0.000000	0.000000

2	0.693147	0.693147
3	1.791759	1.791759
4	3.178054	3.178054
5	4.787492	4.787492
6	6.579251	6.579251
7	8.525162	8.525162
8	10.604603	10.604603
9	12.801827	12.801827
10	15.104413	15.104413
11	17.502308	17.502308
12	19.987215	19.987215
13	22.552164	22.552164
14	25.191221	25.191221
15	27.899273	27.899271
20	42.335617	42.335617
25	58.003605	58.003605
30	74.658234	74.658234

(5) 验证 BETA 的程序 D4R5 如下:

```

PROGRAM D4R5
! Driver for routine BETA
CHARACTER TEXT * 13
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Beta Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T7,A1,T15,A1,T26,A6,T43,A9)') &
      'W','Z','Actual','BETA(W,Z)'
DO I=1,NVAL
  READ(5,*) W,Z,VALUE
  WRITE(*,'(2F8.2,2E18.6)') W,Z,VALUE,BETA(W,Z)

```

```

END DO
CLOSE(5)
END

```

计算结果如下:

Beta Function

W	Z	Actual	BETA(W,Z)
1.00	1.00	0.100000E+01	0.100000E+01
0.20	1.00	0.500000E+01	0.500000E+01
1.00	0.20	0.500000E+01	0.500000E+01
0.40	1.00	0.250000E+01	0.250000E+01
1.00	0.40	0.250000E+01	0.250000E+01
0.60	1.00	0.166667E+01	0.166667E+01
0.80	1.00	0.125000E+01	0.125000E+01
6.00	6.00	0.360750E-03	0.360750E-03
6.00	5.00	0.793651E-03	0.793651E-03
6.00	4.00	0.198413E-02	0.198413E-02
6.00	3.00	0.595238E-02	0.595238E-02
6.00	2.00	0.238095E-01	0.238095E-01
7.00	7.00	0.832501E-04	0.832501E-04
5.00	5.00	0.158730E-02	0.158730E-02
4.00	4.00	0.714286E-02	0.714285E-02

4.2 不完全 Γ 函数、误差函数

1. 功能

本节包括五个函数过程和两个子过程,它们的功能分别是:

(1) 函数过程 GAMMP, 对于不完全 Γ 函数

$$P(a, x) \equiv \frac{\gamma(a, x)}{\Gamma(a)} \equiv \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt \quad (a > 0)$$

根据收敛快慢调用 GSER 或 GCF 计算函数值.

(2) 函数过程 GAMMQ, 对于不完全 Γ 函数的余函数

$$Q(a, x) \equiv 1 - P(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt \quad (a > 0)$$

根据收敛快慢调用 GSER 或 GCF 计算函数值.

(3) 子过程 GSER, GCF, 用级数展开式和连分式展开分别计算 $P(a, x)$ 和 $Q(a, x)$ 的函数值.

(4) 函数过程 ERF, ERFC 计算误差函数 $\operatorname{erf}(x)$ 和余误差函数 $\operatorname{erfc}(x)$ 的函数值.

(5) 函数过程 ERFCC, 计算相对误差小于 1.2×10^{-7} 的余误差函数.

2. 方法

(1) 不完全 Γ 函数.

不完全 Γ 函数为

$$P(a, x) = \nu(a, x) / \Gamma(a)$$

其中

$$\nu(a, x) = \int_0^x e^{-t} t^{a-1} dt \quad (a > 0)$$

易见 $P(a, 0) = 0, P(a, \infty) = 1$.

对于 $\nu(a, x)$ 有如下级数展式

$$\nu(a, x) = e^{-x} x^a \sum_{n=0}^{\infty} \frac{\Gamma(a)}{\Gamma(a+1+n)} x^n$$

而 $\Gamma(a+1) = a\Gamma(a)$.

当级数中的某项如第 k 项与前 k 项的和相比已很小时, 求和结束, 这时

$$P(a, x) \approx e^{-x} x^a \sum_{n=0}^k \frac{\Gamma(a)}{\Gamma(a+n+1)}$$

(2) 不完全 Γ 函数的余函数.

不完全 Γ 函数的余函数为

$$Q(a, x) = 1 - P(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

其中

$$\Gamma(a, x) = \int_x^{\infty} e^{-t} t^{a-1} dt \quad (a > 0)$$

对于 $\Gamma(a, x)$ 有连分式展式

$$\Gamma(a, x) = e^{-x} x^a \left(\frac{1}{x+1} \frac{1-a}{1+} \frac{1}{x+1} \frac{2-a}{1+} \frac{2}{x+1} \dots \right) \quad (x > 0)$$

当 $x < a+1$ 时, 用级数展式表示法计算收敛快, 当 $x \geq a+1$ 时, 用连分式展式计算收敛快.

(3) 误差函数、余误差函数.

误差函数是

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \begin{cases} -P(0.5, x^2) & (x < 0) \\ P(0.5, x^2) & (x \geq 0) \end{cases}$$

余误差函数是

$$\operatorname{erfc}(x) \equiv 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

容易看出:

$$\operatorname{erf}(0) = 0, \quad \operatorname{erf}(\infty) = 1, \quad \operatorname{erf}(-x) = -\operatorname{erf}(x)$$

$$\operatorname{erf}(x) = P\left(\frac{1}{2}, x^2\right), \quad x \geq 0, \quad \operatorname{erfc}(x) = Q\left(\frac{1}{2}, x^2\right), \quad x \geq 0$$

(4) 累积泊松概率函数.

设随机变量 ξ 服从参数为 x 的泊松分布, 则

$$P_x\{\xi = m\} = \frac{x^m}{m!} e^{-x}, \quad m = 0, 1, 2, \dots$$

而

$$P_x\{\xi < k\} = \sum_{m=0}^{k-1} P_x\{\xi = m\} = Q(k, x)$$

(5) χ^2 概率函数.

设 $P(\chi^2|\nu)$ 表示服从自由度为 ν 的 χ^2 分布的随机变量小于值 χ^2 的概率, 由 χ^2 分布的密度函数

$$P(x, \nu) = \begin{cases} \frac{1}{2^{\nu/2} \Gamma(\nu/2)} e^{-\frac{x}{2}} x^{\frac{\nu}{2}-1} & (x > 0) \\ 0 & (x < 0) \end{cases}$$

其中 ν 表示 χ^2 分布的自由度, 则有

$$P(\chi^2|\nu) = P\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

$$Q(\chi^2|\nu) \equiv 1 - P(\chi^2|\nu) = Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

3. 使用说明

GAMMP(A, X)

GAMMQ(A, X)

GSER(GAMSER, A, X, GLN)

GCF(GAMMCF, A, X, GLN)

ERF(X), ERFC(X), ERFC(X)

A	实型变量,输入参数, Γ 函数的参数
X	实型变量,输入参数,自变量
GAMSER	实型变量,输出参数,存放 $P(a, x)$ 的近似值
GLN	实型变量,输出参数,存放 $\ln\Gamma(a)$
GAMMCF	实型变量,输出参数,存放 $Q(a, x) = 1 - P(a, x)$ 的近似值

4. 过程

(1) 函数过程 GAMMP.

```

FUNCTION gammp(a,x)
REAL a,gammp,x
! USES gcf,gser
REAL gammcf,gamser,gln
if(x<0. .or. a<=0.) pause 'bad arguments in gammp'
if(x<a+1.)then
    call gser(gamser,a,x,gln)
    gammp=gamser
else
    call gcf(gammcf,a,x,gln)
    gammp=1.-gammcf
endif
END FUNCTION gammp

```

(2) 函数过程 GAMMQ.

```

FUNCTION gammq(a,x)
REAL a,gammq,x
! USES gcf,gser
REAL gammcf,gamser,gln
if(x<0. .or. a<=0.) pause 'bad arguments in gammq'
if(x<a+1.) then
    call gser(gamser,a,x,gln)
    gammq=1.-gamser
else
    call gcf(gammcf,a,x,gln)
    gammq=gammcf
endif
END FUNCTION gammq

```

(3) 子过程 GSER.

```

SUBROUTINE gser(gamser,a,x,gln)
INTEGER ITMAX
REAL a,gamser,gln,x,EPS
PARAMETER (ITMAX=100,EPS=3.e-7)
! USES gammln
INTEGER n
REAL ap,del,sum,gammln
gln=gammln(a)
if(x<=0.) then
    if(x<0.) pause 'x < 0 in gser'
    gamser=0.
    return
endif
ap=a
sum=1./a
del=sum
do n=1,ITMAX
    ap=ap+1.
    del=del*x/ap
    sum=sum+del
    if(abs(del)<abs(sum)*EPS) then
        gamser=sum*exp(-x+a*log(x)-gln)
        return
    end if
end do
pause 'a too large, ITMAX too small in gser'
END SUBROUTINE gser

```

(4) 子过程 GCF.

```

SUBROUTINE gcf(gammcf,a,x,gln)
INTEGER ITMAX
REAL a,gammcf,gln,x,EPS,FPMIN
PARAMETER (ITMAX=100,EPS=3.e-7,FPMIN=1.e-30)
! USES gammln
INTEGER i

```

```

REAL an,b,c,d,del,h,gammln
gln=gammln(a)
b=x+1.-a
c=1./FPMIN
d=1./b
h=d
do i=1,ITMAX
  an=-i*(i-a)
  b=b+2.
  d=an*d+b
  if(abs(d)<FPMIN) d=FPMIN
  c=b+an/c
  if(abs(c)<FPMIN) c=FPMIN
  d=1./d
  del=d*c
  h=h*del
  if(abs(del-1.)<EPS) then
    gammcf=exp(-x+a*log(x)-gln)*h
    return
  endif
end do
pause 'a too large, ITMAX too small in gcf'
END SUBROUTINE gcf

```

(5) 函数过程 ERF.

```

FUNCTION erf(x)
REAL erf,x
! USES gammp
REAL gammp
if(x<0.)then
  erf=-gammp(.5,x**2)
else
  erf=gammp(.5,x**2)
endif
END FUNCTION erf

```

(6) 函数过程 ERFC.

```

FUNCTION erfc(x)
REAL erfc,x

```

```

! USES gammp,gammq
REAL gammp,gammq
if(x<0.)then
    erfc=1.+gammp(.5,x * * 2)
else
    erfc=gammq(.5,x * * 2)
endif
END FUNCTION erfc

```

(7) 函数过程 ERFCC.

```

FUNCTION erfcc(x)
REAL erfcc,x
REAL t,z
z=abs(x)
t=1./(1.+0.5*z)
erfcc=t*exp(-z*z-1.26551223+t*(1.00002368+t*(.37409196+t*&
(.09678418-t*(-.18628806+t*(.27886807+t*(-1.13520398+t*&
(1.48851587+t*(-.82215223+t*.17087277))))))))))
if (x<0.) erfcc=2.-erfcc
END FUNCTION erfcc

```

5. 例子

在每个例子中,数值计算的结果均与精确解进行比较.

(1) 验证 GAMMP 的程序 D4R6 如下:

```

PROGRAM D4R6
! Driver for routine GAMMP
CHARACTER TEXT * 25
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
READ(5,'(A)') TEXT
IF (.NOT. TEXT/='Incomplete Gamma Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T16,A,T25,A,T35,A)')&
'A','X','Actual','GAMMP(A,X)'

```

```

DO I=1,NVAL
  READ(5,*) A,X,VALUE
  WRITE(*, '(1X,F6.2,3F12.6)') A,X,VALUE,GAMMP(A,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Incomplete Gamma Function

A	X	Actual	GAMMP(A,X)
0.10	0.031623	0.742026	0.742026
0.10	0.316228	0.911975	0.911975
0.10	1.581139	0.989896	0.989896
0.50	0.070711	0.293128	0.293128
0.50	0.707107	0.765642	0.765642
0.50	3.535534	0.992166	0.992166
1.00	0.100000	0.095163	0.095163
1.00	1.000000	0.632121	0.632120
1.00	5.000000	0.993262	0.993262
1.10	0.104881	0.075747	0.075747
1.10	1.048809	0.607646	0.607646
1.10	5.244044	0.993343	0.993342
2.00	0.141421	0.009105	0.009105
2.00	1.414214	0.413064	0.413064
2.00	7.071068	0.993145	0.993145
6.00	2.449490	0.038732	0.038732
6.00	12.247449	0.982594	0.982594
11.00	16.583124	0.940427	0.940427
26.00	25.495098	0.486387	0.486387
41.00	44.821869	0.735971	0.735971

(2) 验证 GAMMQ 的程序 D4R7 如下:

```
PROGRAM D4R7
```

```

1 Driver for routine GAMMQ
CHARACTER TEXT * 25
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Incomplete Gamma Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T16,A,T25,A,T35,A)') &
      'A','X','Actual','GAMMQ(A,X)'
DO I=1,NVAL
  READ(5,*) A,X,VALUE
  WRITE(*,'(1X,F6.2,3F12.6)') A,X,1.-VALUE,GAMMQ(A,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Incomplete Gamma Function

A	X	Actual	GAMMQ(A,X)
0.10	0.031623	0.257974	0.257974
0.10	0.316228	0.088025	0.088025
0.10	1.581139	0.010104	0.010104
0.50	0.070711	0.706872	0.706872
0.50	0.707107	0.234358	0.234358
0.50	3.535534	0.007834	0.007834
1.00	0.100000	0.904837	0.904837
1.00	1.000000	0.367879	0.367880
1.00	5.000000	0.006738	0.006738
1.10	0.104881	0.924253	0.924253
1.10	1.048809	0.392354	0.392354
1.10	5.244044	0.006657	0.006658
2.00	0.141421	0.990895	0.990895

2.00	1.414214	0.586936	0.586936
2.00	7.071068	0.006855	0.006855
6.00	2.449490	0.961268	0.961268
6.00	12.247449	0.017406	0.017406
11.00	16.583124	0.059573	0.059573
26.00	25.495098	0.513613	0.513613
41.00	44.821869	0.264029	0.264029

(3) 验证 GSER 的程序 D4R8 如下:

```

PROGRAM D4R8
! Driver for routine GSER
CHARACTER TEXT * 25
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT.TEXT/='Incomplete Gamma Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T16,A,T25,A,T36,A,T47,A,T62,A)')&
  'A','X','Actual','GSER(A,X)','GAMMLN(A)','GLN'
DO I=1,NVAL
  READ(5,*) A,X,VALUE
  CALL GSER(GAMSER,A,X,GLN)
  WRITE(*,'(1X,F6.2,5F12.6)') A,X,VALUE,GAMSER,&
    GAMMLN(A),GLN
END DO
CLOSE(5)
END

```

计算结果如下:

Incomplete Gamma Function

A	X	Actual	GSER(A,X)	GAMMLN(A)	GLN
0.10	0.031623	0.742026	0.742026	2.252713	2.252713

0.10	0.316228	0.911975	0.911975	2.252713	2.252713
0.10	1.581139	0.989896	0.989896	2.252713	2.252713
0.50	0.070711	0.293128	0.293128	0.572365	0.572365
0.50	0.707107	0.765642	0.765642	0.572365	0.572365
0.50	3.535534	0.992166	0.992166	0.572365	0.572365
1.00	0.100000	0.095163	0.095163	0.000000	0.000000
1.00	1.000000	0.632121	0.632120	0.000000	0.000000
1.00	5.000000	0.993262	0.993262	0.000000	0.000000
1.10	0.104881	0.075747	0.075747	-0.049872	-0.049872
1.10	1.048809	0.607646	0.607646	-0.049872	-0.049872
1.10	5.244044	0.993343	0.993343	-0.049872	-0.049872
2.00	0.141421	0.009105	0.009105	0.000000	0.000000
2.00	1.414214	0.413064	0.413064	0.000000	0.000000
2.00	7.071068	0.993145	0.993145	0.000000	0.000000
6.00	2.449490	0.038732	0.038732	4.787492	4.787492
6.00	12.247449	0.982594	0.982594	4.787492	4.787492
11.00	16.583124	0.940427	0.940427	15.104413	15.104413
26.00	25.495098	0.486387	0.486387	58.003605	58.003605
41.00	44.821869	0.735971	0.735970	110.320641	110.320641

(4) 验证 GCF 的程序 D4R9 如下:

```

PROGRAM D4R9
! Driver for routine GCF
CHARACTER TEXT * 25
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT/='Incomplete Gamma Function') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T5,A,T16,A,T25,A,T36,A,T47,A,T62,A)') &
  'A', 'X', 'Actual', 'GCF(A,X)', 'GAMMLN(A)', 'GLN'
DO I=1,NVAL

```

```

READ(5,*) A,X,VALUE
IF(X.GE.A+1.0) THEN
  CALL GCF(GAMMCF,A,X,GLN)
  WRITE(*,'(1X,F6.2,5F12.6)') A,X,1.-VALUE,&
    GAMMCF,GAMMLN(A),GLN
ENDIF
END DO
CLOSE(5)
END

```

计算结果如下:

Incomplete Gamma Function

A	X	Actual	GCF(A,X)	GAMMLN(A)	GLN
0.10	1.581139	0.010104	0.010104	2.252713	2.252713
0.50	3.535534	0.007834	0.007834	0.572365	0.572365
1.00	5.000000	0.006738	0.006738	0.000000	0.000000
1.10	5.244044	0.006657	0.006658	-0.049872	-0.049872
2.00	7.071068	0.006855	0.006855	0.000000	0.000000
6.00	12.247449	0.017406	0.017406	4.787492	4.787492
11.00	16.583124	0.059573	0.059573	15.104413	15.104413
41.00	44.821869	0.264029	0.264029	110.320641	110.320641

(5) 验证 ERF 的程序 D4R10 如下:

```

PROGRAM D4R10
! Driver for routine ERF
CHARACTER TEXT * 14
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT.TEXT/='Error Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A1,T12,A6,T24,A6)')&
  'X','Actual','ERF(X)'

```

```

DO I=1,NVAL
  READ(5,*) X,VALUE
  WRITE(*,'(1X,F6.2,2F12.7)') X,VALUE,ERF(X)
END DO
CLOSE(5)
END

```

计算结果如下：

Error Function

X	Actual	ERF(X)
0.00	0.0000000	0.0000000
0.10	0.1124629	0.1124629
0.20	0.2227026	0.2227026
0.30	0.3286268	0.3286267
0.40	0.4283924	0.4283924
0.50	0.5204999	0.5204998
0.60	0.6038561	0.6038561
0.70	0.6778012	0.6778012
0.80	0.7421010	0.7421011
0.90	0.7969082	0.7969083
1.00	0.8427008	0.8427008
1.10	0.8802051	0.8802050
1.20	0.9103140	0.9103138
1.30	0.9340079	0.9340079
1.40	0.9522851	0.9522851
1.50	0.9661051	0.9661052
1.60	0.9763484	0.9763484
1.70	0.9837905	0.9837905
1.80	0.9890905	0.9890905
1.90	0.9927904	0.9927904

(6) 验证 ERFC 的程序 D4R11 如下：

```

PROGRAM D4R11
! Driver for routine ERFC
CHARACTER TEXT * 14
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')

```

```

DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT/='Error Function') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) 'Complementary error function'
WRITE(*, '(1X,T5,A1,T12,A6,T23,A7)') &
      'X', 'Actual', 'ERFC(X)'
DO I=1, NVAL
  READ(5, *) X, VALUE
  VALUE=1.0-VALUE
  WRITE(*, '(1X,F6.2,2F12.7)') X, VALUE, ERFC(X)
END DO
CLOSE(5)
END

```

计算结果如下：

Complementary error function

X	Actual	ERFC(X)
0.00	1.0000000	1.0000000
0.10	0.8875371	0.8875371
0.20	0.7772974	0.7772974
0.30	0.6713732	0.6713732
0.40	0.5716076	0.5716076
0.50	0.4795001	0.4795002
0.60	0.3961439	0.3961439
0.70	0.3221988	0.3221988
0.80	0.2578990	0.2578989
0.90	0.2030918	0.2030917
1.00	0.1572992	0.1572992
1.10	0.1197949	0.1197950
1.20	0.0896860	0.0896862
1.30	0.0659921	0.0659921
1.40	0.0477149	0.0477149
1.50	0.0338949	0.0338948
1.60	0.0236516	0.0236516
1.70	0.0162095	0.0162095

1.80	0.0109095	0.0109095
1.90	0.0072096	0.0072096

(7) 验证 ERFCC 的程序 D4R12 如下:

```

PROGRAM D4R12
! Driver for routine ERFCC
CHARACTER TEXT * 14
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Error Function') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) 'Complementary error function'
WRITE(*, '(1X,T5,A1,T12,A6,T23,A8)') &
  'X', 'Actual', 'ERFCC(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE
  VALUE=1.0-VALUE
  WRITE(*, '(1X,F6.2,2F12.7)') X,VALUE,ERFCC(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Complementary error function

X	Actual	ERFCC(X)
0.00	1.0000000	1.0000000
0.10	0.8875371	0.8875371
0.20	0.7772974	0.7772975
0.30	0.6713732	0.6713732
0.40	0.5716076	0.5716076
0.50	0.4795001	0.4795001
0.60	0.3961439	0.3961439
0.70	0.3221988	0.3221988
0.80	0.2578990	0.2578991

0.90	0.2030918	0.2030918
1.00	0.1572992	0.1572992
1.10	0.1197949	0.1197949
1.20	0.0896860	0.0896860
1.30	0.0659921	0.0659921
1.40	0.0477149	0.0477149
1.50	0.0338949	0.0338949
1.60	0.0236516	0.0236516
1.70	0.0162095	0.0162095
1.80	0.0109095	0.0109095
1.90	0.0072096	0.0072096

4.3 不完全贝塔函数

1. 功能

本节过程的功能是计算不完全贝塔函数

$$I_x(a, b) \equiv B_x(a, b) / B(a, b)$$

其中

$$B_x(a, b) \equiv \int_0^x t^{a-1} (1-t)^{b-1} dt \quad (a, b > 0)$$

函数过程 BETAI 用连分式表示式计算不完全贝塔函数值; 函数过程 BETACF 用于计算不完全贝塔函数函数中的连分式.

2. 方法

(1) 不完全贝塔函数.

由不完全贝塔函数的定义可得:

$$I_0(a, b) = 0, \quad I_1(a, b) = 1$$

$$I_x(a, b) = 1 - I_{1-x}(b, a) \quad (\text{对称性})$$

$$I_x(a, b) = \frac{x^a(1-x)^b}{aB(a, b)} \left[1 + \sum_{n=0}^{\infty} \frac{B(a+1, n+1)}{B(a+b, n+1)} x^{n+1} \right]$$

但计算中采用如下更有用的连分式表示式:

$$I_x(a, b) = \frac{x^a(1-x)^b}{aB(a, b)} \left[\frac{1}{1 + \frac{d_1}{1 + \frac{d_2}{1 + \dots}}} \right]$$

其中

$$d_{2m+1} = - \frac{(a+m)(a+b+m)}{(a+2m)(a+2m+1)} x$$

$$d_{2m} = \frac{m(b-m)}{(a+2m-1)(a+2m)} x$$

当 $x < (a+1)/(a+b+2)$ 时, 应用连分式计算不完全贝塔函数收敛速度非常快, 最坏情形也只需 $O(\sqrt{\max(a,b)})$ 次迭代, 而对 $x > (a+1)/(a+b+2)$, 由对称性, 采用等价的连分式计算.

(2) t 分布概率函数.

设 ξ 为服从自由度为 ν 的 t 分布随机变量, $A(t|\nu) = P\{|\xi| < t\}$, 则

$$\begin{aligned} A(t|\nu) &= \frac{1}{\nu^{1/2} B\left(\frac{1}{2}, \frac{\nu}{2}\right)} \int_{-t}^t \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}} dx \\ &= 1 - I_{\mu}\left(\frac{\nu}{2}, \frac{1}{2}\right), \quad \mu = \frac{\nu}{\nu + t^2} \end{aligned}$$

(3) F 分布的分布概率函数.

自由度为 (ν_1, ν_2) 的 F 分布的分布密度函数是

$$p(x; \nu_1, \nu_2) = \begin{cases} \frac{\Gamma\left(\frac{\nu_1 + \nu_2}{2}\right)}{\Gamma\left(\frac{\nu_1}{2}\right) \Gamma\left(\frac{\nu_2}{2}\right)} \left(\frac{\nu_1}{\nu_2}\right) \left(\frac{\nu_1}{\nu_2} x\right)^{\frac{\nu_1}{2}-1} \left(1 + \frac{\nu_1}{\nu_2} x\right)^{-\frac{\nu_1 + \nu_2}{2}} & (x > 0) \\ 0 & (x < 0) \end{cases}$$

因而 F 分布的概率分布函数为

$$\begin{aligned} P(F|\nu_1, \nu_2) &= \int_0^F p(x; \nu_1, \nu_2) dx = 1 - I_{\mu}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right) \\ Q(F|\nu_1, \nu_2) &\equiv 1 - P(F|\nu_1, \nu_2) = I_{\mu}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right) \end{aligned}$$

其中 $\mu = \frac{\nu_2}{\nu_2 + \nu_1 F}$, F 为自变量.

(4) 累积二项概率分布.

在 n 重伯努利试验中, 假设每次试验成功的概率为 p , 则累积二项概率是成功的次数大于等于 k 的概率

$$P \equiv \sum_{j=k}^n \binom{n}{j} p^j (1-p)^{n-j} = I_p(k; n-k+1)$$

当 $n > 12$ 时, 用不完全贝塔函数计算累积二项概率比直接计算二项式系数好得多, 而当 $n \leq 12$ 时, 二者都是可接受的.

3. 使用说明

BETACF (A,B,X)

BETAI (A,B,X)

A 实型变量,输入参数,不完全贝塔函数的参数 $-x$ 的幂次

B 实型变量,输入参数,不完全贝塔函数的参数 $-(1-x)$ 的幂次

X 实型变量,输入参数,自变量

4. 过程

(1) 函数过程 BETACF.

```

FUNCTION betacf(a,b,x)
INTEGER maxit
REAL betacf,a,b,x,EPS,fpmin
PARAMETER (maxit=100,EPS=3.e-7,fpmin=1.e-30)
INTEGER m,m2
REAL aa,c,d,del,h,qab,qam,qap
qab=a+b
qap=a+1.
qam=a-1.
c=1.
d=1.-qab*x/qap
if(abs(d)<fpmin)d=fpmin
d=1./d
h=d
do m=1,maxit
  m2=2*m
  aa=m*(b-m)*x/((qam+m2)*(a+m2))
  d=1.+aa*d
  if(abs(d)<fpmin)d=fpmin
  c=1.+aa/c
  if(abs(c)<fpmin)c=fpmin
  d=1./d
  h=h*d*c
  aa=-(a+m)*(qab+m)*x/((a+m2)*(qap+m2))
  d=1.+aa*d
  if(abs(d)<fpmin)d=fpmin

```

```

c=1. +aa/c
if(abs(c)<fpmin)c=fpmin
d=1./d
del=d*c
h=h*del
if(abs(del-1.)<eps) then
    betacf=h
    return
end if
end do
pause 'a or b too big, or maxit too small in betacf'
END FUNCTION betacf

```

(2) 函数过程 BETAI.

```

FUNCTION betai(a,b,x)
REAL betai,a,b,x
! USES betacf,gammln
REAL bt,betacf,gammln
if(x<0. .or. x>1. ) pause 'bad argument x in betai'
if(x==0. .or. x==1. ) then
    bt=0.
else
    bt=exp(gammln(a+b)-gammln(a)-gammln(b)+a*log(x)&
        +b*log(1.-x))
endif
if(x<(a+1.)/(a+b+2.)) then
    betai=bt*betacf(a,b,x)/a
    return
else
    betai=1.-bt*betacf(b,a,1.-x)/b
    return
endif
END FUNCTION betai

```

5. 例子

验证 BETAI 和 BETACF 的程序 D4R13 如下:

```

PROGRAM D4R13
! Driver for routine BETAI,BETACF
CHARACTER TEXT * 24
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',SIATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT / -'Incomplete Beta Function') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T6,A,T16,A,T28,A,T37,A,T48,A)') &
  'A','B','X','Actual','BETAI(X)'
DO I=1,NVAL
  READ(5, *) A,B,X,VALUE
  WRITE(*, '(1X,F6.2,4F12.6)') A,B,X,VALUE,BETAI(A,B,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Incomplete Beta Function

A	B	X	Actual	BETAI(X)
0.50	0.500000	0.010000	0.063769	0.063769
0.50	0.500000	0.100000	0.204833	0.204833
0.50	0.500000	1.000000	1.000000	1.000000
1.00	0.500000	0.010000	0.005013	0.005013
1.00	0.500000	0.100000	0.051317	0.051317
1.00	0.500000	1.000000	1.000000	1.000000
1.00	1.000000	0.500000	0.500000	0.500000
5.00	5.000000	0.500000	0.500000	0.500000
10.00	0.500000	0.900000	0.151641	0.151641
10.00	5.000000	0.500000	0.089783	0.089783
10.00	5.000000	1.000000	1.000000	1.000000
10.00	10.000000	0.500000	0.500000	0.499999

20.00	5.000000	0.800000	0.459877	0.459878
20.00	10.000000	0.600000	0.214682	0.214682
20.00	10.000000	0.800000	0.950737	0.950736
20.00	20.000000	0.500000	0.500000	0.500004
20.00	20.000000	0.600000	0.897941	0.897942
30.00	10.000000	0.700000	0.224130	0.224129
30.00	10.000000	0.800000	0.758641	0.758641
40.00	20.000000	0.700000	0.700178	0.700178

4.4 零阶、一阶和任意整数阶的第一、二类贝塞尔函数

1. 功能

本节有六个过程,它们的功能分别是:

- (1) 函数过程 BESSJ0,对实数 x ,用双精度计算第一类零阶贝塞尔函数.
- (2) 函数过程 BESSY0,对实数 $x > 0$,用双精度计算第二类零阶贝塞尔函数.
- (3) 函数过程 BESSJ1,对实数 x ,用双精度计算第一类一阶贝塞尔函数.
- (4) 函数过程 BESSY1,对实数 $x > 0$ 用双精度计算第二类一阶贝塞尔函数.
- (5) 函数过程 BESSY,对实数 $x > 0$,计算第二类 N 阶贝塞尔函数.
- (6) 函数过程 BESSJ,对实数 $x > 0$,计算第一类 N 阶贝塞尔函数.

2. 方法

第一类 ν 阶贝塞尔函数定义为

$$J_{\nu}(x) = \left(\frac{1}{2}x\right)^{\nu} \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{4}x^2\right)^k}{k! \Gamma(\nu + k + 1)} \quad (\nu \text{ 为实数})$$

第二类 ν 阶贝塞尔函数定义为

$$Y_{\nu}(x) = \frac{J_{\nu}(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \quad (\nu \text{ 为实数})$$

- (1) 计算 $J_0(x), J_1(x), Y_0(x), Y_1(x)$.

当 $|x| < 8$ 时,用 x 的有理函数多项式分别近似 $J_0(x), Y_0(x), J_1(x), Y_1(x)$,记 $y=x^2$,则

$$\begin{aligned}
J_0(x) &\approx (r_1 + r_2y + r_3y^2 + r_4y^3 + r_5y^4 + r_6y^5) / \\
&\quad (s_1 + s_2y + s_3y^2 + s_4y^3 + s_5y^4 + s_6y^5) \\
Y_0(x) &\approx (r_1 + r_2y + r_3y^2 + r_4y^3 + r_5y^4 + r_6y^5) / \\
&\quad (s_1 + s_2y + s_3y^2 + s_4y^3 + s_5y^4 + s_6y^5) + \frac{2}{\pi} J_0(x) \cdot \ln x \\
J_1(x) &\approx x(r_1 + r_2y + r_3y^2 + r_4y^3 + r_5y^4 + r_6y^5) / \\
&\quad (s_1 + s_2y + s_3y^2 + s_4y^3 + s_5y^4 + s_6y^5) \\
Y_1(x) &\approx x(r_1 + r_2y + r_3y^2 + r_4y^3 + r_5y^4 + r_6y^5) / \\
&\quad (s_1 + s_2y + s_3y^2 + s_4y^3 + s_5y^4 + s_6y^5 + s_7y^6) \\
&\quad + \frac{2}{\pi} \left[J_1(x) \ln x - \frac{1}{x} \right]
\end{aligned}$$

其中系数具体见各自的程序,即 $J_0(x)$ 的近似公式中的 $r_i, s_i (1 \leq i \leq 6)$ 见函数过程 BESSJ0, $Y_0(x)$ 的近似公式中的 $r_i, s_i (1 \leq i \leq 6)$ 见函数过程 BESSY0, 等等.

当 $x \geq 8$ 时,令 $z = 8/x$, 采用下列近似公式 ($n=0, 1$):

$$\begin{aligned}
J_n(x) &\approx \sqrt{\frac{2}{\pi x}} [P_n(z) \cos(X_n) - Q_n(z) \sin(X_n)] \\
Y_n(x) &\approx \sqrt{\frac{2}{\pi x}} [P_n(z) \sin(X_n) + Q_n(z) \cos(X_n)]
\end{aligned}$$

其中

$$X_n \equiv x - \frac{2n+1}{4}\pi$$

而

$$\begin{aligned}
P_0(z) &= p_1 + p_2z^2 + p_3z^4 + p_4z^6 + p_5z^8 \\
Q_0(z) &= q_1z + q_2z^3 + q_3z^5 + q_4z^7 + q_5z^9
\end{aligned}$$

系数 p_i 和 $q_i (1 \leq i \leq 5)$ 见函数过程 BESSJ0 和 BESSY0.

$$\begin{aligned}
P_1(z) &= p_1 + p_2z^2 + p_3z^4 + p_4z^6 + p_5z^8 \\
Q_1(z) &= q_1z + q_2z^3 + q_3z^5 + q_4z^7 + q_5z^9
\end{aligned}$$

系数 $p_i, q_i (1 \leq i \leq 5)$ 见函数过程 BESSJ1 或 BESSY1.

(2) 计算任意整数阶的贝塞尔函数 $J_n(x), Y_n(x)$.

① 利用递推公式

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

由 $Y_0(x)$ 和 $Y_1(x)$ 计算 $Y_n(x) (n \geq 2, x > 0)$.

② 用逆递推方法 (Miller 算法) 计算 $J_n(x) (x > 0, n \geq 2)$.

对于给定的 $x \geq n$, 选定适当大的正整数 M , 令

$$J_{M+1}^* = 0, \quad J_M^* = 1$$

以递推公式

$$J_{n-1}^* = \frac{2n}{x} J_n^* - J_{n+1}^*, \quad n = M, M-1, \dots, 1$$

计算出 $J_{M-1}^*, J_{M-2}^*, \dots, J_1^*$. 由公式

$$BK = J_0^* + 2 \sum_{j=1}^K J_{2j}^* \quad \left(\text{其中 } K = \left[\frac{M}{2} \right] \text{ 为 } \frac{M}{2} \text{ 的整数部分} \right)$$

则有

$$J_n(x) = J_n^* / BK \quad (n \geq 2)$$

对于 $x < n$, 以顺递推公式

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

计算 $J_n(x)$ ($n \geq 2, x > 0$).

3. 使用说明

(1) BESSJ0 (X)

X 实自变量, 输入参数

(2) BESSY0 (X)

X 正的实自变量, 输入参数

该函数过程要调用函数过程 BESSJ0.

(3) BESSJ1 (X)

X 实自变量, 输入参数

(4) BESSY1 (X)

X 正的实自变量, 输入参数

该函数过程要调用函数过程 BESSJ1.

(5) BESSY (N, X)

N 整型变量, 输入参数, 第二类贝塞尔函数的阶, $N \geq 2$

X 正的实自变量, 输入参数

该函数过程要调用函数过程 BESSY0, BESSY1.

(6) BESSJ (N, X)

N 整型变量, 输入参数, 第一类贝塞尔函数的阶, $N \geq 2$

X 正的实自变量, 输入参数

该函数过程要调用函数过程 BESSJ0, BESSJ1.

4. 过程

(1) 函数过程 BESSJ0.

```

FUNCTION bessj0(x)
REAL bessj0,x
REAL ax,xx,z
DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,&
    r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,s5,s6,y
SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,&
    s1,s2,s3,s4,s5,s6
DATA p1,p2,p3,p4,p5/1.d0,-.1098628627d-2,.2734510407d-4,&
    -.2073370639d-5,.2093887211d-6/, q1,q2,q3,q4,q5&
    /- .1562499995d-1,.1430488765d-3,-.6911147651d-5,&
    .7621095161d-6,-.934945152d-7/
DATA r1,r2,r3,r4,r5,r6/57568490574.d0,-13362590354.d0,&
    651619640.7d0,-11214424.18d0,77392.33017d0,&
    -184.9052456d0/,s1,s2,s3,s4,s5,s6/57568490411.d0,&
    1029532985.d0,9494680.718d0,59272.64853d0,&
    267.8532712d0,1.d0/
if(abs(x)<8.) then
    y=x * * 2
    bessj0=(r1+y * (r2+y * (r3+y * (r4+y * (r5+y * r6)))))/&
        (s1+y * (s2+y * (s3+y * (s4+y * (s5+y * s6))))))
else
    ax=abs(x)
    z=8./ax
    y=z * * 2
    xx=ax-.785398164
    bessj0=sqrt(.636619772/ax) * (cos(xx) * (p1+y * (p2+y * &
        (p3+y * (p4+y * p5))))-z * sin(xx) * (q1+y * (q2+y * &
        (q3+y * (q4+y * q5))))))
endif
END FUNCTION bessj0

```

(2) 函数过程 BESSY0.

```

FUNCTION bessy0(x)

```

```

REAL bessy0,x
! USES bessj0
REAL xx,z,bessj0
DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,&
    r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,s5,s6,y
SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,&
    s1,s2,s3,s4,s5,s6
DATA p1,p2,p3,p4,p5/1.d0,-.1098628627d-2,.2734510407d-4,&
    -.2073370639d-5,.2093887211d-6/, q1,q2,q3,q4,q5/&
    -.1562499995d-1,.1430488765d-3,-.6911147651d-5,&
    .7621095161d-6,-.934945152d-7/
DATA r1,r2,r3,r4,r5,r6/-2957821389.d0,7062834065.d0,&
    -512359803.6d0,10879881.29d0,-86327.92757d0,&
    228.4622733d0/,s1,s2,s3,s4,s5,s6/40076544269.d0,&
    745249964.8d0,7189466.438d0,47447.26470d0,&
    226.1030244d0,1.d0/
if(x<8.)then
    y=x * * 2
    bessy0=(r1+y * (r2+y * (r3+y * (r4+y * (r5+y * r6)))))/&
        (s1+y * (s2+y * (s3+y * (s4+y * (s5+y * s6)))))+&
        .636619772 * bessj0(x) * log(x)
else
    z=8./x
    y=z * * 2
    xx=x-.785398164
    bessy0=sqrt(.636619772/x) * (sin(xx) * (p1+y * (p2+y * &
        (p3+y * (p4+y * p5)))))+z * cos(xx) * (q1+y * (q2+y * &
        (q3+y * (q4+y * q5))))
endif
END FUNCTION bessy0

```

(3) 函数过程 BESSJ1.

```

FUNCTION bessj1(x)
REAL bessj1,x
REAL ax,xx,z
DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,&
    r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,s5,s6,y

```



```

SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,&
      s1,s2,s3,s4,s5,s6
DATA r1,r2,r3,r4,r5,r6/72362614232.d0,-7895059235.d0,&
      242396853.1d0,-2972611.439d0,15704.48260d0,&
      30.16036606d0/,s1,s2,s3,s4,s5,s6/144725228442.d0,&
      2300535178.d0,18583304.74d0,99447.43394d0,&
      376.9991397d0,1.d0/
DATA p1,p2,p3,p4,p5/1.d0,.183105d-2,-.3516396496d-4,&
      .2457520174d-5,-.240337019d-6/,q1,q2,q3,q4,q5/&
      .04687499995d0,-.2002690873d-3,.8449199096d-5,&
      -.88228987d-6,.105787412d-6/
if(abs(x)<8.) then
  y=x * * 2
  bessj1=x * (r1+y * (r2+y * (r3+y * (r4+y * (r5+y * r6)))))/&
      (s1+y * (s2-y * (s3-y * (s4+y * (s5+y * s6))))))
else
  ax=abs(x)
  z=8./ax
  y=z * * 2
  xx=ax-2.356194491
  bessj1=sqrt(.636619772/ax) * (cos(xx) * (p1+y * (p2+y * &
      (p3+y * (p4+y * p5)))))-z * sin(xx) * (q1+y * (q2+y * &
      (q3+y * (q4+y * q5)))) * sign(1.,x)
endif
END FUNCTION bessj1

```

(4) 函数过程 BESSY1.

```

FUNCTION bessy1(x)
REAL bessy1,x
! USES bessj1
REAL xx,z,bessj1
DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,&
      r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,s5,s6,s7,y
SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,&
      s1,s2,s3,s4,s5,s6,s7
DATA p1,p2,p3,p4,p5/1.d0,.183105d-2,-.3516396496d-4,&
      .2457520174d-5,-.240337019d-6/,q1,q2,q3,q4,q5/&

```

```

      .04687499995d0, -.2002690873d-3, .8449199096d-5, &
      -.88228987d-6, .105787412d-6/
DATA r1,r2,r3,r4,r5,r6/-.4900604943d13, .1275274390d13, &
      -.5153438139d11, .7349264551d9, -.4237922726d7, &
      .8511937935d4/, s1,s2,s3,s4,s5,s6,s7/.2499580570d14, &
      .4244419664d12, .3733650367d10, .2245904002d8, &
      .1020426050d6, .3549632885d3, 1.d0/
if(x<8.)then
  y=x * * 2
  bessyl=x * (r1+y * (r2+y * (r3+y * (r4+y * (r5+y * r6)))))/&
      (s1+y * (s2+y * (s3+y * (s4+y * (s5+y * (s6+y * s7))))))&
      +.636619772 * (bessjl(x) * log(x) - 1. /x)
else
  z=8. /x
  y=z * * 2
  xx=x-2.356194491
  bessyl=sqrt(.636619772/x) * (sin(xx) * (p1+y * (p2+y * &
      (p3+y * (p4+y * p5))))+z * cos(xx) * (q1+y * (q2+y * &
      (q3+y * (q4+y * q5))))))
endif
END FUNCTION bessyl

```

(5) 函数过程 BESSY.

```

FUNCTION bessy(n,x)
INTEGER n
REAL bessy,x
! USES bessy0,bessyl
INTEGER j
REAL by,bym,byp,tox,bessy0,bessyl
if(n<2) pause 'bad argument n in bessy'
tox=2. /x
by=bessyl(x)
bym=bessy0(x)
do j=1,n-1
  byp=j * tox * by-bym
  bym=by
  by=byp

```

```

end do
bessy=by
END FUNCTION bessy

```

(6) 函数过程 BESSJ.

```

FUNCTION bessj(n,x)
INTEGER n,IACC
REAL bessj,x,BIGNO,BIGNI
PARAMETER (IACC=40,BIGNO=1.e10,BIGNI=1.e-10)
! USES bessj0,bessj1
INTEGER j,jsum,m
REAL ax,bj,bjm,bjp,sum,tox,bessj0,bessj1
if(n<2) pause 'bad argument n in bessj'
ax=abs(x)
if(ax==0.) then
    bessj=0.
else if(ax>float(n)) then
    tox=2./ax
    bjm=bessj0(ax)
    bj=bessj1(ax)
    do j=1,n-1
        bjp=j * tox * bj-bjm
        bjm=bj
        bj=bjp
    end do
    bessj=bj
else
    tox=2./ax
    m=2*((n+int(sqrt(float(IACC*n)))))/2
    bessj=0.
    jsum=0
    sum=0.
    bjp=0.
    bj=1.
    do j=m,1,-1
        bjm=j * tox * bj-bjp
        bjp=bj

```

```

      bj=bjm
      if(abs(bj)>BIGNO) then
        bj=bj * BIGNI
        bjp=bjp * BIGNI
        bessj=bessj * BIGNI
        sum=sum * BIGNI
      endif
      if(jsum/=0) sum=sum+bj
      jsum=1-jsum
      if(j==n) bessj=bjp
    end do
    sum=2. * sum-bj
    bessj=bessj/sum
  endif
  if(x<0. . and. mod(n,2)==1) bessj=-bessj
END FUNCTION bessj

```

5. 例子

(1) 验证 BESSJ0 的程序 D4R14 如下：

```

PROGRAM D4R14
! Driver for routine BESSJ0
CHARACTER TEXT * 18
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Bessel Function J0') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*, '(1X,T7,A1,T19,A6,T33,A9)') &
  'X','Actual','BESSJ0(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE
  WRITE(*, '(1X,F8.2,2F16.7)') X,VALUE,BESSJ0(X)
END DO
CLOSE(5)

```

END

计算结果如下:

Bessel Function J0

X	Actual	BESSJ0(X)
-5.00	-0.1775968	-0.1775968
-4.00	-0.3971498	-0.3971498
-3.00	-0.2600520	-0.2600520
-2.00	0.2238908	0.2238908
-1.00	0.7651976	0.7651977
0.00	1.0000000	1.0000000
1.00	0.7651977	0.7651977
2.00	0.2238908	0.2238908
3.00	-0.2600520	-0.2600520
4.00	-0.3971498	-0.3971498
5.00	-0.1775968	-0.1775968
6.00	0.1506453	0.1506453
7.00	0.3000793	0.3000793
8.00	0.1716508	0.1716508
9.00	-0.0903336	-0.0903335
10.00	-0.2459358	-0.2459358
11.00	-0.1711903	-0.1711904
12.00	0.0476893	0.0476892
13.00	0.2069261	0.2069261
14.00	0.1710735	0.1710735

(2) 验证 BESSY0 的程序 D4R15 如下:

```
PROGRAM D4R15
```

```
! Driver for routine BESSY0
```

```
CHARACTER TEXT * 18
```

```
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
```

```
DO
```

```
  READ(5,'(A)') TEXT
```

```

      IF (.NOT. TEXT/='Bessel Function Y0') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*, '(1X,T7,A1,T19,A6,T33,A9)') &
      'X', 'Actual', 'BESSY0(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE
  WRITE(*, '(1X,F8.2,2F16.7)') X,VALUE,BESSY0(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Bessel Function Y0

X	Actual	BESSY0(X)
0.10	-1.5342387	-1.5342386
1.00	0.0882570	0.0882570
2.00	0.5103757	0.5103757
3.00	0.3768500	0.3768500
4.00	-0.0169407	-0.0169407
5.00	-0.3085176	-0.3085176
6.00	-0.2881947	-0.2881947
7.00	-0.0259497	-0.0259497
8.00	0.2235215	0.2235215
9.00	0.2499367	0.2499367
10.00	0.0556712	0.0556712
11.00	-0.1688473	-0.1688473
12.00	-0.2252373	-0.2252373
13.00	-0.0782079	-0.0782079
14.00	0.1271926	0.1271925

(3) 验证 BESSJ1 的程序 D4R16 如下:

```
PROGRAM D4R16
```

```

! Driver for routine BESSJ1
CHARACTER TEXT * 18
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT.TEXT/='Bessel Function J1') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T7,A1,T19,A6,T33,A9)')&
  'X','Actual','BESSJ1(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE
  WRITE(*,'(1X,F8.2,2F16.7)') X,VALUE,BESSJ1(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Bessel Function J1

X	Actual	BESSJ1(X)
-5.00	0.3275791	0.3275791
-4.00	0.0660433	0.0660433
-3.00	-0.3390590	-0.3390590
-2.00	-0.5767248	-0.5767248
-1.00	-0.4400506	-0.4400506
0.00	0.0000000	0.0000000
1.00	0.4400506	0.4400506
2.00	0.5767248	0.5767248
3.00	0.3390590	0.3390590
4.00	-0.0660433	-0.0660433
5.00	-0.3275791	-0.3275791
6.00	-0.2766839	-0.2766839
7.00	-0.0046828	-0.0046828

8.00	0.2346364	0.2346363
9.00	0.2453118	0.2453118
10.00	0.0434728	0.0434727
11.00	-0.1767853	-0.1767853
12.00	-0.2234471	-0.2234471
13.00	-0.0703181	-0.0703181
14.00	0.1333752	0.1333752

(4) 验证 BESSY1 的程序 D4R17 如下:

```

PROGRAM D4R17
  ! Driver for routine BESSY1
  CHARACTER TEXT * 18
  OPEN(5,FILE:='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
  DO
    READ(5, '(A)') TEXT
    IF (.NOT. TEXT/='Bessel Function Y1') EXIT
  END DO
  READ(5, *) NVAL
  WRITE(*, *) TEXT
  WRITE(*, '(1X,T7,A1,T19,A6,T33,A9)') &
    'X', 'Actual', 'BESSY1(X)'
  DO I=1,NVAL
    READ(5, *) X,VALUE
    WRITE(*, '(1X,F8.2,2F16.7)') X,VALUE,BESSY1(X)
  END DO
  CLOSE(5)
END

```

计算结果如下:

Bessel Function Y1

X	Actual	BESSY1(X)
0.10	-6.4589510	-6.4589505
1.00	-0.7812128	-0.7812128
2.00	-0.1070324	-0.1070324

3.00	0.3246744	0.3246744
4.00	0.3979257	0.3979257
5.00	0.1478631	0.1478631
6.00	--0.1750103	-0.1750103
7.00	-0.3026672	-0.3026672
8.00	-0.1580605	-0.1580605
9.00	0.1043146	0.1043146
10.00	0.2490154	0.2490154
11.00	0.1637055	0.1637055
12.00	-0.0570992	--0.0570992
13.00	-0.2100814	-0.2100814
14.00	-0.1666448	-0.1666448

(5) 验证 BESSY 的程序 D4R18 如下:

```

PROGRAM D4R18
! Driver for routine BESSY
CHARACTER TEXT * 18
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Bessel Function Yn') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T7,A,T16,A,T27,A,T43,A)') &
  'N', 'X', 'Actual', 'BESSY(N,X)'
DO I=1,NVAL
  READ(5, *) N,X,VALUE
  WRITE(*, '(1X,I6,F10.2,2E18.6)') N,X,VALUE,BESSY(N,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Bessel Function Y_n

N	X	Actual	BESSY(N,X)
2	1.00	-0.165068E+01	-0.165068E+01
2	2.00	-0.617408E+00	-0.617408E+00
2	5.00	0.367663E+00	0.367663E+00
2	10.00	-0.586808E-02	-0.586816E-02
2	50.00	0.957932E-01	0.957931E-01
5	1.00	-0.260406E+03	-0.260406E+03
5	2.00	-0.993599E+01	-0.993599E+01
5	5.00	-0.453695E+00	-0.453695E+00
5	10.00	0.135403E+00	0.135403E+00
5	50.00	-0.785484E-01	-0.785485E-01
10	1.00	--0.121618E+09	0.121618E+09
10	2.00	-0.129185E+06	-0.129185E+06
10	5.00	-0.251291E+02	-0.251291E+02
10	10.00	-0.359814E+00	-0.359814E+00
10	50.00	0.572390E-02	0.572377E-02
20	1.00	-0.411397E+23	-0.411397E+23
20	2.00	-0.408165E+17	-0.408165E+17
20	5.00	-0.593397E+09	-0.593397E+09
20	10.00	-0.159748E+04	-0.159748E+04
20	50.00	0.164426E-01	0.164425E-01

(6) 验证 BESSJ 的程序 D4R19 如下:

```

PROGRAM D4R19
! Driver for routine BESSJ
CHARACTER TEXT * 18
OPEN(5,FILE = 'D:\VF_SHU\DATA\FNCVAL.DAT',STATUS = 'OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Bessel Function Jn') EXIT
END DO

```

```

READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T7,A,T16,A,T27,A,T43,A)')&
      'N','X','Actual','BESSJ(N,X)'
DO I=1,NVAL
  READ(5,*) N,X,VALUE
  WRITE(*,'(1X,I6,F10.2,2E18.6)') N,X,VALUE,BESSJ(N,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Bessel Function Jn

N	X	Actual	BESSJ(N,X)
2	1.00	0.114903E+00	0.114903E+00
2	2.00	0.352834E+00	0.352834E+00
2	5.00	0.465651E-01	0.465651E-01
2	10.00	0.254630E+00	0.254630E+00
2	50.00	-0.597128E-01	-0.597130E-01
5	1.00	0.249758E-03	0.249758E-03
5	2.00	0.703963E-02	0.703963E-02
5	5.00	0.261141E+00	0.261141E+00
5	10.00	-0.234062E+00	-0.234062E+00
5	50.00	-0.814002E-01	-0.814002E-01
10	1.00	0.263062E-09	0.263062E-09
10	2.00	0.251539E-06	0.251539E-06
10	5.00	0.146780E-02	0.146780E-02
10	10.00	0.207486E+00	0.207486E+00
10	50.00	-0.113848E+00	-0.113848E+00
20	1.00	0.387350E-24	0.387350E-24
20	2.00	0.391897E-18	0.391897E-18
20	5.00	0.277033E-10	0.277033E-10
20	10.00	0.115134E-04	0.115134E-04
20	50.00	-0.116704E+00	-0.116704E+00

4.5 零阶、一阶和任意整数阶的第一、二类变形贝塞尔函数

1. 功能

本节有六个函数过程, 它们的功能分别是:

- (1) 函数过程 BESSI0, 对实数 x , 用双精度计算 $I_0(x)$.
- (2) 函数过程 BESSK0, 对实数 $x > 0$, 用双精度计算 $K_0(x)$.
- (3) 函数过程 BESSI1, 对实数 x , 用双精度计算 $I_1(x)$.
- (4) 函数过程 BESSK1, 对实数 $x > 0$, 用双精度计算 $K_1(x)$.
- (5) 函数过程 BESSK, 对实数 $x > 0$, 计算 N 阶第二类变形贝塞尔函数 $K_N(x)$.
- (6) 函数过程 BESSI, 对实数 $x > 0$, 计算 N 阶第一类变形贝塞尔函数 $I_N(x)$.

2. 方法

计算变形的贝塞尔函数 $I_N(x)$ 和 $K_N(x)$, 等同于计算纯虚自变量的 $J_N(x)$ 和 $Y_N(x)$, 它们之间有关系:

$$I_N(x) = (-1)^n J_N(ix)$$

$$K_N(x) = \frac{\pi}{2} i^{n+1} [J_N(ix) + iY_N(ix)]$$

计算它们的公式均是由 Abramowitz 和 Stegun 给出的近似多项式, 具体如下:

(1) 计算 $I_0(x)$.

当 $|x| < 3.75$ 时,

$$I_0(x) \approx p_1 + p_2 y + p_3 y^2 + p_4 y^3 + p_5 y^4 + p_6 y^5 + p_7 y^6$$

当 $|x| \geq 3.75$ 时,

$$I_0(x) \approx \left(\frac{e^x}{\sqrt{|x|}} \right) \cdot (q_1 + q_2 z + q_3 z^2 + q_4 z^3 + q_5 z^4 + q_6 z^5 + q_7 z^6 + q_8 z^7 + q_9 z^8)$$

其中

$$y = (x/3.75)^2, \quad z = 3.75/|x|$$

系数 $p_i (1 \leq i \leq 7)$, $q_i (1 \leq i \leq 9)$ 见函数过程 BESSI0.

(2) 计算 $K_0(x)$.

当 $0 < x \leq 2$ 时,

$$K_0(x) \approx I_0(x) \ln z + p_1 + p_2 y + p_3 y^2 + p_4 y^3 + p_5 y^4 + p_6 y^5 + p_7 y^6$$

当 $2 < x < +\infty$ 时,

$$K_0(x) \approx \left(\frac{e^{-x}}{\sqrt{x}} \right) (q_1 + q_2 z + q_3 z^2 + q_4 z^3 + q_5 z^4 + q_6 z^5 + q_7 z^6)$$

其中

$$y = x^2/4, \quad z = 2/x$$

系数 $p_i, q_i (1 \leq i \leq 7)$ 见函数过程 BESSK0.

(3) 计算 $I_1(x)$.

当 $|x| < 3.75$ 时,

$$I_1(x) \approx x(p_1 + p_2 y + p_3 y^2 + p_4 y^3 + p_5 y^4 + p_6 y^5 + p_7 y^6)$$

当 $|x| \geq 3.75$ 时,

$$I_1(x) \approx \left(\frac{e^x}{\sqrt{|x|}} \right) \cdot (q_1 + q_2 z + q_3 z^2 + q_4 z^3 + q_5 z^4 + q_6 z^5 + q_7 z^6 + q_8 z^7 + q_9 z^8)$$

其中

$$y = (x/3.75)^2, \quad z = 3.75/|x|$$

系数 $p_i (1 \leq i \leq 7), q_i (1 \leq i \leq 9)$ 见函数过程 BESSI1.

(4) 计算 $K_1(x)$.

当 $0 < x \leq 2$ 时,

$$K_1(x) \approx -I_1(x) \ln z + 2z(p_1 + p_2 y + p_3 y^2 + p_4 y^3 + p_5 y^4 + p_6 y^5 + p_7 y^6)$$

当 $2 < x < +\infty$ 时,

$$K_1(x) \approx \left(\frac{e^{-x}}{\sqrt{x}} \right) (q_1 + q_2 z + q_3 z^2 + q_4 z^3 + q_5 z^4 + q_6 z^5 + q_7 z^6)$$

其中

$$y = x^2/4, \quad z = 2/x$$

系数 $p_i, q_i (1 \leq i \leq 7)$ 见函数过程 BESSK1.

(5) 计算 $K_N(x)$ ($x > 0, N \geq 2$).

$K_N(x)$ 由 $K_0(x), K_1(x)$ 利用下面递推公式计算:

$$K_{N+1}(x) = \frac{2N}{x} K_N(x) + K_{N-1}(x)$$

(6) $I_N(x)$ ($x > 0, N \geq 2$) 用逆递推法计算:

对给定的 x , 选定适当的正偶数 $M > N$, 令

$$I_{M+1}^* = 0, \quad I_M^* = 1$$

以递推公式

$$I_{N-1}^* = \frac{2n}{x} I_N^* + I_{N+1}^*, \quad N = M, M-1, \dots, 1$$

计算出 $I_{M-1}^*, I_{M-2}^*, \dots, I_1^*, I_0^*$. 得到 $I_N^*(x)$ 后再用 BESSIO 进行规范化得到 $I_N(x)$.

3. 使用说明

(1) BESSIO (X)

X 实自变量, 输入参数

(2) BESSK0 (X)

X 实自变量, 输入参数

该函数过程要调用函数过程 BESSIO

(3) BESSI1 (X)

X 实自变量, 输入参数

(4) BESSK1 (X)

X 实自变量, 输入参数

该函数过程要调用函数过程 BESSI1.

(5) BESSK (N, X)

X 实自变量, 输入参数

N 整型变量, 输入参数, 第二类变形贝塞尔函数的阶, $N \geq 2$

该函数过程要调用函数过程 BESSK0, BESSK1.

(6) BESSI (N, X)

X 实自变量, 输入参数

N 整型变量, 输入参数, 第一类变形贝塞尔函数的阶, $N \geq 2$

该函数过程要调用函数过程 BESSIO.

4. 过程

(1) 函数过程 BESSIO.

```
FUNCTION bessio(x)
```

```
REAL bessio, x
```

```
REAL ax
```

```
SAVE p1, p2, p3, p4, p5, p6, p7, q1, q2, q3, q4, q5, q6, &
```

```
q7, q8, q9
```

```

DATA p1,p2,p3,p4,p5,p6,p7/1.0d0,3.5156229d0,&.
      3.0899424d0,1.2067492d0,0.2659732d0,0.360768d-1,&.
      0.45813d-2/
DATA q1,q2,q3,q4,q5,q6,q7,q8,q9/0.39894228d0,&.
      0.1328592d-1,0.225319d-2,-0.157565d-2,&.
      0.916281d-2,-0.2057706d-1,0.2635537d-1,&.
      -0.1647633d-1,0.392377d-2/
if (abs(x)<3.75) then
  y=(x/3.75) * * 2
  bess10=p1+y*(p2+y*(p3+y*(p4+y*(p5+y*(p6+y*p7))))
else
  ax=abs(x)
  y=3.75/ax
  bess10=(exp(ax)/sqrt(ax))* (q1+y*(q2+y*(q3+y*(q4+y*&.
      (q5+y*(q6+y*(q7+y*(q8+y*q9)))))))
endif
END FUNCTION bess10

```

(2) 函数过程 BESSK0.

```

FUNCTION bessk0(x)
REAL bessk0,x
! USES bess10
REAL bess10
DOUBLE PRECISION p1,p2,p3,p4,p5,p6,p7,&.
      q1,q2,q3,q4,q5,q6,q7,y
SAVE p1,p2,p3,p4,p5,p6,p7,q1,q2,q3,q4,q5,q6,q7
DATA p1,p2,p3,p4,p5,p6,p7/-0.57721566d0,0.42278420d0,&.
      0.23069756d0,0.3488590d-1,0.262698d-2,0.10750d-3,0.74d-5/
DATA q1,q2,q3,q4,q5,q6,q7/1.25331414d0,-0.7832358d-1,&.
      0.2189568d-1,-0.1062446d-1,0.587872d-2,-0.251540d-2,&.
      0.53208d-3/
if (x<=2.0) then
  y=x*x/4.0
  bessk0=(-log(x/2.0)*bess10(x)-(p1+y*(p2+y*(p3+y*&.
      (p4+y*(p5+y*(p6+y*p7))))))
else
  y=(2.0/x)

```

```

      bessk0=(exp(-x)/sqrt(x))* (q1+y * (q2+y * (q3+y * (q4+y * &
          (q5+y * (q6+y * q7))))))
endif
END FUNCTION bessk0

```

(3) 函数过程 BESSJ1.

```

FUNCTION bessj1(x)
REAL bessj1,x
REAL ax
DOUBLE PRECISION p1,p2,p3,p4,p5,p6,p7,q1,q2,&
          q3,q4,q5,q6,q7,q8,q9,y
SAVE p1,p2,p3,p4,p5,p6,p7,q1,q2,q3,q4,q5,q6,q7,q8,q9
DATA p1,p2,p3,p4,p5,p6,p7/0.5d0,0.87890594d0,&
    0.51498869d0,0.15084934d0,0.2658733d-1,&
    0.301532d-2,0.32411d-3/
DATA q1,q2,q3,q4,q5,q6,q7,q8,q9/0.39894228d0,&
    -0.3988024d-1,-0.362018d-2,0.163801d-2,&
    -0.1031555d-1,0.2282967d-1,-0.2895312d-1,&
    0.1787654d-1,-0.420059d-2/
if (abs(x)<3.75) then
    y=(x/3.75)* * 2
    bessj1= x * (p1+y * (p2+y * (p3+y * (p4+y * (p5+y * (p6+y * p7))))))
else
    ax=abs(x)
    y=3.75/ax
    bessj1=(exp(ax)/sqrt(ax))* (q1+y * (q2+y * (q3+y * (q4+y * &
        (q5+y * (q6+y * (q7+y * (q8+y * q9))))))
    if(x<0.)bessj1=-bessj1
endif
END FUNCTION bessj1

```

(4) 函数过程 BESSK1.

```

FUNCTION bessk1(x)
REAL bessk1,x
! USES bessj1
REAL bessj1

```



```

DOUBLE PRECISION p1,p2,p3,p4,p5,p6,p7,&
               q1,q2,q3,q4,q5,q6,q7,y
SAVE p1,p2,p3,p4,p5,p6,p7,q1,q2,q3,q4,q5,q6,q7
DATA p1,p2,p3,p4,p5,p6,p7/1.0d0,0.15443144d0,&
     -0.67278579d0,-0.18156897d0,-0.1919402d-1,&
     -0.110404d-2,-0.4686d-4/
DATA q1,q2,q3,q4,q5,q6,q7/1.25331414d0,0.23498619d0,&
     -0.3655620d-1,0.1504268d-1,-0.780353d-2,&
     0.325614d-2,-0.68245d-3/
if (x<=2.0) then
  y=x*x/4.0
  bessk1=(log(x/2.0)*bessil(x))+ (1.0/x)*(p1+y*&
      (p2+y*(p3+y*(p4+y*(p5+y*(p6+y*p7))))))
else
  y=2.0/x
  bessk1=(exp(-x)/sqrt(x))*(q1+y*(q2+y*(q3+y*(q4+&
      y*(q5+y*(q6+y*q7))))))
endif
END FUNCTION bessk1

```

(5) 函数过程 BESSK.

```

FUNCTION bessk(n,x)
INTEGER n
REAL bessk,x
! USES bessk0,bessk1
INTEGER j
REAL bk,bkm,bkp,tox,bessk0,bessk1
if (n<2) pause 'bad argument n in bessk'
tox=2.0/x
bkm=bessk0(x)
bk=bessk1(x)
do j=1,n-1
  bkp=bkm+j*tox*bk
  bkm=bk
  bk=bkp
end do
bessk=bk

```

```
END FUNCTION bessk
```

(6) 函数过程 BESSI.

```
FUNCTION bess(n,x)
INTEGER n,IACC
REAL bess,x,BIGNO,BIGNI
PARAMETER (IACC=40,BIGNO=1.0e10,BIGNI=1.0e-10)
! USES bess0
INTEGER j,m
REAL bi,bim,bip,tox,bess0
if (n<2) pause 'bad argument n in bess'
if (x==0.) then
    bess=0.
else
    tox=2.0/abs(x)
    bip=0.0
    bi=1.0
    bess=0.
    m=2*((n+int(sqrt(float(IACC*n))))))
    do j=m,1,-1
        bim=bip+float(j)*tox*bi
        bip=bi
        bi=bim
        if (abs(bi)>BIGNO) then
            bess=bess*BIGNI
            bi=bi*BIGNI
            bip=bip*BIGNI
        endif
        if (j==n) bess=bip
    end do
    bess=bess*bess0(x)/bi
    if (x<0. .and. mod(n,2) /= 1) bess=-bess
endif
END FUNCTION bess
```

5. 例子

(1) 验证 BESSI0 的程序 D4R20 如下:

```

PROGRAM D4R20
! Driver for routine BESSI0
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Modified Bessel Function I0') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T7,A,T19,A,T35,A)')&
  'X','Actual','BESSI0(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE
  WRITE(*,'(1X,F8.2,2E18.7)') X,VALUE,BESSI0(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function I0

X	Actual	BESSI0(X)
0.00	0.1000000E+01	0.1000000E+01
0.20	0.1010025E+01	0.1010025E+01
0.40	0.1040402E+01	0.1040402E+01
0.60	0.1092045E+01	0.1092045E+01
0.80	0.1166515E+01	0.1166515E+01
1.00	0.1266066E+01	0.1266066E+01
1.20	0.1393726E+01	0.1393726E+01
1.40	0.1553395E+01	0.1553395E+01
1.60	0.1749981E+01	0.1749981E+01
1.80	0.1989559E+01	0.1989559E+01
2.00	0.2279585E+01	0.2279585E+01
2.50	0.3289839E+01	0.3289839E+01

3.00	0.4880793E-01	0.4880793E-01
3.50	0.7378203E+01	0.7378203E+01
4.00	0.1130192E+02	0.1130192E+02
4.50	0.1748117E+02	0.1748117E+02
5.00	0.2723987E+02	0.2723987E+02
6.00	0.6723441E+02	0.6723441E+02
8.00	0.4275641E+03	0.4275641E+03
10.00	0.2815717E+04	0.2815717E+04

(2) 验证 BESSK0 的程序 D4R21 如下:

```

PROGRAM D4R21
! Driver for routine BESSK0
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Modified Bessel Function K0') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T7,A,T19,A,T33,A)') &
  'X', 'Actual', 'BESSK0(X)'
DO I=1,NVAL
  READ(5, *) X,VALUE
  WRITE(*, '(1X,F8.2,2E18.7)') X,VALUE,BESSK0(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function K0

X	Actual	BESSK0(X)
0.10	0.2427069E+01	0.2427069E+01
0.20	0.1752704E+01	0.1752704E+01

0.40	0.1114529E+01	0.1114529E+01
0.60	0.7775221E+00	0.7775220E+00
0.80	0.5653471E+00	0.5653470E+00
1.00	0.4210244E+00	0.4210244E+00
1.20	0.3185082E+00	0.3185082E+00
1.40	0.2436551E+00	0.2436551E+00
1.60	0.1879548E+00	0.1879548E+00
1.80	0.1459314E+00	0.1459314E+00
2.00	0.1138939E+00	0.1138939E+00
2.50	0.6234755E-01	0.6234755E-01
3.00	0.3473950E-01	0.3473950E-01
3.50	0.1959890E-01	0.1959890E-01
4.00	0.1115968E-01	0.1115968E-01
4.50	0.6399857E-02	0.6399857E-02
5.00	0.3691098E-02	0.3691098E-02
6.00	0.1243994E-02	0.1243994E-02
8.00	0.1464707E-03	0.1464707E-03
10.00	0.1778006E-04	0.1778006E-04

(3) 验证 BESS11 的程序 D4R22 如下:

```

PROGRAM D4R22
! Driver for routine BESS11
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT.TEXT/='Modified Bessel Function 11') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T7,A,T19,A,T35,A)')&
  'X','Actual','BESS11(X)'
DO I=1,NVAL
  READ(5,*) X,VALUE

```

```

      WRITE(*, '(1X,F8.2,2E18.7)') X,VALUE,BESSI1(X)
    END DO
  CLOSE(5)
END

```

计算结果如下：

Modified Bessel Function I1

X	Actual	BESSI1(X)
0.00	0.0000000E+00	0.0000000E+00
0.20	0.1005008E+00	0.1005008E+00
0.40	0.2040267E+00	0.2040268E+00
0.60	0.3137040E+00	0.3137040E+00
0.80	0.4328648E+00	0.4328648E+00
1.00	0.5651591E+00	0.5651591E+00
1.20	0.7146779E+00	0.7146780E+00
1.40	0.8860919E+00	0.8860919E+00
1.60	0.1084811E+01	0.1084811E+01
1.80	0.1317167E+01	0.1317167E+01
2.00	0.1590637E+01	0.1590637E+01
2.50	0.2516716E+01	0.2516716E+01
3.00	0.3953370E+01	0.3953370E+01
3.50	0.6205835E+01	0.6205835E+01
4.00	0.9759465E+01	0.9759465E+01
4.50	0.1538922E+02	0.1538922E+02
5.00	0.2433564E+02	0.2433564E+02
6.00	0.6134194E+02	0.6134194E+02
8.00	0.3998731E+03	0.3998731E+03
10.00	0.2670988E+04	0.2670988E+04

(4) 验证 BESSK1 的程序 D4R23 如下：

```

PROGRAM D4R23
! Driver for routine BESSK1
CHARACTER TEXT * 27

```

```

OPEN(5,FILE 'D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT/='Modified Bessel Function K1') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T7,A,T19,A,T35,A)') &
  'X', 'Actual', 'BESSK1(X)'
DO I=1,NVAL
  READ(5, *) X,VALUE
  WRITE(*, '(1X,F8.2,2E18.7)') X,VALUE,BESSK1(X)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function K1

X	Actual	BESSK1(X)
0.10	0.9853845E+01	0.9853845E+01
0.20	0.4775972E+01	0.4775972E+01
0.40	0.2184354E+01	0.2184354E+01
0.60	0.1302835E+01	0.1302835E+01
0.80	0.8617817E+00	0.8617816E+00
1.00	0.6019073E+00	0.6019073E+00
1.20	0.4345924E+00	0.4345924E+00
1.40	0.3208359E+00	0.3208359E+00
1.60	0.2406339E+00	0.2406339E+00
1.80	0.1826231E+00	0.1826231E+00
2.00	0.1398659E+00	0.1398659E+00
2.50	0.7389081E-01	0.7389081E-01
3.00	0.4015643E-01	0.4015643E-01
3.50	0.2223939E-01	0.2223939E-01
4.00	0.1248350E-01	0.1248350E-01

4.50	0.7078095E-02	0.7078095E-02
5.00	0.4044613E-02	0.4044613E-02
6.00	0.1343920E-02	0.1343920E-02
8.00	0.1553692E-03	0.1553692E-03
10.00	0.1864877E-04	0.1864877E-04

(5) 验证 BESSK 的程序 D4R24 如下:

```

PROGRAM D4R24
! Driver for routine BESSK
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Modified Bessel Function Kn') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
WRITE(*, '(1X,T5,A,T14,A,T25,A,T41,A)') &
  'N', 'X', 'Actual', 'BESSK(N,X)'
DO I=1,NVAL
  READ(5, *) N,X,VALUE
  WRITE(*, '(1X,I4,F10.2,2E18.6)') N,X,VALUE,BESSK(N,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function Kn

N	X	Actual	BESSK(N,X)
2	0.20	0.495124E+02	0.495124E+02
2	1.00	0.162484E+01	0.162484E+01
2	2.00	0.253760E+00	0.253760E+00
2	2.50	0.121460E+00	0.121460E+00
2	3.00	0.615105E-01	0.615105E-01

2	5.00	0.530894E--02	0.530894E- 02
2	10.00	0.215098E-04	0.215098E-04
2	20.00	0.632954E--09	0.632954E-09
3	1.00	0.710126E+01	0.710126E+01
3	2.00	0.647385E+00	0.647385E+00
3	5.00	0.829177E-02	0.829177E-02
3	10.00	0.272527E--04	0.272527E-04
3	50.00	0.372794E-22	0.372794E-22
5	1.00	0.360961E+03	0.360961E+03
5	2.00	0.943105E+01	0.943105E+01
5	5.00	0.327063E-01	0.327063E-01
5	10.00	0.575418E-04	0.575419E-04
5	50.00	0.436718E-22	0.436718E-22
10	1.00	0.180713E+09	0.180713E+09
10	2.00	0.162482E+06	0.162482E+06
10	5.00	0.975856E+01	0.975856E+01
10	10.00	0.161426E-02	0.161426E-02
10	50.00	0.915099E-22	0.915099E-22
20	1.00	0.629437E+23	0.629437E+23
20	2.00	0.577086E+17	0.577086E+17
20	5.00	0.482700E+09	0.482700E+09
20	10.00	0.178744E+03	0.178744E+03
20	50.00	0.170615E-20	0.170615E-20

(6) 验证 BESSI 的程序 D4R25 如下:

```

PROGRAM D4R25
! Driver for routine BESSI
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Modified Bessel Function In') EXIT
END DO

```

```

READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T14,A,T25,A,T41,A)')&
      'N','X','Actual','BESSI(N,X)'
DO I=1,NVAL
  READ(5,*) N,X,VALUE
  WRITE(*,'(1X,I4,F10.2,2E18.6)') N,X,VALUE,BESSI(N,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function In

N	X	Actual	BESSI(N,X)
2	0.20	0.501669E-02	0.501669E-02
2	1.00	0.135748E+00	0.135748E+00
2	2.00	0.688948E+00	0.688948E+00
2	2.50	0.127647E+01	0.127647E+01
2	3.00	0.224521E+01	0.224521E+01
2	5.00	0.175056E+02	0.175056E+02
2	10.00	0.228152E+04	0.228152E+04
2	20.00	0.393128E+08	0.393128E+08
3	1.00	0.221684E-01	0.221684E-01
3	2.00	0.212740E+00	0.212740E+00
3	5.00	0.103312E+02	0.103312E+02
3	10.00	0.175838E+02	0.175838E+04
3	50.00	0.267776E+21	0.267776E+21
5	1.00	0.271463E-03	0.271463E-03
5	2.00	0.982568E-02	0.982568E-02
5	5.00	0.215797E+01	0.215797E+01
5	10.00	0.777188E+03	0.777188E+03
5	50.00	0.227855E+21	0.227855E+21
10	1.00	0.275295E-09	0.275295E-09

10	2.00	0.301696E-06	0.301696E-06
10	5.00	0.458004E-02	0.458004E-02
10	10.00	0.218917E+02	0.218917E+02
10	50.00	0.107160E+21	0.107160E+21
20	1.00	0.396684E-24	0.396684E-24
20	2.00	0.431056E-18	0.431056E-18
20	5.00	0.502424E-10	0.502424E-10
20	10.00	0.125080E-03	0.125080E-03
20	50.00	0.544201E+19	0.544201E+19

4.6 分数阶第一类贝塞尔函数和变形贝塞尔函数

1. 功能

ν 阶(ν 为实数)第一类贝塞尔函数定义为

$$J_{\nu}(x) = \left(\frac{1}{2}x\right)^{\nu} \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{4}x^2\right)^k}{k! \Gamma(\nu + k + 1)}$$

ν 阶第一类变形贝塞尔函数定义为

$$I_{\nu}(x) = i^{-\nu} J_{\nu}(ix) = e^{-\frac{\nu\pi i}{2}} J_{\nu}(ix) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{\Gamma(\nu + k + 1)}$$

本算法用于计算一组第一类分数阶的贝塞尔函数 $J_{a+n}(x)$ 和一组第一类分数阶变形贝塞尔函数 $I_{a+n}(x)$ 的值,使其具有用户所需的精度.其中自变量 $x > 0$, a 为需要计算的贝塞尔函数的阶的分数部分,当 $n=0,1,2,\dots,n_{\max}$ 时, $0 \leq a < 1$,当 $n=-1,-2,\dots,-n_{\max}$ 时, $0 < a < 1$, n_{\max} 为需要计算的贝塞尔函数的阶的整数部分的上限.本算法对小的 x 值或适当大的 x 值较为有效.

2. 方法

本算法使用计算贝塞尔函数的递推法.

对于真分数 a ,有公式

$$\sum_{k=0}^{\infty} \frac{(a+2k)\Gamma(a+k)}{k!} \left(\frac{x}{2}\right)^{-a} J_{a+2k} = 1$$

设 J_l^* 是柱函数,引进记号

$$K = \sum_{l=0}^N \frac{(a+2l)\Gamma(a+l)}{l!} \left(\frac{x}{2}\right)^{-a} J_{a+2l}^*$$

$$B = \left(\frac{x}{2}\right)^{-a} \Gamma(a+1)/N!$$

$$A_N = 1, \quad A_l = \frac{A_{l+1}(l+1)}{a+l} \quad (l = N-1, N-2, \dots, 1, 0)$$

则

$$\begin{aligned} K &= B \sum_{l=0}^N (a+2l) A_l J_{a+2l}^* \\ &= \frac{\Gamma(1+a)}{A_1} \left(\frac{x}{2}\right)^{-a} \sum_{l=0}^N (a+2l) A_l J_{a+2l}^* \end{aligned}$$

J_{a+n} 的计算步骤如下:

(1) 选择适当小的 a 和适当大的 $p = a + n_{\max} + L > x$, 其中 L 为整数, 并引进辅助函数 $J_n^*(x)$. 令

$$J_p^*(x) = 0, \quad J_{p-1}^*(x) = a \neq 0$$

(2) 用柱函数的递推关系式

$$J_{\nu-1}^* = \frac{2\nu}{x} J_{\nu}^* - J_{\nu+1}^*$$

逆递推得到序列 $J_{p-2}^*, J_{p-3}^*, \dots, J_1^*, J_0^*$.

(3) 令 $N_1 = \left[\frac{p}{2}\right]$, 其中 $\left[\frac{p}{2}\right]$ 表示 $\frac{p}{2}$ 的整数部分, 则有

$$A_{N_1} = 1, \quad A_l = \frac{A_{l+1}(l+1)}{a+l} \quad (l = N_1-1, N_1-2, \dots, 1, 0)$$

$$K = \frac{\Gamma(1+a)}{A_1} \left(\frac{x}{2}\right)^{-a} \sum_{l=0}^{N_1} (a+2l) A_l J_{a+2l}^*$$

(4) 计算

$$J_{a+l}(x) = J_{a+l}^*/K \quad (l = 0, 1, \dots, n_{\max})$$

(5) 给 p 以增量 5, 重复步骤 (1)~(4), 直至相邻两次求得的 $J_{a+l}(x)$ 的值的相对误差小于给定的精度要求.

计算 $J_{a+n}(x)$ 的步骤完全与此类似.

3. 使用说明

(1) BESJAN (X, A, NM, IH, F)

X 实型变量, 输入参数, 自变量

A 实型变量, 输入参数, 阶的分数部分

NM 整型变量, 输入参数, 阶的整数部分的上界

IH 整型变量, 输入参数, $IH=1$ 时计算 $J_{a+n}(x)$, $IH=-1$ 时, 计算

$$J_{a-n}(x)$$

F 实型一维数组,输出参数,存放一组 $J_{a+n}(x)$

(2) BESJAN (X,A,NM,IH,F)

X,A,NM 与 BESSJAN 中相同

IH 整型变量,输入参数, $IH=1$ 时计算 $I_{a+n}(x)$, $IH=-1$ 时,计算 $I_{a-n}(x)$

F 实型一维数组,输出参数,存放一组 $I_{a+n}(x)$

4. 过程

(1) 子过程 BESJAN.

```

SUBROUTINE besjan(x,a,nm,ih,f)
  DIMENSION f(0:nm)
  if (ih==1) then
    call jap(x,a,nm,f)
    return
  else
    call jap(x,a,1,f)
    f(1)=2. * a * f(0)/x - f(1)
    do i=1,nm-1
      f(i+1)=2. * (a-i) * f(i)/x - f(i-1)
    end do
    return
  endif
END SUBROUTINE besjan

SUBROUTINE jap(x,a,nmax,f)
  PARAMETER (nmax=10,eps=0.0005)
  DIMENSION f(0:nmax),fa(0:mmax),rr(0:mmax)
  ! USE gammln
  do n=0,nmax
    fa(n)=0.0
  end do
  sum=exp(gammln(1.+a))
  sum=exp(a * log(x/2.))/sum
  d1=2.3026 * e + 1.3863
  if(nmax>0) then

```

```

      r=t(0.5 * d1/nmax) * nmax
    else
      r=0.0
    endif
    s=t(0.73576 * d1/x) * 1.3591 * x
    if(r<=s) then
      nu=1 + int(s)
    else
      nu=1 + int(r)
    endif
    do
      m=0
      al=1.
      li=int(nu/2)
      do
        m=m+1
        al=al * (m+a)/(m+1)
        if(.not. m<li) exit
      end do
      n=2 * m
      r=0.0
      s=0.0
      do
        r=1./(2. * (a+n)/x-r)
        if(int(n/2)/=float(n)/2.) then
          am=0.0
        else
          al=al * (n+2)/(n+2. * a)
          am=al * (n+a)
        endif
        s=r * (am+s)
        if(n<=nmax) rr(n-1)=r
        n=n-1
        if(.not. n>=1) exit
      end do
      f(0)=sum/(1. +s)
      do n=0,nmax-1

```

```

      f(n+1)=rr(n)*f(n)
    end do
    do n=0,nmax
      aaa=1.
      if(abs((f(n)-fa(n))/f(n))>eps) then
        aaa=-1.
        do m=0,nmax
          fa(m)=f(m)
        end do
        nu=nu+5
        if(.not.aaa>0) exit
      endif
    end do
    if(.not.aaa== -1.) exit
  end do
END SUBROUTINE jap

```

```

FUNCTION t(y)
  if(y<=10.) then
    t=((((0.000057941*y-0.00176148)*y+0.0208645)*y-&
      0.129013)*y+0.85777)*y+1.0125
  else
    z=log(y)-0.775
    p=(0.775-log(z))/(1.+z)
    t=y/(p+1.)/z
  endif
END FUNCTION t

```

(2) 子过程 BESIAN.

```

SUBROUTINE besian(x,a,nm,ih,f)
  DIMENSION f(0:nm)
  if (ih==1) then
    call iap(x,a,nm,f)
    return
  else
    call iap(x,a,1,f)
    f(1)=2.*a*f(0)/x+f(1)
  end if

```

```

      do i=1,nm-1
        f(i+1)=2.* (a-i) * f(i)/x+f(i-1)
      end do
      return
    endif
  END SUBROUTINE besian
  SUBROUTINE iap(x,a,nmax,f)
    ! USE gammln
    PARAMETER (mmax=10,eps=0.00005)
    DIMENSION f(0:nmax),fa(0:mmax),rr(0:mmax)
    do n=0,nmax
      fa(n)=0.0
    end do
    sum=exp(gammln(1.+a))
    sum=exp(a*log(x/2.)) * exp(x)/sum
    d1=2.3026 * e+1.3863
    if(nmax>0) then
      r=t(0.5 * d1/nmax) * nmax
    else
      r=0.0
    endif
    if(x<d1) then
      s=t(0.73576 * (d1-x)/x) * 1.3591 * x
    else
      s=1.3591 * x
    endif
    if(r<=s) then
      nu=1-int(s)
    else
      nu=1+int(r)
    endif
    do
      n=n+1
      al=1.
      do
        n=n+1
        al=al * (n+2. * a)/(n+1.)
      end do
    end do
  end do
end do

```



```

        if (.not. n<nu) exit
    end do
    r=0.0
    s=0.0
    do
        r=1./(2.*(a+n)/x+r)
        a1=a1*(n+1.)/(n+2.*a)
        am=2.*(n+a)*a1
        s=r*(am+s)
        if (n<=nmax) rr(n+1)=r
        n=n+1
        if (.not. n>-1) exit
    end do
    f(0)=sum/(1.+s)
    do n=0,nmax-1
        f(n+1)=rr(n)*f(n)
    end do
    do n=0,nmax
        aaa=1.
        if(abs((f(n)-fa(n))/f(n))>eps) then
            aaa=-1.
            do m=0,nmax
                fa(m)=f(m)
            end do
            nu=nu+5
            if (.not. aaa>0.) exit
        endif
    end do
    if (.not. aaa== -1.) exit
end do
END SUBROUTINE iap
FUNCTION t(y)
if(y<=10.) then
    t=((((0.000057941*y-0.00176148)*y+0.0208645)*y-&
    0.129013)*y+0.85777)*y+1.0125
else
    z=log(y)-0.775

```

```

      p=(0.775-log(z))/(1.+z)
      t=y/(p+1.)/z
endif
END FUNCTION t

```

5. 例子

(1) 用以控制精度的 ϵ , 存放在 BESJAN 中的子过程 JAP 中. 验证程序 D4R26 如下:

```

PROGRAM D4R26
! Driver for routine BESJAN
DIMENSION VALUE(0:18),F(0:18),N(0:18),A(0:18),X(0:18)
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5,'(A)') TEXT
  IF (.NOT. TEXT/='Bessel Function Ja+n') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
DO I=0,NVAL-1
  READ(5,*) N(I),A(I),X(I),VALUE(I)
END DO
IH=1
WRITE(*,'(1X,T5,A,T10,A,T15,A,T25,A,T35,A,T50,A)')&
  'N','A','IH','X','Actual','BESJAN'
CALL BESJAN(X(0),A(0),4,IH,F)
DO I=0,4
  WRITE(*,'(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I),A(I),&
    IH,X(I),VALUE(I),F(I)
END DO
CALL BESJAN(X(5),A(5),4,IH,F)
DO I=5,9
  WRITE(*,'(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I),A(I),&
    IH,X(I),VALUE(I),F(I-5)
END DO
IH=-1

```

```

CALL BESJAN(X(10),A(10),4,IH,F)
DO I=10,13
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I),A(I),&
    IH,X(I),VALUE(I),F(I-9)
END DO
CALL BESJAN(X(14),A(14),4,IH,F)
DO I=14,17
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I),A(I),&
    IH,X(I),VALUE(I),F(I-13)
END DO
CLOSE(5)
END

```

计算结果如下:

Bessel Function $J_{\alpha+n}$

N	A	IH	X	Actual	BESJAN
0	0.5	1	0.500000	0.540974E+00	0.540974E+00
1	0.5	1	0.500000	0.917017E-01	0.917017E-01
2	0.5	1	0.500000	0.923641E-02	0.923641E-02
3	0.5	1	0.500000	0.662379E-03	0.662379E-03
4	0.5	1	0.500000	0.368921E-04	0.368921E-04
0	0.5	1	3.500000	-0.149605E+00	-0.149605E+00
1	0.5	1	3.500000	0.356643E+00	0.356643E+00
2	0.5	1	3.500000	0.455298E+00	0.455298E+00
3	0.5	1	3.500000	0.293783E+00	0.293783E+00
4	0.5	1	3.500000	0.132269E+00	0.132269E+00
-1	0.5	-1	0.500000	0.990246E+00	0.990246E+00
-2	0.5	-1	0.500000	-0.252147E+01	-0.252147E+01
-3	0.5	-1	0.500000	0.141385E+02	0.141385E+02
-4	0.5	-1	0.500000	-0.138864E+03	-0.138864E+03
-1	0.5	-1	3.500000	-0.399387E+00	-0.399387E+00
-2	0.5	-1	3.500000	0.263715E+00	0.263715E+00
-3	0.5	-1	3.500000	0.173345E+00	0.173345E+00

-4 0.5 -1 3.500000 -0.511351E+00 -0.511351E+00

(2) 用以控制精度的 eps, 存放在 BESIAN 中的子过程 IAP 中. 验证程序 D4R27 如下:

```

PROGRAM D4R27
! Driver for routine BESIAN
DIMENSION VALUE(0:18),F(0:18),N(0:18),A(0:18),X(0:18)
CHARACTER TEXT * 30
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT/='Modified Bessel Function Ia+n') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT
DO I=0, NVAL-1
  READ(5, *) N(I), A(I), X(I), VALUE(I)
END DO
IH=1
WRITE(*, '(1X,T5,A,T10,A,T15,A,T25,A,T35,A,T50,A)') &
  'N', 'A', 'IH', 'X', 'Actual', 'BESIAN'
CALL BESIAN(X(0), A(0), 3, IH, F)
DO I=0, 3
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I), A(I), &
    IH, X(I), VALUE(I), F(I)
END DO
CALL BESIAN(X(4), A(4), 3, IH, F)
DO I=4, 7
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I), A(I), &
    IH, X(I), VALUE(I), F(I-4)
END DO
IH=-1
CALL BESIAN(X(8), A(8), 3, IH, F)
DO I=8, 10
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I), A(I), &
    IH, X(I), VALUE(I), F(I-7)

```

```

END DO
CALL BESIAN(X(11),A(11),3,IH,F)
DO I=11,13
  WRITE(*, '(3X,I2,F6.1,3X,I2,F12.6,2E15.6)') N(I),A(I),&
    IH,X(I),VALUE(I),F(I-10)
END DO
CLOSE(5)
END

```

计算结果如下:

Modified Bessel Function Ia + n

N	A	IH	X	Actual	BESIAN
0	0.5	1	0.200000	0.359208E+00	0.359208E+00
1	0.5	1	0.200000	0.238836E-01	0.238836E-01
2	0.5	1	0.200000	0.954254E-03	0.954255E-03
3	0.5	1	0.200000	0.272471E-04	0.272471E-04
0	0.5	1	1.000000	0.937675E+00	0.937675E+00
1	0.5	1	1.000000	0.293525E+00	0.293525E+00
2	0.5	1	1.000000	0.570989E-01	0.570989E-01
3	0.5	1	1.000000	0.803078E-02	0.803078E-02
-1	0.5	-1	0.200000	0.181993E+01	0.181993E+01
-2	0.5	-1	0.200000	-0.874042E+01	-0.874042E+01
-3	0.5	-1	0.200000	0.132926E+03	0.132926E+03
-1	0.5	-1	1.000000	0.123120E+01	0.123120E+01
-2	0.5	-1	1.000000	-0.293525E+00	-0.293525E+00
-3	0.5	-1	1.000000	0.211178E+01	0.211178E+01

4.7 指数积分和定指数积分

1. 功能

本节有三个过程.

(1) 函数过程 EX 计算实指数积分

$$E_1(x) = -E_0(-x) = \int_x^{\infty} (e^{-u}/u) du, \quad x > 0$$

的值.

(2) 函数过程 AS 计算定指数积分

$$A_{n-1}(b) = \int_1^{\infty} x^{n-1} e^{(-bx)} dx$$

的值.

(3) 函数过程 BS 计算定指数积分

$$B_{n-1}(a) = \int_{-1}^{+1} x^{n-1} e^{(-ax)} dx$$

的值.

2. 方法

(1) 计算实指数积分 $E_1(x)$.

当 $0 < x < 1$ 时采用逼近公式:

$$-E_1(-x) + \ln(x) \approx R_6 + R_5x + R_4x^2 + R_3x^3 + R_2x^4 + R_1x^5$$

当 $1 \leq x < \infty$ 时采用逼近公式:

$$-E_1(-x) = \frac{e^{-x}}{x} \left\{ \frac{T_4 + T_3x + T_2x^2 + T_1x^3 + x^4}{S_4 + S_3x + S_2x^2 + S_1x^3 + x^4} \right\}$$

其中 $R_6, \dots, R_1, T_4, \dots, T_1, S_4, \dots, S_1$ 的值见函数过程 EX.

(2) 计算定指数积分 $A_n(b)$ 的公式为:

$$A_0(b) = e^{-b}/b, \quad A_n(b) = A_0(b) + (n/b)A_{n-1}(b)$$

(3) 计算定指数积分 $B_n(a)$ 的公式为: 当 $|a| < 8$ 时, 采用 $e^{(-ax)}$ 的展开式计算; 否则, 采用公式

$$B_0(a) = 2\sinh(a)/a, \quad B_n(a) = [(-1)^n e^a - e^{-a} + nB_{n-1}(a)]/a$$

计算.

3. 使用说明

(1) EX (X)

X 实型变量, 输入参数

(2) AS (N, B)

N 整型变量, 输入参数, x 的幂参数

B 实型变量, 输入参数, e 的幂参数

(3) BS(N, A)

N 整型变量, 输入参数, x 的幂参数

A 实型变量, 输入参数, e 的幂参数

4. 过程

(1) 函数过程 EX.

```

FUNCTION ex(x)
  REAL * 8 y,w,r1,r2,r3,r4,r5,r6,t1,t2,t3,t4,s1,s2,s3,s4
  DATA r1,r2,r3,r4,r5,r6/0.107857d-2,-0.976004d-2,&
    0.5519968d-1,-0.24991055d0,0.99999193d0,-0.57721566d0/
  DATA t1,t2,t3,t4/8.5733287401d0,1.8059016973d1,&
    8.6347608925d0,0.2677737343d0/
  DATA s1,s2,s3,s4/9.5733223454d0,2.56329561486d1,&
    2.10996530827d1,3.9584969228d0/
  if (x<1.) then
    ex=((((r1 * x+r2) * x+r3) * x+r4) * x+r5) * x+r6-log(x)
  else
    y=((x+t1) * x+t2) * x+t3) * x+t4
    w=((x+s1) * x+s2) * x+s3) * x+s4
    ex=(y/w)/(exp(x) * x)
  endif
END FUNCTION ex

```

(2) 函数过程 AS.

```

FUNCTION as(n,b)
  DIMENSION a(15)
  INTEGER n,i
  REAL db
  a(1)=exp(-b)/b
  if (n==1) then
    as=a(n)
  else
    db=1./b
    do i=2,n
      a(i)=a(1)+db * (i-1) * a(i-1)
    end do
    as=a(n)
  end if
END FUNCTION as

```

(3) 函数过程 BS.

```

FUNCTION bs(N,A)
DIMENSION b(20)
REAL a,del,r,eps,s,ak,a2,da,ajp,ajm,q1,q2
if (a==0.0) then
    if (n/2==float(n)/2.) then
        b(n)=0.0
        bs=b(n)
        return
    endif
    b(n)=2./n
    bs=b(n)
    return
endif
if (abs(a)<8.) then
    del=1.e-8
    if (n/2/=float(n)/2.) then
        r=2./float(n)
        eps=r*del
        s=r
        ak=0.
        a2=a*a
        do
            ak=ak+2.
            r=r*a2*(n+ak-2.)/(ak*(ak-1.)*(n+ak))
            s=s+r
            if (.not. r>eps) exit
        end do
        b(n)=s+r
        bs=b(n)
        return
    endif
    r=2.*a/(n+1.)
    omeg=abs(r*del)
    s=r
    ak=1.

```



```

a2=a * a
do
ak=ak + 2.
r=r * a2 * (n+ak-2.)/(ak * (ak-1.) * (n+ak))
s=s+r
if (.not. abs(r)>omeg) exit
end do
b(n)=- (s+r)
bs=b(n)
return
endif
da=1./a
ajp=da * exp(a)
ajm=(da * da)/ajp
b(1)=ajp-ajm
if(n==1) then
bs=b(n)
return
end if
q1=-1.
q2=1.
do m=2,n
b(m)=q1 * ajp-ajm+q2 * da * b(m-1)
q1=-q1
q2=q2+1.
end do
bs=b(n)
END FUNCTION bs

```

5. 例子

(1) 验证 EX 的程序 D4R28 如下:

```

PROGRAM D4R28
! Driver for routine EX
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF _SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO

```

```

      READ(5, '(A)') TEXT
      IF (.NOT. TEXT /= 'Exponential Integral Ex') EXIT
    END DO
    READ(5, *) NVAL
    WRITE(*, *) TEXT
    WRITE(*, '(1X, T6, A, T15, A, T31, A)') &
      'X', 'Actual', 'EX(X)'
    DO I=1, NVAL
      READ(5, *) X, VALUE
      WRITE(*, '(1X, F6.2, 2E16.7)') X, VALUE, EX(X)
    END DO
    CLOSE(5)
  END

```

计算结果如下:

Exponential Integral Ex

X	Actual	EX(X)
0.10	0.1822924E+01	0.1822924E+01
0.50	0.5597736E+00	0.5597735E+00
1.00	0.2193839E+00	0.2193839E+00
1.50	0.1000196E+00	0.1000196E+00
3.00	0.1304838E-01	0.1304838E-01
5.00	0.1148299E-02	0.1148296E-02

(2) 验证 AS 的程序 D4R29 如下:

```

PROGRAM D4R29
! Driver for routine AS
CHARACTER TEXT * 27
OPEN(5, FILE='D:\VF_SHU\DATA\FNCVAL.DAT', STATUS='OLD')
DO
  READ(5, '(A)') TEXT
  IF (.NOT. TEXT /= 'Exponential Integral As') EXIT
END DO
READ(5, *) NVAL
WRITE(*, *) TEXT

```

```

WRITE( *, '(1X,T5,A,T11,A,T21,A,T36,A)') &
    'N','B','Actual','AS(N,B)'
DO I=1,NVAL
    READ(5, *) N,B,VALUE
    WRITE( *, '(1X,I4,2X,F6.2,2E16.7)') N,B,VALUE,AS(N,B)
END DO
CLOSE(5)
END

```

计算结果如下:

Exponential Integral As

N	B	Actual	AS(N,B)
1	0.25	0.3115203E+01	0.3115203E+01
3	0.25	0.1277233E+03	0.1277233E+03
5	0.25	0.2457584E+05	0.2457584E+05
7	0.25	0.1179648E+08	0.1179648E+08
9	0.25	0.1056965E+11	0.1056965E+11
11	0.25	0.1522029E+14	0.1522029E+14
1	24.00	0.1572973E-11	0.1572973E-11
3	24.00	0.1709515E-11	0.1709516E-11
5	24.00	0.1870750E-11	0.1870750E-11
7	24.00	0.2063651E-11	0.2063651E-11
9	24.00	0.2297930E-11	0.2297930E-11
11	24.00	0.2587429E-11	0.2587430E-11

(3) 验证 BS 的程序 D4R30 如下:

```

PROGRAM D4R30
! Driver for routine BS
CHARACTER TEXT * 27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO
    READ(5, '(A)') TEXT
    IF (.NOT. TEXT /= 'Exponential Integral Bs') EXIT
END DO

```

```

READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T10,A,T21,A,T36,A)')&
      'N','B','Actual','BS(N,B)'
DO I=1,NVAL
  READ(5,*) N,B,VALUE
  WRITE(*,'(1X,I4,2X,F6.2,2E16.7)') N,B,VALUE,BS(N,B)
END DO
CLOSE(5)
END

```

计算结果如下:

Exponential Integral Bs

N	B	Actual	BS(N,B)
2	0.25	-0.1677107E+00	-0.1677107E+00
4	0.25	-0.1007459E+00	-0.1007459E+00
6	0.25	-0.7200876E-01	-0.7200876E-01
8	0.25	-0.5603029E-01	-0.5603029E-01
10	0.25	-0.4585627E-01	-0.4585627E-01
12	0.25	-0.3880972E-01	-0.3880972E-01
1	24.00	0.1103713E+10	0.1103713E+10
3	24.00	0.1015570E+10	0.1015570E+10
5	24.00	0.9409189E+09	0.9409188E+09
7	24.00	0.8767912E+09	0.8767912E+09
9	24.00	0.8210525E+09	0.8210525E+09
11	24.00	0.7721223E+09	0.7721223E+09

4.8 连带勒让德函数

1. 功能

计算实变元的连带勒让德(Associated Legendre)多项式函数 $P_l^m(x)$ 的值, 其中整数 m 和 l 满足 $l \geq m \geq 0$, $x \in [-1, 1]$. 利用它可对球面调和函数 $Y_{lm}(\theta, \varphi)$ 进行计算.

2. 方法

(1) 连带勒让德多项式.

连带勒让德多项式由普通勒让德多项式定义

$$P_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_l(x)$$

其中 $P_l(x)$ 为普通勒让德多项式

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l, \quad x \in [-1, 1], \quad l = 0, 1, 2, \dots$$

对于 $P_l^m(x)$ 有递推关系

$$(l-m)P_l^m = x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m$$

而对初始值有

$$P_m^m = (-1)^m (2m-1)! (1-x^2)^{m/2}$$

$$P_{m+1}^m = x(2m+1)P_m^m \quad (\text{或取 } P_{m-1}^m = 0)$$

(2) 球面调和函数 $Y_{lm}(\theta, \varphi)$.

球面调和函数 $Y_{lm}(\theta, \varphi)$ ($-l \leq m \leq l$) 是球面上的两个坐标 θ, φ 的函数, 定义为

$$Y_{lm}(\theta, \varphi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} \cdot P_l^m(\cos\theta) e^{im\varphi}$$

显然有

$$Y_{l,-m}(\theta, \varphi) = (-1)^m Y_{lm}^*(\theta, \varphi)$$

3. 使用说明

PLGNDR (L,M,X)

L 整型变量, 输入参数, 勒让德函数的次数

M 整型变量, 输入参数, 勒让德函数的阶数

X 实自变量, 输入参数

4. 过程

函数过程 PLGNDR.

```
FUNCTION plgndr(l,m,x)
```

```
INTEGER l,m
```

```
REAL plgndr,x
```

```
INTEGER i,ll
```

```

REAL fact, pll, pmm, pmmp1, somx2
if(m<0. or. m>1. or. abs(x)>1. ) pause&
      'bad arguments in plgndr'

pmm=1.
if(m>0) then
  somx2=sqrt((1.-x)*(1.+x))
  fact=1.
  do i=1,m
    pmm=-pmm*fact*somx2
    fact=fact+2.
  end do
endif
if(l==m) then
  plgndr=pmm
else
  pmmp1=x*(2*m+1)*pmm
  if(l==m+1) then
    plgndr=pmmp1
  else
    do ll=m+2,l
      pll=(x*(2*ll-1)*pmmp1-(ll+m-1)*pmm)/(ll-m)
      pmm=pmmp1
      pmmp1=pll
    end do
    plgndr=pll
  endif
endif
END FUNCTION plgndr

```

5. 例子

验证 PLGNDR 的程序 D4R31 如下：

```

PROGRAM D4R31
! Driver for routine PLGNDR
CHARACTER TEXT*27
OPEN(5,FILE='D:\VF_SHU\DATA\FNCVAL.DAT',STATUS='OLD')
DO

```

```

READ(5,'(A)') TEXT
IF (.NOT.TEXT/='Legendre Polynomials') EXIT
END DO
READ(5,*) NVAL
WRITE(*,*) TEXT
WRITE(*,'(1X,T5,A,T9,A,T20,A,T35,A,T49,A)') &
    'N','M','X','Actual','PLGNDR(N,M,X)'
DO I=1,NVAL
    READ(5,*) N,M,X,VALUE
    FAC=1.0
    IF (M>0) THEN
        DO J=N-M+1,N+M
            FAC=FAC * J
        END DO
    ENDIF
    FAC=2.0 * FAC / (2.0 * N + 1.0)
    VALUE=VALUE * SQRT(FAC)
    WRITE(*,'(1X,2I4,3E17.6)') N,M,X,VALUE,PLGNDR(N,M,X)
END DO
CLOSE(5)
END

```

计算结果如下:

Legendre Polynomials

N	M	X	Actual	PLGNDR(N,M,X)
1	0	0.100000E+01	0.100000E+01	0.100000E+01
10	0	0.100000E+01	0.100000E+01	0.100000E+01
20	0	0.100000E+01	0.100000E+01	0.100000E+01
1	0	0.707107E+00	0.707106E+00	0.707107E+00
10	0	0.707107E+00	0.115112E+00	0.115112E+00
20	0	0.707107E+00	-0.193065E+00	-0.193065E+00
1	0	0.000000E+00	0.000000E+00	0.000000E+00
10	0	0.000000E+00	-0.246094E+00	-0.246094E+00
20	0	0.000000E+00	0.176197E+00	0.176197E+00
2	2	0.707107E+00	0.150000E+01	0.150000E+01

10	2	0.707107E+00	-0.688843E+01	-0.688839E+01
20	2	0.707107E+00	0.842319E+02	0.842319E+02
2	2	0.000000E+00	0.300000E+01	0.300000E+01
10	2	0.000000E+00	0.270703E+02	0.270703E+02
20	2	0.000000E+00	-0.740027E+02	-0.740028E+02
10	10	0.707107E+00	0.204601E+08	0.204603E+08
20	10	0.707107E+00	-0.133551E+13	-0.133551E+13
10	10	0.000000E+00	0.654729E+09	0.654729E+09
20	10	0.000000E+00	-0.161205E+13	-0.161205E+13

附 录

文件 FNCVAL.DAT:

Values of Special Functions in format x,F(x) or x,y,F(x,y)

Gamma Function

17 Values

1.0	1.000000
1.2	0.918169
1.4	0.887264
1.6	0.893515
1.8	0.931384
2.0	1.000000
0.2	4.590845
0.4	2.218160
0.6	1.489192
0.8	1.164230
-0.2	-5.821149
-0.4	-3.722981
-0.6	-3.696933
-0.8	-5.738555
10.0	3.6288000E05
20.0	1.2164510E17
30.0	8.8417620E30

N-factorial

18 Values

1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1.3076735E12
20	2.4329042E18
25	1.5511222E25
30	2.6525281E32

Binomial Coefficients

20 Values

1	0	1
6	1	6
6	3	20
6	5	6
15	1	15
15	3	455
15	5	3003
15	7	6435
15	9	5005
15	11	1365
15	13	105
25	1	25
25	3	2300
25	5	53130
25	7	480700
25	9	2042975
25	11	4457400

23	13	5200300
----	----	---------

25	15	3268760
----	----	---------

25	17	1081575
----	----	---------

Beta Function

15 Values

1.0	1.0	1.000000
-----	-----	----------

0.2	1.0	5.000000
-----	-----	----------

1.0	0.2	5.000000
-----	-----	----------

0.4	1.0	2.500000
-----	-----	----------

1.0	0.4	2.500000
-----	-----	----------

0.6	1.0	1.666667
-----	-----	----------

0.8	1.0	1.250000
-----	-----	----------

6.0	6.0	3.607504E-04
-----	-----	--------------

6.0	5.0	7.936508E-04
-----	-----	--------------

6.0	4.0	1.984127E-03
-----	-----	--------------

6.0	3.0	5.952381E-03
-----	-----	--------------

6.0	2.0	0.238095E-01
-----	-----	--------------

7.0	7.0	8.325008E-05
-----	-----	--------------

5.0	5.0	1.587302E-03
-----	-----	--------------

4.0	4.0	7.142857E-03
-----	-----	--------------

3.0	3.0	0.333333E-01
-----	-----	--------------

2.0	2.0	1.666667E-01
-----	-----	--------------

Incomplete Gamma Function

20 Values

0.1	3.1622777E-02	0.7420263
-----	---------------	-----------

0.1	3.1622777E-01	0.9119753
-----	---------------	-----------

0.1	1.5811388	0.9898955
-----	-----------	-----------

0.5	7.0710678E-02	0.2931279
-----	---------------	-----------

0.5	7.0710678E-01	0.7656418
-----	---------------	-----------

0.5	3.5355339	0.9921661
-----	-----------	-----------

1.0	0.1000000	0.0951626
-----	-----------	-----------

1.0	1.0000000	0.6321206
-----	-----------	-----------

1.0	5.0000000	0.9932621
-----	-----------	-----------

1.1	1.0488088E-01	0.0757471
-----	---------------	-----------

1.1	1.0488088	0.6076457
-----	-----------	-----------

1.1	5.2440442	0.9933425
-----	-----------	-----------

2.0	1.4142136E-01	0.0091054
-----	---------------	-----------

2.0	1.4142136	0.4130643
2.0	7.0710678	0.9931450
6.0	2.4494897	0.0387318
6.0	12.247449	0.9825937
11.0	16.583124	0.9404267
26.0	25.495098	0.4863866
41.0	44.821870	0.7359709

Error Function

20 Values

0.0	0.000000
0.1	0.1124629
0.2	0.2227026
0.3	0.3286268
0.4	0.4283924
0.5	0.5204999
0.6	0.6038561
0.7	0.6778012
0.8	0.7421010
0.9	0.7969082
1.0	0.8427008
1.1	0.8802051
1.2	0.9103140
1.3	0.9340079
1.4	0.9522851
1.5	0.9661051
1.6	0.9763484
1.7	0.9837905
1.8	0.9890905
1.9	0.9927904

Incomplete Beta Function

20 Values

0.5	0.5	0.01	0.0637686
0.5	0.5	0.10	0.2048328
0.5	0.5	1.00	1.0000000
1.0	0.5	0.01	0.0050126
1.0	0.5	0.10	0.0513167
1.0	0.5	1.00	1.0000000

1.0	1.0	0.5	0.5000000
5.0	5.0	0.5	0.5000000
10.0	0.5	0.9	0.1516409
10.0	5.0	0.5	0.0897827
10.0	5.0	1.0	1.0000000
10.0	10.0	0.5	0.5000000
20.0	5.0	0.8	0.4598773
20.0	10.0	0.6	0.2146816
20.0	10.0	0.8	0.9507365
20.0	20.0	0.5	0.5000000
20.0	20.0	0.6	0.8979414
30.0	10.0	0.7	0.2241297
30.0	10.0	0.8	0.7586405
40.0	20.0	0.7	0.7001783

Bessel Function J0

20 Values

-5.0	-0.1775968
-4.0	-0.3971498
-3.0	-0.2600520
-2.0	0.2238908
-1.0	0.7651976
0.0	1.0000000
1.0	0.7651977
2.0	0.2238908
3.0	-0.2600520
4.0	-0.3971498
5.0	-0.1775968
6.0	0.1506453
7.0	0.3000793
8.0	0.1716508
9.0	-0.0903336
10.0	-0.2459358
11.0	-0.1711903
12.0	0.0476893
13.0	0.2069261
14.0	0.1710735
15.0	-0.0142245

Bessel Function Y0

15 Values

0.1	-1.5342387
1.0	0.0882570
2.0	0.51037567
3.0	0.37685001
4.0	-0.0169407
5.0	-0.3085176
6.0	-0.2881947
7.0	-0.0259497
8.0	0.2235215
9.0	0.2499367
10.0	0.0556712
11.0	-0.1688473
12.0	-0.2252373
13.0	-0.0782079
14.0	0.1271926
15.0	0.2054743

Bessel Function J1

20 Values

-5.0	0.3275791
-4.0	0.0660433
-3.0	-0.3390590
-2.0	-0.5767248
-1.0	-0.4400506
0.0	0.0000000
1.0	0.4400506
2.0	0.5767248
3.0	0.3390590
4.0	-0.0660433
5.0	-0.3275791
6.0	-0.2766839
7.0	-0.0046828
8.0	0.2346364
9.0	0.2453118
10.0	0.0434728
11.0	-0.1767853

12.0	-0.2234471
13.0	-0.0703181
14.0	0.1333752
15.0	0.2051040

Bessel Function Y1

15 Values

0.1	-6.4589511
1.0	-0.7812128
2.0	-0.1070324
3.0	0.3246744
4.0	0.3979257
5.0	0.1478631
6.0	-0.1750103
7.0	-0.3026672
8.0	-0.1580605
9.0	0.1043146
10.0	0.2490154
11.0	0.1637055
12.0	-0.0570992
13.0	-0.2100814
14.0	-0.1666448
15.0	0.0210736

Bessel Function Jn, $n \geq 2$

20 Values

2	1.0	1.149034849E-01
2	2.0	3.528340286E-01
2	5.0	4.656511628E-02
2	10.0	2.546303137E-01
2	50.0	-5.971280079E-02
5	1.0	2.497577302E-04
5	2.0	7.039629756E-03
5	5.0	2.611405461E-01
5	10.0	-2.340615282E-01
5	50.0	-8.140024770E-02
10	1.0	2.630615124E-10
10	2.0	2.515386283E-07
10	5.0	1.467802647E-03

10	10.0	2.074861066E-01
10	50.0	-1.138478491E-01
20	1.0	3.873503009E-25
20	2.0	3.918972805E-19
20	5.0	2.770330052E-11
20	10.0	1.151336925E-05
20	50.0	-1.167043528E-01

Bessel Function Y_n , $n \geq 2$

20 Values

2	1.0	-1.650682607
2	2.0	-6.174081042E-01
2	5.0	3.676628826E-01
2	10.0	-5.868082460E-03
2	50.0	9.579316873E-02
5	1.0	-2.604058666E02
5	2.0	-9.935989128
5	5.0	-4.536948225E-01
5	10.0	1.354030477E-01
5	50.0	-7.854841391E-02
10	1.0	-1.216180143E08
10	2.0	-1.291845422E05
10	5.0	-2.512911010E01
10	10.0	-3.598141522E-01
10	50.0	5.723897182E-03
20	1.0	-4.113970315E22
20	2.0	-4.081651389E16
20	5.0	-5.933965297E08
20	10.0	-1.597483848E03
20	50.0	1.644263395E-02

Modified Bessel Function I_0

20 Values

0.0	1.0000000
0.2	1.0100250
0.4	1.0404018
0.6	1.0920453
0.8	1.1665149
1.0	1.2660658

1.2	1.3937256
1.4	1.5533951
1.6	1.7499807
1.8	1.9895593
2.0	2.2795852
2.5	3.2898391
3.0	4.8807925
3.5	7.3782035
4.0	11.301922
4.5	17.481172
5.0	27.239871
6.0	67.234406
8.0	427.56411
10.0	2815.7167

Modified Bessel Function K0

20 Values

0.1	2.4270690
0.2	1.7527038
0.4	1.1145291
0.6	0.77752208
0.8	0.56534710
1.0	0.42102445
1.2	0.31850821
1.4	0.24365506
1.6	0.18795475
1.8	0.14593140
2.0	0.11389387
2.5	6.2347553E-02
3.0	3.4739500E-02
3.5	1.9598897E-02
4.0	1.1159676E-02
4.5	6.3998572E-03
5.0	3.6910983E-03
6.0	1.2439943E-03
8.0	1.4647071E-04
10.0	1.7780062E-05

Modified Bessel Function I1

20 Values

0.0	0.00000000
0.2	0.10050083
0.4	0.20402675
0.6	0.31370403
0.8	0.43286480
1.0	0.56515912
1.2	0.71467794
1.4	0.88609197
1.6	1.0848107
1.8	1.3171674
2.0	1.5906369
2.5	2.5167163
3.0	3.9533700
3.5	6.2058350
4.0	9.7594652
4.5	15.389221
5.0	24.335643
6.0	61.341937
8.0	399.87313
10.0	2670.9883

Modified Bessel Function K_1

20 Values

0.1	9.8538451
0.2	4.7759725
0.4	2.1843544
0.6	1.3028349
0.8	0.86178163
1.0	0.60190724
1.2	0.43459241
1.4	0.32083589
1.6	0.24063392
1.8	0.18262309
2.0	0.13986588
2.5	7.3890816E-02
3.0	4.0156431E-02
3.5	2.2239393E-02

4.0	1.2483499E-02
4.5	7.0780949E-03
5.0	1.0446134E-03
6.0	1.3439197E-03
8.0	1.5536921E-04
10.0	1.8648773E-05

Modified Bessel Function K_n , $n \geq 2$

28 Values

2	0.2	49.512430
2	1.0	1.6248389
2	2.0	2.5375975E-01
2	2.5	1.2146021E-01
2	3.0	6.1510459E-02
2	5.0	5.3089437E-03
2	10.0	2.1509817E-05
2	20.0	6.3295437E-10
3	1.0	7.101262825
3	2.0	6.473853909E-01
3	5.0	8.291768415E-03
3	10.0	2.725270026E-05
3	50.0	3.72793677E-23
5	1.0	3.609605896E02
5	2.0	9.431049101
5	5.0	3.270627371E-02
5	10.0	5.754184999E-05
5	50.0	4.36718224E-23
10	1.0	1.807132899E08
10	2.0	1.624824040E05
10	5.0	9.758562829
10	10.0	1.614255300E-03
10	50.0	9.15098819E-23
20	1.0	6.294369369E22
20	2.0	5.770856853E16
20	5.0	4.827000521E08
20	10.0	1.787442782E02
20	50.0	1.70614838E-21

Modified Bessel Function I_n , $n \geq 2$

28 Values

2	0.2	5.0166876E-03
2	1.0	1.3574767E-01
2	2.0	6.8894844E-01
2	2.5	1.2764661
2	3.0	2.2452125
2	5.0	17.505615
2	10.0	2281.5189
2	20.0	3.9312785E07
3	1.0	2.216842492E-02
3	2.0	2.127399592E-01
3	5.0	1.033115017E01
3	10.0	1.758380717E01
3	50.0	2.67776414E20
5	1.0	2.714631560E-04
5	2.0	9.825679323E-03
5	5.0	2.157974547
5	10.0	7.771882864E02
5	50.0	2.27854831E20
10	1.0	2.752948040E-10
10	2.0	3.016963879E-07
10	5.0	4.580044419E-03
10	10.0	2.189170616E01
10	50.0	1.07159716E20
20	1.0	3.966835986E-25
20	2.0	4.310560576E-19
20	5.0	5.024239358E-11
20	10.0	1.250799736E-04
20	50.0	5.44200840E18

Bessel Function $J_{\alpha+n}$

18 Values

0	0.5	0.5	0.5409738
1	0.5	0.5	0.09170170
2	0.5	0.5	0.009236408
3	0.5	0.5	0.0006623786
4	0.5	0.5	0.00003689213
0	0.5	3.5	-0.1496046

1	0.5	3.5	0.3566427
2	0.5	3.5	0.4552983
3	0.5	3.5	0.2937835
4	0.5	3.5	0.1322688
-1	0.5	0.5	0.9902459
-2	0.5	0.5	-2.521466
-3	0.5	0.5	14.13855
-4	0.5	0.5	-138.8640
-1	0.5	3.5	-0.3993868
-2	0.5	3.5	0.2637151
-3	0.5	3.5	0.1733453
-4	0.5	3.5	-0.5113513

Modified Bessel Function I_{a+n}

14 Values

0	0.5	0.2	0.3592084
1	0.5	0.2	0.02388361
2	0.5	0.2	0.0009542545
3	0.5	0.2	0.00002724712
0	0.5	1.0	0.9376749
1	0.5	1.0	0.2935253
2	0.5	1.0	0.05709891
3	0.5	1.0	0.008030780
-1	0.5	0.2	1.819926
-2	0.5	0.2	-8.740420
-3	0.5	0.2	132.9262
-1	0.5	1.0	1.231200
-2	0.5	1.0	-0.2935253
-3	0.5	1.0	2.111776

Exponential Integral Ex

6 Values

0.1	1.8229237549
0.5	0.5597735950
1.0	0.2193839344
1.5	0.1000195826
3.0	0.0130483813
5.0	0.0011482986

Exponential Integral As

12 Values

1	0.25	0.31152031E1
3	0.25	0.12772332E3
5	0.25	0.24575836E5
7	0.25	0.11796478E8
9	0.25	0.10569646E11
11	0.25	0.15220290E14
1	24.0	0.15729727E-11
3	24.0	0.17095154E-11
5	24.0	0.18707497E-11
7	24.0	0.20636507E-11
9	24.0	0.22979296E-11
11	24.0	0.25874295E-11

Exponential Integral E_s

12 Values

2	0.25	-0.16771066
4	0.25	-0.10074586
6	0.25	-0.07200876
8	0.25	-0.05603029
10	0.25	-0.04585627
12	0.25	-0.03880972
1	24.0	1103713422.07
3	24.0	1015569641.84
5	24.0	940918885.93
7	24.0	876791258.53
9	24.0	821052542.62
11	24.0	772122289.32

Legendre Polynomials

19 Values

1	0	1.0	1.224745
10	0	1.0	3.240370
20	0	1.0	4.527693
1	0	0.7071067	0.866025
10	0	0.7071067	0.373006
20	0	0.7071067	-0.874140
1	0	0.0	0.000000
10	0	0.0	-0.797435

20	0	0.0	0.797766
2	2	0.7071067	0.484123
10	2	0.7071067	-0.204789
20	2	0.7071067	0.910208
2	2	0.0	0.968246
10	2	0.0	0.804785
20	2	0.0	-0.799672
10	10	0.7071067	0.042505
20	10	0.7071067	-0.707252
10	10	0.0	1.360172
20	10	0.0	-0.853705

第5章 函数逼近

在函数求值中,经常用级数来逼近函数,从而计算函数值.

5.1节是用交错级数的欧拉变换方法对级数求和.5.2节计算多项式函数及其各阶导数,并计算两个多项式的商多项式及余多项式(注意,是代数计算,即对有理多项式的代数计算).5.3节用克伦肖的递推公式计算函数的切比雪夫逼近多项式,在克伦肖的递推公式中,为了使数值稳定,一般是按下降次序即下标减少的次序来计算的.但在极少数情况下,反而按上升次序进行递推,才能使数值稳定.5.4节利用函数的切比雪夫逼近级数得到函数的积分和导数的切比雪夫逼近级数.这对函数逼近计算的灵活运用是非常方便的.5.5节包含了切比雪夫逼近和多项式逼近间的转换.

本章对于函数的逼近是很基本的内容,读者应能把本章的一般技巧和其它各章的方法结合起来考虑.

5.1 级数求和

1. 功能

用欧拉变换方法计算交错级数的和,进而也可用于求正项级数的和.本算法收敛速度快,对于交错级数特别有效.

2. 方法

(1) 埃特金的 δ^2 -过程.

设有级数 $\sum_{k=0}^{\infty} u_k$ 收敛,记 $S_n = \sum_{k=0}^n u_k$,则埃特金的 δ^2 -过程是

$$S'_n = S_{n+1} - \frac{(S_{n+1} - S_n)^2}{S_{n+1} - 2S_n + S_{n-1}}$$

或更一般的

$$S'_n = S_{n+p} - \frac{(S_{n+p} - S_n)^2}{S_{n+p} - 2S_n + S_{n-p}} \quad (p \text{ 为整数})$$

(2) 交错级数的欧拉变换方法.

交错级数的欧拉变换公式

$$\sum_{s=0}^{\infty} (-1)^s u_s = u_0 - u_1 + u_2 - \cdots - u_{n-1} + \sum_{s=0}^p (-1)^s \frac{1}{2^{s+1}} [\Delta^s u_n]$$

其中 n 为偶数,

$$\Delta u_n = u_{n+1} - u_n$$

$$\Delta^s u_n = \Delta[\Delta^{s-1} u_n], \quad s = 2, 3, \dots$$

记

$$\sigma_n = \sum_{s=0}^{n-1} (-1)^s u_s, \quad w_{p,n} = \sum_{s=0}^p (-1)^s \frac{\Delta^s u_n}{2^{s+1}}$$

$$S_{n+p} = \sigma_n + w_{p,n}$$

则取

$$S_{n+p+1} = \begin{cases} S_{(n+1)+p} = \sigma_{n+1} + w_{p,n+1}, & \text{当 } n \text{ 为奇数时} \\ S_{n+(p+1)} = \sigma_n + w_{p+1,n}, & \text{当 } n \text{ 为偶数时} \end{cases}$$

(3) 对于正项级数有

$$\sum_{r=1}^{\infty} v_r = \sum_{r=1}^{\infty} (-1)^{r-1} w_r$$

$$w_r = \sum_{k=0}^{\infty} 2^k v_{(2^k)r}$$

由于 w_r 中 v_r 的指标是以 2 的幂次增加的, 因而收敛很快, 实际计算仅需少数几项.

3. 使用说明

EULSUM(SUM, TERM, JTERM, WKSP)

SUM 实型变量, 输入、输出参数, 输入时存放已知的部分和, 输出时存放新的部分和

JTERM 整型变量, 输入参数, 输入需要合并到 SUM 中项的指标, JTERM = 1, 2, ...

TERM 实型变量, 输入参数, 存放级数的第 JTERM 项 $(-1)^{JTERM-1} u_{JTERM}$

WKSP 实型数组, 工作单元

4. 过程

子过程 EULSUM.

SUBROUTINE eulsum(sum, term, jterm, wksp)

INTEGER jterm

REAL sum, term, wksp(jterm)


```

INTEGER j,nterm
REAL dum,tmp
SAVE nterm
if(jterm=-1)then
    nterm=1
    wksp(1)=term
    sum=0.5 * term
else
    tmp=wksp(1)
    wksp(1)=term
    do j=1,nterm-1
        dum=wksp(j+1)
        wksp(j+1)=0.5 * (wksp(j)+tmp)
        tmp=dum
    end do
    wksp(nterm+1)=0.5 * (wksp(nterm)+tmp)
    if(abs(wksp(nterm+1))<=abs(wksp(nterm))) then
        sum=sum + 0.5 * wksp(nterm+1)
        nterm=nterm+1
    else
        sum=sum + wksp(nterm+1)
    endif
endif
END SUBROUTINE eulsum

```

5. 例子

子过程 EULSUM 利用欧拉变换对交错级数进行求和. 我们的例子是计算下列级数的和

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, \quad -1 < x < 1$$

验证程序开始后需要输入 MVAL, 表示随后将 MVAL 次调用子过程 EULSUM, 变量 TERM 取值为 $(-1)^{j+1}x^j/j$. 最后将数值计算结果与精确解进行比较. 如果 MVAL 取小于 1 或大于 40, 程序将结束运行. 验证程序 D5R1 如下:

```

PROGRAM D5R1
! Driver for routine EULSUM
PARAMETER (NVAL=40)

```

```

DIMENSION WKSP(NVAL)
! Evaluate  $\ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 \dots$  for  $-1 < x < 1$ 
DO
  WRITE(*,*) 'How many terms in polynomial?'
  WRITE(*, '(1X,A,I2,A)') 'Enter n between 1 and ', &
    NVAL, '. Enter n=0 to end.'
  READ(*,*) MVAL
  IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
  WRITE(*, '(1X,T9,A1,T18,A6,T28,A10)') 'X', 'Actual', &
    'Polynomial'
  DO I=-8,8,1
    X=I/10.0
    SUM=0.0
    XPOWER=-1.
    DO J=1,MVAL
      XPOWER=-X*XPOWER
      TERM=XPOWER/J
      CALL EULSUM(SUM,TERM,J,WKSP)
    END DO
    WRITE(*, '(3F12.6)') X, LOG(1.0+X), SUM
  END DO
END DO
END

```

计算结果如下:

How many terms in polynomial?

Enter n between 1 and 40. Enter n=0 to end.

5

X	Actual	Polynomial
-0.800000	-1.609438	-1.425835
-0.700000	-1.203973	-1.136165
-0.600000	-0.916291	-0.892176
-0.500000	-0.693147	-0.685417
⋮	⋮	⋮
0.500000	0.405465	0.404427
0.600000	0.470004	0.468744
0.700000	0.530628	0.532260

```

0.800000      0.587787      0.589429
How many terms in polynomial?
Enter n between 1 and 40. Enter n=0 to end.
20

```

X	Actual	Polynomial
-0.800000	-1.609438	-1.607258
-0.700000	-1.203973	-1.203872
-0.600000	-0.916291	-0.916287
-0.500000	-0.693147	-0.693147
⋮	⋮	⋮
0.500000	0.405465	0.405465
0.600000	0.470004	0.470004
0.700000	0.530628	0.530628
0.800000	0.587787	0.587787

```

How many terms in polynomial?
Enter n between 1 and 40. Enter n=0 to end.
0

```

5.2 多项式和有理函数

1. 功能

本节包括两个函数过程:DDPOLY 与 POLDIV. 第一个用于计算 $n-1$ 阶多项式在给定点的函数值及其各阶导数值;第二个用于计算给定的两个多项式的商多项式及余多项式.

2. 方法

(1) 设已知一多项式函数

$$P(x) = c_n x^{n-1} + c_{n-1} x^{n-2} + \cdots + c_2 x + c_1$$

则 $P(x)$ 的导函数为

$$P'(x) = (n-1)c_n x^{n-2} + (n-2)c_{n-1} x^{n-3} + \cdots + c_2$$

一般地,其 k 阶导函数为

$$P^{(k)}(x) = \sum_{j=k+1}^n (j-1)(j-2)\cdots(j-k)c_j x^{j-k-1}$$

(2) 设已知两个多项式函数

$$U(x) = \sum_{j=1}^n u_j x^{j-1}$$

$$V(x) = \sum_{j=1}^m v_j x^{j-1}$$

则由综合除法有

$$U(x) = V(x)Q(x) + R(x)$$

其中 $Q(x)$ 即为 $U(x)$ 与 $V(x)$ 的商多项式, $R(x)$ 为其商的余多项式.

3. 使用说明

(1) DDPOLY (C,NC,X,PD,ND)

- NC 整型变量,输入参数,多项式阶数加 1
 C NC 个元素的一维实型数组,输入参数,存放多项式的系数,其中 $C(1)$ 为常数项
 X 实型变量,输入参数,需求值的自变量
 ND 整型变量,输入参数,输入所需多项式,导数值的最高阶数加 1
 PD ND 个元素的一维实型数组,输出参数,存放输出结果,其中 $PD(1)$ 为多项式在 X 处的值, $PD(j)$ 为多项式在 X 处的 $j-1$ 阶导数值, $j=2, \dots, ND$

(2) POLDIV(U,N,V,NV,Q,R)

- N 整型变量,输入参数,多项式 $U(X)$ 的阶数加 1
 NV 整型变量,输入参数,多项式 $V(X)$ 的阶数加 1
 U N 个元素的一维实型数组,输入参数,多项式 $U(X)$ 的系数,其中 $U(1)$ 为常数项
 V NV 个元素的一维实型数组,输入参数,多项式 $V(X)$ 的系数,其中 $V(1)$ 为其常数项
 Q N 个元素的一维实型数组,输出参数,存放商多项式 $Q(x)$ 的系数,其中较高项的系数 $Q(j)=0, j=N-NV+1, \dots, N$
 R NV 个元素的一维实型数组,输出参数,存放余多项式 $R(x)$ 的系数,其中较高指数的项的系数为零

4. 过程

(1) 子过程 DDPOLY.

```
SUBROUTINE ddpoly(c,nc,x,pd,nd)
INTEGER nc,nd
```

```

REAL x,c(nc),pd(nd)
INTEGER i,j,nnd
REAL const
pd(1)=c(nc)
do j=2,nd
    pd(j)=0.
end do
do i=nc-1,1,-1
    nnd=min(nd,nc+1-i)
    do j=nnd,2,-1
        pd(j)=pd(j)*x+pd(j-1)
    end do
    pd(1)=pd(1)*x+c(i)
end do
const=2.
do i=3,nd
    pd(i)=const*pd(i)
    const=const*i
end do
END SUBROUTINE ddpoly

```

(2) 子过程 POLDIV.

```

SUBROUTINE poldiv(u,n,v,nv,q,r)
INTEGER n,nv
REAL q(n),r(n),u(n),v(nv)
INTEGER j,k
do j=1,n
    r(j)=u(j)
    q(j)=0.
end do
do k=n-nv,0,-1
    q(k+1)=r(nv+k)/v(nv)
    do j=nv+k-1,k-1,-1
        r(j)=r(j)-q(k+1)*v(j-k)
    end do
end do
do j=nv,n

```

```

      r(j)=0.
    end do
  END SUBROUTINE poldiv

```

5. 例子

(1) 为了验证子过程 DDPOLY, 我们所举的例子为

$$(x-1)^5 = -1 + 5x - 10x^2 + 10x^3 - 5x^4 + x^5$$

(当然, 这是一个很简单的例子, 每个人都会用笔计算出结果, 但是它给了我们验证数值计算结果的一条方便途径). 我们所计算的 x 的区间为 $[0.0, 2.0]$. 该多项式的各阶导数的精确解为

$$f^{(n-1)}(x) = \frac{5!}{(6-n)!} (x-1.0)^{6-n}, \quad n = 1, \dots, 5$$

验证程序 D5R2 中为了计算阶乘, 采用了第 4 章中的子过程 FACTRL. 验证程序 D5R2 如下:

```

PROGRAM D5R2
  ! Driver for routine DDPOLY
  ! Polynomial (X-1) * * 5
  PARAMETER (NC=6, NCM1=5, NP=20)
  DIMENSION C(NC), PD(NCM1), D(NCM1, NP)
  ! USES FACTRL
  CHARACTER A(NCM1) * 15
  DATA A/'polynomial:', 'first deriv:', 'second deriv:', &
        'third deriv:', 'fourth deriv:' /
  DATA C / -1.0, 5.0, -10.0, 10.0, -5.0, 1.0 /
  DO I=1, NP
    X=0.1 * I
    CALL DDPOLY(C, NC, X, PD, NC-1)
    DO J=1, NC-1
      D(J, I) = PD(J)
    END DO
  END DO
  DO I=1, NC-1
    WRITE(*, '(IX, T7, A)') A(I)
    WRITE(*, '(IX, T13, A, T25, A, T40, A)') 'X', 'DDPOLY', &
      'ACTUAL'
    DO J=1, NP

```

```

X=0.1 * J
WRITE( * , '(1X,3F15.6)') X,D(I,J),&
    FACTRL(NC-1)/FACTRL(NC-I) * ((X-1.0) * * (NC-I))
END DO
WRITE( * , * ) 'Press ENTER to END DO...'
READ( * , * )
END DO
END

```

计算结果如下:

polynomial:

X	DDPOLY	ACTUAL
0.100000	-0.590490	-0.590490
0.200000	-0.327680	-0.327680
0.300000	-0.168070	-0.168070
⋮	⋮	⋮
1.800000	0.327682	0.327680
1.900000	0.590491	0.590490
2.000000	1.000000	1.000000

Press ENTER to END DO...

first deriv:

X	DDPOLY	ACTUAL
0.100000	3.280500	3.280499
0.200000	2.048000	2.048000
0.300000	1.200500	1.200500
⋮	⋮	⋮
1.800000	2.048005	2.048001
1.900000	3.280502	3.280499
2.000000	5.000000	5.000000

Press ENTER to END DO...

second deriv:

X	DDPOLY	ACTUAL
0.100000	-14.580000	-14.579998
0.200000	-10.240001	-10.240001
0.300000	-6.859999	-6.860000
⋮	⋮	⋮
1.800000	10.240008	10.240003

1.900000	14.580003	14.579998
2.000000	20.000000	20.000000

Press ENTER to END DO...

third deriv:

X	DDPOLY	ACTUAL
0.100000	48.599998	48.599998
0.200000	38.399998	38.400002
0.300000	29.400002	29.400000
:	:	:
1.800000	38.400013	38.400005
1.900000	48.600002	48.599998
2.000000	60.000000	60.000000

Press ENTER to END DO...

fourth deriv:

X	DDPOLY	ACTUAL
0.100000	-108.000015	-108.000000
0.200000	-96.000023	-96.000000
0.300000	-83.999985	-84.000000
:	:	:
1.800000	96.000015	96.000008
1.900000	108.000000	108.000000
2.000000	120.000000	120.000000

Press ENTER to END DO...

(2) 为了验证 POLDIV, 我们所举的例子为

$$U = -1 + 5x - 10x^2 + 10x^3 - 5x^4 + x^5 = (x-1)^5$$

$$V = 1 + 3x + 3x^2 + x^3 = (x+1)^3$$

$$Q = 31 - 8x + x^2$$

$$R = -32 - 80x - 80x^2$$

验证程序 D5R3 如下:

```
PROGRAM D5R3
```

```
! Driver for routine POLDIV
```

```
! (X-1) * * 5 / (X+1) * * 3
```

```
PARAMETER (N=6,NV=4)
```

```
DIMENSION U(N),V(NV),Q(N),R(N)
```

```
DATA U / -1.0,5.0,-10.0,10.0,-5.0,1.0 /
```



```

DATA V/1.0,3.0,3.0,1.0
CALL POLDIV(U,N,V,NV,Q,R)
WRITE(*, '(//1X,6(7X,A)/)') &
      'X^ 0', 'X^ 1', 'X^ 2', 'X^ 3', 'X^ 4', 'X^ 5'
WRITE(*, *) 'Quotient polynomial coefficients;'
WRITE(*, '(1X,6F10.2/)' ) (Q(I), I=1,6)
WRITE(*, *) 'Expected quotient coefficients;'
WRITE(*, '(1X,6F10.2/)' ) 31.0, -8.0, 1.0, 0.0, 0.0, 0.0
WRITE(*, *) 'Remainder polynomial coefficients;'
WRITE(*, '(1X,4F10.2/)' ) (R(I), I=1,4)
WRITE(*, *) 'Expected quotient coefficients;'
WRITE(*, '(1X,4F10.2/)' ) -32.0, -80.0, -80.0, 0.0
END

```

计算结果如下:

X^ 0	X^ 1	X^ 2	X^ 3	X^ 4	X^ 5
Quotient polynomial coefficients;					
31.00	-8.00	1.00	0.00	0.00	0.00
Expected quotient coefficients;					
31.00	-8.00	1.00	0.00	0.00	0.00
Remainder polynomial coefficients;					
-32.00	-80.00	-80.00	0.00		
Expected quotient coefficients;					
-32.00	-80.00	-80.00	0.00		

5.3 切比雪夫逼近

1. 功能

给定函数 $f(x)$, $x \in [a, b]$, 求其切比雪夫(Чебышев)逼近级数

$$f(x) \approx \sum_{k=1}^n c_k T_{k-1}(y) - c_1/2, \quad y = \left[x - \frac{1}{2}(b+a) \right] / \left[\frac{1}{2}(b-a) \right]$$

其中 $T_k(y)$ 为 k 阶切比雪夫多项式, 并用切比雪夫级数求函数 $f(x)$ 在给定点 x 处的值.

子过程 CHEBFT 用于计算切比雪夫级数的系数 $c_k (k=1, \dots, n)$, 而子过程 CHEBEV 用切比雪夫级数计算函数在给定点 x 处的值.

2. 方法

(1) 设 $x_k (k=1, \dots, n)$ 为 $T_n(x) (x \in [-1, 1])$ 的 n 个零点, 则切比雪夫级数的系数为

$$c_j = \frac{2}{n} \sum_{k=1}^n f(x_k) T_{j-1}(x_k), \quad x_k = \cos \left[\frac{T_1 \left(k - \frac{1}{2} \right)}{n} \right], \quad j = 1, \dots, n$$

逼近公式为

$$f(x) \approx \sum_{k=1}^n c_k T_{k-1}(x) - \frac{1}{2} c_1, \quad x \in [-1, 1]$$

(2) 克伦肖 (Clenshaw) 的递推公式

设已知

$$f(x) = \sum_{k=0}^n c_k F_k(x)$$

$$F_{n+1}(x) = \alpha(n, x) F_n(x) + \beta(n, x) F_{n-1}(x)$$

其中 $\alpha(n, x), \beta(n, x)$ 为给定的函数.

定义下列递推关系:

$$y_{n+2} = y_{n+1} = 0$$

$$y_k = \alpha(k, x) y_{k+1} + \beta(k+1, x) y_{k+2} + c_k, \quad k = n, n-1, \dots, 1$$

则

$$f(x) = \beta(1, x) F_0(x) y_2 + F_1(x) y_1 + F_0(x) c_0$$

(3) 对切比雪夫逼近

$$f(x) = \sum_{k=1}^n c_k T_{k-1}(x) - \frac{1}{2} c_1$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n \geq 1, \quad x \in [-1, 1]$$

应用克伦肖递推公式定义

$$d_{n+2} = d_{n+1} = 0$$

$$d_j = 2x d_{j+1} - d_{j+2} + c_j, \quad j = n, n-1, \dots, 2$$

则

$$f(x) = d_0 = x d_2 - d_3 + \frac{1}{2} c_1$$

(4) 对 $x \in [a, b]$ 作变换

$$y = \frac{x - \frac{1}{2}(b+a)}{\frac{1}{2}(b-a)}$$

则 $y \in [-1, 1]$.

3. 使用说明

(1) CHEBFT(A,B,C,N,FUNC)

- N 整型变量,输入参数,所求切比雪夫级数的项数
 A 实型变量,输入参数,区间 $[a, b]$ 的 a
 B 实型变量,输入参数,区间 $[a, b]$ 的 b
 C N 个元素的一维实型数组,输出参数,存放切比雪夫级数的系数
 FUNC 函数子程序,输入参数,要求逼近的函数 $f(x)$,用户自编。

(2) CHEBEV(A,B,C,M,X)

- M 整型变量,输入参数,所用切比雪夫级数的项数
 A 实型变量,输入参数,区间 $[a, b]$ 的 a
 B 实型变量,输入参数,区间 $[a, b]$ 的 b
 C M 个元素的一维实型数组,输入参数,输入由函数过程 CHEBFT 输出的 C 中的前 M 个元素
 X 实型变量,输入参数,要求求值的点 $x \in [a, b]$

4. 过程

(1) 子过程 CHEBFT.

```
SUBROUTINE chebft(a,b,c,n,func)
  INTEGER n,NMAX
  REAL a,b,c(n),func,PI
  EXTERNAL func
  PARAMETER (NMAX=50, PI=3.141592653589793d0)
  INTEGER j,k
  REAL bma,bpa,lac,y,f(NMAX)
  DOUBLE PRECISION sum
  bma=0.5*(b-a)
  bpa=0.5*(b+a)
  do k=1,n
    y=cos(PI*(k-0.5)/n)
    f(k)=func(y*bma+bpa)
  end do
  fac=2./n
  do j=1,n
```

```

    sum=0.d0
    do k=1,n
        sum=sum+f(k)*cos((PI*(j-1))*(k-0.5d0)/n))
    end do
    c(j)=fac*sum
end do
END SUBROUTINE chebft

```

(2) 函数过程 CHEBEV.

```

FUNCTION chebev(a,b,c,m,x)
INTEGER m
REAL chebev,a,b,x,c(m)
INTEGER j
REAL d,dd,sv,y,y2
if ((x-a)*(x-b)>0.) pause 'x not in range in chebev'
d=0.
dd=0.
y=(2.*x-a-b)/(b-a)
y2=2.*y
do j=m,2,-1
    sv=d
    d=y2*d-dd+c(j)
    dd=sv
end do
chebev=y*d-dd+0.5*c(1)
END FUNCTION chebev

```

5. 例子

(1) 验证子过程 CHEBFT 的例子是 $\text{FUNC} = x^2(x^2 - 2)\sin x$, $x \in (-\pi/2, \pi/2)$. 所求出的切比雪夫级数的最大项数 $\text{NVAL} = 40$. 为了和精确解进行比较, 可以随意确定所需要的项数. 验证程序 D5R4 如下:

```

PROGRAM D5R4
1 Driver for routine CHEBFT
EXTERNAL FUNC
PARAMETER (NVAL=40,PIO2=1.5707963,EPS=1.E-6)

```

```

DIMENSION C(NVAL)
A=-PI/2
B=PI/2
CALL CHEBFT(A,B,C,NVAL,FUNC)
! Test result
DO
  WRITE(*,*) 'How many terms in Chebyshev evaluation?'
  WRITE(*, '(1X,A,12,A)') 'Enter n between 6 and ',NVAL&
    ,'. Enter n=0 to end.'
  READ(*,*) MVAL
  IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
  WRITE(*, '(1X,T10,A,T19,A,T28,A)') 'X','Actual',&
    'Chebyshev fit'
  DO I=-8,8,1
    X=I*PI/2/10.0
    Y=(X-0.5*(B+A))/(0.5*(B-A))
    ! Evaluate Chebyshev polynomial without using routine CHEBEV
    T0=1.0
    T1=Y
    F=C(2)*T1+C(1)*0.5
    DO J=3,MVAL
      DUM=-T1
      T1=2.0*Y*T1-T0
      T0=DUM
      TERM=C(J)*T1
      F=F+TERM
    END DO
    WRITE(*, '(1X,3F12.6)') X,FUNC(X),F
  END DO
END DO
END
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC

```

计算结果如下:

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

10

X	Actual	Chebyshev fit
-1.256637	0.632073	0.632055
-1.099557	0.852077	0.852096
-0.942478	0.798917	0.798935
-0.785398	0.603301	0.603288
⋮	⋮	⋮
0.785398	-0.603301	-0.603288
0.942478	-0.798917	-0.798935
1.099557	-0.852077	-0.852096
1.256637	-0.632073	-0.632055

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

20

X	Actual	Chebyshev fit
-1.256637	0.632073	0.632073
-1.099557	0.852077	0.852077
-0.942478	0.798917	0.798916
-0.785398	0.603301	0.603301
⋮	⋮	⋮
0.785398	-0.603301	-0.603301
0.942478	-0.798917	-0.798917
1.099557	-0.852077	-0.852077
1.256637	-0.632073	-0.632073

(2) 验证 CHEBEV 的例子和前面验证 CHEBFT 的例子相同. 事实上, 以下的验证程序 D5R5 和上面的 D5R4 几乎一样, 只不过用 CHEBEV 来替代前面的多项式求和. 验证程序 D5R5 如下:

```

PROGRAM D5R5
! Driver for routine CHEBEV
EXTERNAL FUNC
! USES CHEBFT
PARAMETER (NVAL=40,PIO2=1.5707963)
DIMENSION C(NVAL)
A=-PIO2
B=PIO2

```

```

CALL CHEBFT(A,B,C,NVAL,FUNC)
! Test Chebyshev evaluation routine
DO
  WRITE(*,*) 'How many terms in Chebyshev evaluation?'
  WRITE(*,'(1X,A,I2,A)') 'Enter n between 6 and ',NVAL&
    ,'. Enter n=0 to end.'
  READ(*,*) MVAL
  IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
  WRITE(*,'(1X,T10,A,T19,A,T28,A)') 'X','Actual',&
    'Chebyshev fit'
  DO I=-8,8,1
    X=I*PIO2/10.0
    WRITE(*,'(1X,3F12.6)') X,FUNC(X),CHEBEV(A,B,C,&
      MVAL,X)
  END DO
END DO
END
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC

```

计算结果如下:

```

How many terms in Chebyshev evaluation?
Enter n between 6 and 40. Enter n=0 to end.

```

10

X	Actual	Chebyshev fit
-1.256637	0.632073	0.632055
-1.099557	0.852077	0.852096
-0.942478	0.798917	0.798935
-0.785398	0.603301	0.603288
⋮	⋮	⋮
0.785398	-0.603301	-0.603288
0.942478	-0.798917	-0.798935
1.099557	-0.852077	-0.852096
1.256637	-0.632073	-0.632055

```

How many terms in Chebyshev evaluation?
Enter n between 6 and 40. Enter n=0 to end.

```

20

X	Actual	Chebyshev fit
-1.256637	0.632073	0.632073
-1.099557	0.852077	0.852077
-0.942478	0.798917	0.798917
-0.785398	0.603301	0.603301
⋮	⋮	⋮
0.785398	-0.603301	-0.603301
0.942478	-0.798917	-0.798916
1.099557	-0.852077	-0.852077
1.256637	-0.632073	-0.632073

5.4 积分和导数的切比雪夫逼近

1. 功能

给定函数 $f(x)$, $x \in [a, b]$, 利用 $f(x)$ 的切比雪夫逼近, 求函数 $f(x)$ 的不定积分和导函数的切比雪夫逼近级数.

子过程 CHINT 用于计算函数不定积分的切比雪夫逼近级数的系数, 并选择积分常数使不定积分在 a 处的值为零; 子过程 CHDER 用于计算函数导函数的切比雪夫逼近级数的系数.

2. 方法

(1) 设 $c_i (i=1, \dots, m)$ 为 $f(x)$ 的切比雪夫逼近级数的系数, $C_i (i=1, \dots, m)$ 为 $f(x)$ 的不定积分的切比雪夫逼近级数的系数, 则

$$C_i = \frac{c_{i-1} - c_{i+1}}{2(i-1)} \quad (i > 1)$$

(2) 设 $c'_i (i=1, 2, \dots, m)$ 为函数的导数的切比雪夫逼近级数的系数, 则

$$c'_m = c'_{m+1} = 0$$

$$c'_{i-1} = c'_{i+1} - 2(i-1)c_i, \quad i = m-1, m-2, \dots, 2$$

3. 使用说明

(1) CHINT(A,B,C,CINT,N)

N 整型变量, 输入参数, 要求逼近的阶数

A 实型变量, 输入参数, 区间 $[a, b,]$ 的 a

- B 实型变量,输入参数,区间 $[a,b,]$ 的 b
- C N 个元素的一维实型数组,输入参数,输入由子过程 CHEBFT 输出的 C
- CINT N 个元素的一维实型数组,输出参数,存放函数不定积分的切比雪夫逼近级数的系数
- (2) CHDER(A,B,C,CDER,N)
- N,A,B,C 同子过程 CHINT
- CDER N 个元素的一维实型数组,输出参数,存放导函数的切比雪夫逼近级数的系数

4. 过程

(1) 子过程 CHINT.

```

SUBROUTINE chint(a,b,c,cint,n)
INTEGER n
REAL a,b,c(n),cint(n)
INTEGER j
REAL con,fac,sum
con=0.25*(b-a)
sum=0.
fac=1.
do j=2,n-1
    cint(j)=con*(c(j-1)-c(j+1))/(j-1)
    sum=sum+fac*cint(j)
    fac=-fac
end do
cint(n)=con*c(n-1)/(n-1)
sum=sum+fac*cint(n)
cint(1)=2.*sum
END SUBROUTINE chint

```

(2) 子过程 CHDER.

```

SUBROUTINE chder(a,b,c,cder,n)
INTEGER n
REAL a,b,c(n),cder(n)
INTEGER j

```

```

REAL con
cder(n)=0.
cder(n-1)=2*(n-1)*c(n)
do j=n-2,1,-1
    cder(j)=cder(j+2)+2*j*c(j+1)
end do
con=2./(b-a)
do j=1,n
    cder(j)=cder(j)*con
end do
END SUBROUTINE chder

```

5. 例子

验证 CHINT 和 CHDER 的例子是相同的, 即 $\text{FUNC} = x^2(x^2 - 2)\sin x$. 该函数的积分为

$$\text{FINT} = 4x^2(x^2 - 7)\sin x - (x^4 - 14x^2 + 28)\cos x$$

该函数的导函数为

$$\text{FDER} = 4x^2(x^2 - 1)\sin x + x^2(x^2 - 2)\cos x$$

在计算过程中, 首先要调用 5.3 节中的函数过程 CHEBFT, 计算被积分函数的切比雪夫系数; 然后, 再调用 CHINT 或 CHDER; 最后, 再调用 5.3 节中的 CHEBEV 计算出积分值和导函数的值, 并与精确解进行比较.

(1) 验证 CHINT 的程序 D5R6 如下:

```

PROGRAM D5R6
! Driver for routine CHINT
EXTERNAL FUNC,FINT
PARAMETER (NVAL=40,PIO2=1.5707963)
! USES CHEBFT,CHEBEV
DIMENSION C(NVAL),CINT(NVAL)
A=-PIO2
B=PIO2
CALL CHEBFT(A,B,C,NVAL,FUNC)
! Test integral
DO
    WRITE(*,*) 'How many terms in Chebyshev evaluation?'
    WRITE(*, '(1X,A,I2,A)') 'Enter n between 6 and ',NVAL&

```

```

      ,'. Enter n=0 to end.'
READ(*,*) MVAL
IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
CALL CHINT(A,B,C,CINT,MVAL)
WRITE(*, '(1X,T10,A,T19,A,T28,A)') 'X','Actual',&
      'Cheby. Integ.'
DO I=-8,8,1
  X=I*PIO2/10.0
  WRITE(*, '(1X,3F12.6)') X,FINT(X)-FINT(-PIO2),&
      CHEBEV(A,B,CINT,MVAL,X)
END DO
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC
FUNCTION FINT(X)
! Integral of FUNC
  FINT=4.0*X*((X**2)-7.0)*SIN(X)-((X**4)-14.0*(X**2)+&
      28.0)*COS(X)
END FUNCTION FINT

```

计算结果如下:

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

10

X	Actual	Cheby. Integ.
-1.256637	-0.026791	-0.026791
-1.099557	0.094254	0.094399
-0.942478	0.226606	0.226871
-0.785398	0.337824	0.338027
:	:	:
0.785398	0.337824	0.338027
0.942478	0.226606	0.226871
1.099557	0.094254	0.094399
1.256637	-0.026791	-0.026791

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

20

X	Actual	Cheby. Integ.
-1.256637	-0.026791	-0.026791
-1.099557	0.094254	0.094254
-0.942478	0.226606	0.226605
-0.785398	0.337824	0.337825
⋮	⋮	⋮
0.785398	0.337824	0.337825
0.942478	0.226606	0.226605
1.099557	0.094254	0.094254
1.256637	-0.026791	-0.026791

(2) 验证 CHDER 的程序 D5R7 如下:

```

PROGRAM D5R7
! Driver for routine CHDER
EXTERNAL FUNC,FDER
! USES CHEBFT,CHBEV
PARAMETER (NVAL=40,PIO2=1.5707963)
DIMENSION C(NVAL),CDER(NVAL)
A=-PIO2
B=PIO2
CALL CHEBFT(A,B,C,NVAL,FUNC)
! Test derivative
DO
  WRITE(*,*) 'How many terms in Chebyshev evaluation?'
  WRITE(*, '(1X,A,I2,A)') 'Enter n between 6 and ',NVAL&
    ,'. Enter n=0 to end.'
  READ(*,*) MVAL
  IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
  CALL CHDER(A,B,C,CDER,MVAL)
  WRITE(*, '(1X,T10,A,T19,A,T28,A)') 'X','Actual',&
    'Cheby. Deriv.'
  DO I= 8,8,1
    X=I*PIO2/10.0
    WRITE(*, '(1X,3F12.6)') X,FDER(X),&
      CHEBEV(A,B,CDER,MVAL,X)
  
```

```

END DO
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC
FUNCTION FDER(X)
! Derivative of FUNC
  FDER=4.0*X*((X**2)-1.0)*SIN(X)+(X**2)*&
      (X**2-2.0)*COS(X)
END FUNCTION FDER

```

计算结果如下:

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

10

X	Actual	Cheby. Deriv.
-1.256637	2.563207	2.563418
-1.099557	0.384989	0.385147
-0.942478	-0.921232	-0.921387
-0.785398	-1.454446	-1.454627
:	:	:
0.785398	1.454446	-1.454627
0.942478	-0.921232	-0.921387
1.099557	0.384989	0.385148
1.256637	2.563207	2.563417

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

20

X	Actual	Cheby. Deriv.
-1.256637	2.563207	2.563206
-1.099557	0.384989	0.384990
-0.942478	-0.921232	-0.921233
-0.785398	-1.454446	-1.454445
:	:	:
0.785398	-1.454446	-1.454445
0.942478	-0.921232	-0.921233

1.099557	0.384989	0.384990
1.256637	2.563207	2.563206

5.5 用切比雪夫逼近求函数的多项式逼近

1. 功能

给定函数 $f(x)$, $x \in [a, b]$, 利用 $f(x)$ 的切比雪夫逼近级数

$$f(x) \approx T(y) = \sum_{k=1}^n c_k T_{k-1}(y)$$

$$y = \left[x - \frac{1}{2}(b+a) \right] / \left[\frac{1}{2}(b-a) \right]$$

求其普通多项式逼近 $P(x) = \sum_{k=1}^n g_k x^{k-1}$, 使 $P(x) = T(y)$, $x \in [a, b]$.

本节两个子过程 CHEBPC 和 PCSHFT, 前者求 d_k ($k=1, \dots, n$) 使 $\sum_{k=1}^n d_k y^{k-1} = T(y)$, $y \in [-1, 1]$; 后者计算 g_k ($k=1, \dots, n$) 使 $P(x) = \sum_{k=1}^n d_k y^{k-1} = \sum_{k=1}^n g_k x^{k-1}$, $x \in [a, b]$, $y \in [-1, 1]$.

2. 方法

(1) 由克伦肖递推公式定义:

$$b_{n+2} = b_{n+1} = 0$$

$$b_j = 2yb_{j+1} - b_{j+2} + c_j, \quad j = n, n-1, \dots, 2$$

则

$$T(y) = b_0 = yb_2 - b_3 + \frac{1}{2}c_1$$

于是由

$$\sum_{k=1}^n d_k y^{k-1} = yb_2 - b_3 + \frac{1}{2}c_1$$

递推即可得出所求系数.

(2) 由于

$$\sum_{k=1}^n d_k y^{k-1} = \sum_{k=1}^n d_k \left\{ \left[x - \frac{1}{2}(b+a) \right] / \left[\frac{1}{2}(b-a) \right] \right\}^{k-1}$$

$$= \sum_{k=1}^n d_k \left(\frac{2}{b-a} \right)^{k-1} \left[x - \frac{1}{2}(b+a) \right]^{k-1}$$

令

$$\tilde{a} = \frac{1}{2}(b+a), \quad z = x - \tilde{a}, \quad \tilde{d}_k = d_k \left(\frac{2}{b-a} \right)^{k-1}$$

则

$$\sum_{k=1}^n d_k y^{k-1} = \sum_{k=1}^n \tilde{d}_k z^{k-1}$$

$$\sum_{k=1}^n g_k x^{k-1} = \sum_{k=1}^n g_k (z + \tilde{a})^{k-1}$$

而要求 $\sum_{k=1}^n d_k y^{k-1} = \sum_{k=1}^n g_k x^{k-1}$ 知 g_1 即是 $\sum_{k=1}^n \tilde{d}_k z^{k-1}$ 除以 $z + \tilde{a}$ 的余数, 依此类推, 即可得到其余 g_k .

3. 使用说明

(1) CHEBPC(C,D,N)

N 整型变量, 输入参数, 要求逼近的阶数

C N 个元素的一维实型数组, 输入参数, 存放由子过程 CHEBFT 输出的 C

D N 个元素的一维实型数组, 输出参数, 存放多项式 $\sum_{k=1}^n d_k y^{k-1}$ 的系数

(2) PCSHFT(A,B,D,N)

N 整型变量, 输入参数, 多项式系数的个数

A 实型变量, 输入参数, 区间 $[a, b]$ 的 a

B 实型变量, 输入参数, 区间 $[a, b]$ 的 b

D N 个元素的一维实型数组, 输入、输出参数, 输入时存放多项式

$\sum_{k=1}^n d_k y^{k-1} (y \in [-1, 1])$ 的系数 $d_k (k=1, \dots, n)$, 输出时存放多项式 $\sum_{k=1}^n g_k x^{k-1} (x \in [a, b])$ 的系数 $g_k (k=1, \dots, n)$

4. 过程

(1) 子过程 CHEBPC.

```
SUBROUTINE chebpc(c,d,n)
```

```
INTEGER n,NMAX
```

```
REAL c(n),d(n)
```

```
PARAMETER (NMAX=50)
```

```

INTEGER j,k
REAL sv,dd(NMAX)
do j=1,n
    d(j)=0.
    dd(j)=0.
end do
d(1)=c(n)
do j=n-1,2,-1
    do k=n-j+1,2,-1
        sv=d(k)
        d(k)=2.*d(k-1)-dd(k)
        dd(k)=sv
    end do
    sv=d(1)
    d(1)=-dd(1)+c(j)
    dd(1)=sv
end do
do j=n,2,-1
    d(j)=d(j-1)-dd(j)
end do
d(1)=-dd(1)+0.5*c(1)
END SUBROUTINE chebpc

```

(2) 子过程 PCSHFT.

```

SUBROUTINE pcsht(a,b,d,n)
INTEGER n
REAL a,b,d(n)
INTEGER j,k
REAL const,fac
const=2./(b-a)
fac=const
do j=2,n
    d(j)=d(j)*fac
    fac=fac*const
end do
const=0.5*(a+b)
do j=1,n-1

```



```

do k=n-1,j,-1
    d(k)=d(k)-const*d(k+1)
end do
end do
END SUBROUTINE pcshft

```

5. 例子

(1) 验证 CHEBPC 的例子为 $\text{FUNC} = x^2(x^2 - 2)\sin x, x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. 在计算过程中, 首先利用 5.3 节中的子过程 CHEBFT 求得该函数的切比雪夫系数, 然后再调用 CHEBPC 得到关于 y 的多项式系数, 并与精确解进行比较. 验证程序 D5R8 如下:

```

PROGRAM D5R8
! Driver for routine CHEBPC
EXTERNAL FUNC
! USES CHEBFT
PARAMETER (NVAL=40,PIO2=1.5707963)
DIMENSION C(NVAL),D(NVAL)
A=-PIO2
B=PIO2
CALL CHEBFT(A,B,C,NVAL,FUNC)
DO
    WRITE(*,*) 'How many terms in Chebyshev evaluation?'
    WRITE(*, '(1X,A,I2,A)') 'Enter n between 6 and ',NVAL&
        ', '. Enter n=0 to end.'
    READ(*,*) MVAL
    IF ((MVAL<-0).OR.(MVAL>NVAL)) stop
    CALL CHEBPC(C,D,MVAL)
! Test polynomial
    WRITE(*, '(1X,T10,A,T19,A,T28,A)') 'X', 'Actual', &
        'Polynomial'
    DO I=-8,8,1
        X=I*PIO2/10.0
        Y=(X-(0.5*(B+A)))/(0.5*(B-A))
        POLY=D(MVAL)
        DO J=MVAL-1,1,-1

```

```

      POLY = POLY * Y + D(J)
    END DO
    WRITE(*,'(1X,3F12.6)') X, FUNC(X), POLY
  END DO
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC = (X * * 2) * (X * * 2 - 2.0) * SIN(X)
END FUNCTION FUNC

```

计算结果如下:

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

10

X	Actual	Polynomial
-1.256637	0.632073	0.632055
-1.099557	0.852077	0.852096
-0.942478	0.798917	0.798935
-0.785398	0.603301	0.603288
⋮	⋮	⋮
0.785398	-0.603301	-0.603288
0.942478	-0.798917	-0.798935
1.099557	-0.852077	-0.852096
1.256637	-0.632073	-0.632055

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

20

X	Actual	Polynomial
-1.256637	0.632073	0.632073
-1.099557	0.852077	0.852076
-0.942478	0.798917	0.798916
-0.785398	0.603301	0.603301
⋮	⋮	⋮
0.785398	-0.603301	-0.603301
0.942478	-0.798917	-0.798916
1.099557	-0.852077	-0.852076
1.256637	-0.632073	-0.632072

(2) 验证 PCSHFT 的例子和上面验证 CHEBPC 的例子相同, 且验证程序 D5R9 和 D5R8 基本相同, 只是再加入调用了过程 PCSHFT 来直接求多项

式 $\sum_{k=1}^n g_k x^{k-1}$ 的值, 验证程序 D5R9 如下:

```

PROGRAM D5R9
! Driver for routine PCSHFT
EXTERNAL FUNC
! USES CHEBFT,CHEBPC
PARAMETER (NVAL=40,PIO2=1.5707963)
DIMENSION C(NVAL),D(NVAL)
A=-PIO2
B=PIO2
CALL CHEBFT(A,B,C,NVAL,FUNC)
DO
  WRITE(*,*) 'How many terms in Chebyshev evaluation?'
  WRITE(*, '(1X,A,I2,A)') 'Enter n between 6 and ',NVAL&
    ,'. Enter n=0 to end.'
  READ(*,*) MVAL
  IF ((MVAL<=0).OR.(MVAL>NVAL)) STOP
  CALL CHEBPC(C,D,MVAL)
  CALL PCSHFT(A,B,D,MVAL)
! Test shifted polynomial
  WRITE(*, '(1X,T10,A,T19,A,T28,A)') 'X','Actual',&
    'Polynomial'
  DO I=-8,8,1
    X=I*PIO2/10.0
    POLY=D(MVAL)
    DO J=MVAL-1,1,-1
      POLY=POLY*X+D(J)
    END DO
    WRITE(*, '(1X,3F12.6)') X,FUNC(X),POLY
  END DO
END DO
END PROGRAM
FUNCTION FUNC(X)
  FUNC=(X**2)*(X**2-2.0)*SIN(X)
END FUNCTION FUNC

```

计算结果如下:

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

10

X	Actual	Polynomial
-1.256637	0.632073	0.632055
-1.099557	0.852077	0.852096
-0.942478	0.798917	0.798935
-0.785398	0.603301	0.603288
:	:	:
0.785398	-0.603301	-0.603288
0.942478	-0.798917	-0.798935
1.099557	-0.852077	-0.852096
1.256637	-0.632073	-0.632055

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

20

X	Actual	Polynomial
-1.256637	0.632073	0.632072
-1.099557	0.852077	0.852077
-0.942478	0.798917	0.798917
-0.785398	0.603301	0.603301
:	:	:
0.785398	-0.603301	-0.603301
0.942478	-0.798917	-0.798916
1.099557	-0.852077	-0.852076
1.256637	-0.632073	-0.632073

How many terms in Chebyshev evaluation?

Enter n between 6 and 40. Enter n=0 to end.

0

第6章 随 机 数

本章包括常用分布随机数、随机位的产生及用蒙特卡罗方法计算多重积分。

在理论计算中,所加入的随机因素可以采用产生随机数或模拟随机过程的办法来实现。从计算方法本身讲,随机数又可作为蒙特卡罗方法的基本工具。在实际应用中,随机过程模拟及具有任意分布的随机数都由 $(0,1)$ 区间上的均匀分布随机数来实现。因此在6.1节中介绍的是关于 $(0,1)$ 区间上的均匀分布随机数发生器的程序,在这几个程序中,一般情况下,我们推荐大家使用RAN1,它由三个线性同余数发生器联合产生 $(0,1)$ 上的均匀分布随机数,因此,其统计性质较好。有时可能还需考虑运行速度,那么可以使用子过程RAN2。如果使用的计算机中本身带有随机数发生器,这时可利用子过程RAN0来改善这个随机数发生器产生的随机数的统计性质。关于随机数的统计检验,可参考第13章的相关内容。6.4节中介绍的方法对于高级语言不是很有用,但对于能进入机器语言或能容易有效地建立专门的硬件时,它可能是相当有用的。应该注意,不要把从这些程序中得到的随机位序列作为一个大的、想象的随机整数,也不能作为一个想象的浮点随机数的尾数。

蒙特卡罗方法算法比较简单,便于在计算机上编制程序,它的收敛速度与维数无关,因此对于解决高维问题比较适宜。本章仅以特殊的三重积分为例说明方法,读者不难编制自己的程序。

6.1 均匀分布随机数

1. 功能

本节给出四个函数过程,均产生 $(0,1)$ 区间上的均匀分布随机数。

子过程RAN0用于改善系统(计算机的程序库中)提供的均匀分布随机数发生器RAN。它改变RAN产生的随机数列的序列相关性。

子过程RAN1,RAN2,RAN3是三个袖珍型均匀分布随机数发生器。RAN1联合使用三个线性同余数发生器产生 $(0,1)$ 区间上的一个均匀分布随机数,其统计规律明显优于单独使用一个线性同余数发生器,实用上,可认为它产生的随机数列的周期是无穷的;RAN2用一个线性同余数发生器产生 $(0,1)$ 区间上的一个

均匀分布随机数,其速度较快;RAN3 用减法产生(0,1)区间上的一个均匀分布随机数.

2. 方法

(1) 改善系统提供的随机数发生器.

设 RAN 为系统程序库中的一个随机数发生器.它是一个函数子过程,一般是用线性同余数法,即采用递推关系

$$I_{j+1} = aI_j + c(\bmod m), \quad r_{j+1} = I_{j+1}/m$$

其中 m, a, c 为常数,均为正整数.

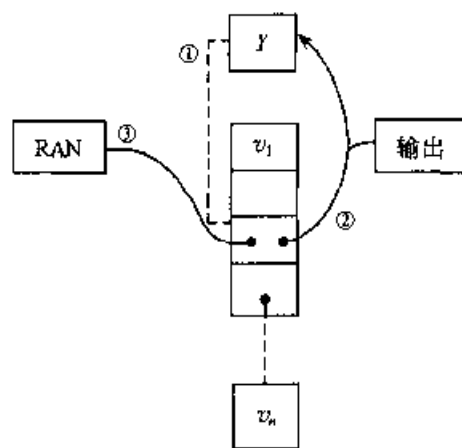


图 6.1 混洗过程,①,②,
③表示混洗顺序

线性同余数法的优点是计算量少,速度快,缺点是在逐次调用中产生的随机数是序列相关的.为破坏这种序列相关性,采用如下方法:设 v_1, v_2, \dots, v_n 是由 RAN 生成的 n 个随机数,现随机地取一正整数 j ($1 \leq j \leq n$),取 v_j 为一要求的随机数,而 v_j 再由 RAN 生成的另一随机数替换,替换后再由 v_1, v_2, \dots, v_n 中随机地取一个为下一次要求的随机数,依此重复.这种过程称为混洗(shuffling)过程, $v = (v_1, \dots, v_n)$ 称为混洗数组,如图 6.1 所示.

(2) 线性同余数法的联合使用.

设用三个线性同余数法产生(0,1)区间上的均匀分布随机数,则第一个线性同余数用于产生随机数的最高有效位部分,第二个用于产生随机数的最低有效位部分,第三个用于控制混洗过程,其中线性同余数法中的常数如表 6.1 所示.

(3) 减去法.

这里我们采用克努斯(Knuth)的建议,给定一个整数上界 M ,并随机地取 55 个正整数 a_1, \dots, a_{55} ,均是 1 与 M 之间的数,然后再通过多次互相减的方法搅乱这 55 个整数,一旦在减的过程中有某个元小于 0,则用这个元加上 M .最后在 1 与 55 之间随机地取两个正整数 i, j ,满足 $|i - j| = 31$,定义 $r = (a_i - a_j)/M$,若 $a_i - a_j < 0$,则定义 $r = (a_i - a_j + M)/M$, r 即为要求的(0,1)区间上的一个均匀分布随机数.

表 6.1 线性同余数法中的最佳常数

Constants for Portable Random Number Generators							
overflow at	m	a	c	overflow at	m	a	c
2^{20}	6075	106	1283	2^{28}	117128	1277	24749
2^{21}	7875	211	1663		312500	741	66037
2^{22}	7875	421	1663		121500	2041	25673
2^{23}	11979	430	2531	2^{29}	120050	2311	25367
	6655	936	1399		214326	1807	45289
	6075	1366	1283		244944	1597	51749
2^{24}	53125	171	11213		233280	1861	49297
	11979	859	2531		175000	2661	36979
	29282	419	6173		121500	4081	25673
	14406	967	3041		145800	3661	30809
2^{25}	134456	141	28411	2^{30}	139968	3877	29573
	31104	625	6571		214326	3613	45289
	14000	1541	2957		714025	1366	150889
	12960	1741	2731	2^{31}	134456	8121	28411
	21870	1291	4621		243000	4561	51349
	139968	205	29573		259200	7141	54773
2^{26}	81000	421	17117	2^{32}	233280	9301	49297
	29282	1255	6173		714025	4096	150889
	134456	282	28411	2^{33}	1771875	2416	374441
2^{27}	86436	1093	18257	2^{34}	510300	17221	107839
	259200	421	54773		312500	36261	66037
	116640	1021	24631	2^{35}	217728	84589	45989
	121500	1021	25673				

3. 使用说明

RAN0(IDUM), RAN1(IDUM), RAN2(IDUM), RAN3(IDUM)

IDUM 整型变量,输入、输出参数,第一次调用该过程时输入一个负整数,为产生随机数提供初值,输出时除 RAN2 外,其余输出值均为 1, RAN2 中的是一个小于 $M=714025$ 的正整数,为产生下一个随机数提供初值

4. 过程

(1) 函数过程 RAN0.

```

FUNCTION ran0(idum)
INTEGER idum,IA,IM,IQ,IR,MASK
REAL ran0,AM
PARAMETER (IA=16807,IM=2147483647,AM=1./IM,&
            IQ=127773,IR=2836,MASK=123459876)
INTEGER k
idum=ieor(idum,MASK)
k=idum/IQ
idum=IA*(idum-k*IQ)-IR*k
if (idum<0) idum=idum+IM
ran0=AM*idum
idum=ieor(idum,MASK)
END FUNCTION ran0

```

(2) 函数过程 RAN1.

```

FUNCTION ran1(idum)
INTEGER idum,ia,im,iq,ir,ntab,ndiv
REAL ran1,am,eps,rnmix
PARAMETER (ia=16807,im=2147483647,am=1./im,&
            iq=127773,ir=2836,ntab=32,ndiv=1+(im-1)/ntab,&
            eps=1.2e-7,rnmix=1.-eps)
INTEGER j,k,iv(ntab),iy
SAVE iv,iy
DATA iv /ntab*0/, iy /0/
if (idum<=0. or. iy/=0) then
    idum=max(-idum,1)
    do j=ntab+8,1,-1
        k=idum/iq
        idum=ia*(idum-k*iq)-ir*k
        if (idum<0) idum=idum+im
        if (j<=ntab) iv(j)=idum
    end do
    iy=iv(1)

```



```

endif
k=idum/iq
idum=ia * (idum-k * iq)-ir * k
if (idum<0) idum=idum+im
j=1+iy/ndiv
iy=iv(j)
iv(j)=idum
ran1=min(am * iy,rnmX)
END FUNCTION ran1

```

(3) 函数过程 RAN2.

```

FUNCTION ran2(idum)
INTEGER idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,&
    IR1,IR2,NTAB,NDIV
REAL ran2,AM,EPS,RNMX
PARAMETER (IM1=2147483563,IM2=2147483399,AM=1./IM1,&
    IMM1=IM1-1,IA1=40014,IA2=40692,IQ1=53668,&
    IQ2=52774,IR1=12211,IR2=3791,NTAB=32,&
    NDIV=1+IMM1/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
INTEGER idum2,j,k,iv(NTAB),iy
SAVE iv,iy,idum2
DATA idum2/123456789/, iv/NTAB * 0/, iy/0/
if (idum<=0) then
    idum=max(-idum,1)
    idum2=idum
    do j=NTAB+8,1,-1
        k=idum/IQ1
        idum=IA1 * (idum-k * IQ1)-k * IR1
        if (idum<0) idum=idum+IM1
        if (j<=NTAB) iv(j)=idum
    end do
    iy=iv(1)
endif
k=idum/IQ1
idum=IA1 * (idum-k * IQ1)-k * IR1
if (idum<0) idum=idum+IM1
k=idum2/IQ2

```

```

idum2=IA2 * (idum2-k * IQ2)-k * IR2
if (idum2<0) idum2=idum2+IM2
j=1+iy/NDIV
iy=iv(j)-idum2
iv(j)=idum
if(iy<1) iy=iy+IMM1
ran2=min(AM * iy,RNMX)
END FUNCTION ran2

```

(4) 函数过程 RAN3.

```

FUNCTION ran3(idum)
INTEGER idum
INTEGER MBIG,MSEED,MZ
! REAL MBIG,MSEED,MZ
REAL ran3,FAC
PARAMETER (MBIG=1000000000,MSEED=161803398,MZ=0,FAC=1./MBIG)
! PARAMETER (MBIG=4000000.,MSEED=1618033.,MZ=0.,FAC=1./MBIG)
INTEGER i,iff,ii,inext,inextp,k
INTEGER mj,mk,ma(55)
! REAL mj,mk,ma(55)
SAVE iff,inext,inextp,ma
DATA iff /0/
if(idum<0.or.iff==0) then
    iff=1
    mj=MSEED-iabs(idum)
    mj=mod(mj,MBIG)
    ma(55)=mj
    mk=1
    do i=1,54
        ii=mod(21*i,55)
        ma(ii)=mk
        mk=mj-mk
        if(mk<MZ) mk=mk+MBIG
        mj=ma(ii)
    end do
    do k=1,4
        do i=1,55

```

```

      ma(i)=ma(i)-ma(1+mod(i+30,55))
      if(ma(i)<MZ) ma(i)=ma(i)+MBIG
    end do
  end do
  inext=0
  inextp=31
  idum=1
endif
inext=inext+1
if(inext==56) inext=1
inextp=inextp+1
if(inextp==56) inextp=1
mj=ma(inext)-ma(inextp)
if(mj<MZ) mj=mj+MBIG
ma(inext)=mj
ran3=mj*FAC
END FUNCTION ran3

```

5. 例子

验证程序 D6R1 到 D6R4 除了每个程序分别调用不同的随机数发生器 (RAN0 到 RAN3) 以外, 实际上其余都是相同的. 首先, 它们用随机数发生器连续取出四个随机数 X_1, \dots, X_4 . 然后, 将这些数作为一个点的坐标. 例如, 取 (X_1, X_2) 作为二维空间中的一个点, (X_1, X_2, X_3) 作为三维空间中的一个点……这些点都在各自的 n 维空间的单位 n 面体中. 但是, 它们可以在该空间的单位球体内或者在球体外. 当 $n=2, 3, 4$ 时, 我们寻找一个点在球体内的概率. 这个数在理论上是很容易计算的. 当 $n=2$ 时, 为 $\pi/4$; 当 $n=3$ 时, 为 $\pi/6$; 当 $n=4$ 时, 为 $\pi^2/32$. 如果随机数发生器没有毛病的话, 由此产生的点将以一定的比例落入单位球体内, 且结果随点数的增加而提高精度. 在验证程序 D6R1 到 D6R4 中, 我们为方便起见取 2^n 为因子, 且采用随机发生器去确定值 $\pi, 4\pi/3$ 和 $\pi^2/2$ 的统计均值.

(1) 验证 RAN0 的程序 D6R1 如下:

```

PROGRAM D6R1
! Driver for routine RAN0
! Calculates pi statistically using volume of unit n-sphere
PARAMETER (PI=3.1415926)
DIMENSION IY(3), YPROB(3)

```

```

FNC(X1,X2,X3,X4)=SQRT(X1 * * 2+X2 * * 2+X3 * * 2+X4 * * 2)
IDUM=-1
DO I=1,3
    IY(I)=0
END DO
WRITE(*,'(1X,/,T15,A)') 'Volume of unit n-sphere, n=2,3,4'
WRITE(*,'(1X,/,T3,A,T17,A,T26,A,T37,A)') &
    '# points', 'PI', '(4/3) * PI', '(1/2) * PI ^ 2'
DO J=1,15
    DO K=2 * * (J-1),2 * * J
        X1=RAN0(IDUM)
        X2=RAN0(IDUM)
        X3=RAN0(IDUM)
        X4=RAN0(IDUM)
        IF(FNC(X1,X2,0.0,0.0)<1.0) IY(1)=IY(1)+1
        IF(FNC(X1,X2,X3,0.0)<1.0) IY(2)=IY(2)+1
        IF(FNC(X1,X2,X3,X4)<1.0) IY(3)=IY(3)+1
    END DO
    DO I=1,3
        YPROB(I)=1.0 * (2 * * (I+1)) * IY(I)/(2 * * J)
    END DO
    WRITE(*,'(1X,I8,3F12.6)') 2 * * J,(YPROB(I),I=1,3)
END DO
WRITE(*,'(1X,/,T4,A,3F12.6/)') 'actual',PI,&
    4.0 * PI/3.0,0.5 * (PI * * 2)
END

```

计算结果如下：

Volume of unit n sphere, n=2,3,4			
# points	PI	(4/3) * PI	(1/2) * PI ^ 2
2	2.000000	0.000000	0.000000
4	3.000000	4.000000	8.000000
8	2.500000	4.000000	6.000000
16	3.250000	5.000000	4.000000
32	3.500000	4.500000	4.500000
64	3.187500	4.125000	5.250000
128	3.187500	4.250000	4.625000

256	3.093750	4.000000	4.437500
512	3.031250	3.921875	4.531250
1024	3.074219	3.898438	4.750000
2048	3.121094	4.113281	5.078125
4096	3.108398	4.103516	5.000000
8192	3.105469	4.097656	4.964844
16384	3.119873	4.139160	4.976563
32768	3.128906	4.156738	4.922852
actual	3.141593	4.188790	4.934802

(2) 验证 RAN1 的程序 D6R2 如下:

```

PROGRAM D6R2
! Driver for routine RAN1
! Calculates pi statistically using volume of unit n-sphere
PARAMETER (PI=3.1415926)
DIMENSION IY(3),YPROB(3)
FNC(X1,X2,X3,X4)=SQRT(X1**2+X2**2+X3**2+X4**2)
IDUM=-1
DO I=1,3
    IY(I)=0
END DO
WRITE(*, '(1X,/,T15,A)') 'Volume of unit n-sphere, n=2,3,4'
WRITE(*, '(1X,/,T3,A,T17,A,T26,A,T37,A)') &
    ' # points', 'PI', '(4/3)*PI', '(1/2)*PI^2'
DO J=1,15
    DO K=2*(J-1),2*J
        X1=RAN1(IDUM)
        X2=RAN1(IDUM)
        X3=RAN1(IDUM)
        X4=RAN1(IDUM)
        IF(FNC(X1,X2,0,0).LT.1.0) IY(1)=IY(1)+1
        IF(FNC(X1,X2,X3,0).LT.1.0) IY(2)=IY(2)+1
        IF(FNC(X1,X2,X3,X4).LT.1.0) IY(3)=IY(3)+1
    END DO
    DO I=1,3
        YPROB(I)=1.0*(2*(I+1))*IY(I)/(2*J)
    END DO

```

```

      WRITE(*, '(1X,I8,3F12.6)') 2 * J, (YPROB(I), I=1,3)
    END DO
    WRITE(*, '(1X,/,T4,A,3F12.6/)' ) 'actual', PI, &
      4.0 * PI/3.0, 0.5 * (PI * * 2)
  END

```

计算结果如下:

Volume of unit n-sphere, n=2,3,4			
# points	PI	(4/3) * PI	(1/2) * PI ²
2	2.000000	4.000000	0.000000
4	3.000000	2.000000	0.000000
8	2.500000	3.000000	4.000000
16	3.250000	4.500000	6.000000
32	3.375000	5.250000	6.000000
64	3.125000	4.125000	4.750000
128	3.281250	4.687500	5.375000
256	3.171875	4.312500	5.000000
512	3.226563	4.296875	4.812500
1024	3.261719	4.390625	5.031250
2048	3.152344	4.246094	4.976563
4096	3.125000	4.166016	4.921875
8192	3.139160	4.238281	5.023438
16384	3.128662	4.202637	4.930664
32768	3.141724	4.182617	4.902344
actual	3.141593	4.188790	4.934802

(3) 验证 RAN2 的程序 D6R3 如下:

```

PROGRAM D6R3
! Driver for routine RAN2
! Calculates pi statistically using volume of unit n sphere
PARAMETER (PI=3.1415926)
DIMENSION IY(3), YPROB(3)
FNC(X1,X2,X3,X4)=SQRT(X1 * * 2+X2 * * 2+ X3 * * 2+X4 * * 2)
IDUM=-1
DO I=1,3
  IY(I)=0

```

```

END DO
WRITE(*, '(1X,/,T15,A)') 'Volume of unit n-sphere, n=2,3,4'
WRITE(*, '(1X,/,T3,A,T17,A,T26,A,T37,A)') &
    '# points', 'PI', '(4/3) * PI', '(1/2) * PI ^ 2'
DO J=1,15
    DO K=2 * * (J-1), 2 * * J
        X1=RAN2(IDUM)
        X2=RAN2(IDUM)
        X3=RAN2(IDUM)
        X4=RAN2(IDUM)
        IF(FNC(X1,X2,0.0,0.0)<1.0) IY(1)=IY(1)+1
        IF(FNC(X1,X2,X3,0.0)<1.0) IY(2)=IY(2)+1
        IF(FNC(X1,X2,X3,X4)<1.0) IY(3)=IY(3)+1
    END DO
    DO I=1,3
        YPROB(I)=1.0 * (2 * * (I+1)) * IY(I) / (2 * * J)
    END DO
    WRITE(*, '(1X,I8,3F12.6)') 2 * * J, (YPROB(I), I=1,3)
END DO
WRITE(*, '(1X,/,T4,A,3F12.6/))' 'actual', PI, &
    4.0 * PI/3.0, 0.5 * (PI * * 2)
END

```

计算结果如下:

Volume of unit n-sphere, n=2,3,4			
# points	PI	(4/3) * PI	(1/2) * PI ^ 2
2	4.000000	4.000000	8.000000
4	5.000000	6.000000	12.000000
8	4.500000	5.000000	10.000000
16	4.250000	5.000000	7.000000
32	3.875000	4.250000	5.500000
64	3.562500	3.875000	5.500000
128	3.375000	3.812500	4.625000
256	3.328125	3.968750	4.750000
512	3.273438	4.109375	5.156250
1024	3.226563	4.085938	4.875000
2048	3.220703	4.156250	4.796875

4096	3.166016	4.089844	4.746094
8192	3.160645	4.115234	4.744141
16384	3.149902	4.131348	4.792969
32768	3.151489	4.167969	4.876953
actual	3.141593	4.188790	4.934802

(4) 验证 RAN3 的程序 D6R4 如下:

```

PROGRAM D6R4
1 Driver for routine RAN3
1 Calculates pi statistically using volume of unit n-sphere
PARAMETER (PI=3.1415926)
DIMENSION IY(3),YPROB(3)
FNC(X1,X2,X3,X4)=SQRT(X1**2+X2**2+X3**2+X4**2)
IDUM=-1
DO I=1,3
    IY(I)=0
END DO
WRITE(*, '(1X,/,T15,A)') 'Volume of unit n-sphere, n=2,3,4'
WRITE(*, '(1X,/,T3,A,T17,A,T26,A,T37,A)') &
    '# points', 'PI', '(4/3)*PI', '(1/2)*PI^2'
DO J=1,15
    DO K=2*(J-1),2*J
        X1=RAN3(IDUM)
        X2=RAN3(IDUM)
        X3=RAN3(IDUM)
        X4=RAN3(IDUM)
        IF(FNC(X1,X2,0.0,0.0)<1.0) IY(1)=IY(1)+1
        IF(FNC(X1,X2,X3,0.0)<1.0) IY(2)=IY(2)+1
        IF(FNC(X1,X2,X3,X4)<1.0) IY(3)=IY(3)+1
    END DO
    DO I=1,3
        YPROB(I)=1.0*(2*(I+1))*IY(I)/(2*J)
    END DO
    WRITE(*, '(1X,I8,3F12.6)') 2*J,(YPROB(I),I=1,3)
END DO
WRITE(*, '(1X,/,T4,A,3F12.6/)') 'actual',PI,&
    4.0*PI/3.0,0.5*(PI**2)

```


END

计算结果如下:

Volume of unit n-sphere, n=2,3,4			
# points	PI	(4/3) * PI	(1/2) * PI ²
2	4.000000	4.000000	0.000000
4	5.000000	4.000000	4.000000
8	3.500000	4.000000	6.000000
16	3.500000	4.500000	6.000000
32	3.500000	4.000000	5.500000
64	3.375000	4.500000	4.750000
128	3.375000	4.437500	5.250000
256	3.234375	4.343750	5.062500
512	3.257813	4.281250	4.906250
1024	3.230469	4.265625	4.937500
2048	3.185547	4.089844	4.773438
4096	3.182617	4.167969	4.691406
8192	3.149414	4.129883	4.726363
16384	3.151855	4.149414	4.820313
32768	3.139893	4.130859	4.854004
actual	3.141593	4.188790	4.934802

6.2 变换方法——指数分布和正态分布随机数

1. 功能

用变换方法产生服从指数分布的随机数和服从标准正态分布的随机数.

函数过程 EXPDEV 产生一个具有单位均值的正指数分布随机数.

函数过程 GASDEV 产生一个标准正态分布随机数.

2. 方法

(1) 定理.

定理 设 m 维随机向量 ξ 有密度 $f_{\xi}(x)$, 函数 $y=g(x)$ ($x, y \in R^m$) 满足:

(a) $y=g(x)$ 有惟一反函数 $x=h(y)$;

(b) $y=g(x)$ 和 $x=h(y)$ 连续;

(c) 雅可比行列式 $J = \frac{dx}{dy} = \frac{\partial(x_1, \dots, x_n)}{\partial(y_1, \dots, y_n)}$ 存在且连续. 则 m 维随机向量 $\eta = g(\xi)$ 是连续型的, 其密度为

$$f_{\eta}(y) = \begin{cases} f_{\xi}[h(y)] \cdot |h'(y)| & (y \in G) \\ 0 & (y \notin G) \end{cases}$$

其中 G 是函数 $y=g(x)$ ($x \in R^m$) 的值域.

(2) 指数分布.

作变换 $y=g(x)=-\ln(x)$, 则当 x 为均匀分布随机数时, y 是密度为 $p(y)=e^{-y}$ 的指数分布随机数.

(3) 正态分布.

设 x_1, x_2 是 $(0, 1)$ 上的均匀分布随机数, 作变换

$$\begin{aligned} y_1 &= \sqrt{-2\ln x_1} \cos 2\pi x_2 \\ y_2 &= \sqrt{-2\ln x_1} \sin 2\pi x_2 \end{aligned}$$

等价地

$$\begin{aligned} x_1 &= \exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] \\ x_2 &= \frac{1}{2\pi} \operatorname{arctg} \frac{y_2}{y_1} \end{aligned}$$

由定理, y_1, y_2 是两个独立的标准正态分布随机数.

进一步, 由于三角函数的计算量大, 因此, 把取 x_1, x_2 换为在单位圆 (中心在原点) 内取随机点 (v_1, v_2) , 由 $r = v_1^2 + v_2^2$ 代替 x_1 , 而该点与 v_1 轴的角 θ 代替 $2\pi x_2$, 则

$$v_1 = \sqrt{r} \cos \theta, \quad v_2 = \sqrt{r} \sin \theta$$

实际上有结论 (Box-Muller):

设 x_1, x_2 是 $(0, 1)$ 区间上的独立的均匀随机数, $v_1 = 2x_1 - 1$, $v_2 = 2x_2 - 1$, $r = v_1^2 + v_2^2$, 显然, v_1, v_2 是 $(-1, 1)$ 上的均匀随机数, 若 $r \leq 1$, 则

$$\begin{aligned} y_1 &= v_1 [(-2\ln r)/r]^{1/2} \\ y_2 &= v_2 [(-2\ln r)/r]^{1/2} \end{aligned}$$

是两个独立的标准正态分布随机数.

3. 使用说明

FUNCTION EXPDEV (IDUM)

FUNCTION GASDEV (IDUM)

IDUM 整型变量,输入、输出参数,输入时为调用程序提供初值,其输入值为一负整数,输出值为 1,为下一次调用该子程序提供参数值

4. 过程

(1) 函数过程 EXPDEV.

```

FUNCTION expdev(idum)
INTEGER idum
REAL expdev
! USES ranl
REAL dum,ranl
do
    dum=ranl(idum)
    if(.not. dum = 0.) exit
end do
expdev=-log(dum)
END FUNCTION expdev

```

(2) 函数过程 GASDEV.

```

FUNCTION gasdev(idum)
INTEGER idum
REAL gasdev
! USES ranl
INTEGER iset
REAL fac,gset,rsq,v1,v2,ranl
SAVE iset,gset
DATA iset/0/
if (iset==0) then
    do
        v1=2. * ranl(idum)-1.
        v2=2. * ranl(idum)-1.
        rsq=v1 * * 2+v2 * * 2
        if(.not. rsq > -1. .or. rsq==0.) exit
    end do
    fac=sqrt(-2. * log(rsq)/rsq)
    gset=v1 * fac

```

```

      gasdev=v2*fac
      iset=1
    else
      gasdev=gset
      iset=0
    endif
  END FUNCTION gasdev

```

5. 例子

(1) 为了验证子过程 EXPDEV, 在验证程序 D6R5 中, 我们调用了 EXPDEV 一万次, 且视其结果将数组 $X(J)$ 中某一元素向上累加 1. 然后, 将该数组中所有元素的值求和 TOTAL. 由于某一元素可能数值太大, 我们将 $X(I)$ 除以总和 TOTAL 得到其比例. 最后将计算结果与相似的规范化的指数结果 EXPECT 进行比较. 验证程序 D6R5 如下:

```

PROGRAM D6R5
  ! Driver for routine EXPDEV
  PARAMETER (NPTS=10000,EE=2.718281828)
  DIMENSION TRIG(21),X(21)
  DO I=1,21
    TRIG(I)=(I-1)/20.0
    X(I)=0.0
  END DO
  IDUM=-1
  DO I=1,NPTS
    Y=EXPDEV(IDUM)
    DO J=2,21
      IF((Y<TRIG(J)).AND.(Y>TRIG(J-1))) THEN
        X(J)=X(J)+1.0
      ENDIF
    END DO
  END DO
  TOTAL=0.0
  DO I=2,21
    TOTAL=TOTAL+X(I)
  END DO
  WRITE(*, '(1X,A,I6,A)') 'Exponential distribution with', &

```

```

      NPTS,' points:'
WRITE(*, '(1X,T5,A,T19,A,T31,A)')&
      'interval','observed','expected'
DO I=2,21
  X(I)=X(I)/TOTAL
  EXPECT=EXP(-(TRIG(I-1)+TRIG(I))/2.0)
  EXPECT=EXPECT*0.05*EE/(EE-1.)
  WRITE(*, '(1X,2F6.2,2F12.4)')&
      TRIG(I-1),TRIG(I),X(I),EXPECT
END DO
END

```

计算结果如下:

Exponential distribution with 10000 points:

interval	observed	expected
0.00 0.05	0.0826	0.0771
0.05 0.10	0.0696	0.0734
0.10 0.15	0.0716	0.0698
0.15 0.20	0.0645	0.0664
0.20 0.25	0.0625	0.0632
0.25 0.30	0.0614	0.0601
0.30 0.35	0.0521	0.0572
0.35 0.40	0.0526	0.0544
0.40 0.45	0.0529	0.0517
0.45 0.50	0.0457	0.0492
0.50 0.55	0.0504	0.0468
0.55 0.60	0.0438	0.0445
0.60 0.65	0.0426	0.0423
0.65 0.70	0.0448	0.0403
0.70 0.75	0.0416	0.0383
0.75 0.80	0.0338	0.0364
0.80 0.85	0.0334	0.0347
0.85 0.90	0.0341	0.0330
0.90 0.95	0.0272	0.0314
0.95 1.00	0.0328	0.0298

(2) 为了验证子过程 GASDEV,在验证程序 D6R6 中,我们调用了函数过

程 GASDEV 一万次,并将其结果分进 21 个区段里.为了论证正态分布的钟形状特征,我们简单地以图形表示这些区段.虽然很简略,但快速完整地验证了该过程.验证程序 D6R6 如下:

```

PROGRAM D6R6
  ! Driver for routine GASDEV
  PARAMETER (N=20,NP1=N+1,NOVER2=N/2,NPTS=10000,&
             ISCAL=400,LLEN=50)
  DIMENSION DIST(NP1)
  CHARACTER TEXT(50)*1
  IDUM=-13
  DO J=1,NP1
    DIST(J)=0.0
  END DO
  DO I=1,NPTS
    J=NINT(0.25*N*GASDEV(IDUM))+NOVER2+1
    IF ((J>=1).AND.(J<=NP1)) DIST(J)=DIST(J)+1
  END DO
  WRITE(*, '(1X,A,I6,A)') &
    'Normally distributed deviate of ',NPTS,' points'
  WRITE(*, '(1X,T6,A,T14,A,T23,A)') 'x','p(x)','graph:'
  DO J=1,NP1
    DIST(J)=DIST(J)/NPTS
    DO K=1,50
      TEXT(K)=' '
    END DO
    KLIM=INT(ISCAL*DIST(J))
    IF (KLIM>LLEN) KLIM=LLEN
    DO K=1,KLIM
      TEXT(K)='*'
    END DO
    X=FLOAT(J)/(0.25*N)
    WRITE(*, '(1X,F7.2,F10.4,4X,50A1)') &
      X,DIST(J),(TEXT(K),K=1,50)
  END DO
END

```

计算结果如下:

Normally distributed deviate of 10000 points

x	p(x)	graph:
0.20	0.0115	* * *
0.40	0.0176	* * * * *
0.60	0.0213	* * * * * * * * * *
0.80	0.0291	* * * * * * * * * *
1.00	0.0403	* * * * * * * * * * * * * *
1.20	0.0479	* * * * * * * * * * * * * *
1.40	0.0591	* * * * * * * * * * * * * *
1.60	0.0674	* * * * * * * * * * * * * *
1.80	0.0769	* * * * * * * * * * * * * *
2.00	0.0730	* * * * * * * * * * * * * *
2.20	0.0795	* * * * * * * * * * * * * *
2.40	0.0799	* * * * * * * * * * * * * *
2.60	0.0751	* * * * * * * * * * * * * *
2.80	0.0696	* * * * * * * * * * * * * *
3.00	0.0573	* * * * * * * * * * * * * *
3.20	0.0476	* * * * * * * * * * * * * *
3.40	0.0391	* * * * * * * * * * * * *
3.60	0.0293	* * * * * * * * * *
3.80	0.0208	* * * * * * * *
4.00	0.0154	* * * * * *
4.20	0.0101	* * * *

6.3 舍选法—— Γ 分布、泊松分布和二项式分布随机数

1. 功能

用舍选法产生 Γ 分布、泊松(Poisson)分布、二项式分布随机数。

函数过程 GAMDEV 产生整数阶 Γ 分布随机数;函数过程 POIDEV 产生泊松分布随机数;函数过程 BNLDEV 产生二项式分布随机数。

2. 方法

舍选法基于一个简单的几何事实:

设 $p(x)$ 是要生成的随机数所服从分布的密度函数. 在二维空间中画出 $p(x)$ 的图像, 如果能产生二维随机点 (x, y) , 使其均匀地分布在 $p(x)$ 曲线下方的区域中, 则相应的 x 值有所要求的分布.

为产生二维随机点, 现设有另一曲线函数 $f(x) \geq p(x), x \in [a, b]$, 其中 $[a, b]$ 是 $p(x)$ 的定义域. 假定 $f(x)$ 的不定积分解析可逆, $A = \int_a^b f(x) dx$ 有限 (由 $\int_a^b p(x) dx = 1$), 则称 $f(x)$ 为比较函数. 取均匀随机数 $r \in (0, A)$, 则 $r = \int_a^x f(x) dx$, 对某个 x , 即 $x = F^{-1}(r)$, 其中 $F(x) = \int_a^x f(x) dx$, 然后再取均匀随机数 $y \in (0, f(x))$, 则 (x, y) 位于曲线 $f(x)$ 的下方区域中. 如果 (x, y) 在 $p(x)$ 的下方, 则接受这个点, 否则舍弃点 (x, y) , 重复之.

等价地, 可取 $(0, 1)$ 上的均匀随机数 r_0 , 若 $r_0 < p(x)/f(x)$, 则接受 x , 否则就舍弃 x , 重复之.

(1) Γ 分布.

整数阶 α 的 Γ 分布的密度函数为

$$p_\alpha(x) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)}, \quad x > 0$$

比较函数取为

$$f(x) = c_0 / [1 + (x - x_0)^2 / a_0^2]$$

于是, x 由下式产生:

$$x = a_0 \operatorname{tg}(\pi u) + x_0$$

其中 u 为 $(0, 1)$ 区间上的均匀随机数. 这里选择常数 a_0, c_0, x_0 使 $f(x) \geq p(x)$, $x \in (-\infty, +\infty)$, 且使 a_0, c_0 尽可能小. 如果 $x < 0$ 则舍弃.

对于较小的整数 α , 采用如下直接方法: 首先产生 α 个 $(0, 1)$ 区间上的均匀随机数 r_1, \dots, r_α , 则 $x = -\ln(\prod_{i=1}^{\alpha} r_i)$ 为 Γ 分布随机数. 这里直接法基于下述结论:

若 ξ, η 是两个独立的 Γ 分布随机变量, 其密度函数分别为 $p_{\alpha_1}(x), p_{\alpha_2}(x)$, 则 $\xi + \eta$ 是密度为 $p_{\alpha_1 + \alpha_2}(x)$ 的 Γ 分布随机变量.

(2) 泊松分布.

均值为 λ 的泊松分布概率函数是

$$p_\lambda(m) = \frac{\lambda^m}{m!} e^{-\lambda}, \quad m \text{ 为整数 } \geq 0$$

这是一个离散分布. 为此令

$$q_\lambda(x) = \frac{\lambda^{[x]} e^{-\lambda}}{[x]!}, \quad x \geq 0$$

其中 $[x]$ 表示 x 取整. 如果用舍选法产生 $q_\lambda(x)$ 的随机数, 然后取它的整数部分, 则将是 $p_\lambda(m)$ 的随机数.

同 Γ 分布, 取比较函数为

$$f(x) = c_0 / [1 + (x - x_0)^2 / a_0]$$

如果 λ 为小值, 则应用直接法.

定理 设有一串 $(0, 1)$ 区间上的均匀随机数 $r_i, i=0, 1, \dots, \lambda > 0$, 若关系式

$$\prod_{i=0}^m r_i \geq e^{-\lambda} > \prod_{i=0}^{m+1} r_i$$

成立, 则 m 是一个以 λ 为均值的泊松分布随机数.

(3) 二项式分布.

二项式分布的概率函数是

$$p_{n,q}(m) = \binom{n}{m} q^m (1-q)^{n-m}$$

其中 n 是伯努利试验的重数, q 为每次试验成功的概率, m 是 n 重伯努利试验中成功的次数.

对于较小的试验重数 n , 用直接法: 取 n 个 $(0, 1)$ 区间上的均匀随机数 $r_i (i=1, \dots, n)$, 若 $r_i \leq q$, 则令 $r_i=1$, 否则令 $r_i=0$, 则

$$m = \sum_{i=1}^n r_i$$

为二项式分布随机数.

如果 n 较大而 nq 很小, 则有下列结论:

泊松定理 如果 q 满足 $nq \rightarrow \lambda > 0 (n \rightarrow \infty)$, 则对于任意正整数 $m \geq 1$, 有

$$\lim_{n \rightarrow \infty} p_{n,q}(m) = \frac{\lambda^m}{m!} e^{-\lambda}$$

舍选法的比较函数同泊松分布即同 Γ 分布.

3. 使用说明

(1) GAMDEV (IA, IDUM)

IA 整型变量, 输入参数, 存放整数阶 Γ 分布的阶数

IDUM 整型变量, 输入、输出参数, 输入时为产生随机数提供初值, 第一次调用该程序时输入一个负整数, 输出时其值为1, 为下一次调用该程序提供初值

(2) POIDEV (XM, IDUM)

XM 实型变量, 输入参数, 存放泊松分布的均值

IDUM 同子程序 GAMDEV

(3) BNLDEV (PP, N, IDUM)

N 整型变量, 输入参数, 试验的重数

PP 实型变量, 输入参数, 存放每次试验成功的概率

IDUM 同子程序 GAMDEV

4. 过程

(1) 函数过程 GAMDEV.

```

FUNCTION gamdev(ia,idum)
INTEGER ia,idum
REAL gamdev
! USES ranl
INTEGER j
REAL am,e,s,v1,v2,x,y,ranl
if(ia<1) pause 'bad argument in gamdev'
if(ia<6) then
    x=1.
    do j=1,ia
        x=x * ranl(idum)
    end do
    x=-log(x)
else
    do
        do
            do
                v1=ranl(idum)
                v2=2. * ranl(idum)-1.
                if(.not. v1 * * 2+v2 * * 2>1.) exit
            end do
            y=v2/v1
            am=ia-1
            s=sqrt(2. * am+1.)
            x=s * y+am
            if(.not. x<=0.) exit
        end do
    end do

```

```

      e=(1.+y**2)*exp(am*log(x/am)-s*y)
      if(.not.ran1(idum)>e) exit
    end do
  endif
  gamdev=x
END FUNCTION gamdev

```

(2) 函数过程 POIDEV.

```

FUNCTION poidev(xm,idum)
INTEGER idum
REAL poidev,xm,PI
PARAMETER (PI=3.141592654)
! USES gammln,ran1
REAL alxm,cm,g,oldm,sq,t,y,gammln,ran1
SAVE alxm,g,oldm,sq
DATA oldm /-1./
if (xm<12.)then
  if (xm/=oldm) then
    oldm=xm
    g=exp(-xm)
  endif
  cm=-1
  t=1.
  do
    cm=cm+1.
    t=t*ran1(idum)
    if (.not. t>g) exit
  end do
else
  if (xm/=oldm) then
    oldm=xm
    sq=sqrt(2.*xm)
    alxm=log(xm)
    g=xm*alxm-gammln(xm+1.)
  endif
  do
    do

```

```

      y=tan(PI * ran1(idum))
      em=sq * y+xm
      if (.not. em<0.) exit
    end do
      em=int(em)
      t=0.9 * (1. + y * * 2) * exp(em * alxm - gammaln(em+1.) - g)
      if (.not. ran1(idum)>t) exit
    end do
  endif
  poidev=em
END FUNCTION poidev

```

(3) 函数过程 BNLDDEV.

```

FUNCTION bnlddev(pp,n,idum)
INTEGER idum,n
REAL bnlddev,pp,PI
! USES gammaln,ran1
PARAMETER (PI=3.141592654)
INTEGER j,nold
REAL am,em,en,g,oldg,p,pc,pclog,plog,pold,&
      sq,t,y,gammaln,ran1
SAVE nold,pold,pc,plog,pclog,en,oldg
DATA nold /-1/, pold /-1./
if(pp<=0.5) then
  p=pp
else
  p=1.-pp
endif
am=n * p
if (n<25) then
  bnlddev=0.
  do j=1,n
    if(ran1(idum)<p) bnlddev=bnlddev+1.
  end do
else if (am<1.) then
  g=exp(-am)
  t=1.

```

```

do j=0,n
  t=t*ran1(idum)
  if (t<g) exit
end do
if(t<g) then
  bnldev=j
else
  j=n
  bnldev=j
end if
else
  if (n/=nold) then
    en=n
    oldg=gammln(en+1.)
    nold=n
  endif
  if (p/=pold) then
    pc=1.-p
    plog=log(p)
    pclog=log(pc)
    pold=p
  endif
  sq=sqrt(2.*am*pc)
  do
    do
      y=tan(PI*ran1(idum))
      em=sq*y+am
      if (.not. (em<0. .or. em>=en+1.)) exit
    end do
    em=int(em)
    t=1.2*sq*(1.+y**2)*exp(oldg-gammln(em+1.)-&
      gammln(en-em+1.)+em*plog+(en-em)*pclog)
    if (.not. ran1(idum)>t) exit
  end do
  bnldev=em
endif
if (p/=pp) bnldev=n-bnldev

```

```
END FUNCTION bnldev
```

5. 例子

这里要介绍的三个验证程序 D6R7 到 D6R9 和上一节的验证程序是相同的,只不过这里的每一个验证程序调用了不同的随机数发生器子程序且产生了不同的图像. D6R7 调用 GAMDEV 显示了由用户所确定的 IA 阶 Γ 分布. D6R8 调用 POIDEV 产生了由用户确定的 XM 均值的一个泊松分布. D6R9 调用 BNLDEV 产生了一个由用户确定的 XM 的二项式分布.

(1) 验证 GAMDEV 的程序 D6R7 如下:

```
PROGRAM D6R7
! Driver for routine GAMDEV
PARAMETER (N=20,NPTS=10000,ISCAL=200,LLEN=50)
DIMENSION DIST(21)
CHARACTER TEXT(50)*1
IDUM=-13
DO
  DO
    DO J=1,21
      DIST(J)=0.0
    END DO
    WRITE(*,*)&
      'Order of Gamma distribution (n=1..20); -1 to end.'
    READ(*,*) IA
    IF (IA<=0) STOP
    IF (.NOT. IA>20) EXIT
  END DO
  DO I=1,NPTS
    J=INT(GAMDEV(IA,IDUM))+1
    IF ((J>=1).AND.(J<=21)) DIST(J)=DIST(J)+1
  END DO
  WRITE(*,'(1X,A,I2,A,I6,A)')&
    'Gamma ~distribution deviate, order ',IA,&
    ' of ',NPTS,' points'
  WRITE(*,'(1X,T6,A,T14,A,T23,A)') 'x','p(x)','graph:'
  DO J=1,20
    DIST(J)=DIST(J)/NPTS
```

```

      DO K=1,50
        TEXT(K)=' '
      END DO
      KLIM=INT(ISCAL * DIST(J))
      IF (KLIM>LLEN) KLIM=LLEN
      DO K=1,KLIM
        TEXT(K)='*'
      END DO
      WRITE(*,'(1X,F7.2,F10.4,4X,50A1)')&
        FLOAT(J),DIST(J),(TEXT(K),K=1,50)
    END DO
  END DO
END

```

计算结果如下:

Order of Gamma distribution (n=1..20); -1 to end.

5

Gamma-distribution deviate, order 5 of 10000 points

x	p(x)	graph:
1.00	0.0033	
2.00	0.0506	* * * * *
3.00	0.1322	* * * * * * * * * * * * * * * * * * * *
4.00	0.1878	* * * * * * * * * * * * * * * * * * * *
5.00	0.1939	* * * * * * * * * * * * * * * * * * * *
6.00	0.1503	* * * * * * * * * * * * * * * * * * *
7.00	0.1089	* * * * * * * * * * * * * * * *
8.00	0.0729	* * * * * * * * * *
9.00	0.0446	* * * * * * *
10.00	0.0259	* * * *
11.00	0.0126	* *
12.00	0.0085	*
13.00	0.0042	
14.00	0.0023	
15.00	0.0010	
16.00	0.0006	
17.00	0.0003	
18.00	0.0001	

19.00 0.0000

20.00 0.0000

Order of Gamma distribution ($n=1..20$); -1 to end.

-1

(2) 验证 POIDEV 的程序 D6R8 如下:

```
PROGRAM D6R8
```

```
1 Driver for routine POIDEV
```

```
PARAMETER (N=20,NPTS=10000,ISCAL=200,LLEN=50)
```

```
DIMENSION DIST(21)
```

```
CHARACTER TEXT(50)*1
```

```
IDUM=-13
```

```
DO
```

```
DO
```

```
DO J=1,21
```

```
DIST(J)=0.0
```

```
END DO
```

```
WRITE(*,*)&
```

```
'Mean of Poisson distrib. &
```

```
(x=0. to 20.); -1. to end'
```

```
READ(*,*) XM
```

```
IF (XM<0.) STOP
```

```
IF (.NOT. XM>20.) EXIT
```

```
END DO
```

```
DO I=1,NPTS
```

```
J=INT(POIDEV(XM,IDUM))+1
```

```
IF ((J>=1). AND. (J<=21)) DIST(J)=DIST(J)+1
```

```
END DO
```

```
WRITE(*, '(1X,A,F5.2,A,I6,A)')&
```

```
'Poisson distribution deviate, mean ',XM,' of ',&
```

```
NPTS,' points'
```

```
WRITE(*, '(1X,T6,A,T14,A,T23,A)') 'x','p(x)','graph:'
```

```
DO J=1,20
```

```
DIST(J)=DIST(J)/NPTS
```

```
DO K=1,50
```

```
TEXT(K)=' '
```

```
END DO
```



```

KLIM=INT(ISCAL * DIST(J))
IF (KLIM>LLEN) KLIM=LLEN
DO K=1,KLIM
    TEXT(K)= ' * '
END DO
WRITE( *, '(1X,F7.2,F10.4,4X,50A1)') &
    FLOAT(J),DIST(J),(TEXT(K),K=1,50)
END DO
END DO
END

```

计算结果如下:

Mean of Poisson distrib. (x=0. to 20.): -1. to end.

5

Poisson distribution deviate, mean 5.00 of 10000 points

x	p(x)	graph:
1.00	0.0070	*
2.00	0.0355	* * * * *
3.00	0.0834	* * * * * * * * *
4.00	0.1360	* * * * * * * * * * *
5.00	0.1698	* * * * * * * * * * * * *
6.00	0.1792	* * * * * * * * * * * * * * *
7.00	0.1483	* * * * * * * * * * * * * *
8.00	0.1066	* * * * * * * * * * *
9.00	0.0645	* * * * * * * * *
10.00	0.0358	* * * * * *
11.00	0.0188	* * *
12.00	0.0096	*
13.00	0.0037	
14.00	0.0012	
15.00	0.0005	
16.00	0.0001	
17.00	0.0000	
18.00	0.0000	
19.00	0.0000	
20.00	0.0000	

Mean of Poisson distrib. (x=0. to 20.): -1. to end

- 1

(3) 验证 BNLDEV 的程序 D6R9 如下:

```

PROGRAM D6R9
! Driver for routine BNLDEV
PARAMETER (N=20,NPTS=10000,ISCAL=200,LLEN=50,NN=100)
DIMENSION DIST(21)
CHARACTER TEXT(50)*1
IDUM=-13
DO
  DO
    DO J=1,21
      DIST(J)=0.0
    END DO
    WRITE(*,*) &
      'Mean of binomial distrib. &
      (x=0. to 20.); -1. to end'
    READ(*,*) XM
    IF (XM<0.) STOP
    IF (.not. XM>20.) EXIT
  END DO
  PP=XM/NN
  DO I=1,NPTS
    J=INT(BNLDEV(PP,NN,IDUM))
    IF ((J>=0). AND. (J<=20)) DIST(J+1)=DIST(J+1)+1
  END DO
  WRITE(*, '(1X,T3,A,T10,A,T18,A)') 'x', 'p(x)', 'graph'
  DO J=1,20
    DIST(J)=DIST(J)/NPTS
    DO K=1,50
      TEXT(K)= ' '
    END DO
    TEXT(1)= ' * '
    KLIM=INT(ISCAL * DIST(J))
    IF (KLIM.GT.LLEN) KLIM=LLEN
    DO K=1,KLIM
      TEXT(K)= ' * '
    END DO
  END DO
END DO

```

```

END DO
WRITE(*, '(1X,F5.1,F8.4,3X,50A1)')&
      FLOAT(J-1),DIST(J),(TEXT(K),K=1,50)
END DO
END DO
END

```

计算结果如下:

Mean of binomial distrib. (x=0. to 20.); -1. to end.

5

x	p(x)	graph
0.0	0.0076	*
1.0	0.0297	* * * * *
2.0	0.0856	* * * * * * * * * * * * * * *
3.0	0.1440	* * * * * * * * * * * * * * * * * * *
4.0	0.1743	* *
5.0	0.1838	* *
6.0	0.1535	* *
7.0	0.1008	* * * * * * * * * * * * * * * * * *
8.0	0.0611	* * * * * * * * * *
9.0	0.0330	* * * * *
10.0	0.0161	* * *
11.0	0.0062	*
12.0	0.0027	*
13.0	0.0009	*
14.0	0.0004	*
15.0	0.0002	*
16.0	0.0001	*
17.0	0.0000	*
18.0	0.0000	*
19.0	0.0000	*

Mean of binomial distrib. (x=0. to 20.); -1. to end

-1

6.4 随机位的产生

1. 功能

由模 2 本原多项式理论产生随机位,即等概率地产生随机位 0 与 1.

本节两个函数过程 IRBIT1 与 IRBIT2, 按 18 阶模 2 本原多项式 $x^{18} + x^5 + x^2 + x + 1$ 定义产生随机位的递推关系.

函数过程 IRBIT1 适用于硬件, 而函数过程 IRBIT2 适用于软件.

2. 方法

对每个 n 阶模 2 本原多项式

$$f(x) = x^n + g_{n-1}x^{n-1} + \cdots + g_1x + g_0$$

其中 $g_i \in GF(2) = \{0, 1\}$, $i = 1, 2, \dots, n-1$, 定义一个递推关系, 它从 n 个已知位产生一个新的随机位, 即对任意的 n 个初始位, 产生一个随机位序列. 该序列在重复 n 个初始位前已获得 $2^n - 1$ 个随机位, 即序列的周期是 $2^n - 1$, 若 $f(x)$ 不是本原多项式, 则序列的周期将小于 $2^n - 1$.

给定任一整数, 取其 n 个低有效位, 记为 a_1, \dots, a_n , 由递推关系产生一个新的随机位 a_0 , 然后舍去 a_n , 并用 a_{i-1} 替换 a_i , $i = 1, \dots, n$, 重复之.

算法 1 (如图 6.2(a) 所示)

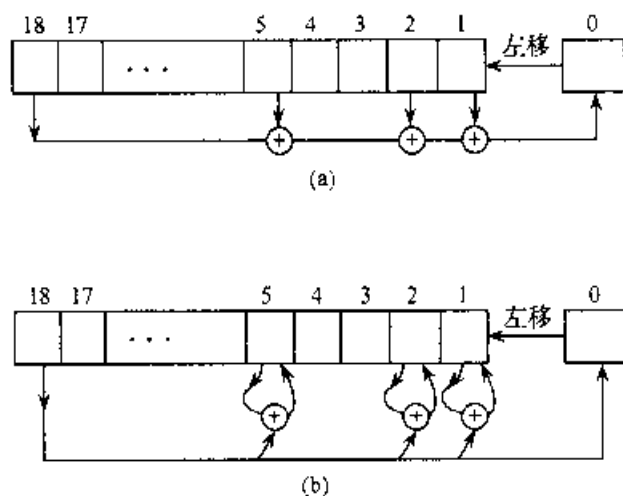


图 6.2 产生随机位的两种方法

1) 给定一个整数 S , 其二进制表示为

$$S = (a_m \cdots a_{n+1} a_n a_{n-1} \cdots a_2 a_1)_2$$

2)

$$a_0 = a_n \oplus g_{n-1}a_{n-1} \oplus \cdots \oplus g_1a_1$$

(对上述 18 阶本原多项式有 $a_0 = a_{18} \oplus a_5 \oplus a_2 \oplus a_1$).

这里 \oplus 表示按位逻辑异或, 即两个位相同时, 结果为 0, 否则结果为 1.

3) $a_i := a_{i-1}$, $i = 1, 2, \dots, n$.

4) $S := (a_n a_{n-1} \cdots a_2 a_1)_2$, 转 2).

算法 I (如图 6.2(b)所示)

1) 取一个整数 S 的 n 个低有效位 a_1, a_2, \dots, a_n .

2) $a_0 \leftarrow a_n, a_i \leftarrow a_i \oplus g_i a_0, i=1, 2, \dots, n-1$.

3) $a_i \leftarrow a_{i-1}, i=1, 2, \dots, n$, 转 2).

自然, 也可采用不同于这两个算法中的递推关系. 另外, 表 6.2 列出了 100 次以内的模 2 本原多项式.

表 6.2 模 2 本原多项式

Primitive Polynomials Module 2	
(1,0)	(51,6,3,1,0)
(2,1,0)	(52,3,0)
(3,1,0)	(53,6,2,1,0)
(4,1,0)	(54,6,5,1,3,2,0)
(5,2,0)	(55,6,2,1,0)
(6,1,0)	(56,7,4,2,0)
(7,1,0)	(57,5,3,2,0)
(8,4,3,2,0)	(58,6,5,1,0)
(9,4,0)	(59,6,5,4,3,1,0)
(10,3,0)	(60,1,0)
(11,2,0)	(61,5,2,1,0)
(12,6,4,1,0)	(62,6,5,3,0)
(13,4,3,1,0)	(63,1,0)
(14,5,3,1,0)	(64,4,3,1,0)
(15,1,0)	(65,4,3,1,0)
(16,5,3,2,0)	(66,8,6,5,3,2,0)
(17,3,0)	(67,5,2,1,0)
(18,5,2,1,0)	(68,7,5,1,0)
(19,5,2,1,0)	(69,6,5,2,0)
(20,3,0)	(70,5,3,1,0)
(21,2,0)	(71,5,3,1,0)
(22,1,0)	(72,6,4,3,2,1,0)
(23,5,0)	(73,4,3,2,0)
(24,4,3,1,0)	(74,7,4,3,0)
(25,3,0)	(75,6,3,1,0)
(26,6,2,1,0)	(76,5,4,2,0)
(27,5,2,1,0)	(77,6,5,2,0)
(28,3,0)	(78,7,2,1,0)

续表

Primitive Polynomials Module 2	
(29,2,0)	(79,4,3,2,0)
(30,6,4,1,0)	(80,7,5,3,2,1,0)
(31,3,0)	(81,4,0)
(32,7,5,3,2,1,0)	(82,8,7,6,4,1,0)
(33,6,4,1,0)	(83,7,4,2,0)
(34,7,6,5,2,1,0)	(84,8,7,5,3,1,0)
(35,2,0)	(85,8,2,1,0)
(36,6,5,4,2,1,0)	(86,6,5,2,0)
(37,5,4,3,2,1,0)	(87,7,5,1,0)
(38,6,5,1,0)	(88,8,5,4,3,1,0)
(39,4,0)	(89,6,5,3,0)
(40,5,4,3,0)	(90,9,3,2,0)
(41,3,0)	(91,7,6,5,3,2,0)
(42,5,4,3,2,1,0)	(92,6,5,2,0)
(43,6,4,3,0)	(93,2,0)
(44,6,5,2,0)	(94,6,5,1,0)
(45,4,3,1,0)	(95,6,5,4,2,1,0)
(46,8,5,3,2,1,0)	(96,7,6,4,3,2,0)
(47,5,0)	(97,6,0)
(48,7,5,4,2,1,0)	(98,7,4,3,2,1,0)
(49,6,5,1,0)	(99,7,5,4,0)
(50,4,3,2,0)	(100,8,7,2,0)

3. 使用说明

FUNCTION IRBIT1 (ISEED)

FUNCTION IRBIT2 (ISEED)

ISEED 整型变量,输入、输出参数,输入一个整数,为产生随机位提供初值,输出值为下一次调用该子程序提供初值

4. 过程

(1) 函数过程 IRBIT1.

FUNCTION irbit1(iseed)

INTEGER irbit1,iseed,IB1,IB2,IB5,IB18

PARAMETER (IB1=1,IB2=2,IB5=16,IB18=131072)

```

LOGICAL newbit
newbit=tand(iseed,IB18)/=0
if(iand(iseed,IB5)/=0) newbit=.not. newbit
if(iand(iseed,IB2)/=0) newbit=.not. newbit
if(iand(iseed,IB1)/=0) newbit=.not. newbit
irbit1=0
iseed=iand(ishft(iseed,1),not(IB1))
if(newbit) then
    irbit1=1
    iseed=ior(iseed,IB1)
endif
END FUNCTION irbit1

```

(2) 函数过程 IRBIT2.

```

FUNCTION irbit2(iseed)
INTEGER irbit2,iseed,IB1,IB2,IB5,IB18,MASK
PARAMETER (IB1=1,IB2=2,IB5=16,IB18=131072,&
            MASK=IB1+IB2+IB5)
if(iand(iseed,IB18)/=0) then
    iseed=ior(ishft(icor(iseed,MASK),1),IB1)
    irbit2=1
else
    iseed=iand(ishft(iseed,1),not(IB1))
    irbit2=0
endif
END FUNCTION irbit2

```

5. 例子

子过程 IRBIT1 和 IRBIT2 两者都随机地产生 1 和 0 的序列. 它们的验证程序 D6R10 和 D6R11 是相同的, 且它们验证这个序列有正确的统计特性(或更准确地说, 至少有一个正确的特性). 验证程序在该序列中寻找 1 并计算在下一个 1 出现前有多少个零. 例如, 没有零的可能性应该是 50%, 有一个零的可能性为 25% 等.

(1) 验证 IRBIT1 的程序 D6R10 如下:

```
PROGRAM D6R10
```

```

! Driver for routine IRBIT1
! Calculate distribution of runs of zeros
PARAMETER (NBIN=15,NTRIES=10000)
DIMENSION DELAY(NBIN)
ISEED=12345
DO I=1,NBIN
    DELAY(I)=0.0
END DO
IPTS=0
DO I=1,NTRIES
    IF (IRBIT1(ISEED)==1) THEN
        IPTS=IPTS+1
        IFLG=0
        DO J=1,NBIN
            IF ((IRBIT1(ISEED)==1).AND.(IFLG==0)) THEN
                IFLG=1
                DELAY(J)=DELAY(J)+1.0
            ENDIF
        END DO
    ENDIF
END DO
WRITE(*,*) 'Distribution of runs of N zeros'
WRITE(*, '(1X,T7,A,T16,A,T38,A)') 'N','Probability',&
    'Expected'
DO N=1,NBIN
    WRITE(*, '(1X,I6,F18.6,F20.6)') N-1,DELAY(N)/IPTS,&
        1./(2.*N)
END DO
END

```

计算结果如下：

Distribution of runs of N zeros		
N	Probability	Expected
0	0.501207	0.500000
1	0.247586	0.250000
2	0.122084	0.125000
3	0.067377	0.062500

4	0.029968	0.031250
5	0.015688	0.015625
6	0.007039	0.007813
7	0.004626	0.003906
8	0.002615	0.001953
9	0.001006	0.000977
10	0.000603	0.000488
11	0.000000	0.000244
12	0.000000	0.000122
13	0.000201	0.000061
14	0.000000	0.000031

(2) 验证 IRBIT2 的程序 D6R11 如下:

```

PROGRAM D6R11
! Driver for routine IRBIT2
! Calculate distribution of runs of zeros
PARAMETER (NBIN=15,NTRIES=10000)
DIMENSION DELAY(NBIN)
ISEED=12345
DO I=1,NBIN
    DELAY(I)=0.0
END DO
IPTS=0
DO I=1,NTRIES
    IF (IRBIT2(ISEED)==1) THEN
        IPTS=IPTS+1
        IFLG=0
        DO J=1,NBIN
            IF ((IRBIT2(ISEED)==1).AND.(IFLG==0)) THEN
                IFLG=1
                DELAY(J)=DELAY(J)+1.0
            ENDIF
        END DO
    ENDIF
END DO
WRITE(*,*) 'Distribution of runs of N zeros'
WRITE(*, '(1X,T7,A,T16,A,T38,A)') 'N','Probability',&

```

```

      'Expected'
DO N=1,NBIN
WRITE( *, '(IX,I6,F18.6,F20.6)' ) N-1,DELAY(N)/IPTS,&
      1./(2. * * N)
END DO
END

```

计算结果如下:

Distribution of runs of N zeros

N	Probability	Expected
0	0.499201	0.500000
1	0.253895	0.250000
2	0.118658	0.125000
3	0.066520	0.062500
4	0.030763	0.031250
5	0.015981	0.015625
6	0.007591	0.007813
7	0.003995	0.003906
8	0.001798	0.001953
9	0.001398	0.000977
10	0.000200	0.000488
11	0.000000	0.000244
12	0.000000	0.000122
13	0.000000	0.000061
14	0.000000	0.000031

6.5 蒙特卡罗积分法

1. 功能

用蒙特卡罗(Monte Carlo)积分法求下列积分的近似值:

$$\int_w \rho dx dy dz, \int_w x \rho dx dy dz, \int_w y \rho dx dy dz, \int_w z \rho dx dy dz \quad (6-1)$$

其中积分区域为

$$W = \{(x, y, z) \in R^3; x \geq 1, y \geq -3, z^2 + (\sqrt{x^2 + y^2} - 3)^2 \leq 1\}$$

而密度 ρ 为 $\rho(x, y, z) = e^{xz}$.

对其它积分完全类似.

2. 方法

(1) 设积分

$$I = \int_W g(x) dW$$

在积分区域 W 中均匀地取 n 个均匀分布随机数 x_1, x_2, \dots, x_n , 则

$$I \approx |W| \cdot \langle g \rangle \pm ER$$

其中 $|W|$ 表示区域 W 的体积, 而

$$\langle g \rangle \equiv \frac{1}{n} \sum_{i=1}^n g(x_i), \quad ER = |W| \cdot \sqrt{(\langle g^2 \rangle - \langle g \rangle^2)/n}$$

ER 是一个标准偏差误差估计, 它仅是一个可能误差的粗糙表示.

(2) 如果积分区域 W 的形状很复杂, 则取一个容易取样的、尽可能小的区域 $V \supset W$, 定义

$$f(x) = \begin{cases} g(x), & x \in W \\ 0, & x \in V \setminus W \end{cases}$$

则

$$I = \int_W g(x) dW = \int_V f(x) dV$$

(3) 对于

$$W = \{(x, y, z) \in R^3; 2^2 + (\sqrt{x^2 + y^2} - 3)^2 \leq 1, x \geq 1, y \geq -3\}$$

取

$$V = \{(x, y, z) \in R^3; 1 \leq x \leq 4, -3 \leq y \leq 4, -1 \leq z \leq 1\}$$

程序可采用如下形式:

```

N=
SW=0.0
SWX=0.0
SWY=0.0
SWZ=0.0
VARW=0.0
VARX=0.0
VARY=0.0
VARZ=0.0
IDUM=
SS=0.2*(EXP(5.0)-EXP(-5.0))

```

```

VOL=3.0*7.0*SS
DO J=1,N
  X=1.0+3.0*RAN2(IDUM)
  Y=-3.0+7.0*RAN2(IDUM)
  S=0.00135+SS*RAN2(IDUM)
  Z=0.2*LOG(5.0*S)
  IF (Z**2+(SQRT(X**2+Y**2)-3.0)**2 < 1.0) THEN
    SW=SW+1.
    SWX=SWX+X
    SWY=SWY+Y
    SWZ=SWZ+Z
    VARW=VARW+1.
    VARX=VARX+X**2
    VARY=VARY+Y**2
    VARZ=VARZ+Z**2
  END IF
END DO
W=VOL*SW/N
X=VOL*SWX/N
Y=VOL*SWY/N
Z=VOL*SWZ/N
DW=VOL*SQRT((VARW/N-(SW/N)**2)/N)
DX=VOL*SQRT((VARX/N-(SWX/N)**2)/N)
DY=VOL*SQRT((VARY/N-(SWY/N)**2)/N)
DZ=VOL*SQRT((VARZ/N-(SWZ/N)**2)/N)

```

其中 N 表示所要求的样本点总数, W, X, Y, Z 依次表示式(6-1)中的四个积分, DW, DX, DY, DZ 依次表示这四个积分对应的计算误差估计。

第7章 排 序

本章的内容本不属于数值方法,然而,对一个优秀的程序员来说,掌握排序的实用技术是必要的.作为数值技术方面的专家,不应该忽略像排序这样的基础课题.

本章的主要内容有:(1) 排序,即把一个数组元素按大小排序;(2) 把一个数组元素按大小排序并同时对应的另一个或多个数组作相应的重排,使得在原所有数组元素之间的对应性保持不变;(3) 给定一个数组,配置一个索引表,即配置一个能够告知排序后的数组中每一个元素在原数组中的位置的指针表;(4) 给定一个数组,配置一个等级表,即配置一个能告知原数组中的每一元素在排序后数组中的数值等级表.

对于 n 个元素的数组的排序,最佳算法的运行时间是 $O(n \log_2 n) \approx p \times n \log_2 n$. 堆排序(7.2节)与快速排序(7.4节)是两个最佳算法.

对于大 n (如 $n > 1000$),快速排序又叫分区交换排序,是较快的,因子 p 大约为 1.5 或 2,可是它要求一些附加内存,且程序也稍复杂一点.它是由 C. A. R. Hoare 在 1962 年提出来的.冒泡排序与快速排序都属于交换排序,但冒泡排序是一个 $O(n^2)$ 算法,是一个决不应该考虑的方法,因为对于 $O(n^2)$ 算法,计算机运行时间之长是非常可怕的.

堆排序对于小 n ,其优点并不突出,但当 n 很大时,它的高效率是分外明显的.该算法的运行时间主要花在建堆和调整操作上,在最坏情况下,其时间复杂度为 $O(n \log_2 n)$,且堆排序程序相对更紧凑,因此也容易修改.总的来说,我们更倾向于使用堆排序法.它是由 J. W. J. Williams 于 1964 年提出的.

对于小 n (大体上是 $n < 50$),直接插入法(7.1节)是简单且足够快的方法,但它是一个 $O(n^2)$ 算法,因此对于大 n 并不适用.对于 $50 < n < 1000$ 的 n ,希尔(Shell)方法(7.1节)只是比直接插入法的程序稍复杂的方法,是通常选用的方法.这个方法在最坏情形下是一个 $O(n^{3/2})$ 算法,但它通常是较快的.

7.1 直接插入法和 Shell 方法

1. 功能

子过程 PIKSRT 和 PIKSR2 均为直接插入方法,适用于较小的 n ,如 $n < 50$.

前者用于对某一给定的数组,将其元素按数值升序进行重排,后者是对第一个数组按升序进行重排,而同时对第二个数组按第一个数组的新次序重新整理次序.

子过程 SHELL 采用 Shell-Mozgar 算法进行排序.

对于直接插入法,最小比较次数是 n ,最大比较次数是 $(n+2)(n+1)/2 \approx n^2/2$;最小移动次数是 $2(n-1) \approx 2n$,最大移动次数约为 $n^2/2$.

对于 Shell 方法,其平均比较次数和平均移动次数均为 $n^{1.3}$ 左右.

2. 方法

(1) 直接插入法.

取数组的第二个元素与第一个元素进行比较,把第二个元素放到相对于第一个元素的合适位置.然后再取第三个元素与前两个元素比较,并把第三个元素插入到相对于前两个元素的合适位置,直至把第 n 个元素插入到前 $n-1$ 个已排序的元素的合适位置中.

(2) Shell 方法.

先取定一个整数 $d_1 = \left\lceil \frac{n}{2} \right\rceil$,把全部元素分成 d_1 个组,所有距离为 d_1 位置的元素放在一组中,在各组内进行排序;然后取 $d_2 = \left\lceil \frac{d_1}{2} \right\rceil$ 重复上述分组和排序工作;直到取 $d_i = 1$ 即所有元素放在一组中排序为止.各组内的排序可以采用直接插入法.

3. 使用说明

PIKSRT (N,ARR)

PIKSRT2 (N,ARR,BRR)

SHELL (N,ARR)

N 整型变量,输入参数,待排序数组元素的个数

ARR N 个元素的一维实型数组,输入、输出参数,输入时存放待排序的数组,输出时存放排序后的数组

BRR N 个元素的一维实型数组,输入、输出参数,输入时存放需要作相应重排的第二个数组,输出时存放排序后的第二个数组

4. 过程

(1) 子过程 PIKSRT.

SUBROUTINE piksrt(n,arr)

```
INTEGER n
REAL arr(n)
INTEGER i,j
REAL a
do j=2,n
    a=arr(j)
    do i=j-1,1,-1
        if(arr(i)<=a) exit
        arr(i+1)=arr(i)
    end do
    if(arr(i)<=a) then
        arr(i+1)=a
    else
        i=0
        arr(i+1)=a
    end if
end do
END SUBROUTINE piksrt
```

(2) 子过程 PIKSR2.

```
SUBROUTINE piksr2(n,arr,brr)
INTEGER n
REAL arr(n),brr(n)
INTEGER i,j
REAL a,b
do j=2,n
    a=arr(j)
    b=brr(j)
    do i=j-1,1,-1
        if(arr(i)<=a) exit
        arr(i+1)=arr(i)
        brr(i+1)=brr(i)
    end do
    if(arr(i)<=a) then
        arr(i+1)=a
    else
        i=0

```

```

        arr(i-1)=a
    end if
    brr(i+1)=b
end do
END SUBROUTINE piksr2

```

(3) 子过程 SHELL.

```

SUBROUTINE shell(n,arr)
INTEGER m,k,l,j
REAL arr
! parameter (aln2i=1./0.69314718,tiny=1.e-5)
PARAMETER (aln2i=1.4427,tiny=1.e-5)
DIMENSION arr(n)
lognb2=int(alog(float(n))*aln2i+tiny)
m=n
do nn=1,lognb2
    m=m/2
    k=n-m
    do j=1,k
        i=j
        do
            l=i+m
            if (arr(l)<arr(i)) then
                t=arr(i)
                arr(i)=arr(l)
                arr(l)=t
                i=i-m
                if (.not.i>=1) exit
            endif
        end do
        exit
    end do
end do
END SUBROUTINE shell

```

5. 例子

(1) 验证程序 D7R1 所采用的例子是本章末附录中文件 TARRAY.DAT,

该文件提供了一个 100 个元素的数组. 子过程 PIKSRT 利用直接插入法对数组元素进行排序. 验证程序 D7R1 打印出原数组和已排序的数组以进行比较. 程序 D7R1 如下:

```

PROGRAM D7R1
! Driver for routine PIKSRT
DIMENSION A(100)
OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
READ(5,*) (A(I),I=1,100)
CLOSE(5)
! Print Original array
WRITE(*,*) 'Original array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
! Sort array
CALL PIKSRT(100,A)
! Print sorted array
WRITE(*,*) 'Sorted array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Sorted array:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

(2) 子过程 PIKSR2 对第一个数组进行排序,同时对第二个数组(同样大小)按第一个数组的新次序重新进行整理次序.在验证程序 D7R2 中,第一个数组仍取本章末附录中文件 TARRAY.DAT 的数据.第二个数组定义为 $B(I)=I$. 换句话说, B 是原本已排好序的数组,而 A 为没有排序的数组.在调用 PIKSR2 后, A 则成为排好序的数组,而 B 则按 A 的新次序重新进行排列.在第二次调用时, B 和 A 的位置进行对调,这样 A 和 B 将回归原来的次序,验证程序 D7R2 如下:

```

PROGRAM D7R2
  ! Driver for routine PIKSR2
  DIMENSION A(100),B(100)
  OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
  READ(5,*) (A(I),I=1,100)
  CLOSE(5)
  ! Generate B-array
  DO I=1,100
    B(I)=I
  END DO
  ! Sort A and mix B
  CALL PIKSR2(100,A,B)
  WRITE(*,*) 'After sorting A and mixing B, array A is:'
  DO I=1,10
    WRITE(*, '(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
  END DO
  WRITE(*,*) '...and array B is:'
  DO I=1,10

```

```

      WRITE( *, '(1X,10F6.2)') (B(10*(I-1)+J),J=1,10)
END DO
WRITE( *, *) 'Press RETURN to END DO...'
READ( *, *)
! Sort B and mix A
CALL PIKSR2(100,B,A)
WRITE( *, *) 'After sorting A and mixing B, array A is:'
DO I=1,10
      WRITE( *, '(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
WRITE( *, *) '...and array B is:'
DO I=1,10
      WRITE( *, '(1X,10F6.2)') (B(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

After sorting A and mixing B, array A is:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

...and array B is:

61.00	51.00	40.00	3.00	58.00	86.00	79.00	74.00	45.00	44.00
23.00	48.00	62.00	73.00	69.00	49.00	31.00	22.00	92.00	14.00
77.00	87.00	18.00	29.00	96.00	8.00	95.00	15.00	91.00	10.00
65.00	1.00	63.00	78.00	33.00	57.00	70.00	100.00	28.00	35.00
59.00	81.00	88.00	68.00	12.00	55.00	75.00	32.00	20.00	84.00
13.00	24.00	5.00	21.00	67.00	94.00	99.00	90.00	16.00	37.00
50.00	6.00	82.00	19.00	27.00	93.00	76.00	89.00	2.00	11.00
85.00	60.00	30.00	54.00	72.00	25.00	71.00	56.00	26.00	66.00
38.00	36.00	97.00	47.00	17.00	52.00	4.00	7.00	46.00	83.00
98.00	64.00	42.00	9.00	39.00	43.00	53.00	80.00	41.00	34.00

After sorted B and mixing A, array A is:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

... and array B is:

1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00
11.00	12.00	13.00	14.00	15.00	16.00	17.00	18.00	19.00	20.00
21.00	22.00	23.00	24.00	25.00	26.00	27.00	28.00	29.00	30.00
31.00	32.00	33.00	34.00	35.00	36.00	37.00	38.00	39.00	40.00
41.00	42.00	43.00	44.00	45.00	46.00	47.00	48.00	49.00	50.00
51.00	52.00	53.00	54.00	55.00	56.00	57.00	58.00	59.00	60.00
61.00	62.00	63.00	64.00	65.00	66.00	67.00	68.00	69.00	70.00
71.00	72.00	73.00	74.00	75.00	76.00	77.00	78.00	79.00	80.00
81.00	82.00	83.00	84.00	85.00	86.00	87.00	88.00	89.00	90.00
91.00	92.00	93.00	94.00	95.00	96.00	97.00	98.00	99.00	100.00

(3) 子过程 SHELL 对一数组元素进行 Shell 排序. 验证 SHELL 的方式和验证 PIKSRT 一样. 因而, 验证程序也几乎相同. 验证程序 D7R3 如下:

```

PROGRAM D7R3
  ! Driver for routine SHELL
  DIMENSION A(100)
  OPEN(5, FILE='D:\VF_SHU\DATA\TARRAY.DAT', STATUS='OLD')
  READ(5, *) (A(I), I=1, 100)
  CLOSE(5)
  ! Print Original array
  WRITE(*, *) 'Original array:'
  DO I=1, 10
    WRITE(*, '(1X, 10F6.2)') (A(10*(I-1)+J), J=1, 10)
  END DO
  ! Sort array

```

```

CALL SHELL(100,A)
! Print sorted array
WRITE(*,*) 'Sorted array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Sorted array:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

7.2 堆 排 序

1. 功能

应用堆排序算法对数组进行重排序,使其重排后的元素升序。

堆排序适用于数组元素较多的情况,它采用完全二叉树型结构,进行排序所需时间是 $O(n\log_2 n)$,其中 n 为数组元素的个数. 附加存储空间只要求一个用于交换的节点.

子过程 SORT 重排一个数组,而子过程 SORT2 用于同时对另一数组作相应重排.

2. 方法

(1) 堆的定义:一组数 $a_i, i=1, \dots, n$, 如果当 $1 \leq [j/2] < j \leq n$ 时,满足 $a_{[j/2]} \geq a_j$, 则称其形成一个堆.

(2) 堆排序过程.

首先将要排序的数组 $\{a_1, \dots, a_n\}$ 放到一棵完全二叉树的各个节点中,然后从 $i=[n/2]$ 的节点 a_i 开始逐步把以 $a_{[n/2]}, a_{[n/2]-1}, \dots$ 为根的子树排成堆,直到以 a_1 为根的树排成堆,即完成了建堆,建堆完成后便找到了最大的元素,再交换 a_1 与 a_n ,而后对前面 $n-1$ 个节点重复上述过程. 只是重复时,不需从下往上推,而是从顶往下检查. 如此反复进行 $n-1$ 次,便完成了排序过程.

3. 使用说明

SORT (N,RA)

SORT2(N,RA,RB)

N 整型变量,输入参数,数组元素的个数

RA,RB N 个元素的一维实型数组,输入、输出参数,输入时 RA 存放待排序的数组,RB 存放需要作相应重排的另一数组,输出时存放排序后的结果

4. 过程

(1) 子过程 SORT.

```
SUBROUTINE sort(n,ra)
```

```
REAL ra(n)
```

```
INTEGER i,ir,j,l
```

```
REAL a,aaa,rra
```

```
l=n/2+1
```

```
ir=n
```

```
do
```

```
  if (l>1) then
```

```

l=l-1
rra=ra(l)
else
    rra=ra(ir)
    ra(ir)=ra(l)
    ir=ir-1
    if (ir==1) then
        ra(l)=rra
        return
    endif
endif
endif
i=l
j=l+1
do while(j<=ir)
    aaa=1.
    if (j<ir) then
        if (ra(j)<ra(j+1)) j=j+1
    endif
    if (rra<ra(j)) then
        ra(i)=ra(j)
        i=j
        j=j+j
    else
        j=ir+1
    endif
end do
ra(i)=rra
end do
END SUBROUTINE sort

```

(2) 子过程 SORT2.

```

SUBROUTINE sort2(n,ra,rb)
DIMENSION ra(n),rb(n)
l=n/2+1
ir=n
do
    if(l>1) then

```

```

    l=l-1
    rra=ra(l)
    rrb=rb(l)
else
    rra=ra(ir)
    rrb=rb(ir)
    ra(ir)=ra(l)
    rb(ir)=rb(l)
    ir=ir-1
    if(ir==1) then
        ra(l)=rra
        rb(l)=rrb
        return
    endif
endif
i=1
j=l+1
do while(j<=ir)
    if(j.lt.ir) then
        if(ra(j)<ra(j+1)) j=j+1
    endif
    if(rra<ra(j)) then
        ra(i)=ra(j)
        rb(i)=rb(j)
        i=j
        j=j-j
    else
        j=ir+1
    endif
end do
ra(i)=rra
rb(i)=rrb
end do
END SUBROUTINE sort2

```

5. 例子

(1) 子过程 SORT 的验证方式和上节中 PIKSRT 的验证方式一样, SORT

是对单个数组采用堆排序进行排序. 验证程序 D7R4 如下:

```

PROGRAM D7R4
! Driver for routine SORT
DIMENSION A(100)
OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
READ(5,*) (A(I),I=1,100)
CLOSE(5)
! Print Original array
WRITE(*,*) 'Original array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
! Sort array
CALL SORT(100,A)
! Print sorted array
WRITE(*,*) 'Sorted array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	1.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Sorted array:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

(2) 子过程 SORT2 的验证方式和上节中 PIKSR2 的验证方式一样. 子过程 SORT2 采用堆排序对第一个数组进行排序, 而对第二个数组按照第一个数组的新次序进行相应的整理. 验证程序 D7R5 如下:

```

PROGRAM D7R5
  ! Driver for routine SORT2
  DIMENSION A(100),B(100)
  OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
  READ(5,*) (A(I),I=1,100)
  CLOSE(5)
  ! Generate B--array
  DO I=1,100
    B(I)=I
  END DO
  ! Sort A and mix B
  CALL SORT2(100,A,B)
  WRITE(*,*) 'After sorting A and mixing B, array A is:'
  DO I=1,10
    WRITE(*, '(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
  END DO
  WRITE(*,*) '...and array B is:'
  DO I=1,10
    WRITE(*, '(1X,10F6.2)') (B(10*(I-1)+J),J=1,10)
  END DO
  WRITE(*,*) 'Press RETURN to END DO...'
  READ(*,*)
  ! Sort B and mix A

```

```

CALL SORT2(100,B,A)
WRITE(*,*) 'After sorting A and mixing B, array A is:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
WRITE(*,*) '...and array B is:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (B(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

After sorting A and mixing B, array A is:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

...and array B is:

61.00	51.00	40.00	3.00	58.00	86.00	79.00	74.00	45.00	44.00
23.00	48.00	62.00	73.00	69.00	49.00	31.00	22.00	92.00	14.00
77.00	87.00	18.00	29.00	96.00	8.00	95.00	15.00	91.00	10.00
65.00	1.00	63.00	78.00	33.00	57.00	70.00	100.00	28.00	35.00
59.00	81.00	88.00	68.00	12.00	55.00	75.00	32.00	20.00	84.00
13.00	24.00	5.00	21.00	67.00	94.00	99.00	90.00	16.00	37.00
50.00	6.00	82.00	19.00	27.00	93.00	76.00	89.00	2.00	11.00
85.00	60.00	30.00	54.00	72.00	25.00	71.00	56.00	26.00	66.00
38.00	36.00	97.00	47.00	17.00	52.00	4.00	7.00	46.00	83.00
98.00	64.00	42.00	9.00	39.00	43.00	53.00	80.00	41.00	34.00

After sorted B and mixing A, array A is:

```

29.82  71.51   3.30  87.44  53.42  63.16  89.10  25.75  93.16  27.72
71.58  48.34  53.11  18.34  27.13  60.31  83.34  22.81  66.84  52.91
53.42  13.22   8.01  53.39  76.12  79.09  67.61  38.39  24.81  73.21
13.42  52.10  34.86  99.83  38.46  81.59  61.75  79.62  93.39   3.21
99.34  92.22  94.29   7.03   6.67  89.35  83.14   9.01  12.68  62.22
 2.95  85.02  95.82  73.96  49.29  77.72  36.65   3.48  38.98  71.83
 1.41   9.48  32.37  89.95  28.39  79.36  54.05  46.08  11.67  37.78
77.17  74.33  10.13   4.62  49.95  68.40  19.40  34.06   4.11  98.40
42.44  64.14  89.41  52.99  71.79   3.94  19.73  44.91  71.44  59.10
27.54  15.67  67.95  55.61  26.05  25.01  82.09  89.67  57.08  38.27
... and array B is:

```

```

 1.00   2.00   3.00   4.00   5.00   6.00   7.00   8.00   9.00  10.00
11.00  12.00  13.00  14.00  15.00  16.00  17.00  18.00  19.00  20.00
21.00  22.00  23.00  24.00  25.00  26.00  27.00  28.00  29.00  30.00
31.00  32.00  33.00  34.00  35.00  36.00  37.00  38.00  39.00  40.00
41.00  42.00  43.00  44.00  45.00  46.00  47.00  48.00  49.00  50.00
51.00  52.00  53.00  54.00  55.00  56.00  57.00  58.00  59.00  60.00
61.00  62.00  63.00  64.00  65.00  66.00  67.00  68.00  69.00  70.00
71.00  72.00  73.00  74.00  75.00  76.00  77.00  78.00  79.00  80.00
81.00  82.00  83.00  84.00  85.00  86.00  87.00  88.00  89.00  90.00
91.00  92.00  93.00  94.00  95.00  96.00  97.00  98.00  99.00 100.00

```

7.3 索引表和等级表

1. 功能

子过程 INDEXX 用于对含有 n 个记录的数组按其关键词 $K_i (i=1, \dots, n)$ 的大小构造一个索引表 $I=(I_1, \dots, I_n)$, 使 K_{I_1}, \dots, K_{I_n} 为升序排列; 函数过程 SORT3 应用索引表对数组进行排序, 并同时对外另外两个数组作相应重排。

子过程 RANK 应用索引表构造一个等级表 $R=(R_1, \dots, R_n)$, 其中 $R_j (j=1, \dots, n)$ 是原记录的第 j 个记录在排序后的数组中的位置。

2. 方法

(1) 索引表.

采用堆排序方法, 并在堆排序过程中记录元素的交换信息. 若原数组中的第 i 个记录换到了位置 j , 则令 $I_j=i$.

(2) 等级表.

若 $I_j=i$, 则令 $R_i=j$.

3. 使用说明

(1) INDEXX (N,ARRIN,INDX)

N 整型变量,输入参数,数组元素的个数
ARRIN N 个元素的一维实型数组,输入参数,存放要排序的数组
INDX N 个元素的一维整型数组,输出参数,存放索引表

(2) SORT3 (N,RA,RB,RC,WKSP,IWKSP)

N 整型变量,输入参数,数组元素的个数
RA N 个元素的一维实型数组,输入、输出参数,输入时存放要排序的数组,输出时存放排序后的结果
RB,RC N 个元素的一维实型数组,输入、输出参数,输入时存放需要作相应重排的两个数组,输出时存放重排后的结果
WKSP N 个元素的一维实型数组,工作单元
IWKSP N 个元素的一维整型数组,工作单元

(3) RANK (N,INDX,IRANK)

N 整型变量,输入参数,数组元素的个数
INDX N 个元素的一维整型数组,输入参数,输入过程 INDEXX 的输出结果
IRANK N 个元素的一维整型数组,输出参数,输出结果

4. 过程

(1) 子过程 INDEXX.

```
SUBROUTINE indexx(n,arrin,indx)
DIMENSION arrin(n),indx(n)
do j=1,n
    indx(j)=j
end do
l=n/2+1
ir=n
do
    if(l>1)then
        l=l-1
```

```

      indxt=indx(l)
      q=arrin(indxt)
    else
      indxt=indx(ir)
      q=arrin(indxt)
      indx(ir)=indx(1)
      ir=ir-1
      if(ir==1) then
        indx(1)=indxt
        return
      endif
    endif
  endif
  i=1
  j=1+1
  do while(j<=ir)
    if(j<ir) then
      if(arrin(indx(j))<arrin(indx(j+1))) j=j+1
    endif
    if(q<arrin(indx(j))) then
      indx(i)=indx(j)
      i=j
      j=j+1
    else
      j=ir-1
    endif
  end do
  indx(i)=indxt
end do
END SUBROUTINE indexx

```

(2) 子过程 SORT3.

```

SUBROUTINE sort3(n,ra,rb,rc,wksp,iwksp)
  INTEGER n,iwksp(n)
  REAL ra(n),rb(n),rc(n),wksp(n)
  ! USES indexx
  INTEGER j
  call indexx(n,ra,iwksp)

```

```

do j=1,n
    wksp(j)=ra(j)
end do
do j=1,n
    ra(j)=wksp(iwksp(j))
end do
do j=1,n
    wksp(j)=rb(j)
end do
do j=1,n
    rb(j)=wksp(iwksp(j))
end do
do j=1,n
    wksp(j)=rc(j)
end do
do j=1,n
    rc(j)=wksp(iwksp(j))
end do
END SUBROUTINE sort3

```

(3) 子过程 RANK.

```

SUBROUTINE rank(n,indx,irank)
INTEGER n,indx(n),irank(n),j
do j=1,n
    irank(indx(j))=j
end do
END SUBROUTINE rank

```

5. 例子

(1) 子过程 INDEXX 为给定的输入数组制定了数组元素目录. 数组元素目录 INDX(J) 将给出输入数组元素排序后的索引. 就是对输入数组 A 来说, A 的排序后的顺序为 $A(\text{INDX}(J))$. 验证程序 D7R6 所采用的输入数组为本章末附录中的文件 TARRY.DAT. 为了检验 INDEXX, 将打印出数组 $A(\text{INDX}(J))$, $J=1, \dots, 100$. 验证程序 D7R6 如下:

```
PROGRAM D7R6
```

```

! Driver for routine INDEXX
DIMENSION A(100),INDX(100)
OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
READ(5,*) (A(I),I=1,100)
CLOSE(5)
! Generate index for sorted array
CALL INDEXX(100,A,INDX)
! Print Original array
WRITE(*,*) 'Original array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
! Print sorted array
WRITE(*,*) 'Sorted array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(INDX(10*(I-1)+J)),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Sorted array:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99

53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.11	66.84	67.61	67.95	68.40	71.41	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.31	99.83

(2) 子过程 SORT3 在对一数组进行排序的同时,又对另外两个数组进行相应的重新排列. 在验证程序 D7R7 中,第一个数组取自本章末附录 TARRAY. DAT 中前 64 个元素. 第二个和第三个数组元素分别取为(1,2,...,64)与(64,63,...,2,1). 调用子过程 SORT3 后,第一个数组被排序,第二和第三个数组被重新排列,但事实上是以第一个数组的变化作同一方式重新排列. 即假设第一个数组第四个元素被排序后,变化为新数组的第一个元素;那么,相应的第二和第三个数组的第四个元素就变化为新的第二和第三个数组的第一个元素. 依此类推,第一个数组元素被排序后,相应的第二和第三个数组的元素也都作重新排列. 为了证明这一点,一句引文被赋值到一字符型数组. 随后,这些字符根据被重新整理的第二个数组顺序而重新排列. 最后,它们又按照被重新整理的第三个数组的顺序而规则化. 如果正常的话,最后打印出的信息应该是原来引文的反序. 验证程序 D7R7 如下:

```

PROGRAM D7R7
  ! Driver for routine SORT3
  PARAMETER (NLEN=64)
  DIMENSION A(NLEN),IB(NLEN),IC(NLEN),WKSP(NLEN),&
    INDX(NLEN)
  CHARACTER MSG1*33,MSG2*31
  CHARACTER MSG*64,AMSG(64)*1,BMSG(64)*1,CMSG(64)*1
  EQUIVALENCE(MSG,AMSG(1)),(MSG1,AMSG(1)),(MSG2,AMSG(34))
  DATA MSG1/'I'd rather have a bottle in front'/
  DATA MSG2/' of me than a frontal lobotomy.'/
  WRITE(*,*) 'Original message:'
  WRITE(*,'(1X,64A1/)' ) (AMSG(J),J=1,64)
  ! Read array of random numbers
  OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
  READ(5,*)(A(I),I=1,NLEN)
  CLOSE(5)
  ! Create array IB and array IC
  DO I=1,NLEN

```

```

      IB(I)=I
      IC(I)=NLEN+1-I
    END DO
    ! Sort array A while mixing IB and IC
    CALL SORT3(NLEN,A,IB,IC,WKSP,INDX)
    ! Scramble message according to array IB
    DO I=1,NLEN
      J=IB(I)
      BMSG(I)=AMSG(J)
    END DO
    WRITE(*,*) 'Scrambled message:'
    WRITE(*, '(1X,64A1,/)') (BMSG(J),J=1,64)
    ! Unscramble according to array C
    DO I=1,NLEN
      J=IC(I)
      CMSG(J)=BMSG(I)
    END DO
    WRITE(*,*) 'Mirrored message:'
    WRITE(*, '(1X,64A1,/)') (CMSG(J),J=1,64)
  END

```

计算结果如下:

Original message:

I'd rather have a bottle in front of me than a frontal lobotomy.

Scrambled message:

on db nlfmrotv fherlyto ooh noaetr oabn' trl limf at ta. heeaat

Mirrored message:

.ymotobol latnorf a naht em fo unorf ni elttob a evah rehtar d'I

(3) 子过程 RANK 类似于过程 INDEXX, 但它产生了一个等级表, 而不是一个目录数组. 即调用 RANK 后将得到一个数组 IRANK. 该数组的第一个元素表示未排序的原数组第一个元素在排序后数组中的顺序位置, 以下依次类推. 子过程 RANK 不能直接处理原数组, 必须先调用 INDEXX, 然后将 INDEXX 的输出数组 INDX 调入 RANK, 才能得到数组 IRANK. 验证程序 D7R8 中, 原数组采用本章末附录中文件 TARRAY.DAT 的数据. 程序运行结果中将列出数组 IRANK. 为了验证结果, 我们将原数组 A 按照 IRANK 的顺序赋值到数组 B

中,那么 B 应该是数组 A 被排序的情形. 验证程序 D7R8 如下:

```

PROGRAM D7R8
! Driver for routine RANK
DIMENSION A(100),B(10),INDX(100),IRANK(100)
! USES INDEXX
OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
READ(5,*) (A(I),I=1,100)
CLOSE(5)
CALL INDEXX(100,A,INDX)
CALL RANK(100,INDX,IRANK)
WRITE(*,*) 'Original array:'
DO I=1,10
    WRITE(*, '(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
WRITE(*,*) 'Table of ranks is:'
DO I=1,10
    WRITE(*, '(1X,10I6)') (IRANK(10*(I-1)+J),J=1,10)
END DO
WRITE(*,*) 'Press RETURN to END DO...'
READ(*,*)
WRITE(*,*) 'Array sorted according to rank table:'
DO I=1,10
    DO J=1,10
        K=10*(I-1)+J
        DO L=1,100
            IF (IRANK(L)==K) B(J)=A(L)
        END DO
    END DO
    WRITE(*, '(1X,10F6.2)') (B(J),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Table of ranks is:

32	69	4	87	54	62	88	26	94	30
70	45	51	20	28	59	85	23	64	49
53	18	11	52	76	79	65	39	24	73
17	48	35	100	40	82	60	81	95	3
99	93	96	10	9	89	84	12	16	61
2	86	97	74	46	78	36	5	41	72
1	13	33	92	31	80	55	14	15	37
77	75	14	8	47	67	21	34	7	98
42	63	90	50	71	6	22	43	68	58
29	19	66	56	27	25	83	91	57	38

Press RETURN to END DO...

Array sorted according to rank table:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

7.4 快速排序

1. 功能

用快速排序法对数组进行排序,使排序后的数组按升序排列。

在多数机器上,对较大的 n ,快速排序法是已知排序算法中最快的,它的执行时间为 $O(n \log_2 n)$,另外要求一个长度为 $2\log_2 n$ 的辅助存储数组。

2. 方法

设数组为 a_1, a_2, \dots, a_n . 先取出 a_1 , 这样空出前端第一个排序码的位置, 将 a_1 与 a_n 相比较. 若 $a_n > a_1$, 则 a_n 留在原处不动, 继续将 a_1 与 a_{n-1} 相比, \dots ; 若 $a_n \leq a_1$, 则将 a_n 移到原来 a_1 的位置, 从而空出 a_n 的位置, 这时再将 a_1 与 a_2, a_3, \dots 相比, 找出一个大于 a_1 的元素, 将它移到后面刚刚空出的位置, 如此反复比较, 一步一步地往中间夹入, 便将大于 a_1 的元素都移到后部, 而把小于等于 a_1 的元素都移到前部, 最后空出的位置就是 a_1 的位置, 这样便把数组分成了两个独立的子数组: a_1 前面的部分和 a_1 后面的部分. 对分开的两个子数组继续执行上述过程, 当子数组的长度小于等于 M 时, 则用直接插入法排序. M 的取法与机器有关, 我们取为 $M=7$.

然而, 当给定的数组几乎是已排好序时, 这时快速排序法是一个 n^2 方法. 为避免这种情形, 在上述过程中, 首先生成 1 与 n 之间的一个随机整数 q , 用 a_q 代替 a_1 .

3. 使用说明

QCKSRT (N,ARR)

N 整型变量, 输入参数, 待排序数组的长度

ARR N 个元素的一维实型数组, 输入、输出参数, 输入时存放待排序的数组, 输出时存放排序后的结果

4. 过程

子过程 QCKSRT.

```
SUBROUTINE qcksrt(n,arr)
```

```
! PARAMETER (m=7,nstack=50,fm=7875.0,fa=211.,&
```

```
!            fc=1663.,fmi=1./fm)
```

```

PARAMETER (m=7,nstack=50,fm=7875.0,fa=211.,&
            fc=1663.,fmi=1.2698e-4)
DIMENSION arr(n),istack(nstack)
logical done
jstack=0
l=1
ir=n
fx=0.0
do
  if(ir-l<m)then
    do j=l+1,ir
      a=arr(j)
      do i=j-1,1,-1
        if (arr(i)<=a) exit
        arr(i+1)=arr(i)
      end do
      if(arr(i)>a) i=i-1
      arr(i+1)=a
    end do
    if(jstack /= 0) return
    ir=istack(jstack)
    l=istack(jstack-1)
    jstack=jstack-2
  else
    i=1
    j=ir
    fx=mod(fx*fa+fc,fm)
    iq=l+(ir-l+1)*(fx*fmi)
    a=arr(iq)
    arr(iq)=arr(l)
    do
      do
        if (j>0) then
          if(a<arr(j)) then
            j=j-1
            done=-1
          else

```

```

        done = 0
    endif
    if(, not, done) exit
end if
end do
if(j<=i) then
    arr(i)=a
    exit
endif
arr(i)=arr(j)
i=i+1
do
    if(i<=n) then
        if(a>arr(i)) then
            i=i+1
            done=-1
        else
            done=0
        endif
        if(, not, done) exit
    end if
end do
if(j<=i) then
    arr(j)=a
    i=j
    exit
endif
arr(j)=arr(i)
j=j-1
end do
jstack=jstack+2
if(jstack, gt, nstack)&
    pause 'nstack must be made larger.'
if(ir-i>=i-1) then
    istack(jstack)=ir
    istack(jstack-1)=i+1
    ir=i-1

```

```

    else
        istack(jstack)=i-1
        istack(jstack-1)=l
        l=i+1
    endif
endif
end do
END SUBROUTINE qcksrt

```

5. 例子

验证子过程 QCKSRT 和验证 PIKSRT 及 SORT 都一样,所以我们再次提供类似的验证程序 D7R9. 验证程序 D7R9 如下:

```

PROGRAM D7R9
! Driver for routine QCKSRT
DIMENSION A(100)
OPEN(5,FILE='D:\VF_SHU\DATA\TARRAY.DAT',STATUS='OLD')
READ(5,*) (A(I),I=1,100)
CLOSE(5)
! Print Original array
WRITE(*,*) 'Original array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
! Sort array
CALL QCKSRT(100,A)
! Print sorted array
WRITE(*,*) 'Sorted array:'
DO I=1,10
    WRITE(*,'(1X,10F6.2)') (A(10*(I-1)+J),J=1,10)
END DO
END

```

计算结果如下:

Original array:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

Sorted array:

1.41	2.95	3.21	3.30	3.48	3.94	4.11	4.62	6.67	7.03
8.01	9.01	9.48	10.13	11.67	12.68	13.42	15.22	15.67	18.34
19.40	19.73	22.81	24.81	25.01	25.75	26.05	27.13	27.54	27.72
28.39	29.82	32.37	34.06	34.86	36.65	37.78	38.27	38.39	38.46
38.98	42.44	44.91	46.08	48.34	49.29	49.95	52.10	52.91	52.99
53.11	53.39	53.42	53.42	54.05	55.61	57.08	59.10	60.31	61.75
62.22	63.16	64.14	66.84	67.61	67.95	68.40	71.44	71.51	71.58
71.79	71.83	73.21	73.96	74.33	76.12	77.17	77.72	79.09	79.36
79.62	81.59	82.09	83.14	83.34	85.02	87.44	89.10	89.35	89.41
89.67	89.95	92.22	93.16	93.39	94.29	95.82	98.40	99.34	99.83

7.5 等价类的确定

1. 功能

对给定 n 个元素的数组及其上的 m 个等价对,按其等价关系确定数组元素的等价类.

子过程 ECLASS 用树型结构确定等价类,而子过程 ECLAZZ 直接用等价对确定等价类.

确定等价类算法所需时间为 $O(m+n)$.

2. 方法

(1) 用树型结构.

设数组元素的编号为 $1, 2, \dots, n$, 而 m 个等价对所对应的元素的编号为 r_i, s_i ($i=1, \dots, m$), 即 r_i 与 s_i 所对应的数组元素等价, 记作 $r_i \sim s_i$ ($i=1, \dots, m$).

假定每一个等价类构成一棵树. 为求这样的树, 记 $F(j)$ 表示树中节点 j 的父母. 开始设 $F(j) = j (j = 1, 2, \dots, n)$ 为具有一个节点的 n 棵树.

对每个等价对 $j \sim k$, 记 $j_1 = F(j)$, 若 $j_1 = j$, 即 $j_1 = j$ 即是 j 的祖先, 若 $j_1 \neq j$, 设 $j_2 = F(j_1)$, 直至 $F(j_p) = j_p$, 则 j_p 为 j 的祖先; 同样设 k 的祖先是 k_q , 由 $j \sim k$, j 与 k 应有相同的祖先, 因此令 $F(j_p) = k_q$.

最后遍历所有 $j \in \{1, 2, \dots, n\}$, 把 $F(j)$ 设置为 j 的最高祖先, 这样具有相同祖先(父母)的 j 在同一棵树上, 即在同一类.

(2) 直接法.

对每个 $j: 1 \leq j \leq n$, 找出与 j 等价的 $k \leq j-1$. 再找出与 k 等价的 l , 则 l 与 j 等价, 并把它们放在与 j 相同的类.

3. 使用说明

(1) ECLASS (NF, N, LISTA, LISTB, M)

N 整型变量, 输入参数, 数组 LISTA 和 LISTB 中单个的元素个数
M 整型变量, 输入参数, 等价对的数目
LISTA M 个元素的一维整型数组, 输入参数, 依次存放等价对的第一个元素的编号
LISTB M 个元素的一维整型数组, 输入参数, 依次存放等价对的第二个元素的编号
NF N 个元素的一维整型数组, 输出参数, 存放每个元素所在等价类的编号

(2) ECLAZZ (NF, N, EQUIV)

N 整型变量, 输入参数, 数组元素的个数
NF N 个元素的一维整型数组, 输出参数, 存放每个元素所在等价类的编号
EQUIV 逻辑函数子程序, 调用等价对, 若 j 与 k 所对应的元素等价, 则 $\text{EQUIV}(j, k)$ 为真, 否则为假, 用户自编

4. 过程

(1) 子过程 ECLASS.

```
SUBROUTINE eclass(nf,n,lista,listb,m)
INTEGER m,n,lista(m),listb(m),nf(n)
INTEGER j,k,l
```

```

do k=1,n
    nf(k)=k
end do
do l=1,m
    j=lista(l)
    do while (nf(j)/=j)
        j=nf(j)
    end do
    k=listb(l)
    do while(nf(k)/=k)
        k=nf(k)
    end do
    if(j/=k)nf(j)=k
end do
do j=1,n
    do while(nf(j)/=nf(nf(j)))
        nf(j)=nf(nf(j))
    end do
end do
END SUBROUTINE eclass

```

(2) 子过程 ECLAZZ.

```

SUBROUTINE eclazz(nf,n,equiv)
INTEGER n,nf(n)
LOGICAL equiv
EXTERNAL equiv
INTEGER jj,kk
nf(1)=1
do jj=2,n
    nf(jj)=jj
    do kk=1,jj-1
        nf(kk)=nf(nf(kk))
        if (equiv(jj,kk)) nf(nf(nf(kk)))=jj
    end do
end do
do jj=1,n
    nf(jj)=nf(nf(jj))

```

```

end do
END SUBROUTINE eclazz

```

5. 例子

(1) 在验证过程 ECLASS 的程序 D7R10 中,我们采用的例子为

LISTA:1,1,5,2,6,2,7,11,3,4,12

LISTB:5,9,13,6,10,14,3,7,15,8,4

按照这个表,1 和 5 等价,1 和 9 等价,等等.

验证程序 D7R10 如下:

```

PROGRAM D7R10
! Driver for routine ECLASS
PARAMETER (N=15,M=11)
DIMENSION LISTA(M),LISTB(M),NF(N),NFLAG(N),NSAV(N)
DATA LISTA/1,1,5,2,6,2,7,11,3,4,12/
DATA LISTB/5,9,13,6,10,14,3,7,15,8,4/
CALL ECLASS(NF,N,LISTA,LISTB,M)
WRITE(*,*)'  NF(I)=, (I=1,N)'
WRITE(*,'(2X,10I6)') (NF(I),I=1,N)
DO I=1,N
    NFLAG(I)=1
END DO
WRITE(*,'(/1X,A)')&
    'Numbers from 1-15 divided according to'
WRITE(*,'(1X,A/)') 'their value modulo 4:'
LCLAS=0
DO I=1,N
    NCLASS=NF(I)
    IF (NFLAG(NCLASS)/=0) THEN
        NFLAG(NCLASS)=0
        LCLAS=LCLAS+1
        K=0
        DO J=1,N
            IF (NF(J)==NF(I)) THEN
                K=K+1
                NSAV(K)=J
            ENDIF
        END DO
    END IF
END DO

```

```

        END DO
        WRITE(*, '(1X,A,I2,A,1X,5I4)') 'Class', &
            LCLAS, ': ', (NSAV(J), J=1, K)
    ENDIF
END DO
END

```

计算结果如下:

```

NF(I)=, (I=1,N)
13 14 15 8 13 14 15 8 13 14 15 8 13 14 15
Numbers from 1—15 divided according to
their value modulo 4:
Class 1 : 1 5 9 13
Class 2 : 2 6 10 14
Class 3 : 3 7 11 15
Class 4 : 4 8 12

```

(2) 子过程 ECLAZZ 进行了同样的分析,但所描述的等价是来自于一个逻辑函数 $\text{EQUIV}(I, J)$, 该逻辑函数分辨了 I 和 J 是否在同一等价类. 在验证程序 D7R11 中, 如果 $(I \bmod 4)$ 和 $(J \bmod 4)$ 是相等的, EQUIV 被定义为 .TRUE.; 否则为 .FALSE..

验证程序 D7R11 如下:

```

PROGRAM D7R11
! Driver for routine ECLAZZ
EXTERNAL EQUIV
LOGICAL EQUIV
PARAMETER (N=15)
DIMENSION NF(N), NFLAG(N), NSAV(N)
CALL ECLAZZ(NF, N, EQUIV)
WRITE(*, *) 'NF(I)=, (I=1,N)'
WRITE(*, '(3X,10I6)') (NF(I), I=1, N)
DO I=1, N
    NFLAG(I)=1
END DO
WRITE(*, '(1X,A)') &
    'Numbers from 1—15 divided according to'

```

```

WRITE(*,'(1X,A/)') 'their value modulo 4;'
LCLAS=0
DO I=1,N
    NCLASS=NF(I)
    IF (NFLAG(NCLASS)/=0) THEN
        NFLAG(NCLASS)=0
        LCLAS=LCLAS+1
        K=0
        DO J=1,N
            IF (NF(J)==NF(I)) THEN
                K=K+1
                NSAV(K)=J
            ENDIF
        END DO
        WRITE(*,'(1X,A,I2,A,1X,5I4)') 'Class', &
            LCLAS,':',(NSAV(J),J=1,K)
    ENDIF
END DO
END PROGRAM
LOGICAL FUNCTION EQUIV(I,J)
    EQUIV=.FALSE.
    IF (MOD(I,4)==MOD(J,4)) EQUIV=.TRUE.
END FUNCTION EQUIV

```

计算结果如下:

```

NF(I)=, (I=1,N)
13  14  15  12  13  14  15  12  13  14  15  12  13  14  15
Numbers from 1-15 divided according to

```

their value modulo 4;

Class 1 : 1 5 9 13

Class 2 : 2 6 10 14

Class 3 : 3 7 11 15

Class 4 : 4 8 12

附 录

文件 TARRAY.DAT:

29.82	71.51	3.30	87.44	53.42	63.16	89.10	25.75	93.16	27.72
71.58	48.34	53.11	18.34	27.13	60.31	83.34	22.81	66.84	52.91
53.42	15.22	8.01	53.39	76.12	79.09	67.61	38.39	24.81	73.21
13.42	52.10	34.86	99.83	38.46	81.59	61.75	79.62	93.39	3.21
99.34	92.22	94.29	7.03	6.67	89.35	83.14	9.01	12.68	62.22
2.95	85.02	95.82	73.96	49.29	77.72	36.65	3.48	38.98	71.83
1.41	9.48	32.37	89.95	28.39	79.36	54.05	46.08	11.67	37.78
77.17	74.33	10.13	4.62	49.95	68.40	19.40	34.06	4.11	98.40
42.44	64.14	89.41	52.99	71.79	3.94	19.73	44.91	71.44	59.10
27.54	15.67	67.95	55.61	26.05	25.01	82.09	89.67	57.08	38.27

第 8 章 特征值问题

本章介绍实矩阵的特征值与特征向量的计算.

当矩阵 A 为实对称矩阵时, 若其阶数 n 不大, 可采用雅可比方法计算特征值. 它计算方便, 算法稳定, 收敛快, 一般对于阶数不很高的矩阵, 特别是接近于对角的实对称矩阵(非对角元素与对角元素之比的绝对值都较小)时, 更显得有效. 此外, 用雅可比方法求特征向量也很方便. 但要注意, 用该方法计算实对称矩阵的所有特征值和特征向量时, 其计算矩阵序列的工作量较大, 速度也慢. 总的工作量大约是 $12n^3 \sim 20n^3$.

若对称矩阵的阶数较大, 可首先采用子过程 TRED2 把实对称矩阵变为对称三对角矩阵, 再利用子过程 TQLI(隐式 QL 方法)求其全部特征值和特征向量. 子过程 TQLI 的总运算量大约是 $30n^2$. 若还要求特征向量, 则工作量较大, 大约是 $3n^3$. 这种方法对于对称矩阵效果非常满意.

对于非对称矩阵特征值和特征向量的计算, 不可能有像对称矩阵一样满意的方法. 原因是, 第一, 非对称矩阵的特征值对于矩阵元素的微小变化可能非常敏感; 第二, 矩阵本身可能是亏损的(即没有完备的特征向量组). 目前计算一般实矩阵所有特征值与特征向量的最为有效的方法就是 QR 方法. 它首先把实矩阵相似约化为上赫申伯格矩阵, 再求该上赫申伯格矩阵的特征值. 本章中我们采用带有原点位移的双重步 QR 方法, 收敛速度很快, 其基本收敛速度是二次的, 当原矩阵是对称矩阵时, 可达三次收敛速度. 用 QR 方法还可求实系数多项式的全部根.

在 8.4 节中, 用消去法(初等相似变换)把一般实矩阵相似约化为上赫申伯格矩阵, 而在相似约化之前, 应该先对原矩阵进行平衡处理, 平衡过程是一个 $O(n^2)$ 过程. 对于相似约化, 一般还可采用豪斯霍尔德方法, 该方法稳定性好, 但消去法的运算量较小, 效率大约是豪斯霍尔德方法的两倍.

在用 QR 方法时, 如果上赫申伯格矩阵类似正交矩阵, 算法可能失效或者可能误差较大, 不过在实用中这种情况是极其例外的. 在用 QR 方法求得特征值后, 可再用反幂法(反迭代法)求出相应的特征向量. 如果有复共轭特征值, 则应该采用包含特征向量计算的 QR 方法. 请参阅参考文献.

8.1 对称矩阵的雅可比变换

1. 功能

用雅可比(Jacobi)旋转变换求实对称矩阵的全部特征值及其对应的特征向量.

子过程 JACOBI 计算出实对称矩阵的全部特征值及其相应的特征向量;子过程 EIGSRT 用直接插入法对由 JACOBI 求出的特征值按降序进行重排序,并同时重排相应的特征向量.

雅可比方法不是一种效率最高的算法,但它是简单可靠的.

2. 方法

(1) 对矩阵 A 进行一系列雅可比旋转变换,其一次雅可比旋转变换为:设

$$A = (a_{ij})_{n \times n}$$

$$P_{pq} = \begin{bmatrix} 1 & & & & \\ & \dots & & & \\ & & c & \dots & s \\ & & \vdots & 1 & \vdots \\ & & -s & \dots & c & \dots \\ & & & & & 1 \end{bmatrix}$$

其中,除去 p 行 p 列, q 行 q 列的两个元素 c 外,所有对角元素都是单位的.除去两个元素 s 和 $-s$ 之外,所有非对角元素都是零.

$$c = \cos \varphi, \quad s = \sin \varphi, \quad c^2 + s^2 = 1$$

$$A' = P_{pq}^T A P_{pq} = (a'_{ij})$$

则

$$a'_{rp} = ca_{rp} - sa_{rq}$$

$$a'_{rq} = ca_{rq} + sa_{rp}, \quad r \neq p, \quad r \neq q$$

$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2sc a_{pq}$$

$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2sc a_{pq}$$

$$a'_{pq} = (c^2 - s^2) a_{pq} + sc(a_{pp} - a_{qq})$$

由要求 $a'_{pq} = 0$ 得

$$\theta = \cot 2\varphi \equiv \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}}, \quad |\varphi| \leq \frac{\pi}{4}$$

设 $t \equiv s/c$, 则 $t^2 + 2t\theta - 1 = 0$, 其较小的根

$$t = \frac{\operatorname{sgn}(\theta)}{|\theta| + \sqrt{1 + \theta^2}}$$

是使 $|\varphi| \leq \frac{\pi}{4}$ 的根. 于是

$$c = 1/\sqrt{1 + \theta^2}, \quad s = t \cdot c$$

(注意, 在上述计算中, 当 θ 很大时, θ^2 可能会溢出, 但这时由 t 的计算式知可取 $t \equiv 1/(2\theta)$).

(2) 为使舍入误差最小, 把 a'_{ij} 写成如下形式:

$$\begin{aligned} a'_{pq} &= 0 \\ a'_{pp} &= a_{pp} - ta_{pq} \\ a'_{qq} &= a_{qq} + ta_{pq} \\ a'_{ip} &= a_{ip} - s(a_{rq} + \tau a_{rp}) \\ a'_{rq} &= a_{rq} + s(a_{rp} - \tau a_{rq}) \end{aligned}$$

其中 $\tau \equiv \tan(\varphi/2) = s/(1+c)$.

(3) 进一步, 依次作变换 $P_{12}, P_{13}, \dots, P_{1n}; P_{23}, P_{24}, \dots, P_{2n}, \dots$, 共进行 $\frac{n(n-1)}{2}$ 次旋转变换, 称为一次扫描. 在前三次扫描中, 采用阈雅可比方法, 阈值 $\epsilon = \frac{1}{5} \frac{S_0}{n^2}$, $S_0 = \sum_{i,j} |a_{ij}|$. 在以后的扫描中, 采用循环雅可比方法, 直至 $|a_{pq}| \ll |a_{pp}|$ 且 $|a_{pq}| \ll |a_{qq}|$, 其判别准则是考察 $|a_{pq}| < 10^{-(d+2)} \cdot a$, 其中 $a = \min\{|a_{pp}|, |a_{qq}|\}$, d 是机器上有效数字的位数.

(4) 特征向量的计算.

设 $V = P_1 P_2 \dots$, 其中 P_i 为每次所作的雅可比旋转矩阵. 则由

$$D = V^T A V$$

的对角元给出 A 的特征值, V 的列即为相应的特征向量.

取初始的 V 为单位矩阵, 设

$$V' = V \cdot P_i, \quad V = (v_{ij}), \quad V' = (v'_{ij}), \quad P_i = P_{pq}$$

则同对 A 的变换有

$$\begin{aligned} v'_{rs} &= v_{rs}, \quad s \neq p, \quad s \neq q \\ v'_{rp} &= v_{rp} - s(v_{rq} + \tau v_{rp}) \\ v'_{rq} &= v_{rq} + s(v_{rp} - \tau v_{rq}). \end{aligned}$$

其中 s 与 τ 的意义同前.

3. 使用说明

(1) JACOBI (A,N,D,V,NROT)

N 整型变量,输入参数,矩阵阶数
A $N \times N$ 个元素的二维实型数组,输入、输出参数,输入实对称矩阵 A ,输出变换后的矩阵,但仅上三角元变化
D N 个元素的一维实型数组,输出参数,输出 A 的特征值
V $N \times N$ 个元素的二维实型数组,输出参数,存放 A 的正规化的特征向量
NROT 整型变量,输出参数,输出雅可比旋转变换的次数

(2) EIGSRT(D,V,N)

N 整型变量,输入参数,矩阵阶数
D N 个元素的一维实型数组,输入、输出参数,输入时,存放子过程 JACOBI 输出的 D ,输出时,存放重排后的矩阵 A 的特征值
V $N \times N$ 个元素的二维实型数组,输入、输出参数,输入时,存放子过程 JACOBI 输出的 V ,输出时,存放重排后的矩阵的特征向量

4. 过程

(1) 子过程 JACOBI.

```
SUBROUTINE jacobi(a,n,d,v,nrot)
INTEGER n,nrot,NMAX
REAL a(n,n),d(n),v(n,n)
PARAMETER (NMAX=500)
INTEGER i,ip,iq,j
REAL c,g,h,s,sm,t,tau,theta,tresh,b(NMAX),z(NMAX)
do ip=1,n
  do iq=1,n
    v(ip,iq)=0.
  end do
  v(ip,ip)=1.
end do
do ip=1,n
  b(ip)=a(ip,ip)
  d(ip)=b(ip)
```

```

      z(ip)=0.
end do
nrot=0
do i=1,50
  sm=0.
  do ip=1,n-1
    do iq=ip+1,n
      sm=sm+abs(a(ip,iq))
    end do
  end do
  if(sm==0.) return
  if(i<4) then
    tresh=0.2*sm/n**2
  else
    tresh=0.
  endif
  do ip=1,n-1
    do iq=ip+1,n
      g=100.*abs(a(ip,iq))
      if((i>4).and.(abs(d(ip))+g==abs(d(ip)))&
        .and.(abs(d(iq))+g==abs(d(iq)))) then
        a(ip,iq)=0.
      else if(abs(a(ip,iq))>tresh) then
        h=d(iq)-d(ip)
        if(abs(h)+g==abs(h)) then
          t=a(ip,iq)/h
        else
          theta=0.5*h/a(ip,iq)
          t=1./(abs(theta)+sqrt(1.+theta**2))
          if(theta<0.) t=-t
        endif
        c=1./sqrt(1+t**2)
        s=t*c
        tau=s/(1.+c)
        h=t*a(ip,iq)
        z(ip)=z(ip)-h
        z(iq)=z(iq)+h
      end if
    end do
  end do
end do

```

```

d(ip)=d(ip)-h
d(iq)=d(iq)+h
a(ip,iq)=0.
do j=1,ip-1
    g=a(j,ip)
    h=a(j,iq)
    a(j,ip)=g-s*(h+g*tau)
    a(j,iq)=h+s*(g-h*tau)
end do
do j=ip+1,iq-1
    g=a(ip,j)
    h=a(j,iq)
    a(ip,j)=g-s*(h+g*tau)
    a(j,iq)=h+s*(g-h*tau)
end do
do j=iq+1,n
    g=a(ip,j)
    h=a(iq,j)
    a(ip,j)=g-s*(h+g*tau)
    a(iq,j)=h+s*(g-h*tau)
end do
do j=1,n
    g=v(j,ip)
    h=v(j,iq)
    v(j,ip)=g-s*(h+g*tau)
    v(j,iq)=h+s*(g-h*tau)
end do
nrot=nrot+1
endif
end do
end do
do ip=1,n
    b(ip)=b(ip)+z(ip)
    d(ip)=b(ip)
    z(ip)=0.
end do
end do

```

```

pause 'too many iterations in jacobi'
END SUBROUTINE jacobi

```

(2) 子过程 EIGSRT.

```

SUBROUTINE eigsrt(d,v,n)
INTEGER n,np
REAL d(n),v(n,n)
INTEGER i,j,k
REAL p
do i=1,n-1
    k=i
    p=d(i)
    do j=i+1,n
        if(d(j)>=p) then
            k=j
            p=d(j)
        endif
    end do
    if(k/=i) then
        d(k)=d(i)
        d(i)=p
        do j=1,n
            p=v(j,i)
            v(j,i)=v(j,k)
            v(j,k)=p
        end do
    endif
end do
END SUBROUTINE eigsrt

```

5. 例子

(1) 验证 JACOBI 的程序 D8R1 中,将已知矩阵送入 JACOBI,从而算出矩阵的特征值和特征向量.最后,为了验证特征向量,我们将原矩阵乘以特征向量并和特征向量按分量作比,其结果应该都等于特征值.验证程序 D8R1 如下:

```
PROGRAM D8R1
```

```

! Driver for routine JACOBI
PARAMETER(N=3,NMAT=3)
DIMENSION D(N),V(N,N),R(N)
DIMENSION A(3,3),E(N,N)
DATA A/1.0,2.0,3.0,2.0,2.0,3.0,3.0,3.0,3.0/
DO II=1,3
    DO JJ=1,3
        E(II,JJ)=A(II,JJ)
    END DO
END DO
CALL JACOBI(E,3,D,V,NROT)
WRITE(*, '(1X,A,I3)') &
    'Number of JACOBI rotations:', NROT
WRITE(*, '(/1X,A)') 'Eigenvalues:'
DO J=1,n
    WRITE(*, '(1X,5F12.6)') D(J)
END DO
WRITE(*, '(/1X,A)') 'Eigenvectors:'
DO J=1,N
    WRITE(*, '(1X,T5,A,I3)') 'Number', J
END DO
! Eigenvector test
WRITE(*, '(/1X,A)') 'Eigenvector Test'
DO J=1,N
    DO L=1,N
        R(L)=0.0
        DO K=1,N
            IF(K>L) THEN
                KK=L
                LL=K
            ELSE
                KK=K
                LL=L
            ENDIF
            R(L)=R(L)+A(LL,KK)*V(K,J)
        END DO
    END DO
END DO

```

```

WRITE(*, '(/1X,A,I3)' ) 'Vector Number',J
WRITE(*, '(/1X,T7,A,T18,A,T31,A)' ) &
    'Vector', 'Mtrx * Vec.', 'Ratio'
DO L=1,N
    RATIO=R(L)/V(L,J)
    WRITE(*, '(1X,3F12.6)' ) V(L,J),R(L),RATIO
END DO
END DO
END

```

计算结果如下:

Matrix Number 1

Number of JACOBI rotations: 11

Eigenvalues:

-0.338922

7.516538

-1.177617

Eigenvectors:

Number 1

0.425090	-0.829153	0.363048
----------	-----------	----------

Number 2

0.482739	0.546963	0.683955
----------	----------	----------

Number 3

-0.765677	-0.115485	0.632773
-----------	-----------	----------

Eigenvector Test

Vector Number 1

Vector	Mtrx * Vec.	Ratio
0.425090	-0.144072	-0.338921
-0.829153	0.281018	-0.338922
0.363048	-0.123045	-0.338921

Vector Number 2

Vector	Mtrx * Vec.	Ratio
0.482739	3.628530	7.516540
0.546963	4.111269	7.516537
0.683955	5.140972	7.516539

Vector Number 3

Vector	Mtrx * Vec.	Ratio
--------	-------------	-------

-0.765677	0.901671	-1.177617
-0.115485	0.135997	-1.177618
0.632773	-0.745165	-1.177617

Press RETURN to END DO...

(2) 验证 EIGSRT 的程序 D8R2 中,我们采取上一个例子中的二维数组. JACOBI 可算出该矩阵的特征值 D 和特征向量 V . 这些 D 和 V 送进 EIGSRT 后,即可返回按降序排列的特征值. 验证程序 D8R2 如下:

```

PROGRAM D8R2
! Driver for routine EIGSRT
PARAMETER(N=3)
! USES JACOBI
DIMENSION D(N),V(N,N),C(N,N)
DATA C/1.0,2.0,3.0,2.0,2.0,3.0,3.0,3.0,3.0/
CALL JACOBI(C,N,D,V,NROT)
WRITE(*,*) 'Unsorted Eigenvectors:'
DO I=1,N
    WRITE(*, '(1X,A,I3,A,F12.6)') 'Eigenvalue', &
        I, '=', D(I)
    WRITE(*,*) 'Eigenvector:'
    WRITE(*, '(10X,5F12.6)') (V(J,I), J=1,N)
END DO
WRITE(*, '(//,A,//)') ' * * * * * sorting * * * * * '
CALL EIGSRT(D,V,N)
WRITE(*,*) 'Sorted Eigenvectors:'
DO I=1,N
    WRITE(*, '(1X,A,I3,A,F12.6)') 'Eigenvalue', &
        I, '=', D(I)
    WRITE(*,*) 'Eigenvector:'
    WRITE(*, '(10X,5F12.6)') (V(J,I), J=1,N)
END DO
END

```

计算结果如下:

Unsorted Eigenvectors:

Eigenvalue 1 = -0.338922

```

Eigenvector:
      0.425090   -0.829153   0.363048
Eigenvalue 2=7.516538
Eigenvector:
      0.482739   0.546963   0.683955
Eigenvalue 3=-1.177617
Eigenvector:
      -0.765677   -0.115485   0.632773
* * * * * sorting * * * * *
Sorted Eigenvectors:
Eigenvalue 1=7.516538
Eigenvector:
      0.482739   0.546963   0.683955
Eigenvalue 2=-0.338922
Eigenvector:
      0.425090   0.829153   0.363048
Eigenvalue 3=-1.177617
Eigenvector:
      -0.765677   -0.115485   0.632773

```

8.2 变实对称矩阵为三对角对称矩阵

1. 功能

用豪斯霍尔德(Householder)矩阵把实对称矩阵变形到对称三对角矩阵,并可积累变换矩阵.当仅求特征值时,其工作量大约是 $2n^3/3$,若还要求特征向量,其工作量大约是 $4n^3/3$.

2. 方法

(1) 对 A 施行 $n-2$ 次正交变换,其每一次为:

设

$$P = I - \frac{uu^T}{H}, \quad H = \frac{1}{2} \|u\|_2^2$$

记

$$p = \frac{Au}{H}$$

$$k = \frac{u^T p}{2II}$$

$$q = p - ku$$

则

$$A' = PAP = A - qu^T - uq^T$$

(2) 变换从第 n 列开始, 即对 $m=1, 2, \dots, n-2$, 每次变换的向量 u 具有形式:

$$u = (a_{i1}, a_{i2}, \dots, a_{i, n-2}, a_{i, n-1} \pm \sqrt{\sigma}, 0, \dots, 0)^T$$

其中

$$i = n - m + 1 = n, \quad n-1, \dots, 3$$

$$\sigma = a_{i1}^2 + \dots + a_{i, n-2}^2$$

定义

$$\varepsilon = \sum_{k=1}^{i-1} |a_{ik}|$$

若 $\varepsilon=0$, 则跳过这次变换, 否则先把 a_{ik} 变为 a_{ik}/ε .

(3) 若需要求特征向量, 则需积累变换矩阵

$$Q = P_1 P_2 \dots P_{n-2}$$

其中 $P_i (i=1, \dots, n-2)$ 为每次的变换矩阵. 令

$$Q_{n-2} = P_{n-2}$$

$$Q_j = P_j Q_{j+1}, \quad j = n-3, \dots, 1$$

则

$$Q = Q_1$$

3. 使用说明

TRED2 (A, N, D, E)

N 整型变量, 输入参数, 实对称矩阵的阶数

A $N \times N$ 个元素的二维实型数组, 输入、输出参数, 输入时存放实对称矩阵 A , 输出时存放积累的正交变换矩阵 Q

D N 个元素的一维实型数组, 输出参数, 存放变形后的三对角矩阵的对角线元素

E N 个元素的一维实型数组, 输出参数, 存放变形后的三对角矩阵的非对角线元素, 其中 $E(1)=0$

4. 过程

子过程 TRED2.

```

SUBROUTINE tred2(a,n,d,e)
  INTEGER n
  REAL a(n,n),d(n),e(n)
  INTEGER i,j,k,l
  REAL f,g,h,hh,scale
  do i=n,2,-1
    l=i-1
    h=0.
    scale=0.
    if(l>1)then
      do k=1,l
        scale=scale+abs(a(i,k))
      end do
      if(scale==0.) then
        e(i)=a(i,l)
      else
        do k=1,l
          a(i,k)=a(i,k)/scale
          h=h+a(i,k)*a(i,k)
        end do
        f=a(i,l)
        g=-sign(sqrt(h),f)
        e(i)=scale*g
        h=h-f*g
        a(i,l)=f-g
        f=0.
        do j=1,l
          ! Omit following line if finding only eigenvalues
          a(j,i)=a(i,j)/h
          g=0.
          do k=1,j
            g=g+a(j,k)*a(i,k)
          end do
          do k=j+1,l
            g=g+a(k,j)*a(i,k)
          end do
          e(j)=g/h
        end do
      end if
    end if
  end do

```

```

        f=f+e(j)*a(i,j)
    end do
    hh=f/(h+h)
    do j=1,l
        f=a(i,j)
        g=e(j)-hh*f
        e(j)=g
        do k=1,j
            a(j,k)=a(j,k)-f*e(k)-g*a(i,k)
        end do
    end do
endif
else
    c(i)=a(i,l)
endif
d(i)=h
end do
! Omit following line if finding only eigenvalues.
d(1)=0.
e(1)=0.
do i=1,n
! Delete lines from here ...
    l=i-1
    if(d(i)/=0.)then
        do j=1,l
            g=0.
            do k=1,l
                g=g+a(i,k)*a(k,j)
            end do
            do k=1,l
                a(k,j)=a(k,j)-g*a(k,i)
            end do
        end do
    endif
!... to here when finding only eigenvalues.
    d(i)=a(i,i)
! Also delete lines from here ...

```

```

      a(i,i)=1.
      do j=1,l
        a(i,j)=0.
        a(j,i)=0.
      end do
      !... to here when finding only eigenvalues.
    end do
  END SUBROUTINE tred2

```

5. 例子

在验证 TRED2 的程序 D8R3 中,我们再次引用前面例子中的矩阵. 将矩阵 A 复制到矩阵 C 中,以便被子过程 TRED2 调用. 矩阵 A 被留下用于检查计算结果. 执行子过程 TRED2 即打印出三对角矩阵中对角线上的元素及非对角线上的元素. 为了验证计算结果,我们利用公式 $F=A^TCA$,将 TRED2 输出的 A 代入上式,计算出三对角矩阵 F ,而该三对角矩阵对角线上的元素和非对角线上的元素均与 TRED2 计算结果一致. 验证程序 D8R3 如下:

```

PROGRAM D8R3
! Driver for routine TRED2
PARAMETER(N=3)
DIMENSION A(N,N),C(N,N),D(N),E(N),F(N,N)
DATA C/1.0,2.0,3.0,2.0,2.0,3.0,3.0,3.0,3.0/
DO I=1,N
  DO J=1,N
    A(I,J)=C(I,J)
  END DO
END DO
CALL TRED2(A,N,D,E)
WRITE(*, '(1X,A)') 'Diagonal elements'
WRITE(*, '(1X,5F12.6)') (D(I),I=1,N)
WRITE(*, '(1X,A)') 'Off-diagonal elements'
WRITE(*, '(1X,5F12.6)') (E(I),I=2,N)
! Check transformation matrix
DO J=1,N
  DO K=1,N
    F(J,K)=0.0
    DO L=1,N

```

```

      DO M=1,N
        F(J,K)=F(J,K)+A(L,J)*C(L,M)*A(M,K)
      END DO
    END DO
  END DO
  ! How does it look?
  WRITE(*, '(1X,A)') 'Tridiagonal matrix'
  DO I=1,N
    WRITE(*, '(1X,10F12.6)') (F(I,J),J=1,N)
  END DO
END

```

计算结果如下:

Diagonal elements

-0.500000 3.500000 3.000000

Off-diagonal elements

0.500000 -4.242640

Tridiagonal matrix

-0.500000 0.500000 0.000000

0.500000 3.500000 -4.242640

0.000000 -4.242640 3.000000

8.3 三对角矩阵的特征值和特征向量

1. 功能

本算法为隐式 QL 算法, 确定一个实对称三对角矩阵的特征值和特征向量, 本算法运行时间短, 占用内存较少, 舍入误差也小.

2. 方法

QL 算法是 QR 算法在实对称三对角矩阵情况下的变形, 其一次迭代方法如下.

(1) 作分解.

$$A_s = k_s I + Q_s L_s$$

$$A_{s+1} = L_s Q_s + k_s I = Q_s^T A_s Q_s$$

其中 Q_s 为正交矩阵, L_s 为下三角矩阵, 而位移 k_s 如下选取:

假设已求得 $r-1$ 个特征值, 则从 A_s 中删除前 $r-1$ 行和前 $r-1$ 列, 留下

$$A_s = \begin{bmatrix} 0 & & & & & 0 \\ & \dots & & & & \\ \vdots & & 0 & & & \vdots \\ & & d_r^{(s)} & e_r^{(s)} & & \\ \vdots & & e_r^{(s)} & d_{r+1}^{(s)} & & \\ & & & \dots & & 0 \\ & & & & d_{n-1}^{(s)} & e_{n-1}^{(s)} \\ 0 & & \dots & & e_{n-1}^{(s)} & d_n^{(s)} \end{bmatrix}$$

取 k_s 为 2×2 子矩阵

$$\begin{bmatrix} d_r^{(s)} & e_r^{(s)} \\ e_r^{(s)} & d_{r+1}^{(s)} \end{bmatrix}$$

接近于 $d_r^{(s)}$ 的特征值, 即

$$k_s = d_r^{(s)} - e_r^{(s)} / (g + \text{sign}(g) \sqrt{1 + g^2})$$

$$g = (d_{r+1}^{(s)} - d_r^{(s)}) / (2e_r^{(s)})$$

于是

$$Q_s^T = P_1^{(s)} \cdots P_{n-1}^{(s)}, \quad L_s = Q_s^T (A_s - k_s I)$$

其中 $P_i^{(s)}$ 是在 $(i, i+1)$ 平面上为了消失在 $(i, i+1)$ 位置上的元素所作的旋转变换, 即

$$P_i^{(s)} = \begin{bmatrix} 1 & & & & & \\ & \dots & & & & \\ & & 1 & & & \\ & & & c & s & \\ & & & -s & c & \\ & & & & & 1 \\ & & & & & \dots & 1 \end{bmatrix} \quad \begin{array}{l} \dots (\text{第 } i \text{ 行}) \\ \dots (\text{第 } i+1 \text{ 行}) \end{array}$$

(2) 在矩阵的元素的阶差很大时, 为不失模较小的特征值的精度, 采用隐式位移的 QL 算法.

由分解的惟一性, 若有

$$A_{s+1} = Q_s^T A_s Q_s$$

其中 Q_s^T 正交且其最后一行与 Q_s^T 的最后一行相同, 使 \bar{A}_{s+1} 为三对角矩阵, 则 $\bar{Q}_s = Q_s$, $A_{s+1} = A_{s+1}$.

由于 Q_s^T 的最后一行为 $P_{n-1}^{(s)}$ 的最后一行, $P_{n-1}^{(s)}$ 为 $(n-1, n)$ 平面上消去 $A_s - k_s I$ 的 $(n-1, n)$ 位置的元素的旋转变换, 所以 $P_{n-1}^{(s)}$ 中的参数为

$$c = \frac{d_n^{(s)} - k_s}{\sqrt{(e_n^{(s)})^2 + (d_n^{(s)} - k_s)^2}}, \quad s = \frac{-e_n^{(s)}}{\sqrt{(e_n^{(s)})^2 + (d_n^{(s)} - k_s)^2}}$$

又矩阵 $P_{n-1}^{(s)} A_s P_{n-1}^{(s)T}$ 形如

$$\begin{bmatrix} \cdots & & & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \otimes \\ & & & \times & \times & \times \\ & & & \otimes & \times & \times \end{bmatrix}$$

取 $P_i^{(s)}$ 为 $(i, i+1)$ 平面上消去 $(i, i+2)$, $(i+2, i)$ 位置元素的旋转变换 $(i = n-2, n-1, \dots, 1)$, 则

$$Q_s^T = \bar{P}_1^{(s)} \bar{P}_2^{(s)} \cdots \bar{P}_{n-2}^{(s)} P_{n-1}^{(s)}$$

使 $A_{s+1} = \bar{Q}_s^T A_s \bar{Q}_s$ 为三对角对称矩阵, 且 \bar{Q}_s^T 与 $P_{n-1}^{(s)}$ 的最后一行相同.

3. 使用说明

TQLI (D, E, N, Z)

N 整型变量, 输入参数, 矩阵阶数

D N 个元素的一维实型数组, 输入、输出参数, 输入时存放对称三对角矩阵的对角元, 输出时存放矩阵的特征值

E N 个元素的一维实型数组, 输入参数, 存放对称三对角矩阵的非对角元素, 其中 $E(1)$ 任意

Z $N \times N$ 个元素的二维实型数组, 输入、输出参数, 如果是求三对角矩阵的特征向量, 则输入为单位矩阵, 若是求由一般对称矩阵变形到三对角矩阵的一般对称矩阵的特征向量, 则输入子过程 TRED2 的输出矩阵 A, 输出时其第 K 列存放对应于特征值 $D(K)$ 的规范化特征向量

4. 过程

子过程 TQLI.

SUBROUTINE tqli(d, e, n, z)

```

INTEGER n
REAL d(n),e(n),z(n,n)
INTEGER i,iter,k,l,m
LOGICAL done
REAL b,c,dd,f,g,p,r,s
do i=2,n
    e(i-1)=e(1)
end do
e(n)=0.
do l=1,n
    iter=0
    do
        done=.false.
        do m=l,n-1
            dd=abs(d(m))+abs(d(m+1))
            if (abs(e(m))+dd==dd) exit
        end do
        if ((abs(c(m))+dd)/=-dd) m=n
    if(m/=l) then
        if(iter==30) pause 'too many iterations in tqli'
        iter=iter+1
        g=(d(l-1)-d(l))/(2.*e(l))
        r=sqrt(g*g+1.)
        g=d(m)-d(l)+e(l)/(g+sign(r,g))
        s=1.
        c=1.
        p=0.
        do i=m-1,l,-1
            f=s*e(i)
            b=c*e(i)
            if(abs(f)>=abs(g)) then
                c=g/f
                r=sqrt(c*c+1.)
                e(i+1)=f*r
                s=1/r
                c=c*s
            else

```

```

      s=f/g
      r=sqrt(s**2+1.)
      c(i+1)=g*r
      c=1/r
      s=c*s
    endif
    g=d(i+1)-p
    r=(d(i)-g)*s+2.*c*b
    p=s*r
    d(i+1)=g+p
    g=c*r-b
! Omit lines from here ...
    do k=1,n
      f=z(k,i+1)
      z(k,i+1)=s*z(k,i)+c*f
      z(k,i)=c*z(k,i)-s*f
    end do
! ... to here when finding only eigenvalues.
  end do
  d(l)=d(l)-p
  e(l)=g
  c(m)=0.
  done=-1
endif
  if (.not. done) exit
end do
end do
END SUBROUTINE tqli

```

5. 例子

在验证 TQLI 的程序 D8R4 中, 首先将矩阵 A 复制进数组 C 中, 然后, 将数组 C 送入子过程 TRED2(见 8.2 节)中, 以便将矩阵 C 简化为实对称三对角矩阵; 最后, 将 TRED2 所输出的对角线上的元素数组 D 和非对角线上的元素 E 及正交变换矩阵 C 送入 TQLI, 即可得特征值和特征向量. 为了验证计算结果, 我们将特征向量和原矩阵 A 相乘并除以特征向量(按分量), 该结果应该和特征值相等(注意: 在某些情况下, 一些特征向量的元素是零或靠近零, 这时用

“div. by 0”表示). 验证程序 D8R4 如下:

```

PROGRAM D8R4
! Driver for routine TQLI
PARAMETER (N=3, TINY=1.0E-6)
! USES tred2, tqli
DIMENSION A(N,N), C(N,N), D(N), E(N), F(N)
DATA A/1.0, 2.0, 3.0, 2.0, 2.0, 3.0, 3.0, 3.0, 3.0/
DO I=1, N
    DO J=1, N
        C(I,J)=A(I,J)
    END DO
END DO
CALL TRED2(C, N, D, E)
CALL TQLI(D, E, N, C)
WRITE(*, '(/1X, A)') &
    'Eigenvectors for a real symmetric matrix'
DO I=1, N
    DO J=1, N
        F(J)=0.0
        DO K=1, N
            F(J)=F(J)+A(J,K)*A(K,I)
        END DO
    END DO
    WRITE(*, '(/1X, A, I3, A, F10.6)') 'Eigenvalue', &
        I, '=', D(I)
    WRITE(*, '(/1X, T7, A, T17, A, T31, A)') 'Vector', &
        'Mtrx * vect.', 'Ratio'
    DO J=1, N
        IF (ABS(A(J,I))<TINY) THEN
            WRITE(*, '(1X, 2F12.6, A12)') C(J,I), F(J), &
                'div. by 0'
        ELSE
            WRITE(*, '(1X, 2F12.6, E14.6)') C(J,I), F(J), &
                F(J)/A(J,I)
        ENDIF
    END DO
END DO

```

```

WRITE(*, '(//1X,A)') 'Press ENTER to END DO...'
READ(*, *)
END DO
END

```

计算结果如下:

Eigenvectors for a real symmetric matrix

Eigenvalue 1= 0.338922

Vector	Mtrx * vect.	Ratio
0.425090	-0.144072	-0.338921E+00
-0.829153	0.281018	-0.338922E+00
0.363048	-0.123045	-0.338922E+00

Press ENTER to END DO...

Eigenvalue 2= -1.177616

Vector	Mtrx * vect.	Ratio
0.765677	-0.901674	-0.117762E+01
0.115485	-0.135997	-0.117762E+01
-0.632773	0.745165	-0.117762E+01

Press ENTER to END DO...

Eigenvalue 3= 7.516539

Vector	Mtrx * vect.	Ratio
-0.482739	-3.628530	0.751654E+01
-0.546963	-4.111269	0.751654E+01
-0.683955	-5.140972	0.751654E+01

Press ENTER to END DO...

8.4 变一般矩阵为赫申伯格矩阵

1. 功能

把实的非对称矩阵化简为上赫申伯格(Hessenberg)矩阵.

子过程 BALANC 用于对一般实矩阵进行平衡处理. 平衡处理用相似变换减小矩阵的范数, 改变算法在执行过程中特征值对舍入误差的灵敏性. 它对对称矩阵无影响.

子过程 ELMHES 用消去法将一般实矩阵化为上赫申伯格矩阵. 它对非实对称矩阵特别有用.

2. 方法

(1) 平衡(balacing).

设 $A = \text{diag}(A) + A_0$, 从 A_0 开始形成如下矩阵序列 $\{A_k\}$:

① 令 R_k, C_k 分别为 $A_{k-1} (k=1, 2, \dots)$ 的第 i 行和第 i 列元素的绝对值之和, i 的选取法是 $i-1 \equiv k-1 (k=1, 2, \dots)$, 此等式以 n 为周期, $C_k \neq 0, R_k \neq 0$. 取 $-\beta > 0$ 为底数, 则存在惟一的一个整数 σ 满足 $\beta^{2\sigma-1} < \frac{R_k}{C_k} < \beta^{2\sigma+1}$, 令 $f = \beta^\sigma$.

② 对给定的常数 $v \leq 1$, 令

$$\bar{D}_k = \begin{cases} I + (f-1)e_i e_i^T & \text{当 } C_k f + R_k/f < v(C_k + R_k) \text{ 时} \\ I & \text{当 } C_k f + R_k/f > v(C_k + R_k) \text{ 时} \end{cases}$$

其中 $I = (e_1, \dots, e_n)$ 是单位矩阵.

③ 形成:

$$D_k = \bar{D}_k D_{k-1}, \quad D_0 = I; \quad A_k = \bar{D}_k^{-1} A_{k-1} \bar{D}_k \quad (k=1, 2, \dots)$$

如果对于一个完整的周期 $D_k = I$, 则迭代结束.

$$\textcircled{4} \lim_{k \rightarrow \infty} A_k = \bar{A}, \quad \bar{A} = D^{-1} A D, \quad \|A\|_1 \geq \inf \|D_\beta^{-1} A D_\beta\|_1.$$

(2) 用消去法把矩阵化为上赫申伯格矩阵.

设 $A_1 \equiv A$ 已变成 A_r , 其前 $r-1$ 行和前 $r-1$ 列是上 H -型矩阵, 则第 r 步是:

① 求 $|a_{rr}^{(r)}| = \max_{r+1 \leq i \leq n} \{|a_{ir}^{(r)}|\}$, 若 $a_{rr}^{(r)} = 0$, 则第 r 步已完成, 否则转下一步.

② 交换第 $r+1$ 行和第 r' 行, 并交换第 $r+1$ 列和第 r' 列.

③ 对 $i=r+2, r+3, \dots, n$, 计算乘子 $n_{i,r+1} = a_{ir}/a_{r+1,r}$, 从第 i 行减去 $n_{i,r+1}$ 乘第 $r+1$ 行, 并由第 i 列乘以 $n_{i,r+1}$ 加到第 $r+1$ 列.

共进行 $n-2$ 步.

3. 使用说明

(1) BALANC (A, N)

N 整型变量, 输入参数, 矩阵阶数

A $N \times N$ 个元素的二维实型数组, 输入、输出参数, 输入需要平衡的 N 阶矩阵, 输出时存放平衡后的矩阵

(2) ELMHES (A,N)

N 整型变量,输入参数,矩阵 A 的阶数

A $N \times N$ 个元素的二维实型数组,输入、输出参数,输入时存放所要变形的矩阵,可为平衡后的矩阵;输出时存放上 H 型矩阵,而对 $I > j+1$ 的元素认为是零,但输出值是随机的

4. 过程

(1) 子过程 BALANC.

```

SUBROUTINE balanc(a,n)
INTEGER n
REAL a(n,n),RADIX,SQRDX
PARAMETER (RADIX=2.,SQRDX=RADIX * 2)
INTEGER i,j,last
REAL c,f,g,r,s
do
  last=1
  do i=1,n
    c=0.
    r=0.
    do j=1,n
      if(j/=i) then
        c=c+abs(a(j,i))
        r=r+abs(a(i,j))
      endif
    enddo
    end do
    if(c/=0. . and. r/=0. ) then
      g=r/RADIX
      f=1.
      s=c+r
      do while(c<g)
        f=f * RADIX
        c=c * SQRDX
      end do
      g=r * RADIX
      do while(c>g)
        f=f/RADIX

```

```

      c=c/SQRDX
    end do
    if((c+r)/f<0.95*s) then
      last=0
      g=1./f
      do j=1,n
        a(i,j)=a(i,j)*g
      end do
      do j=1,n
        a(j,i)=a(j,i)*f
      end do
    endif
  endif
end do
if(last/=0) exit
end do
END SUBROUTINE balanc

```

(2) 子过程 ELMHES.

```

SUBROUTINE elmhes(a,n)
INTEGER n
REAL a(n,n)
INTEGER i,j,m
REAL x,y
do m=2,n-1
  x=0.
  i=m
  do j=m,n
    if(abs(a(j,m-1))>abs(x)) then
      x=a(j,m-1)
      i=j
    endif
  end do
  if(i/=m) then
    do j=m-1,n
      y=a(i,j)
      a(i,j)=a(m,j)

```



```

        a(m,j)=-y
    end do
    do j=1,n
        y=a(j,i)
        a(j,i)=a(j,m)
        a(j,m)=y
    end do
endif
if(x/=0.) then
    do i=m+1,n
        y=a(i,m-1)
        if(y/=0.) then
            y=y/x
            a(i,m-1)=y
            do j=m,n
                a(i,j)=a(i,j)-y*a(m,j)
            end do
            do j=1,n
                a(j,m)=a(j,m)+y*a(j,i)
            end do
        endif
    end do
endif
end do
END SUBROUTINE elmhes

```

5. 例子

(1) BALANC 的作用是在计算非对称矩阵的特征值时减小误差. 为了做到这一点, 该子过程调整矩阵的相应行和列的范数, 使它们是可比较的, 但不改变矩阵的特征值. 验证程序 D8R5 调用子过程 BALANC 对下面的矩阵进行调整:

$$\begin{bmatrix} 1 & 100 & 1 & 100 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 100 & 1 & 100 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 100 & 1 & 100 & 1 \end{bmatrix}$$

计算结果中先打印出 5 行和 5 列的范数, 可以很清楚地看到 5 行中有三个范数、

5 列中有两个范数比其他的大得多. 经过 BALANC 的平衡后, 它们的范数重新被计算, 这时可以看到无论是行还是列, 它们的范数都比较接近. 验证程序 D8R5 如下:

```

PROGRAM D8R5
  ! Driver for routine BALANC
  PARAMETER(N=5)
  DIMENSION A(N,N),R(N),C(N)
  DATA A/1.0,1.0,1.0,1.0,1.0,100.0,1.0,100.0,1.0,100.0,&
    1.0,1.0,1.0,1.0,1.0,100.0,1.0,100.0,1.0,100.0,&
    1.0,1.0,1.0,1.0,1.0/
  ! Print norms
  DO I=1,N
    R(I)=0.0
    C(I)=0.0
    DO J=1,N
      R(I)=R(I)+ABS(A(I,J))
      C(I)=C(I)+ABS(A(J,I))
    END DO
  END DO
  WRITE(*,*) 'Rows:'
  WRITE(*,*) (R(I),I=1,N)
  WRITE(*,*) 'Columns:'
  WRITE(*,*) (C(I),I=1,N)
  WRITE(*, '(1X,A/)' ) ' * * * * * Balancing Matrix * * * * * '
  CALL BALANC(A,N)
  ! Print norms
  DO I=1,N
    R(I)=0.0
    C(I)=0.0
    DO J=1,N
      R(I)=R(I)+ABS(A(I,J))
      C(I)=C(I)+ABS(A(J,I))
    END DO
  END DO
  WRITE(*,*) 'Rows:'
  WRITE(*,*) (R(I), I=1,N)

```

```
WRITE(*,*) 'Columns;'
WRITE(*,*) (C(I), I=1,N)
END
```

计算结果如下:

```
Rows:
  203.0000    5.000000    203.0000    5.000000    203.0000
Columns:
  5.000000    302.0000    5.000000    302.0000    5.000000
```

* * * * * Balancing Matrix * * * * *

```
Rows:
  28.00000    26.00000    28.00000    26.00000    28.00000
Columns:
  19.00000    39.50000    19.00000    39.50000    19.00000
```

(2) 验证程序 D8R6 利用 BALANC 和 ELMHES 将一非对称且没有平衡过的矩阵化为赫申伯格矩阵. 矩阵 A 为

$$\begin{bmatrix} 1 & 2 & 300 & 4 & 5 \\ 2 & 3 & 400 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 600 & 7 & 8 \\ 5 & 6 & 700 & 8 & 9 \end{bmatrix}$$

验证程序 D8R6 在打印出原矩阵 A 后, 将其输入子过程 BALANC, 随之打印出被平衡后的矩阵. 最后利用 ELMHES, 并打印出结果 A . 注意: A 矩阵中的元素当 $I > J+1$ 时都被设定为零. 因为 ELMHES 在矩阵 A 这部分中返回的是随机值, 因而不能认为子过程 ELMHES 即能打印出赫申伯格形式的矩阵. 最重要的是记录中的非零元素. 验证程序 D8R6 如下:

```
PROGRAM D8R6
! Driver for routine ELMHES
PARAMETER(N=5)
! USES BALANC
DIMENSION A(N,N), R(N), C(N)
DATA A/1.0,2.0,3.0,4.0,5.0,2.0,3.0,4.0,5.0,6.0,&
```

```

      300.0,400.0,5.0,600.0,700.0,4.0,5.0,6.0,7.0,8.0,&
      5.0,6.0,7.0,8.0,9.0/
WRITE(*,'(/1X,A/)' )' * * * * * Original Matrix * * x x x '
DO I=1,N
    WRITE(*,'(1X,5F12.2)') (A(I,J),J=1,N)
END DO
WRITE(*,'(/1X,A/)' )' * * * * * Balance Matrix * * * * * '
CALL BALANC(A,N)
DO I=1,N
    WRITE(*,'(1X,5F12.2)') (A(I,J),J=1,N)
END DO
WRITE(*,'(/1X,A/)' )&
    ' * * * * * Reduce to Hessenberg Form * * * * * '
CALL ELMHES(A,N)
DO J=1,N-2
    DO I=J+2,N
        A(I,J)=0.0
    END DO
END DO
DO I=1,N
    WRITE(*,'(1X,5E12.4)') (A(I,J),J=1,N)
END DO
END

```

计算结果如下：

* * * * * Original Matrix * * * * *

1.00	2.00	300.00	4.00	5.00
2.00	3.00	400.00	5.00	6.00
3.00	4.00	5.00	6.00	7.00
4.00	5.00	600.00	7.00	8.00
5.00	6.00	700.00	8.00	9.00

* * * * * Balance Matrix * * * * *

```

1.00  2.00  37.50  4.00  5.00
2.00  3.00  50.00  5.00  6.00
24.00 32.00   5.00 48.00 56.00
4.00  5.00  75.00  7.00  8.00
5.00  6.00  87.50  8.00  9.00

```

```

* * * * * Reduce to Hessenberg Form * * * * *

```

```

0.1000E+01  0.3938E+02  0.9618E+01  0.3333E+01  0.4000E+01
0.2400E+02  0.2733E+02  0.1161E+03  0.4800E+02  0.4800E+02
0.0000E+00  0.8551E+02 -0.4780E+01 -0.1333E+01 -0.2000E+01
0.0000E+00  0.0000E+00  0.5188E+01  0.1447E+01  0.2171E+01
0.0000E+00  0.0000E+00  0.0000E+00 -0.9155E-07  0.7874E-07

```

8.5 实赫申伯格矩阵的 QR 算法

1. 功能

用于计算上赫申伯格矩阵 A 的全部特征值. 采用收敛较快的带原点位移的双重步 QR 算法.

程序中若对矩阵的某个特征值迭代 30 次仍不能求得, 则报告失败.

2. 方法

(1) 从 $A_1 \equiv A$ 开始,

$$A_s - k_s I = Q_s^T R_s$$

$$A_{s+1} = R_s Q_s^T + k_s I$$

$$A_{s+1} - k_{s+1} I = Q_{s+1}^T R_{s+1}$$

$$A_{s+2} = Q_{s+1} Q_s A_s Q_s^T Q_{s+1}^T$$

其中 k_s, k_{s+1} 是原点位移.

定义

$$M = (A_s - k_{s+1} I)(A_s - k_s I)$$

则

$$R = R_{s+1} R_s = Q_{s+1} Q_s M = Q M$$

$$A_s Q^T = Q^T A_{s+2}$$

由 QR 分解的唯一性, 若有

$$A_s \bar{Q}^T = \bar{Q}^T H$$

使 \bar{Q}^T 正交且与 Q^T 的第一列相同, 而 H 为上 H -型矩阵, 则 $\bar{Q} = Q, A_{s+2} = H$.

(2) Q^T 的第一列即为 Q 的第一行, 由 Q 三对角化 M, Q 正交, 则 $Q = P_{n-1} \cdots P_2 P_1$, 其中 P_i 为左上角为 $(i-1) \times (i-1)$ 阶单位阵的 Householder 矩阵 ($i = 1, \cdots, n-1$). 因此 Q 的第一行即是 P_1 的第一行.

因 P_1 由 M 的第一列所确定, 且 A_s 为上 H -型矩阵, 所以 M 的第一列为 $[p_1, q_1, r_1, 0, \cdots, 0]^T$, 其中

$$p_1 = a_{11}^2 - a_{11}(k_s + k_{s+1}) + k_s k_{s+1} + a_{12} a_{21}$$

$$q_1 = a_{21}(a_{11} + a_{22} - k_s - k_{s+1})$$

$$r_1 = a_{21} a_{32}$$

由此 $P_1 = I - 2W_1 W_1^T, W_1 = [\times, \times, \times, 0, \cdots, 0]^T$. 矩阵 $P_1 A_s P_1^T$ 具有形式

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \otimes & \times & \times & \times & \times & \times & \times \\ \otimes & \otimes & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{bmatrix}$$

依次作 Householder 变换:

$$\bar{P}_i = I - 2W_i W_i^T \quad (i = 2, \cdots, n-1)$$

其中 W_i 仅有三个非零分量: 第 i 个, 第 $i+1$ 个, 第 $i+2$ 个分量, 使

$$H = \bar{P}_{n-1} \cdots \bar{P}_2 P_1 A_s P_1^T \bar{P}_2^T \cdots \bar{P}_{n-1}^T$$

为上 H -型矩阵, 则

$$\bar{Q} = Q = P_{n-1} P_2 P_1, A_{s+2} = H$$

注意, \bar{P}_i 作用在第 $i-1$ 列 ($i = 2, \cdots, n-1$).

(3) 原点位移取为 A_s 的右下角的 2×2 子矩阵的特征值, 即

$$k_s + k_{s+1} = a_{n-1, n-1} + a_{nn}$$

$$k_s k_{s+1} = a_{n-1, n-1} a_{nn} - a_{n-1, n} a_{n, n-1}$$

由此

$$p_1 = a_{21} \{ [(a_{nn} - a_{11})(a_{n-1, n-1} - a_{11}) - a_{n-1, n} a_{n, n-1}] / a_{21} + a_{12} \}$$

$$q_1 = a_{21} [a_{22} - a_{11} - (a_{nn} - a_{11}) - (a_{n-1, n-1} - a_{11})]$$

$$r_1 = a_{21} a_{32}$$

(4) 在 $P_1, \bar{P}_i (i = 2, \cdots, n-1)$ 中, $2W_i W_i^T$ 中的非零部分具有形式

$$\begin{bmatrix} (p \pm s)/(\pm s) \\ q/(\pm s) \\ r/(\pm s) \end{bmatrix} [1 \quad q/(p \pm s) \quad r/(p \pm s)]$$

其中, $s^2 = p^2 + q^2 + r^2$.

(5) 在每次迭代中,先考虑矩阵是否可降低以对较低阶的子矩阵迭代.

求最大的 i , 使 $a_{i,i-1}$ 可忽略 (若没有这样的 i 则取 $i=1$).

① 若 $i=n$, 则求得一个特征值.

② 若 $i=n-1$, 则求得两个特征值.

③ 否则, 对第 i 行到第 n 行的子矩阵迭代, 这时还考虑当连续的两个次对角元之积可忽略时, 也可把矩阵分为低阶矩阵来做变换. 其判别准则是依次对 $m=n-2, n-3, \dots, i+1$, 考察

$$|a_{m,m-1}|(|q| + |r|) \ll |p|(|a_{m-1,m+1}| + |a_{mm}| + |a_{m-1,m-1}|)$$

其中

$$p = a_{m+1,m} \{ [(a_{nn} - a_{mm})(a_{n-1,n-1} - a_{mm}) - a_{n-1,n}a_{n,n-1}] / a_{m+1,m} + a_{m,m+1} \}$$

$$q = a_{m+1,m} [a_{m+1,m+1} - a_{mm} - (a_{nn} - a_{mm}) - (a_{n-1,n-1} - a_{mm})]$$

$$r = a_{m+1,m}a_{m+2,m+1}$$

(6) 通常在迭代 10 次后还未确定出一个特征值时, 改变位移量为

$$k_s + k_{s+1} = 1.5(|a_{n,n-1}| + |a_{n-1,n-2}|)$$

$$k_s k_{s+1} = (|a_{n,n-1}| + |a_{n-1,n-2}|)^2$$

3. 使用说明

HQR (A, N, WR, WI)

N 整型变量, 输入参数, 矩阵阶数

A $N \times N$ 个元素的二维实型数组, 输入参数, 存放实的上赫申伯格矩阵 A

WR N 个元素的一维实型数组, 输出参数, 存放 N 个特征值的实部

WI N 个元素的一维实型数组, 输出参数, 存放 N 个特征值的虚部

4. 过程

子过程 HQR.

SUBROUTINE hqr(a,n,wr,wi)

INTEGER n

REAL a(n,n),wi(n),wr(n)

```

INTEGER i,its,j,k,l,m,nn
REAL anorm,p,q,r,s,t,u,v,w,x,y,z
LOGICAL done
anorm=0.
do i=1,n
    do j=max(i-1,1),n
        anorm=anorm+abs(a(i,j))
    end do
end do
nn=n
t=0.
do while(nn>=1)
    its=0
    do
        done=0
        do l=nn,2,-1
            s=abs(a(l-1,l-1))+abs(a(l,l))
            if(s==0.) s=anorm
            if(abs(a(l,l-1))+s==s) exit
        end do
        if(abs(a(l,l-1))+s/=s) l=1
        x=a(nn,nn)
        if(l==nn) then
            wr(nn)=x+t
            w1(nn)=0.
            nn=nn-1
        else
            y=a(nn-1,nn-1)
            w=a(nn,nn-1)*a(nn-1,nn)
            if(l==nn-1) then
                p=0.5*(y-x)
                q=p**2+w
                z=sqrt(abs(q))
                x=x+t
                if(q>=0.) then
                    z=p+sign(z,p)
                    wr(nn)=x+z
                end if
            end if
        end if
    end do
end do

```



```

    wr(nn-1)=wr(nn)
    if(z/=0.) wr(nn)=x-w/z
    wi(nn)=0.
    wi(nn-1)=0.
else
    wr(nn)=x+p
    wr(nn-1)=wr(nn)
    wi(nn)=z
    wi(nn-1)=-z
endif
nn=nn-2
else
    if(its==30) pause 'too many iterations in hqr'
    if(its==10.or.its==20) then
        t=t+x
        do i=1,nn
            a(i,i)=a(i,i)-x
        end do
        s=abs(a(nn,nn-1))+abs(a(nn-1,nn-2))
        x=0.75*s
        y=x
        w=-0.4375*s*s*2
    endif
    its=its+1
    do m=nn-2,1,-1
        z=a(m,m)
        r=x-z
        s=y-z
        p=(r*s-w)/a(m+1,m)+a(m,m+1)
        q=a(m+1,m+1)-z-r-s
        r=a(m+2,m+1)
        s=abs(p)+abs(q)+abs(r)
        p=p/s
        q=q/s
        r=r/s
        if(m==1) exit
        u=abs(a(m,m-1))*(abs(q)+abs(r))
    end do

```

```

      v=abs(p)*(abs(a(m-1,m-1))+abs(z)+abs(a(m+1,m+1)))
      if(u+v==v) exit
end do
do i=m+2,nn
  a(i,i-2)=0.
  if (i/=m+2) a(i,i-3)=0.
end do
do k=m,nn-1
  if(k/=m) then
    p=a(k,k-1)
    q=a(k+1,k-1)
    r=0.
    if(k/=nn-1) r=a(k+2,k-1)
    x=abs(p)+abs(q)+abs(r)
    if(x/=0.) then
      p=p/x
      q=q/x
      r=r/x
    endif
  endif
  s=sign(sqrt(p**2+q**2+r**2),p)
  if(s/=0.) then
    if(k==m) then
      if(l/=m)a(k,k-1)=-a(k,k-1)
    else
      a(k,k-1)=-s*x
    endif
    p=p+s
    x=p/s
    y=q/s
    z=r/s
    q=q/p
    r=r/p
  do j=k,nn
    p=a(k,j)+q*a(k+1,j)
    if(k/=nn-1) then
      p=p+r*a(k+2,j)

```

```

        a(k+2,j)=a(k+2,j)-p*z
    endif
    a(k+1,j)=a(k+1,j)-p*y
    a(k,j)=a(k,j)-p*x
end do
do i=1,min(nn,k+3)
    p=x*a(i,k)+y*a(i,k+1)
    if(k/=nn-1) then
        p=p+z*a(i,k+2)
        a(i,k+2)=a(i,k+2)-p*r
    endif
    a(i,k+1)=a(i,k+1)-p*q
    a(i,k)=a(i,k)-p
end do
endif
end do
done=-1
endif
endif
if (.not. done) exit
end do
end do
END SUBROUTINE hqr

```

5. 例子

验证程序 D8R7 中给定了一个 5×5 矩阵, 是用于验证 HQR 的一个一般非对称矩阵. 这个矩阵首先被子过程 BALANC(见 8.4 节)调用进行平衡, 然后利用子过程 ELMHES(见 8.4 节)产生赫申伯格矩阵, 最后调用子过程 HQR 确定出特征值. 特征值可以是复数, 实部和虚部分别给出. 原矩阵中策略地布置了一些元素为零, 以便容易地手算出特征值, 读者可以进行验证. 验证程序 D8R7 如下:

```

PROGRAM D8R7
! Driver for routine HQR
PARAMETER(N=5)
! USES BALANC,ELMHES
DIMENSION A(N,N),WR(N),WI(N)

```

```

DATA A/1.0,-2.0,3.0,-4.0,-5.0,2.0,3.0,4.0,5.0,6.0,&
      0.0,0.0,50.0,-60.0,-70.0,0.0,0.0,0.0,7.0,8.0,&
      0.0,0.0,0.0,0.0,-9.0/
WRITE(*,'(/1X,A/)') 'Matrix:'
DO I=1,N
  WRITE(*,'(1X,5F12.2)') (A(I,J),J=1,N)
END DO
CALL BALANC(A,N)
CALL ELMHES(A,N)
CALL HQR(A,N,WR,WI)
WRITE(*,'(/1X,A/)') 'Eigenvalues:'
WRITE(*,'(/1X,T9,A,T24,A/)') 'Real','Imag.'
DO I=1,N
  WRITE(*,'(1X,2E15.6)') WR(I),WI(I)
END DO
END

```

计算结果如下:

Matrix:

1.00	2.00	0.00	0.00	0.00
-2.00	3.00	0.00	0.00	0.00
3.00	4.00	50.00	0.00	0.00
-4.00	5.00	-60.00	7.00	0.00
-5.00	6.00	-70.00	8.00	-9.00

Eigenvalues:

Real	Imag.
0.500000E+02	0.000000E+00
0.200000E+01	-0.173205E+01
0.200000E+01	0.173205E+01
0.700000E+01	0.000000E+00
-0.900000E+01	0.000000E+00

第9章 数据拟合

本章包括一般的线性最小二乘法、非线性最小二乘法和绝对值偏差最小的直线拟合。

最小二乘法是曲线拟合中普遍使用的有效方法,可以通过最小二乘法用各种不同形式的曲线去拟合得到的数据。

在线性最小二乘法(见 9.2 节)中,我们用两种方法求解:一种是用全主元高斯-约当消去法(见 1.1 节)求解线性最小二乘问题的正规方程;一种是用矩阵的奇异值分解(SVD)来求解线性最小二乘问题。对于解线性最小二乘问题的正规方程,也可用 LU 分解法(见 1.2 节),不过因这里还要求矩阵的逆,这时高斯-约当消去法的工作量不会比 LU 分解法大,且高斯-约当消去法要方便一些。如果不要协方差矩阵,即不要求矩阵的逆,那么用 LU 分解法计算量要小得多。如果正规方程对舍入误差比较敏感,则应该用矩阵的正交三角分解法(QR 分解法,见 1.11 节)。实际上 9.1 节中的直线拟合所用的方法就是 QR 分解法,而奇异值分解法则是最小二乘问题中的一个通用方法,它能克服别的方法难以克服的问题。

关于非线性最小二乘法,列维布格-麦奎尔特方法实用效果很好,现已成为求解非线性最小二乘问题的标准方法。它实际上是牛顿法的一种改进,因牛顿法要求提供较好的初值,但对于具体问题,选取一个较好的初值往往是困难的,列维布格-麦奎尔特方法则放宽了对初值的限制。

类似于插值,对数据进行分段拟合也是一种常用的方法,可参考第 2 章。

9.1 直线拟合

1. 功能

设随机变量 y 与自变量 x 之间具有函数关系 $y=y(x)=a+bx$, $(x_1, y_1), \dots, (x_n, y_n)$ 是 (x, y) 的 n 个观测值,对每个 $i(1 \leq i \leq n)$, y_i 与真实的 $y(x_i)$ 之间的误差,服从期望值为零、标准差为 σ_i 的正态分布,求 a, b 使

$$\chi^2(a, b) = \sum_{i=1}^n [(y_i - a - bx_i)/\sigma_i]^2$$

达到最小.

2. 方法

由 $\frac{\partial \chi^2(a,b)}{\partial a} = \frac{\partial \chi^2(a,b)}{\partial b} = 0$ 得

$$a = (s_{xx}s_y - s_x s_{xy}) / \Delta$$

$$b = (s s_{xy} - s_x s_y) / \Delta$$

其中

$$s = \sum_{i=1}^n 1/\sigma_i^2, \quad s_x = \sum_{i=1}^n x_i/\sigma_i^2$$

$$s_y = \sum_{i=1}^n y_i/\sigma_i^2, \quad s_{xx} = \sum_{i=1}^n x_i^2/\sigma_i^2$$

$$s_{xy} = \sum_{i=1}^n x_i y_i/\sigma_i^2, \quad \Delta = s s_{xx} - (s_x)^2$$

由此 a 与 b 的方差分别是

$$\sigma_a^2 = s_{xx}/\Delta, \quad \sigma_b^2 = s/\Delta$$

其协方差与相关系数是

$$\text{COV}(a,b) = -s_x/\Delta, \quad r_{ab} = -s_x/\sqrt{s s_{xx}}$$

上述式子对于舍入误差比较敏感, 因此改写如下:

记

$$t_i = (x_i - s_x/s)\sigma_i, \quad i = 1, 2, \dots, n$$

$$s_u = \sum_{i=1}^n t_i^2$$

则

$$b = \frac{1}{s_u} \sum_{i=1}^n (t_i y_i / \sigma_i), \quad a = (s_y - s_x b) / s$$

$$\sigma_a^2 = [1 + s_x^2/(s s_u)] / s, \quad \sigma_b^2 = 1/s_u$$

$$\text{COV}(a,b) = -s_x/(s s_u), \quad r_{ab} = \text{COV}(a,b)/(\sigma_a \sigma_b)$$

由假设 $\chi^2(a,b)$ 服从 χ^2 分布, 自由度 $\nu = n - 2$, 因此, 拟合优度概率是

$$Q(\chi^2 | \nu) = Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

其中 $Q(a, x)$ 是不完全 Γ 函数. 一般地, 如果这个概率大于 0.1, 则拟合优度是可信的; 如果它大于 0.001, 则当误差是非正态的或者误差被适度低估时拟合是可接受的; 如果它小于 0.001, 则模型和(或)估计过程不正确.

3. 使用说明

`FIT(X,Y,NDATA,SIG,MWT,A,B,SIGA,SIGB,CHI2,Q)`

`NDATA` 整型变量,输入参数,观测数据的个数

`X` `NDATA` 个元素的一维实型数组,输入参数,存放自变量 x 的 n 个观测值

`Y` `NDATA` 个元素的一维实型数组,输入参数,存放 y 的 n 个观测值

`SIG` `NDATA` 个元素的一维实型数组,输入参数,存放测量误差的标准差

`MWT` 整型变量,输入参数,输入零与非零,若 `MWT=0`,则 `SIG` 中元素无效,程序中 `SIG` 中的元素全部取为 1,且拟合优度概率 Q 赋值为 1

`A` 实型变量,输出参数,存放系数 a

`B` 实型变量,输出参数,存放系数 b

`SIGA` 实型变量,输出参数, a 的标准差

`SIGB` 实型变量,输出参数, b 的标准差

`CHI2` 实型变量,输出参数, $\chi^2(a,b)$ 的值

`Q` 实型变量,输出参数,拟合优度概率,当 `MWT=0` 时,程序赋 Q 为 1

4. 过程

子过程 `FIT`.

```
SUBROUTINE fit(x,y,ndata,sig,mwt,a,b,siga,sigb,chi2,q)
```

```
INTEGER mwt,ndata
```

```
REAL a,b,chi2,q,siga,sigb,sig(ndata),x(ndata),y(ndata)
```

```
! USES gammq
```

```
INTEGER i
```

```
REAL sigdat,ss,st2,sx,sxoss,sy,t,wt,gammq
```

```
sx=0.
```

```
sy=0.
```

```
st2=0.
```

```
b=0.
```

```
if(mwt/=0) then
```

```
ss=0.
```

```

do i=1,ndata
    wt=1./ (sig(i) * * 2)
    ss=ss+wt
    sx=sx+x(i) * wt
    sy=sy+y(i) * wt
end do
else
do i=1,ndata
    sx=sx+x(i)
    sy=sy+y(i)
end do
ss=float(ndata)
endif
sxoss=sx/ss
if(mwt/=0) then
do i=1,ndata
    t=(x(i)-sxoss)/sig(i)
    st2=st2+t*t
    b=b+t*y(i)/sig(i)
end do
else
do i=1,ndata
    t=x(i)-sxoss
    st2=st2+t*t
    b=b+t*y(i)
end do
endif
b=b/st2
a=(sy-sx*b)/ss
siga=sqrt((1.+sx*sx/(ss*st2))/ss)
sigb=sqrt(1./st2)
chi2=0.
if(mwt==0) then
do i=1,ndata
    chi2=chi2+(y(i)-a-b*x(i))* * 2
end do
q=}.

```



```

sigdat=sqrt(chi2/(ndata-2))
siga=siga * sigdat
sigb=sigb * sigdat
else
do i=1,ndata
  chi2=chi2+((y(i)-a-b*x(i))/sig(i))* * 2
end do
q=gammq(0.5*(ndata-2),0.5*chi2)
endif
END SUBROUTINE fit

```

5. 例子

子过程 FIT 把具有标准差 $SIG(I)$ 的一组 NPT 个数据点 $(X(I), Y(I))$ 拟合到线性模型 $y = Ax + B$. 它采用 χ^2 作为拟合优度. 为了验证 FIT, 我们在验证程序 D9R1 中取了一些噪声数据. 我们设 $x = 0.1I$, y 为 $-2x + 1$ 再加上一些取自于表示噪声的正态分布. 随后, 我们两次调用了 FIT, 第一次执行了没有允许标准差 $SIG(I)$ 的拟合; 第二次执行了带有允许标准差的拟合. 由于 $SIG(I)$ 已经设定为常数值 SPREAD, 所以它不影响结果参数值. 验证程序 D9R1 如下:

```

PROGRAM D9R1
! Driver for routine FIT
PARAMETER(NPT=100,SPREAD=0.5)
! USES GASDEV
DIMENSION X(NPT),Y(NPT),SIG(NPT)
IDUM=-117
DO I=1,NPT
  X(I)=0.1*I
  Y(I)=-2.0*X(I)+1.0+SPREAD*GASDEV(IDUM)
  SIG(I)=SPREAD
END DO
DO MWT=0,1
  CALL FIT(X,Y,NPT,SIG,MWT,A,B,SIGA,SIGB,CHI2,Q)
  IF (MWT==0) THEN
    WRITE(*,'(//1X,A)')&
      'Ignoring standard deviation'
  ELSE
    WRITE(*,'(//1X,A)')&

```

```

                                'Including standard deviation'
ENDIF
WRITE(*,'(1X,T5,A,F9.6,T24,A,F9.6)') 'A = ',A,&
    'Uncertainty: ',SIGA
WRITE(*,'(1X,T5,A,F9.6,T24,A,F9.6)') 'B = ',B,&
    'Uncertainty: ',SIGB
WRITE(*,'(1X,T5,A,4X,F10.6)')&
    'Chi-squared: ',CHI2
WRITE(*,'(1X,T5,A,F10.6)') 'Goodness-of-fit: ',Q
END DO
END

```

在子过程 FIT 中需要调用 GAMMQ(见 4.2 节). 而 GAMMQ 还需调用 GSER,GCF,GAMMLN(见第 4 章). 在 D9R1 中需调用 GASDEV(见 6.2 节). 计算结果如下:

Ignoring standard deviation

```

A =  1.079574      Uncertainty:  0.099821
B = -2.006663      Uncertainty:  0.017161
Chi-squared;      24.047949
Goodness-of-fit:  1.000000

```

Including standard deviation

```

A =  1.079574      Uncertainty:  0.100755
B = -2.006663      Uncertainty:  0.017321
Chi-squared;      96.191795
Goodness-of-fit;   0.532772

```

9.2 线性最小二乘法

1. 功能

若 y 与自变量 x 及参数 a_1, \dots, a_m 之间有函数关系

$$y = y(x) = \sum_{k=1}^m a_k \varphi_k(x)$$

其中 $\varphi_k(x)$ ($k=1, 2, \dots, m$) 是已知函数, 称为基函数; (x_i, y_i) ($i=1, 2, \dots, n$) 是 (x, y) 的 n 对观测值; L 是 $M=\{1, 2, \dots, m\}$ 的一个子集. 假定参数 a_k ($k \in (M-L)$) 为已知, 求参数 a_k ; $k \in L$, 使

$$\chi^2 = \sum_{i=1}^n \left[(y_i - \sum_{k=1}^m a_k \varphi_k(x_i)) / \sigma_i \right]^2$$

达到最小,其中 σ_i 为第 i 个点 (x_i, y_i) ($i=1, 2, \dots, n$) 上测量误差的标准差.

子过程 LFIT 通过求解线性最小二乘问题的正规方程估计参数值,子过程 COVSRT 对接 L 得到的参数的协方差矩阵中的元素进行重排序,使其元素按自然次序排列.

子过程 SVDFIT 应用奇异值分解求解线性最小二乘问题,子过程 SVDVAR 用于计算由子过程 SVDFIT 得到的参数的协方差矩阵.

子过程 FPOLY 与子过程 FLEG 为子过程 LFIT 和 SVDFIT 中子过程参数 FUNCS 的两个例子,前者函数关系 $y=y(x)$ 是普通多项式情形,后者 $y=y(x)$ 是勒让德多项式情形.

2. 方法

(1) 通过正规方程求解.

为使 χ^2 达到极小,则应有

$$\frac{\partial \chi^2}{\partial a_k} = 0, \quad k = 1, 2, \dots, m$$

即

$$[\alpha]a = [\beta] \quad (9-1)$$

其中 $[\alpha]$ 为 $m \times m$ 矩阵,其元素 $\alpha_{kj} = \sum_{i=1}^n \varphi_j(x_i) \varphi_k(x_i) / \sigma_i^2$, $[\beta]$ 为 m 维列向量, $\beta_j = \sum_{i=1}^n y_i \varphi_j(x_i) / \sigma_i^2$.

式(9-1)即为最小二乘问题的正规方程,应用全主元高斯-约当消去法求解.

记 $C = [\alpha]^{-1}$ (假定存在),则 C 即为参数 a_k ($k=1, 2, \dots, m$) 的协方差矩阵,而

$$a_j = \sum_{k=1}^m c_{jk} \beta_k$$

(2) 应用奇异值分解求解最小二乘问题.

在很多情况下,正规方程的系数矩阵很可能接近奇异,若用主元消去法求解正规方程,或者产生零主元,这时无法求解;或者主元很小,得到的解的误差很大.为克服这个问题,可采用矩阵的奇异值分解求解最小二乘问题.

记 A 为 $n \times m$ 矩阵,其元素是 $a_{ij} = \varphi_j(x_i) / \sigma_i$, $b = (b_1, \dots, b_m)^T$, 其元素是 $b_i = y_i / \sigma_i$, $a = (a_1, \dots, a_m)^T$, 则问题变为求参数 a , 使

$$\chi^2 = \|Aa - b\|^2 = \min \quad (9-2)$$

这里 $\|\cdot\|$ 表示欧氏范数.

设 A 的奇异值分解是

$$A = U \Sigma V^T \quad (9-3)$$

其中 $U = (U_1, \dots, U_m)$ 是列正交的 $n \times m$ 矩阵, $\Sigma = \text{diag}(W_1, \dots, W_m)$, V 是 $m \times m$ 正交矩阵, Σ 中的 $W_i (i=1, \dots, m)$ 即是 A 的奇异值, 则式(9-2)的解为

$$\begin{aligned} a &= V \Sigma^+ U^T b \quad (\Sigma^+ \text{ 为 } \Sigma \text{ 的广义逆}) \\ &= \sum_{i=1}^m (\langle U_i, b \rangle / W_i) V_i \end{aligned}$$

这里 $\langle \cdot, \cdot \rangle$ 表示二向量的点积, $U_i (i=1, \dots, m)$ 表示 U 的列. 注意, 若 $W_i = 0$ 或 W_i 很小, 则规定 $1/W_i = 0$.

另外有

$$\text{COV}(a_j, a_k) = \sum_{i=1}^m (V_{ij} V_{ik}) / W_i^2$$

其中 V_{ij} 是正交矩阵 V 的元素.

3. 使用说明

(1) LFIT (X, Y, SIG, NDATA, A, MA, LISTA, MFIT, COVAR, NCVM, CHISQ, FUNCS)

NDATA	整型变量, 输入参数, 数据点的个数 n
MA	整型变量, 输入参数, 参数的总个数 m
MFIT	整型变量, 输入参数, 欲求参数的个数
NCVM	整型变量, 输入参数, 存储协方差矩阵 COVAR 的物理维数
X	NDATA 个元素的一维实型数组, 输入参数, 存放自变量 x 的 n 个观测值
Y	NDATA 个元素的一维实型数组, 输入参数, 存放变量 y 的 n 个观测值
SIG	NDATA 个元素的一维实型数组, 输入参数, 存放 n 个数据点上的测量误差的标准差 $\sigma_1, \dots, \sigma_n$, 若未知, 则全部输入为 1
LISTA	MA 个元素的一维整型数组, 输入参数, 用于对参数重编号, 使其前面的 MFIT 个元素相应于要确定的参数, 其余 $MA - \text{MFIT}$ 个元素相应于已知参数
A	MA 个元素的一维实型数组, 其中某些元素为输入参数, 某些元素为输出所求参数的结果值. 数组 LISTA 中前 MFIT 个元

素值表示了 A 中输出的那些元素的下标值, LISTA 中后 $MA - MFIT$ 个元素值表示了 A 中输入的那些元素的下标值

COVAR $MA \times MA$ 个元素的二维实型数组, 输出参数, 存放参数的协方差矩阵

CHISQ 实型变量, 输出参数, 存放极小化函数 χ^2 的值

FUNCS 子过程名, 形式为 FUNCS($X, AFUNC, MA$), 用于计算 MA 个基函数在 $x=X$ 处的值, 存放在数组 $AFUNC$ 中, 用户自编, 两个例子为子过程 FPOLY, FLEG

(2) COVSRT(COVAR, NCV, MA, LISTA, MFIT)

COVAR $MA \times MA$ 个元素的二维实型数组, 输入、输出参数, 输入时存放已有的协方差矩阵, 输出时存放重排后的协方差矩阵

NCV, MA, LISTA, MFIT 均同子过程 LFIT

(3) SVDFIT($X, Y, SIG, NDATA, A, MA, U, V, W, MP, NP, CHISQ, FUNCS$)

MP, NP 整型变量, 输入参数, 存储矩阵 U, V, W 的物理维数, $MP \geq NDATA, NP \geq MA$

U $NDATA \times MA$ 个元素的二维实型数组, 输出参数, 存放矩阵的奇异值分解的 U , 其存储的物理维数是 $MP \times NP$

V $MA \times MA$ 个元素的二维实型数组, 输出参数, 存放矩阵的奇异值分解的 V , 其存储的物理维数是 $NP \times NP$

W MA 个元素的一维实型数组, 输出参数, 存放矩阵的奇异值, 其存储的物理维数是 NP

FUNCS 字符变量, 存放函数名

$X, Y, SIG, NDATA, A, MA, CHISQ$ 均同子过程 LFIT

(4) SVDVAR(V, MA, NP, W, CVM, NCV)

MA, NP 同子过程 SVDFIT

NCV 整型变量, 输入参数, 存储协方差矩阵 CVM 的物理维数

V $MA \times MA$ 个元素的二维实型数组, 输入参数, 输入由子过程 SVDFIT 输出的 V

W MA 个元素的一维实型数组, 输入参数, 输入由子过程 SVD-FIT 输出的 W

CVM $MA \times MA$ 个元素的二维实型数组, 输出参数, 存放 MA 个参数的协方差矩阵

(5) FPOLY(X,P,NP)

NP 整型变量,输入参数,多项式阶数加 1

X 实型变量,输入参数,自变量 x 的值

P NP 个元素的一维实型数组,输出参数,存放基函数值

(6) FLEG(X,PL,NL)

NL 整型变量,输入参数,所用基函数中的最高阶勒让德多项式的阶数加 1

X 实型变量,输入参数,自变量 x 的值

PL NL 个元素的一维实型数组,输出参数,输出基函数(勒让德多项式)的值

4. 过程

(1) 子过程 LFIT.

```
SUBROUTINE lfit(x,y,sig,ndata,a,ma,lista,mfit,covar,&
               ncvm,chisq,funcs)
```

```
PARAMETER (mmax=50)
```

```
! USES gaussj, covsrt
```

```
REAL x(ndata),y(ndata),sig(ndata),a(ma),beta(mfit),&
     covar(mfit,mfit),afunc(mmax)
```

```
INTEGER lista(ma),j,k,i,kk,ihit
```

```
REAL sum,ym,wt,sig2i
```

```
kk=mfit+1
```

```
do j=1,ma
```

```
    ihit=0
```

```
    do k=1,mfit
```

```
        if(lista(k)==j) ihit=ihit+1
```

```
    end do
```

```
    if(ihit==0) then
```

```
        lista(kk)=j
```

```
        kk=kk+1
```

```
    else if(ihit>1) then
```

```
        pause 'improper set in lista'
```

```
    endif
```

```
end do
```

```
if(kk/=(ma+1)) pause 'improper set in lista'
```

```
do j=1,mfit
```

```
    do k=1,mfit
```

```

        covar(j,k)=0.
    end do
    beta(j)=0.
end do
do i=1,ndata
    call funcn(x(i),afunc,ma)
    ym=y(i)
    if(mfit<ma) then
        do j=mfit+1,ma
            ym=ym-a(lista(j))*afunc(lista(j))
        end do
    endif
    sig2i=1./sig(i)**2
    do j=1,mfit
        wt=afunc(lista(j))*sig2i
        do k=1,j
            covar(j,k)=covar(j,k)+wt*afunc(lista(k))
        end do
        beta(j)=beta(j)+ym*wt
    end do
end do
if(mfit>1) then
    do j=2,mfit
        do k=1,j-1
            covar(k,j)=covar(j,k)
        end do
    end do
endif
call gaussj(covar,mfit,beta)
do j=1,mfit
    a(lista(j))=beta(j)
end do
chisq=0.
do i=1,ndata
    call funcn(x(i),afunc,ma)
    sum=0.
    do j=1,ma

```

```

        sum=sum+a(j)*afunc(j)
    end do
    chisq=chisq+((y(i)-sum)/sig(i))* * 2
end do
call covsrt(covar,ncvm,ma,lista,mfit)
END SUBROUTINE lfit

```

(2) 子过程 COVSRT.

```

SUBROUTINE covsrt(covar,ncvm,ma,lista,mfit)
INTEGER ma,mfit,ncvm,lista(mfit)
REAL covar(ncvm,ncvm)
INTEGER i,j,k
REAL swap
do j=1,ma-1
    do i=j+1,ma
        covar(i,j)=0.
    end do
end do
do i=1,mfit-1
    do j=i+1,mfit
        if(lista(j)>lista(i))then
            covar(lista(j),lista(i))=covar(i,j)
        else
            covar(lista(i),lista(j))=covar(i,j)
        endif
    end do
end do
swap=covar(1,1)
do j=1,ma
    covar(1,j)=covar(j,j)
    covar(j,j)=0.
end do
covar(lista(1),lista(1))=swap
do j=2,mfit
    covar(lista(j),lista(j))=covar(1,j)
end do
do j=2,ma

```



```

do i=1,j-1
    covar(i,j)=covar(j,i)
end do
end do
END SUBROUTINE covsrt

```

(3) 子过程 SVDFIT.

```

SUBROUTINE svdfit(x,y,sig,ndata,a,ma,u,v,w,mp,&
                 np,chisq,funcs)
INTEGER ma,mp,ndata,np,NMAX,MMAX
REAL chisq,a(ma),sig(ndata),u(mp,np),v(np,np),&
    w(np),x(ndata),y(ndata),TOL
EXTERNAL funcs
PARAMETER (NMAX=1000,MMAX=50,TOL=1.e-5)
! USES svbksb,svdcmp
INTEGER i,j
REAL sum,thresh,tmp,wmax,afunc(MMAX),b(NMAX)
do i=1,ndata
    call funcs(x(i),afunc,ma)
    tmp=1./sig(i)
    do j=1,ma
        u(i,j)=afunc(j)*tmp
    end do
    b(i)=y(i)*tmp
end do
call svdcmp(u,ndata,ma,w,v)
wmax=0.
do j=1,ma
    if(w(j)>wmax) wmax=w(j)
end do
thresh=TOL*wmax
do j=1,ma
    if(w(j)<thresh)w(j)=0.
end do
call svbksb(u,w,v,ndata,ma,b,a)
chisq=0.
do i=1,ndata

```

```

      call func(x(i),afunc,ma)
      sum=0.
      do j=1,ma
        sum=sum+a(j)*afunc(j)
      end do
      chisq=chisq+((y(i)-sum)/sig(i))* * 2
    end do
  END SUBROUTINE svdfit

```

(4) 子过程 SVDVAR.

```

SUBROUTINE svdvar(v,ma,np,w,cvm,ncvm)
  INTEGER ma,ncvm,np,MMAX
  REAL cvm(ncvm,ncvm),v(np,np),w(np)
  PARAMETER (MMAX=20)
  INTEGER i,j,k
  REAL sum,wti(MMAX)
  do i=1,ma
    wti(i)=0.
    if(w(i)/=0.) wti(i)=1./(w(i)*w(i))
  end do
  do i=1,ma
    do j=1,i
      sum=0.
      do k=1,ma
        sum=sum-v(i,k)*v(j,k)*wti(k)
      end do
      cvm(i,j)=sum
      cvm(j,i)=sum
    end do
  end do
END SUBROUTINE svdvar

```

(5) 子过程 FPOLY.

```

SUBROUTINE fpoly(x,p,np)
  INTEGER np
  REAL x,p(np)
  INTEGER j
  p(1)=1.

```

```

do j=2,np
  p(j)=p(j-1)*x
end do
END SUBROUTINE fpoly

```

(6) 子过程 FLEG.

```

SUBROUTINE fleg(x,pl,nl)
INTEGER nl
REAL x,pl(nl)
INTEGER j
REAL d,f1,f2,twox
pl(1)=1.
pl(2)=x
if(nl>2) then
  twox=2.*x
  f2=x
  d=1.
  do j=3,nl
    f1=d
    f2=f2+twox
    d=d+1.
    pl(j)=(f2*pl(j-1)-f1*pl(j-2))/d
  end do
endif
END SUBROUTINE fleg

```

5. 例子

(1) 子过程 LFIT 执行了与上节同类的拟合,但这里是对一个更一般的函数做最小二乘法拟合. 在验证程序 D9R2 中,我们选取的例子是由子过程 FUNCS 所执行的 x 的幂的线性求和. 为了检查结果方便,我们取 $y=1+x+x^2+\dots$. 这个级数取的项数由参数 NTERM 来决定,且还要附加上正态干扰去模拟实际数据. SIG(I)为误差常数. 为了拟合同一数据,LFIT 被调用了三次. 第一次 LISTA(I)设定为 I,因而,拟合参数应为 $A(1)\approx 1.0, A(2)\approx 2.0, A(3)\approx 3.0$. 随后,第二次数组 LISTA(I)取为第一次的反序. 这是 LISTA 特点的一种试验. 最后,拟合仅限于奇数参数,而偶数参数固定不变. 在这种情况下,具有固定参数的协方差矩阵的元素应该为零. 验证程序 D9R2 如下:

```

PROGRAM D9R2
! Driver for routine LFIT
PARAMETER(NPT=100,SPREAD=0.1,NTERM=3)
! USES GASDEV
DIMENSION X(NPT),Y(NPT),SIG(NPT),A(NTERM),LISTA(NTERM)&.
        ,COVAR(NTERM,NTERM)
EXTERNAL FUNCS
IDUM=-911
DO I=1,NPT
    X(I)=0.1*I
    Y(I)=FLOAT(NTERM)
    DO J=NTERM-1,1,-1
        Y(I)=J+Y(I)*X(I)
    END DO
    Y(I)=Y(I)+SPREAD*GASDEV(IDUM)
    SIG(I)=SPREAD
END DO
MFIT=NTERM
DO I=1,MFIT
    LISTA(I)=I
END DO
CALL LFIT(X,Y,SIG,NPT,A,NTERM,LISTA,MFIT,COVAR,NTERM,&.
        CHISQ,FUNCS)
WRITE(*, '(/1X,T4,A,T22,A)') 'Parameter','Uncertainty'
DO I=1,NTERM
    WRITE(*, '(1X,T5,A,I1,A,F8.6,F14.6)') 'A(',I,&.
        ')=',A(I),SQRT(COVAR(I,I))
END DO
WRITE(*, '(/3X,A,E12.6)') 'Chi-squared = ',CHISQ
WRITE(*, '(/3X,A)') 'Full covariance matrix'
DO I=1,NTERM
    WRITE(*, '(1X,4E12.2)') (COVAR(I,J),J=1,NTERM)
END DO
WRITE(*, '(/1X,A)') 'Press RETURN to END DO...'
READ(*,*)
! Now test the LISTA feature
DO I=1,NTERM

```

```

LISTA(I) = NTERM + 1 - I
END DO
CALL LFIT(X,Y,SIG,NPT,A,NTERM,LISTA,MFIT,COVAR,NTERM,&
          CHISQ,FUNCS)
WRITE(*, '( /1X,T4,A,T22,A)') 'Parameter', 'Uncertainty'
DO I=1,NTERM
    WRITE(*, '(1X,T5,A,I1,A,F8.6,F14.6)') 'A(', I, &
        ') = ', A(I), SQRT(COVAR(I,I))
END DO
WRITE(*, '( /3X,A,E12.6)') 'Chi-squared = ', CHISQ
WRITE(*, '( /3X,A)') 'Full covariance matrix'
DO I=1,NTERM
    WRITE(*, '(1X,4E12.2)') (COVAR(I,J), J=1,NTERM)
END DO
WRITE(*, '( /1X,A)') 'Press RETURN to END DO. . .'
READ(*, *)
1 Now check results of restricting fit parameters
II=1
DO I=1,NTERM
    IF(MOD(I,2) == 1) THEN
        LISTA(II) = I
        II = II + 1
    ENDIF
END DO
MFIT = II - 1
CALL LFIT(X,Y,SIG,NPT,A,NTERM,LISTA,MFIT,COVAR,NTERM,&
          CHISQ,FUNCS)
WRITE(*, '( /1X,T4,A,T22,A)') 'Parameter', 'Uncertainty'
DO I=1,NTERM
    WRITE(*, '(1X,T5,A,I1,A,F8.6,F14.6)') 'A(', I, &
        ') = ', A(I), SQRT(COVAR(I,I))
END DO
WRITE(*, '( /3X,A,E12.6)') 'Chi-squared = ', CHISQ
WRITE(*, '( /3X,A)') 'Full covariance matrix'
DO I=1,NTERM
    WRITE(*, '(1X,4E12.2)') (COVAR(I,J), J=1,NTERM)
END DO

```

```

END PROGRAM
SUBROUTINE FUNCS(X,AFUNC,MA)
DIMENSION AFUNC(MA)
AFUNC(1)=1.0
DO I=2,MA
    AFUNC(I)=X * AFUNC(I-1)
END DO
END SUBROUTINE FUNCS

```

子过程 LFIT 要调用子过程 COVSRT(见本节)和 GAUSSJ(见 1.1 节), D9R2 要调用 GASDEV(见 6.2 节). 计算结果如下:

Parameter	Uncertainty
A(1)=1.055148	0.030610
A(2)=1.969312	0.013990
A(3)=3.003141	0.001342
Chi-squared = 0.110349E+03	
Full covariance matrix	
0.94E-03	-0.37E-03 0.31E-04
-0.37E-03	0.20E-03 -0.18E-04
0.31E-04	-0.18E-04 0.18E-05

Press RETURN to END DO...

Parameter	Uncertainty
A(1)=1.055148	0.030610
A(2)=1.969312	0.013990
A(3)=3.003141	0.001342
Chi-squared = 0.110349E+03	
Full covariance matrix	
0.94E-03	-0.37E-03 0.31E-04
-0.37E-03	0.20E-03 -0.18E-04
0.31E-04	-0.18E-04 0.18E-05

Press RETURN to END DO...

Parameter	Uncertainty
A(1)=1.055196	0.015047
A(2)=1.969312	0.000000
A(3)=3.003140	0.013990
Chi-squared = 0.110347E+03	
Full covariance matrix	

0.23E-03	0.00E+00	0.11E-06
0.00E+00	0.00E+00	0.00E+00
0.11E-06	0.00E+00	0.20E-03

(2) 子过程 COVSRT 同子过程 LFIT 以及后面的 SVDFIT 一起被调用. 在验证程序 D9R3 中, 设定一个人造的 10×10 协方差矩阵 $\text{COVAR}(I, J)$, 这个矩阵除了左上角 5×5 个元素以外, 其余都是零. 这些非零元素是 $\text{COVAR}(I, J) = I + J - 1$. 我们进行了下面三种试验.

1) 设定 $\text{LISTA}(I) = 2I (I = 1, \dots, 5)$ 和 $\text{MFIT} = 5$, 这时延展了非零元素使得交错为零.

2) 取 $\text{LISTA}(I) = \text{MFIT} + 1 - I (I = 1, \dots, 5)$, 使得左上角的非零元素反序排列.

3) 取 $\text{LISTA}(I) = 12 - 2I (I = 1, \dots, 5)$, 我们将这些元素既延展又反序.

验证程序 D9R3 如下:

```

PROGRAM D9R3
! Driver for routine COVSRT
PARAMETER(MA=10,MFIT=5)
DIMENSION COVAR(MA,MA),LISTA(MFIT)
DO I=1,MA
  DO J=1,MA
    COVAR(I,J)=0.0
    IF (I<=5 .AND. J<=5) THEN
      COVAR(I,J)=I+J-1
    endif
  enddo
enddo
WRITE(*, '(//2X,A)') 'Original matrix'
DO I=1,MA
  WRITE(*, '(1X,10F4.1)') (COVAR(I,J), J=1,MA)
enddo
WRITE(*, *) ' Press RETURN to continue...'
READ(*, *)
! Test 1 - spread by 2
WRITE(*, '(//2X,A)') 'Test #1 - Spread by two'
DO I=1,MFIT
  LISTA(I)=2*I
enddo

```

```

CALL COVSRT(COVAR,MA,MA,LISTA,MFIT)
DO I=1,MA
    WRITE(*,'(1X,10F4.1)')(COVAR(I,J),J=1,MA)
END DO
WRITE(*,*)' Press RETURN to continue...'
READ(*,*)
! Test 2 — reverse
WRITE(*,'(/2X,A)') 'Test #2 — Reverse'
DO I=1,MA
    DO J=1,MA
        COVAR(I,J)=0.0
        IF (I<=5 .AND. J<=5) THEN
            COVAR(I,J)=I+J-1
        ENDIF
    END DO
END DO
DO I=1,MFIT
    LISTA(I)=MFIT+1-I
END DO
CALL COVSRT(COVAR,MA,MA,LISTA,MFIT)
DO I=1,MA
    WRITE(*,'(1X,10F4.1)')(COVAR(I,J),J=1,MA)
END DO
WRITE(*,*)' Press RETURN to continue...'
READ(*,*)
! Test 3 — spread and reverse
WRITE(*,'(/2X,A)') 'Test #3 — Spread and reverse'
DO I=1,MA
    DO J=1,MA
        COVAR(I,J)=0.0
        IF (I<=5 .AND. J<=5) THEN
            COVAR(I,J)=I+J-1
        ENDIF
    END DO
END DO
DO I=1,MFIT
    LISTA(I)=MA+2-2*I

```



```

END DO
CALL COVSRT(COVAR,MA,MA,LISTA,MFIT)
DO I=1,MA
    WRITE(*,'(1X,10F4.1)')(COVAR(I,J),J=1,MA)
END DO
END

```

计算结果如下:

Original matrix

1.0	2.0	3.0	4.0	5.0	0.0	0.0	0.0	0.0	0.0
2.0	3.0	4.0	5.0	6.0	0.0	0.0	0.0	0.0	0.0
3.0	4.0	5.0	6.0	7.0	0.0	0.0	0.0	0.0	0.0
4.0	5.0	6.0	7.0	8.0	0.0	0.0	0.0	0.0	0.0
5.0	6.0	7.0	8.0	9.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Press RETURN to continue...

Test #1 -- Spread by two

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	2.0	0.0	3.0	0.0	4.0	0.0	5.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	2.0	0.0	3.0	0.0	4.0	0.0	5.0	0.0	6.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	3.0	0.0	4.0	0.0	5.0	0.0	6.0	0.0	7.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	4.0	0.0	5.0	0.0	6.0	0.0	7.0	0.0	8.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	5.0	0.0	6.0	0.0	7.0	0.0	8.0	0.0	9.0

Press RETURN to continue...

Test #2 -- Reverse

9.0	8.0	7.0	6.0	5.0	0.0	0.0	0.0	0.0	0.0
8.0	7.0	6.0	5.0	4.0	0.0	0.0	0.0	0.0	0.0
7.0	6.0	5.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0
6.0	5.0	4.0	3.0	2.0	0.0	0.0	0.0	0.0	0.0

```

5.0  4.0  3.0  2.0  1.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

```

Press RETURN to continue...

Test #3 — Spread and reverse

```

0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  9.0  0.0  8.0  0.0  7.0  0.0  6.0  0.0  5.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  8.0  0.0  7.0  0.0  6.0  0.0  5.0  0.0  4.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  7.0  0.0  6.0  0.0  5.0  0.0  4.0  0.0  3.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  6.0  0.0  5.0  0.0  4.0  0.0  3.0  0.0  2.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  5.0  0.0  4.0  0.0  3.0  0.0  2.0  0.0  1.0

```

(3) 子过程 SVDFIT 在执行线性最小二乘法拟合中被推荐优于 LFIT. 验证程序 D9R4 中, 所采用的例子为

$$F(x) = 1 + 2x + 3x^2 + 4x^3 + 5x^4 + \text{Gaussian noise}$$

首先, 将这组数据拟合到一个 5 项多项式和, 然后再拟合到一个 5 项勒让德多项式和, 这两种情况下, 变量 y 方向上波动的 $SIG(I)$ 取为常数. 第一种情况, 计算结果的系数为 $A(I) \approx I$. 第二种情况结果应为: $A(1) \approx 3.0$, $A(2) \approx 4.3$, $A(3) \approx 4.9$, $A(4) \approx 1.6$, $A(5) \approx 1.1$. 验证程序 D9R4 如下:

```

PROGRAM D9R4
! Driver for routine SVDFIT
EXTERNAL FPOLY, FLEG
PARAMETER(NPT=100, SPREAD=0.02, NPOL=5)
! USES SVDVAR, GASDEV, FPOLY, FLEG
DIMENSION X(NPT), Y(NPT), SIG(NPT), A(NPOL), CVM(NPOL, NPOL)
DIMENSION U(NPT, NPOL), V(NPOL, NPOL), W(NPOL)
! Polynomial fit
IDUM=-911
MP=NPT
NP=NPOL

```

```

DO I=1,NPT
  X(I)=0.02*I
  Y(I)=1.0+X(I)*(2.0+X(I)*(3.0+X(I)*(4.0+X(I)*5.0)))
  Y(I)=Y(I)+SPREAD*GASDEV(IDUM)
  SIG(I)=SPREAD
END DO
CALL SVDFIT(X,Y,SIG,NPT,A,NPOL,U,V,W,MP,NP,CHISQ,FPOLY)
CALL SVDVAR(V,NPOL,NP,W,CVM,NPOL)
WRITE(*,*) 'Polynomial fit'
DO I=1,NPOL
  WRITE(*, '(1X,F12.6,A,F10.6)') A(I), '  +- ', &
    SQR(CVM(I,I))
END DO
WRITE(*, '(1X,A,F12.6/)') 'Chi-squared', CHISQ
CALL SVDFIT(X,Y,SIG,NPT,A,NPOL,U,V,W,MP,NP,CHISQ,FLEG)
CALL SVDVAR(V,NPOL,NP,W,CVM,NPOL)
WRITE(*,*) 'Legendre polynomial fit'
DO I=1,NPOL
  WRITE(*, '(1X,F12.6,A,F10.6)') A(I), '  +- ', &
    SQR(CVM(I,I))
END DO
WRITE(*, '(1X,A,F12.6/)') 'Chi-squared', CHISQ
END

```

子过程 SVDFIT 需要调用子过程 SVDCOMP(见 1.8 节)和 SVBKSB(见 1.8 节). 验证程序 D9R4 要调用子过程 SVDVAR(见本节)和 GASDEV(6.2 节), 计算结果如下:

```

Polynomial fit
1.020194  +-  0.010631
1.907697  +-  0.072154
3.112780  |   0.144235
3.946987  +-  0.107031
5.009099  +-  0.026289
Chi-squared 108.720810
Legendre polynomial fit
3.059560  +-  0.061561
4.275996  +-  0.133633

```

```

4.937455  +-  0.110652
1.578820  | -  0.042813
1.144933  +-  0.006009
Chi-squared 108.723434

```

(4) 子过程 SVDVAR 是伴随 SVDFIT 用于计算拟合 MA 个参数的协方差矩阵 CVM. 在验证程序 D9R5 中, 我们输入向量 W 和数组 V , 且计算出由它们所确定的协方差矩阵 CVM. 我们还手算出正确的结果储存在数组 TRU 中以供比较. 验证程序 D9R5 如下:

```

PROGRAM D9R5
! Driver for routine SVDVAR
PARAMETER(MP=6,MA=3,NCVM=MA)
DIMENSION V(MP,MP),W(MP),CVM(NCVM,NCVM),TRU(MA,MA)
DATA W/0.0,1.0,2.0,3.0,4.0,5.0/
DATA V/1.0,2.0,3.0,4.0,5.0,6.0,1.0,2.0,3.0,4.0,5.0,6.0,&
      1.0,2.0,3.0,4.0,5.0,6.0,1.0,2.0,3.0,4.0,5.0,6.0,&
      1.0,2.0,3.0,4.0,5.0,6.0,1.0,2.0,3.0,4.0,5.0,6.0/
DATA TRU/1.25,2.5,3.75,2.5,5.0,7.5,3.75,7.5,11.25/
WRITE(*,'(/1X,A)') 'Matrix V'
DO I=1,MP
  WRITE(*,'(1X,6F12.6)') (V(I,J),J=1,MP)
END DO
WRITE(*,'(/1X,A)') 'Vector W'
WRITE(*,'(1X,6F12.6)') (W(I),I=1,MP)
CALL SVDVAR(V,MA,MP,W,CVM,NCVM)
WRITE(*,'(/1X,A)') 'Covariance matrix from SVDVAR'
DO I=1,MA
  WRITE(*,'(1X,3F12.6)') (CVM(I,J),J=1,MA)
END DO
WRITE(*,'(/1X,A)') 'Expected covariance matrix'
DO I=1,MA
  WRITE(*,'(1X,3F12.6)') (TRU(I,J),J=1,MA)
END DO
END

```

计算结果如下:

Matrix V

1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
3.000000	3.000000	3.000000	3.000000	3.000000	3.000000
4.000000	4.000000	4.000000	4.000000	4.000000	4.000000
5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
6.000000	6.000000	6.000000	6.000000	6.000000	6.000000

Vector W

0.000000	1.000000	2.000000	3.000000	4.000000	5.000000
----------	----------	----------	----------	----------	----------

Covariance matrix from SVDVAR

1.250000	2.500000	3.750000
2.500000	5.000000	7.500000
3.750000	7.500000	11.250000

Expected covariance matrix

1.250000	2.500000	3.750000
2.500000	5.000000	7.500000
3.750000	7.500000	11.250000

(5) 子过程 FPOLY 和 FLEG 在上述验证程序 D9R4 中分别用来计算 x 的幂和勒让德多项式的值. 在验证程序 D9R6 中, 我们调用 FPOLY 列出了 x 的幂值, 它们可以用手工检验. 验证程序 D9R6 如下:

```

PROGRAM D9R6
! Driver for routine FPOLY
PARAMETER(NVAL=15,DX=0.1,NPOLY=5)
DIMENSION AFUNC(NPOLY)
WRITE(*, '(/1X,T29,A)') 'Powers of X'
WRITE(*, '(/1X,T9,A,T17,A,T27,A,T37,A,T47,A,T57,A)') &
    'X', 'X * * 0', 'X * * 1', 'X * * 2', 'X * * 3', 'X * * 4'
DO I=1,NVAL
    X=I * DX
    CALL FPOLY(X,AFUNC,NPOLY)
    WRITE(*, '(1X,6F10.4)') X,(AFUNC(J),J=1,NPOLY)
END DO
END

```

计算结果如下:

	Powers of X				
X	X * * 0	X * * 1	X * * 2	X * * 3	X * * 4

0.1000	1.0000	0.1000	0.0100	0.0010	0.0001
0.2000	1.0000	0.2000	0.0400	0.0080	0.0016
0.3000	1.0000	0.3000	0.0900	0.0270	0.0081
0.4000	1.0000	0.4000	0.1600	0.0640	0.0256
0.5000	1.0000	0.5000	0.2500	0.1250	0.0625
0.6000	1.0000	0.6000	0.3600	0.2160	0.1296
0.7000	1.0000	0.7000	0.4900	0.3430	0.2401
0.8000	1.0000	0.8000	0.6400	0.5120	0.4096
0.9000	1.0000	0.9000	0.8100	0.7290	0.6561
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.1000	1.0000	1.1000	1.2100	1.3310	1.4641
1.2000	1.0000	1.2000	1.4400	1.7280	2.0736
1.3000	1.0000	1.3000	1.6900	2.1970	2.8561
1.4000	1.0000	1.4000	1.9600	2.7440	3.8416
1.5000	1.0000	1.5000	2.2500	3.3750	5.0625

(6) 在验证程序 D9R7 中,为了进行比较,我们利用连带勒让德函数过程 PLGNDR(见 4.8 节). 验证程序 D9R7 如下:

```

PROGRAM D9R7
! Driver for routine FLEG
PARAMETER(NVAL=5,DX=0.2,NPOLY=5)
! USES PLGNDR
DIMENSION AFUNC(NPOLY)
WRITE(*, '(1X,T25,A)') 'Legendre Polynomials'
WRITE(*, '(1X,T8,A,T18,A,T28,A,T38,A,T48,A)') &
    'N=1', 'N=2', 'N=3', 'N=4', 'N=5'
DO I=1,NVAL
    X=I*DX
    CALL FLEG(X,AFUNC,NPOLY)
    WRITE(*, '(1X,A,F6.2)') 'X =', X
    WRITE(*, '(1X,5F10.4,A)') &
        (AFUNC(J),J=1,NPOLY), ' routine FLEG'
    WRITE(*, '(1X,5F10.4,A/)') &
        (PLGNDR(J-1,0,X),J=1,NPOLY), ' routine PLGNDR'
END DO
END

```

计算结果如下:

Legendre Polynomials					
	N=1	N=2	N=3	N=4	N=5
X = 0.20					
	1.0000	0.2000	-0.4400	-0.2800	0.2320 routine FLEG
	1.0000	0.2000	-0.4400	-0.2800	0.2320 routine PLGNDR
X = 0.40					
	1.0000	0.4000	-0.2600	-0.4400	-0.1130 routine FLEG
	1.0000	0.4000	-0.2600	-0.4400	-0.1130 routine PLGNDR
X = 0.60					
	1.0000	0.6000	0.0400	-0.3600	-0.4080 routine FLEG
	1.0000	0.6000	0.0400	-0.3600	-0.4080 routine PLGNDR
X = 0.80					
	1.0000	0.8000	0.4600	0.0800	-0.2330 routine FLEG
	1.0000	0.8000	0.4600	0.0800	-0.2330 routine PLGNDR
X = 1.00					
	1.0000	1.0000	1.0000	1.0000	1.0000 routine FLEG
	1.0000	1.0000	1.0000	1.0000	1.0000 routine PLGNDR

9.3 非线性最小二乘法

1. 功能

设 y 与自变量 x 及参数 $\mathbf{a} = (a_1, \dots, a_m)^T$ 之间具有函数关系 $y = y(x; \mathbf{a})$, $(x_i, y_i) (i=1, \dots, n)$ 是 (x, y) 的 n 对观测值, L 是 $M = \{1, 2, \dots, m\}$ 的一个子集, 参数 $a_k (k \in (M-L))$ 已知, 求参数 $a_k, k \in L$, 使

$$\chi^2(\mathbf{a}) = \sum_{i=1}^n \left[\frac{y_i - y(x_i; \mathbf{a})}{\sigma_i} \right]^2$$

达到最小, 其中 σ_i 为第 i 个点 $(x_i, y_i) (i=1, \dots, n)$ 上测量误差的标准差.

子过程 MRQMIN 为列维布格-麦奎尔特 (Levenberg-Marguardt) 方法的一步迭代, 子过程 MRQCOF 用于估算列维布格-麦奎尔特方法中的线性化拟合矩阵 (海森矩阵, Hessian) 和极小化函数的梯度向量.

子过程 FGAUSS 是当函数关系为

$$y = y(x) = \sum_{k=1}^m h_k \exp \left[- \left(\frac{x - c_k}{g_k} \right)^2 \right]$$

时子过程 MRQMIN 与 MRQCOF 中的一个例子.

2. 方法

问题等价于求解非线性方程

$$\nabla \chi^2(\mathbf{a}) = 0 \quad (9-4)$$

这里 ∇ 表示梯度算子, 解方程(9-4)的牛顿法是: 给 \mathbf{a} 一个初始值 $\mathbf{a}^{(0)}$,

$$\begin{aligned} \mathbf{a}^{(k+1)} &= \mathbf{a}^{(k)} + \Delta \mathbf{a}^{(k)} \\ \nabla^2 \chi^2(\mathbf{a}^{(k)}) \Delta \mathbf{a}^{(k)} &= -\nabla \chi^2(\mathbf{a}^{(k)}) \\ k &= 0, 1, 2, \dots \end{aligned} \quad (9-5)$$

而求解原问题的最速下降法是:

$$\Delta \mathbf{a}^{(k)} = \mu [-\nabla \chi^2(\mathbf{a}^{(k)})] \quad (9-6)$$

对固定的 K , 记

$$\begin{aligned} [\alpha] &= \frac{1}{2} \nabla^2 \chi^2(\mathbf{a}^{(k)}), \quad \beta = -\frac{1}{2} \nabla \chi^2(\mathbf{a}^{(k)}) \\ \Delta \mathbf{a} &= \Delta \mathbf{a}^{(k)} \end{aligned}$$

则由式(9-5)和式(9-6)有

$$[\alpha] \Delta \mathbf{a} = \beta \quad (9-7)$$

$$\Delta \mathbf{a} = \mu \beta \quad (9-8)$$

组合式(9-7)与式(9-8)得列维布格-麦奎尔特方法

$$([\alpha] + \lambda \mathbf{I}) \Delta \mathbf{a} = \beta \quad (9-9)$$

计算中把式(9-9)改写为

$$([\alpha] + \lambda \mathbf{D}) \Delta \mathbf{a} = \beta \quad (9-10)$$

其中 $\mathbf{D} = \text{diag}(\alpha_{11}, \alpha_{22}, \dots, \alpha_{mm})$. 由

$$\begin{aligned} \frac{\partial \chi^2}{\partial a_j} &= -2 \sum_{i=1}^n \frac{[y_i - y(x_i; \mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i; \mathbf{a})}{\partial a_j}, \quad j = 1, \dots, m \\ \frac{\partial^2 \chi^2}{\partial a_l \partial a_j} &= 2 \sum_{i=1}^n \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_l} \frac{\partial y(x_i; \mathbf{a})}{\partial a_j} - [y_i - y(x_i; \mathbf{a})] \left[\frac{\partial^2 y(x_i; \mathbf{a})}{\partial a_l \partial a_j} \right] \right] \end{aligned}$$

因此

$$a_{lj} \approx \sum_{i=1}^n \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_l} \frac{\partial y(x_i; \mathbf{a})}{\partial a_j} \right], \quad l = 1, \dots, m$$

算法:

- ① 给定参数的一个初始值 \mathbf{a} , 计算 $\chi^2(\mathbf{a})$;
- ② 取一个 λ 值, 如 $\lambda = 0.001$, 令 $k = 0$;
- ③ 解线性方程(9-10)得 $\Delta \mathbf{a}$, 计算 $\chi^2(\mathbf{a} + \Delta \mathbf{a})$;
- ④ 若 $\lambda = 0$, 则结束迭代, 否则转⑤;

⑤ 若 $\chi^2(a+\Delta a) \geq \chi^2(a)$, 则取 $a \leftarrow a + \Delta a$, $\lambda \leftarrow 10\lambda$, 转③;

⑥ 若 $\chi^2(a+\Delta a) < \chi^2(a)$, 则当 $|\chi^2(a+\Delta a) - \chi^2(a)| < 0$, 或 $|\chi^2(a+\Delta a) - \chi^2(a)| < 10^{-3} \cdot |\chi^2(a)|$ 时, 如果 $K \leq 2$, 则取 $a \leftarrow a + \Delta a$, $\lambda \leftarrow \lambda/10$, $K \leftarrow K+1$, 转③; 如果 $K > 2$, 则取 $\lambda \leftarrow 0$, $a \leftarrow a + \Delta a$, 转③.

3. 使用说明

(1) MRQMIN (X, Y, SIG, NDATA, A, MA, LISTA, MFIT, COVAR, ALPHA, NCA, CHISQ, FUNCS, ALAMDA)

NDATA	整型变量, 输入参数, 数据点的个数
MA	整型变量, 输入参数, 参数的总个数
MFIT	整型变量, 输入参数, 欲求参数的个数
NCA	整型变量, 输入参数, 存储方阵 COVAR 与 ALPHA 的物理维数
X	NDATA 个元素的一维实型数组, 输入参数, 存放自变量 x 的 n 个观测值
Y	NDATA 个元素的一维实型数组, 输入参数, 存放变量 y 的 n 个观测值
SIG	NDATA 个元素的一维实型数组, 输入参数, 存放 n 个数据点 $(x_i, y_i) (i=1, \dots, n)$ 上的测量误差的标准差 $\sigma_1, \dots, \sigma_n$, 若它们未知, 则全部输入为 1
A	MA 个元素的一维实型数组, LISTA 中最后 MA-MFIT 个元素值表示了 A 中输入的那些元素的下标值, 输入已知参数的值; LISTA 的前 MFIT 个元素值表示了 A 中输出的那些元素的下标值, 输出所求参数的该次迭代的结果值
LISTA	MA 个元素的一维整型数组, 输入参数, 对参数编号, 使 LISTA 的前 MFIT 个元素相应于要确定的参数, 其余 MA-MFIT 个元素相应于已知参数
COVAR, ALPHA	均为 MA×MA 个元素的二维实型数组, 若 ALAMDA $\neq 0$, 则它们为工作单元, 若 ALAMDA = 0, 则它们为输出参数, COVAR 存放参数的协方差矩阵, ALPHA 存放曲率矩阵
CHISQ	实型变量, 输出参数, 存放极小化函数 χ^2 的值
FUNCS	子过程参数, 形式为 FUNCS(X, A, YFIT, DYDA, MA), 用于计算拟合函数 $y=y(x; a)$ 在 $x=X$ 处的函数值 YFIT 与其关于参数的导数 DYDA, 用户自编, 一个例子是 FGAUSS

ALAMDA 实型变量,输入、输出参数,第一次调用子过程时,输入一个负数,最后一次调用时输入为零,中间迭代过程中,输入上一次调用的输出值

(2) MRQCOF (X,Y,SIG,NDATA,A,MA,LISTA,MFIT,ALPHA,BETA,NALP,CHISQ,FUNCS)

NALP 整型变量,输入参数,存储方阵 ALPHA 的物理维数

ALPHA MFIT×MFIT 个元素的二维实型数组,输出参数,存放每次迭代的线性化拟合矩阵

BETA MA 个元素的一维实型数组,输出参数,存放极小化函数 χ^2 关于参数的梯度向量

X,Y,SIG,NDATA,A,MA,LISTA,MFIT,CHISQ,FUNCS 均同子过程 MRQMIN

(3) FGAUSS(X,A,Y,DYDA,NA)

NA 整型变量,输入参数,参数的个数,这里是3的倍数

X 实型变量,输入参数,自变量 x 的值

A NA 个元素的一维实型数组,输入参数,存放参数的值,使 b_k, e_k, g_k 具有相继位置: $A(I) = B_k, A(I+1) = E_k, A(I+2) = G_k, k=1, 2, \dots, NA/3$

Y 实型变量,输出参数,存放函数值

DYDA NA 个元素的一维实型数组,输出参数,存放函数关于参数的导数值

4. 过程

(1) 子过程 MRQMIN.

```
SUBROUTINE mrqmin(X,Y,SIG,NDATA,A,MA,LISTA,MFIT,&
```

```
COVAR,ALPHA,NCA,CHISQ,FUNCS,ALAMDA)
```

```
INTEGER ma,nca,ndata,lista(ma),MMAX
```

```
REAL alambda,chisq,funcs,a(ma),alpha(nca,nca),&
```

```
covar(nca,nca),sig(ndata),x(ndata),y(ndata)
```

```
PARAMETER (MMAX=20)
```

```
!USES covsrt,gaussj,mrqcof
```

```
INTEGER j,k,l,mfit
```

```
REAL ochisq,atry(MMAX),beta(MMAX),da(MMAX)
```

```
if(alambda<0.) then
```

```

kk=mfit+1
do j=1,ma
    ihit=0
    do k=1,mfit
        if(lista(k)==j) ihit=ihit+1
    end do
    if(ihit==0) then
        lista(kk)=j
        kk=kk+1
    else if(ihit>1) then
        pause 'improper permutation in lista'
    endif
end do
if(kk/=(ma+1)) pause 'improper permutation in lista'
alamda=0.001
call mrqcof(x,y,sig,ndata,a,ma,lista,mfit,alpha,&
            beta,nca,chisq,funcs)
ochisq=chisq
do j=1,ma
    atry(j)=a(j)
end do
endif
do j=1,mfit
    do k=1,mfit
        covar(j,k)=alpha(j,k)
    end do
    covar(j,j)=alpha(j,j)*(1.+alamda)
    da(j)=beta(j)
end do
call gaussj(covar,mfit,da)
if(alamda==0.) then
    call covsrt(covar,nca,ma,lista,mfit)
    return
endif
do j=1,mfit
    atry(lista(j))=a(lista(j))+da(j)
end do

```

```

call mrqcof(x,y,sig,ndata,atry,ma,lista,mfit,covar,da,&
            nca,chisq,funcs)
if(chisq<ochisq) then
    alamda=0.1*alamda
    ochisq=chisq
    do j=1,mfit
        do k=1,mfit
            alpha(j,k)=covar(j,k)
        end do
        beta(j)=da(j)
        a(lista(j))=atry(lista(j))
    end do
else
    alamda=10.*alamda
    chisq=ochisq
endif
END SUBROUTINE mrqmin

```

(2) 子过程 MRQCOF.

```

SUBROUTINE mrqcof(x,y,sig,ndata,a,ma,lista,mfit,&
                  alpha,beta,nalp,chisq,funcs)
PARAMETER (mmax=20)
DIMENSION x(ndata),y(ndata),alpha(nalp,nalp),&
          beta(ma),dyda(mmax),lista(mfit),sig(ndata),a(ma)
do j=1,mfit
    do k=1,j
        alpha(j,k)=0.
    end do
    beta(j)=0.
end do
chisq=0.
do i=1,ndata
    call funcs(x(i),a,ymod,dyda,ma)
    sig2i=1./(sig(i)*sig(i))
    dy=y(i)-ymod
    do j=1,mfit
        wt=dyda(lista(j))*sig2i

```

```

      do k=1,j
        alpha(j,k)=alpha(j,k)+wt*dyda(lsta(k))
      end do
      beta(j)=beta(j)+dy*wt
    end do
    chisq=chisq+dy*dy*sig2i
  end do
  do j=2,mfit
    do k=1,j-1
      alpha(k,j)=alpha(j,k)
    end do
  end do
END SUBROUTINE mrqcof

```

(3) 子过程 FGAUSS.

```

SUBROUTINE fgauss(x,a,y,dyda,na)
INTEGER na
REAL x,y,a(na),dyda(na)
INTEGER i
REAL arg,ex,fac
y=0.
do i=1,na-1,3
  arg=(x-a(i+1))/a(i+2)
  ex=exp(-arg**2)
  fac=a(i)*ex*2.*arg
  y=y+a(i)*ex
  dyda(i)=ex
  dyda(i+1)=fac/a(i+2)
  dyda(i+2)=fac*arg/a(i+2)
end do
END SUBROUTINE fgauss

```

5. 例子

(1) 子过程 MRQMIN 与子过程 MRQCOF 一起用列维布格-麦奎尔特方法做非线性最小二乘拟合. 验证程序 D9R8A 采用的人为数据是两个指数(高斯)函数的和再加上干扰:

$$Y(I) = A(1)\exp\{-[(x(I) - A(2))/A(3)]^2\}$$

$$+ A(4)\exp\{-[(x(I) - A(5))/A(6)]^2\} + \text{noise}$$

在拟合开始时,又将初始猜测值 GUES(I)赋予 A(I). 数组 LISTA(I)赋值为 I. 第一次调用子过程 MRQMIN 时, ALAMDA = -1. 子过程 MRQMIN 在 χ^2 值 CHISQ 的一系列测试时进行迭代. 当两次相邻迭代后, CHISQ 的值的减小于 0.1 时,拟合就算完成. 最后, MRQMIN 以 ALAMDA = 0 再调用一次,使得数组 COVAR 将返回协方差矩阵. 误差由 COVAR 对角线元素的平方根所确定. 验证程序 D9R8A 如下:

```

PROGRAM D9R8A
  !Driver for routine MRQMIN
  EXTERNAL FGAUSS
  PARAMETER(NPT=100,MA=6,SPREAD=0.001)
  DIMENSION X(NPT),Y(NPT),SIG(NPT),A(MA),LISTA(MA),&
             COVAR(MA,MA),ALPHA(MA,MA),GUES(MA)
  DATA A/5.0,2.0,3.0,2.0,5.0,3.0/
  DATA GUES/4.5,2.2,2.8,2.5,4.9,2.8/
  IDUM=-911
  !First try a sum of two Gaussians
  DO I=1,100
    X(I)=0.1*I
    Y(I)=0.0
    DO J=1,4,3
      Y(I)=Y(I)+A(J)*EXP(-((X(I)-A(J+1))/A(J+2))* * 2)
    END DO
    Y(I)=Y(I)*(1.0+SPREAD*GASDEV(IDUM))
    SIG(I)=SPREAD*Y(I)
  END DO
  MFIT=MA
  DO I=1,MFIT
    LISTA(I)=I
  END DO
  ALAMDA=-1
  DO I=1,MA
    A(I)=GUES(I)
  END DO
  CALL MRQMIN(X,Y,SIG,NPT,A,MA,LISTA,MFIT,COVAR,ALPHA,&
             MA,CHISQ,FGAUSS,ALAMDA)

```

```

K=1
ITST=0
DO
  WRITE(*, '(/1X,A,I2,T18,A,F10.4,T43,A,E9.2)') &
    'Iteration #', K, 'Chi-squared:', CHISQ, 'ALAMDA:', ALAMDA
  WRITE(*, '(1X,T5,A,T13,A,T21,A,T29,A,T37,A,T45,A)') &
    'A(1)', 'A(2)', 'A(3)', 'A(4)', 'A(5)', 'A(6)'
  WRITE(*, '(1X,6F8.4)') (A(I), I=1,6)
  K=K+1
  OCHISQ=CHISQ
  CALL MRQMIN(X,Y,SIG,NPT,A,MA,LISTA,MFIT,COVAR,ALPHA,&
    MA,CHISQ,FGAUSS,ALAMDA)
  IF (CHISQ>OCHISQ) THEN
    ITST=0
  ELSE IF (ABS(OCHISQ-CHISQ)<0.1) THEN
    ITST=ITST+1
  ENDIF
  IF (ITST>=2) EXIT
END DO
ALAMDA=0.0
CALL MRQMIN(X,Y,SIG,NPT,A,MA,LISTA,MFIT,COVAR,ALPHA,&
  MA,CHISQ,FGAUSS,ALAMDA)
WRITE(*,*) 'Uncertainties:'
WRITE(*, '(1X,6F8.4/)') (SQRT(COVAR(I,I)), I=1,6)
WRITE(*, '(1X,A)') 'Expected results:'
WRITE(*, '(1X,F7.2,5F8.2/)') 5.0,2.0,3.0,2.0,5.0,3.0
END

```

子过程 MRQMIN 需要调用子过程 MRQCOF(见本节)、COVSRT(见9.2节)和 GAUSSJ(见1.1节). 验证程序 D9R8A 需要调用 GASDEV(见6.2节), 计算结果如下:

```

Iteration # 1   Chi-squared: 17729.0430   ALAMDA: 0.10E-03
  A(1)   A(2)   A(3)   A(4)   A(5)   A(6)
  4.7152  1.9326  3.0161  2.1725  4.8002  3.0640
Iteration # 2   Chi-squared:  187.4347   ALAMDA: 0.10E-04
  A(1)   A(2)   A(3)   A(4)   A(5)   A(6)
  4.8586  1.9395  2.9606  2.1701  4.8807  3.0340

```

Iteration # 3 Chi-squared: 184.9744 ALAMDA: 0.10E-05

A(1) A(2) A(3) A(4) A(5) A(6)

5.0050 1.9987 3.0013 1.9982 4.9991 3.0003

Iteration # 4 Chi-squared: 107.8414 ALAMDA: 0.10E-06

A(1) A(2) A(3) A(4) A(5) A(6)

5.0391 2.0155 3.0123 1.9508 5.0355 2.9904

Iteration # 5 Chi-squared: 107.1702 ALAMDA: 0.10E-07

A(1) A(2) A(3) A(4) A(5) A(6)

5.0394 2.0159 3.0126 1.9500 5.0362 2.9902

Iteration # 6 Chi squared: 107.1690 ALAMDA: 0.10E-08

A(1) A(2) A(3) A(4) A(5) A(6)

5.0394 2.0159 3.0126 1.9500 5.0362 2.9902

Uncertainties:

0.0284 0.0124 0.0083 0.0356 0.0263 0.0074

Expected results:

5.00 2.00 3.00 2.00 5.00 3.00

(2) 在验证子过程 MRQCOF 的 D9R8B 中,函数取的和 D9R8A 中的一样.且子过程 MRQCOF 被两次调用.第一次调用时,LISTA(I)=(I)及 MFIT=6,因而6个参数都被用上.第二次调用时,LISTA(I)=I+3且 MFIT=3,这时前三个参数被固定,后三个参数被拟合.验证程序 D9R8B 如下:

```
PROGRAM D9R8B
```

```
!Driver for routine MRQCOF
```

```
EXTERNAL FGAUSS
```

```
PARAMETER(NPT=100,MA=6,SPREAD=0.1)
```

```
!USES GASDEV
```

```
DIMENSION X(NPT),Y(NPT),SIG(NPT),A(MA),LISTA(MA),&
```

```
COVAR(MA,MA),ALPHA(MA,MA),GUES(MA),BETA(MA)
```

```
DATA A/5.0,2.0,3.0,2.0,5.0,3.0/
```

```
DATA GUES/4.9,2.1,2.9,2.1,4.9,3.1/
```

```
IDUM=-911
```

```
!First try a sum of two Gaussians
```

```
DO I=1,100
```

```
  X(I)=0.1*I
```

```
  Y(I)=0.0
```

```
  DO J=1,4,3
```

```
    Y(I)=Y(I)+A(J)*EXP(-((X(I)-A(J+1))/A(J+2))* * 2)
```



```

      END DO
      Y(I)=Y(I)*(1.0+SPREAD*GASDEV(IDUM))
      SIG(I)=SPREAD*Y(I)
    END DO
    MFIT=MA
    DO I=1,MFIT
      LISTA(I)=I
    END DO
    DO I=1,MA
      A(I)=GUES(I)
    END DO
    CALL MRQCOF(X,Y,SIG,NPT,A,MA,LISTA,MFIT,ALPHA,&
      BETA,MA,CHISQ,FGAUSS)
    WRITE(*, '(/1X,A)') 'matrix alpha'
    DO I=1,MA
      WRITE(*, '(1X,6F12.4)') (ALPHA(I,J),J=1,MA)
    END DO
    WRITE(*, '(1X,A)') 'vector beta'
    WRITE(*, '(1X,6F12.4)') (BETA(I),I=1,MA)
    WRITE(*, '(1X,A,F12.4/)') 'Chi-squared:',CHISQ
    !Next fix one line and improve the other
    DO I=1,3
      LISTA(I)=1+3
    END DO
    MFIT=3
    DO I=1,MA
      A(I)=GUES(I)
    END DO
    CALL MRQCOF(X,Y,SIG,NPT,A,MA,LISTA,MFIT,ALPHA,&
      BETA,MA,CHISQ,FGAUSS)
    WRITE(*, '(/1X,A)') 'matrix alpha'
    DO I=1,MFIT
      WRITE(*, '(1X,6F12.4)') (ALPHA(I,J),J=1,MFIT)
    END DO
    WRITE(*, '(1X,A)') 'vector beta'
    WRITE(*, '(1X,6F12.4)') (BETA(I),I=1,MFIT)
    WRITE(*, '(1X,A,F12.4/)') 'Chi-squared:',CHISQ

```

END

计算结果如下:

matrix alpha

137.2021	60.6572	148.2204	147.7027	-12.6093	90.9625
60.6572	500.8828	366.9686	448.3021	287.4483	257.7019
148.2204	366.9686	583.6553	659.6396	553.0108	612.6782
147.7027	448.3021	659.6396	971.0059	1131.7701	1410.9381
-12.6093	287.4483	553.0108	1131.7701	1911.5934	2409.3489
90.9625	257.7019	612.6782	1410.9381	2409.3489	3341.0007

vector beta

-24.7629 -118.2998 -76.5073 -124.1267 -104.8774 -145.4017

Chi-squared: 147.9271

matrix alpha

971.0059	1131.7701	1410.9381
1131.7701	1911.5934	2409.3489
1410.9381	2409.3489	3341.0007

vector beta

-124.1267 -104.8774 -145.4017

Chi-squared: 147.9271

(3) 子过程 FGAUSS 计算了函数值及关于每个可变参数的导数值. 验证程序 D9R9 将计算结果和精确解进行了比较. 验证程序 D9R9 如下:

```
PROGRAM D9R9
```

```
!Driver for routine FGAUSS
```

```
PARAMETER(NPT=3,NLIN=2,NA=3 * NLIN)
```

```
DIMENSION A(NA),DYDA(NA),DF(NA)
```

```
DATA A/3.0,0.2,0.5,1.0,0.7,0.3/
```

```
WRITE(*, '(/1X,T6,A,T14,A,T19,A,T27,A,T35,A,T43,A,&
```

```
      T51,A,T59,A)') 'X','Y','DATA1','DATA2',&
```

```
          'DATA3','DATA4','DATA5','DATA6'
```

```
DO I=1,NPT
```

```
   X=0.3 * I
```

```
   CALL FGAUSS(X,A,Y,DYDA,NA)
```

```
   E1=EXP(-((X-A(2))/A(3)) * * 2)
```

```
   E2=EXP(-((X-A(5))/A(6)) * * 2)
```

```
   F=A(1) * E1+A(4) * E2
```

```
   DF(1)=E1
```

```

DF(4)=E2
DF(2)=A(1)*E1*2.0*(X-A(2))/(A(3)**2)
DF(5)=A(4)*E2*2.0*(X-A(5))/(A(6)**2)
DF(3)=A(1)*E1*2.0*((X-A(2))**2)/(A(3)**3)
DF(6)=A(4)*E2*2.0*((X-A(5))**2)/(A(6)**3)
WRITE(*,'(1X,A/,8F8.4)') 'from FGAUSS',X,Y,&
      (DYDA(J),J=1,6)
WRITE(*,'(1X,A/,8F8.4)') 'independent calc.',X,F,&
      (DF(J),J=1,6)
END DO
END

计算结果如下:

      X      Y    DATA1 DATA2 DATA3 DATA4 DATA5 DATA6
from FGAUSS
      0.3000  3.0514  0.9608  2.3059  0.4612  0.1690 -1.5023  2.0031
independent calc.
      0.3000  3.0514  0.9608  2.3059  0.4612  0.1690 -1.5023  2.0031
from FGAUSS
      0.6000  2.4767  0.5273  5.0620  4.0496  0.8948 -1.9885  0.6628
independent calc.
      0.6000  2.4767  0.5273  5.0620  4.0496  0.8948 -1.9885  0.6628
from FGAUSS
      0.9000  1.0638  0.1409  2.3664  3.3130  0.6412  2.8497  1.8998
independent calc.
      0.9000  1.0638  0.1409  2.3664  3.3130  0.6412  2.8497  1.8998

```

9.4 绝对值偏差最小的直线拟合

1. 功能

设变量 y 是关于自变量 x 的线性函数

$$y = a + bx$$

现给定 (x, y) 的 n 对观测值 $(x_i, y_i) (i=1, 2, \dots, n)$, 求参数 a, b , 使

$$\sum_{i=1}^n |y_i - a - bx_i|$$

达到极小.

子过程 MEDFIT 用于计算拟合参数 a 与 b , 函数子过程 ROFUNC 对给定的 b 计算 $\sum_{i=1}^n x_i \operatorname{sgn}(y_i - a - bx_i)$ 的值.

2. 方法

结论: 一组数 $\{C_1, \dots, C_n\}$ 的中位数 C_M 满足

$$C_M = \min \varphi(C_M), \quad \varphi(C_M) = \sum_{i=1}^n |C_i - C_M|$$

由此得到, 为使 $\sum_{i=1}^n |y_i - a - bx_i|$ 达到极小, 则对固定的 b , 其 a 为

$$a = a(b) = \operatorname{median}\{y_i - bx_i\}$$

而对参数 b 应有

$$F(a, b) = \frac{\partial}{\partial b} \sum_{i=1}^n |y_i - a - bx_i| = 0$$

即

$$F(a, b) = \sum_{i=1}^n x_i \operatorname{sgn}(y_i - a - bx_i) = 0$$

对方程, 用 $a = a(b)$ 代替 a , 则是单变量 b 的方程, 找出其根区间后用二分法求解.

3. 使用说明

(1) MEDFIT(X, Y, NDATA, A, B, ABDEV)

NDATA 整型变量, 输入参数, 观测数据点的个数

X NDATA 个元素的一维实型数组, 输入参数, 存放自变量 x 的 n 个观测值

Y NDATA 个元素的一维实型数组, 输入参数, 存放变量 y 的 n 个观测值

A 实型变量, 输出参数, 存放拟合参数 a

B 实型变量, 输出参数, 存放拟合参数 b

ABDEV 实型变量, 输出参数, 存放平均绝对值偏差 $\frac{1}{n} \sum_{i=1}^n |y_i - a - bx_i|$

(2) ROFUNC(B)

B 实型变量,输入参数,存放变量 b 的值

4. 过程

(1) 子过程 MEDFIT.

```

SUBROUTINE medfit(x,y,ndata,a,b,abdev)
  INTEGER ndata,NMAX,ndatat
  PARAMETER (NMAX=1000)
  REAL a,abdev,b,abdevt,X(NMAX),Y(NMAX)
  !USES rofunc
  INTEGER j
  REAL b1,b2,bb,chisq,del,f,f1,f2,sigb,sx,sxx,sxy,sy,rofunc
  COMMON /ARRAYS/ NDATAt,Xt(NMAX),Yt(NMAX),ARR(NMAX),AA,AB-
DEVt
  sx=0.
  sy=0.
  sxy=0.
  sxx=0.
  do j=1,ndata
    xt(j)=x(j)
    yt(j)=y(j)
    sx=sx+x(j)
    sy=sy+y(j)
    sxy=sxy+x(j)*y(j)
    sxx=sxx+x(j)**2
  end do
  ndatat=ndata
  del=ndata*sxx-sx**2
  aa=(sxx*sy-sx*sxy)/del
  bb=(ndata*sxy-sx*sy)/del
  chisq=0.
  do j=1,ndata
    chisq=chisq+(y(j)-(aa+bb*x(j)))**2
  end do
  sigb=sqrt(chisq/del)
  b1=bb
  f1=rofunc(b1)
  b2=bb+sign(3.*sigb,f1)

```

```

f2=rofunc(b2)
if(b2==b1)then
  a=aa
  b=bb
  abdev=abdevt/ndata
  return
endif
do while(f1*f2>0.)
  bb=b2+1.6*(b2-b1)
  b1=b2
  f1=f2
  b2=bb
  f2=rofunc(b2)
end do
sigb=0.01*sigb
do while(abs(b2-b1)>sigb)
  bb=b1+0.5*(b2-b1)
  if(bb==b1. or. bb==b2) exit
  f=rofunc(bb)
  if(f*f1>=0.)then
    f1=f
    b1=bb
  else
    f2=f
    b2=bb
  endif
end do
a=aa
b=bb
abdev=abdevt/ndata
END SUBROUTINE medfit

```

(2) 函数子过程 ROFUNC.

```

FUNCTION rofunc(b)
PARAMETER (nmax=1000)
!USES sort
COMMON /arrays/ ndata,x(nmax),y(nmax),arr(nmax),aa,abdev

```

```

n1=ndata+1
nml=n1/2
nmh=n1-nml
do j=1,ndata
    arr(j)=y(j)-b*x(j)
end do
call sort(ndata,arr)
aa=0.5*(arr(nml)+arr(nmh))
sum=0.
abdev=0.
do j=1,ndata
    d=y(j)-(b*x(j)+aa)
    abdev=abdev+abs(d)
    sum=sum+x(j)*sign(1.0,d)
end do
rofunc=sum
END FUNCTION rofunc

```

5. 例子

子过程 MEDFIT 是拟合的一种更“稳健”的方法. 它使数据向一直线拟合, 采用了最小绝对值偏差方法, 而不是最小二乘准则. 为了进行比较, 验证程序 D9R10 将有一扰动的线性数据向直线拟合时, 先调用子过程 FIT (见 9.1 节), 然后再调用 MEDFIT. 可以注意到, 能计算出期望的具有扰动振幅为 SPREAD 的绝对值偏差的均值. 子过程 MEDFIT 需要调用子过程 ROFUNC (见本节) 和 SORT (7.2 节). 验证程序 D9R10 如下:

```

PROGRAM D9R10
!Driver for routine MEDFIT
PARAMETER(NPT=100,SPREAD=0.1)
!USES GASDEV,FIT
DIMENSION X(NPT),Y(NPT),SIG(NPT)
IDUM=-1984
DO I=1,NPT
    X(I)=0.1*I
    Y(I)=-2.0*X(I)+1.0+SPREAD*GASDEV(IDUM)
    SIG(I)=SPREAD
END DO

```

```

MWT=1
CALL FIT(X,Y,NPT,SIG,MWT,A,B,SIGA,SIGB,CHI2,Q)
WRITE(*, '(1X,A)')&
    'According to routine FIT the result is:'
WRITE(*, '(1X,T5,A,F8.4,T20,A,F8.4)') 'A = ',A,&
    'Uncertainty:',SIGA
WRITE(*, '(1X,T5,A,F8.4,T20,A,F8.4)') 'B = ',B,&
    'Uncertainty:',SIGB
WRITE(*, '(1X,T5,A,F8.4,A,I4,A)') 'Chi-squared:',CHI2,&
    ' for ',NPT,' points'
WRITE(*, '(1X,T5,A,F8.4)') 'Goodness-of-fit:',Q
WRITE(*, '(1X,A)')&
    'According to routine MEDFIT the result is:'
CALL MEDFIT(X,Y,NPT,A,B,ABDEV)
WRITE(*, '(1X,T5,A,F8.4)') 'A = ',A
WRITE(*, '(1X,T5,A,F8.4)') 'B = ',B
WRITE(*, '(1X,T5,A,F8.4)')&
    'Absolute deviation (per data point): ',ABDEV
WRITE(*, '(1X,T5,A,F8.4,A)')&
    '(note: Gaussian spread is',SPREAD,')'
END

```

计算结果如下:

According to routine FIT the result is:

```

A =   1.0031   Uncertainty:  0.0202
B =  -2.0012   Uncertainty:  0.0035
Chi-squared: 91.9298 for  100 points
Goodness-of-fit:  0.6536

```

According to routine MEDFIT the result is:

```

A =   1.0082
B =  -2.0011
Absolute deviation (per data point):  0.0778
(note: Gaussian spread is  0.1000)

```


第 10 章 方程求根和非线性方程组的解法

本章主要介绍的是求一元方程的根. 子过程 SCRSO 能产生给定函数在确定区间上的一个粗糙图形, 它给出一个函数的图像初级解. 随后介绍两个子过程 ZBRAC 和 ZBRAK, 当设定好函数和区间时, ZBRAC 能几何地延拓区间直至找到一个有根区间为止; 而子过程 ZBRAK 是将某区间划分为 N 个相等子区间, 然后计算出哪些区间包含有方程的根. 一旦包含根的区间确定后, 接着有一系列的子过程能准确确定根的位置. 子过程 RTBIC 采用二分法找出这些根. RTFLSP 和 RTSEC 分别采用试位法和割线法求根. ZBRENT 采用组合的方法, 总能保证收敛且收敛速度比二分法快. RTNEWT 采用牛顿-拉斐森方法求方程的根, 而 RTSAVE 将牛顿-拉斐森方法和二分法结合起来, 它能改善 RTNEWT 的收敛性.

为了求出多项式的根, LAGUER 是手头常用的程序; 而把 LAGUER 和它的驱动程序 ZROOTS 结合起来, 将能得到复系数多项式的全部根. 如果一个实系数多项式有一些不确定的复根, 在 QROOT 中采用贝尔斯托方法能精确地求出.

对非线性方程组求根需要一些预见性, 即如果能确定这些根的近似值, 那么 MNEWT 将采用牛顿-拉斐森方法得到根.

10.1 图 解 法

1. 功能

本子过程能对原始函数 FX 画出其在区间 $[X_1, X_2]$ 中的大致图形, 且改变 X_1, X_2 可得另一区间中的大致图形, 若 $X_1 = X_2$ 则停止执行.

2. 方法

(略)

3. 使用说明

SCRSO(FX)

FX 任意参数

4. 过程

子过程 SCRSHO.

```

SUBROUTINE scrsho(fx)
INTEGER ISCR,JSCR
REAL fx
EXTERNAL fx
PARAMETER (ISCR=60,JSCR=21)
INTEGER i,j,jz
REAL dx,dyj,x,x1,x2,ybig,ysml,y(ISCR)
CHARACTER * 1 scr(ISCR,JSCR),blank,zero,yy,xx,ff
SAVE blank,zero,yy,xx,ff
DATA blank,zero,yy,xx,ff/' ','-' , 'l', ' - ', 'x' /
do
  write (*,*) ' Enter x1,x2 ( = to stop) '
  read (*,*) x1,x2
  if(x1==x2) return
  do j=1,JSCR
    scr(1,j)=yy
    scr(ISCR,j)=yy
  end do
  do i=2,ISCR-1
    scr(i,1)=xx
    scr(i,JSCR)=xx
    do j=2,JSCR-1
      scr(i,j)=blank
    end do
  end do
  dx=(x2-x1)/(ISCR-1)
  x=x1
  ybig=0.
  ysml=ybig
  do i=1,ISCR
    y(i)=fx(x)
    if(y(i)<ysml) ysml=y(i)
    if(y(i)>ybig) ybig=y(i)
    x=x+dx
  end do
end do

```

```

end do
if(ybig==ysml) ybig=yaml+1.
dyj=(JSCR-1)/(ybig-ysml)
jz=1-ysml*dyj
do i=1,ISCR
    scr(i,jz)=zero
    j=1+(y(i)-ysml)*dyj
    scr(i,j)=ff
end do
write (*,'(1x,1pe10.3,1x,80a1)') ybig,&
                                     (scr(i,JSCR),i=1,ISCR)
do j=JSCR-1,2,-1
    write (*,'(12x,80a1)') (scr(i,j),i=1,ISCR)
end do
write (*,'(1x,1pe10.3,1x,80a1)') ysml,&
                                     (scr(i,1),i=1,ISCR)
write (*,'(12x,1pe10.3,40x,e10.3)') x1,x2
end do
END SUBROUTINE scrsho

```

5. 例子

为了验证子过程 SCRSHO,我们采用的例子是零阶贝塞尔函数 J_0 ,验证程序 D10R1 如下:

```

PROGRAM D10R1
! Driver for routine SCRSHO
EXTERNAL BESSJ0
! USES BESSJ0
WRITE(*,*) 'Graph of the Bessel Function J0;'
CALL SCRSHO(BESSJ0)
END

```

计算结果如下:

```

Graph of the Bessel Function J0;
Enter x1,x2 (= to stop)
-5.0,5.0

```

10.2 逐步扫描法和二分法

本节包含三个子过程,它们的功能分别是:

(3) RTBIS, 在有根区间 $[x_1, x_2]$ 中, 用二分法求满足精度要求的一个根.

(1) 确定有根区间, 任选两点 x_1 和 x_2 , 计算 $f(x_1)$ 和 $f(x_2)$.

① 若 $f(x_1)f(x_2)=0$, 则已找到了方程的一个根; 若 $f(x_1)f(x_2)<0$, 则 $[x_1, x_2]$ 即为 $f(x)=0$ 的一个有根区间; 否则进行②.

② 若 $f(x_1)f(x_2)>0$, 沿下降方向重选一点, 即若 $|f(x_1)|<|f(x_2)|$, 则沿 $x_2 \rightarrow x_1$ 的方向重选一点作为新的 x_1 代替旧的 x_1 , 否则沿 $x_1 \rightarrow x_2$ 的方向选一点作为 x_2 而代替旧的 x_2 , 返回①.

这样, 若 $x \rightarrow \pm\infty$ 时, 如果 $f(x)$ 有相反符号, 则总可找到 $f(x)=0$ 的一个有根区间. 当然, 若上述条件不满足, 则可能失败而找不到有根区间.

(2) 逐步扫描法求有根区间及其数目.

设 $f(x)$ 定义在 $[x_1, x_2]$ 中. 将 $[x_1, x_2]$ n 等分得到 $x_1 \equiv \bar{x}_0 < \bar{x}_1 < \cdots < \bar{x}_n \equiv x_2$, $\bar{x}_i = \bar{x}_0 + ih$, $h = (x_2 - x_1)/n$, 且设这些小区间 $\{[x_{i-1}, x_i]\}$ 中最多有 NB 个有根区间. 从区间 $[x_1, x_2]$ 的左端点 x_1 出发, 按步长 h 一步一步向右跨, 每跨一步进行一次根的搜索, 即检验相邻两节点上的函数值的符号, 若 $f(\bar{x}_k)f(\bar{x}_{k-1}) < 0$, 则确定了一个有根区间, $MB \leftarrow MB + 1$, 若 $MB = NB$ 或已搜索完了几个小区间, 搜索停止.

显然, 只要步长 h 取得足够小, 利用这种方法可以得到具有任意精度的近似根 (因有根区间长度即为 h , 近似根可取有根区间的中点), 但当 h 缩小时, 所要搜索的步数相应增加, 计算量增大. 故用这种方法求高精度的近似根是不合算的.

(3) 二分法求方程 $f(x)=0$ 的根.

设 $[x_1, x_2]$ 是方程的有根区间, 且不妨设 $f(x_1) < 0, f(x_2) > 0$, 用区间的中点 $(x_1 + x_2)/2$ 平分区间 (x_1, x_2) 为两个区间, 计算 $f\left(\frac{x_1 + x_2}{2}\right)$, 根据 $f\left(\frac{x_1 + x_2}{2}\right)$ 的值分两种情况:

① $\left|f\left(\frac{x_1 + x_2}{2}\right)\right| < \varepsilon$, ε 是预先给定的精度, 则 $(x_1 + x_2)/2$ 即为所求的根, 过程停止.

② $\left|f\left(\frac{x_1 + x_2}{2}\right)\right| \geq \varepsilon$, 根据 $f\left(\frac{x_1 + x_2}{2}\right)$ 的符号形成新的有根区间 (a_1, b_1) , 当 $f\left(\frac{x_1 + x_2}{2}\right) < 0$ 时, 取 $a_1 = \frac{x_1 + x_2}{2}, b_1 = b$, 当 $f\left(\frac{x_1 + x_2}{2}\right) > 0$ 时, 取 $a_1 = a, b_1 = \frac{x_1 + x_2}{2}$. 这时 $f(a_1)f(b_1) < 0$ 且 $(a_1, b_1) \subset (x_1, x_2), b_1 - a_1 = \frac{1}{2}(x_2 - x_1)$. 用 (a_1, b_1) 代替 (x_1, x_2) 继续上述过程, 此过程可一直进行下去 (本节函数子过程 RTBIS 中设定了一个二分的最大允许次数).

③ 结束标志.

当出现情况①时, 过程停止;

当第 n 次二分过程的有根区间 (a_n, b_n) 满足 $|b_n - a_n| = \frac{1}{2^n} (x_2 - x_1) < \delta$ (子过程中 $\delta = \text{XACC}$) 时, 过程停止;

当二分过程的次数已达到函数过程中设定的二分过程的最大允许次数时, 过程停止.

显然, 前两种结束表明成功, 后一种停止为失败.

3. 使用说明

(1) ZBRAC(FUNC, X1, X2, SUCCES)

SUCCES 逻辑型变量, 输出参数, 当 SUCCES 的值为 TRUE 时, 表示已找到了有根的区间; 当其值为 FALSE 时, 表示没找到

X1, X2 实型变量, 输入、输出参数, 开始时存放根的猜测范围 $[X_1, X_2]$, 结束时存放找到的区间, 且若 SUCCES 的值为 TRUE, 则 $[X_1, X_2]$ 即为有根区间

FUNC 函数子过程 FUNC(X), 由用户自编, 用于计算方程 $f(x) = 0$ 左端函数 $f(x)$ 的值, 其中 X 是实型变量, 输入参数

(2) ZBRAK(FX, X1, X2, N, XB1, XB2, NB)

N 整型变量, 输入参数, 存放将区间 $[X_1, X_2]$ 等分的段数

X1, X2 实型变量, 输入参数, $[X_1, X_2]$ 为函数的定义域

NB 整型变量, 输入、输出系数, 开始时存放需要搜索的根的最大数目, 结束时存放找到的 $[X_1, X_2]$ 中有根区间的个数, $0 \leq NB \leq N$

XB1 一维实型数组, 输出参数, 存放有根区间的左端点

XB2 一维实型数组, 输出参数, 存放有根区间的右端点, 且 $XB1(i)$ 与 $XB2(i)$ 配对, 即 $[XB1(1), XB2(1)], [XB1(2), XB2(2)], \dots$, 为找到的有根区间, 若 $NB > 0$

FX 函数子过程 FX(X), 由用户自编, 用于求方程 $f(x) = 0$ 左端函数 $f(x)$ 的值, 其中 X 是实型变量, 输入参数

(3) RTBIS(FUNC, X1, X2, XACC)

FUNC 函数子过程 FUNC(X), 由用户自编, 用于计算方程 $f(x) = 0$ 左端函数 $f(x)$ 的值, 其中 X 是实型变量, 输入参数

X1, X2 实型变量, 输入参数, $[X_1, X_2]$ 是已知的方程的有根区间

XACC 实型变量, 输入参数, 存放二分过程中区间长度的允许误差

4. 过程

(1) 子过程 ZBRAC.

```

SUBROUTINE zbrac(func,x1,x2,succes)
INTEGER NTRY
REAL x1,x2,func,FACTOR
EXTERNAL func
PARAMETER (FACTOR=1.6,NTRY=50)
INTEGER j
REAL f1,f2
LOGICAL succes
if(x1==x2) &
    pause 'you have to guess an initial range in zbrac'
f1=func(x1)
f2=func(x2)
succes=.true.
do j=1,NTRY
    if(f1*f2<0.) return
    if(abs(f1)<abs(f2)) then
        x1=x1+FACTOR*(x1-x2)
        f1=func(x1)
    else
        x2=x2+FACTOR*(x2-x1)
        f2=func(x2)
    endif
end do
succes=.false.
END SUBROUTINE zbrac

```

(2) 子过程 ZBRAK.

```

SUBROUTINE zbrak(fx,x1,x2,n,xb1,xb2,nb)
INTEGER n,nb
REAL x1,x2,xb1(nb),xb2(nb),fx
EXTERNAL fx
INTEGER i,nbb
REAL dx,fc,fp,x
nbb=0
x=x1
dx=(x2-x1)/n
fp=fx(x)

```

```

do i = 1, n
  x = x + dx
  fc = fx(x)
  if (fc * fp <= 0.) then
    nbb = nbb + 1
    xb1(nbb) = x - dx
    xb2(nbb) = x
    if (nbb == nb) return
  endif
  fp = fc
end do
nb = nbb
END SUBROUTINE zbrak

```

(3) 函数过程 RTBIS.

```

FUNCTION rtbis(func, x1, x2, xacc)
INTEGER JMAX
REAL rtbis, x1, x2, xacc, func
EXTERNAL func
PARAMETER (JMAX=40)
INTEGER j
REAL dx, f, fmid, xmid
fmid = func(x2)
f = func(x1)
if (f * fmid >= 0.) &
      pause 'root must be bracketed in rtbis'
if (f < 0.) then
  rtbis = x1
  dx = x2 - x1
else
  rtbis = x2
  dx = x1 - x2
endif
do j = 1, JMAX
  dx = dx * .5
  xmid = rtbis + dx
  fmid = func(xmid)

```



```

    if(fmid<=0.)rtbis=xmid
    if(abs(dx)<xacc.or.fmid==0.)return
end do
pause 'too many bisections in rtbis'
END FUNCTION rtbis

```

5. 例子

(1) 为了验证 ZBRAC, 我们采用的例子是贝塞尔函数 J_0 , 并以下列 10 个区间 (1.0, 2.0), (2.0, 3.0), ..., (10.0, 11.0) 作为初始猜测区间, 通过调用子过程 ZBRAC 来调整这些区间一直到每个区间包含一个根为止. 然后, 打印出这些区间边界值和函数 J_0 在边界点上的值, J_0 在两个边界点上的值应该符号相反. 验证程序 D10R2 如下:

```

PROGRAM D10R2
! Driver for routine ZBRAC
LOGICAL SUCCES
! USES BESSJ0
EXTERNAL BESSJ0
WRITE(*, '(/1X,T4,A,T29,A/)' ) 'Bracketing values,' &
    'Function values,'
WRITE(*, '(/1X,T6,A,T16,A,T25,A,T39,A/)' ) 'X1','X2' &
    'BESSJ0(X1)', 'BESSJ0(X2)'
DO I=1,10
    X1=FLOAT(I)
    X2=X1+1.0
    CALL ZBRAC(BESSJ0,X1,X2,SUCCES)
    IF(SUCCES) THEN
        WRITE(*, '(1X,F7.2,F10.2,3X,F12.6,2X,F12.6)' ) &
            X1,X2,BESSJ0(X1),BESSJ0(X2)
    ENDIF
END DO
END

```

计算结果如下:

```
Bracketing values;      Function values;
```

X1	X2	BESSJ0(X1)	BESSJ0(X2)
1.00	3.60	0.765198	-0.391769
2.00	3.00	0.223891	-0.260052
1.40	4.00	0.566855	-0.397150
4.00	6.60	-0.397150	0.274043
5.00	6.00	-0.177597	0.150645
4.40	7.00	-0.342257	0.300079
7.00	9.60	0.300079	-0.208979
8.00	9.00	0.171651	-0.090334
7.40	10.00	0.278596	-0.245936
10.00	12.60	-0.245936	0.162607

(2) 子过程 ZBRAK 和子过程 ZBRAC 非常相似,只不过 ZBRAK 先取定一个区间,然后把它分为 N 个相等的部分,随后来确定包含根的子区间. 验证程序 D10R3 寻找贝塞尔函数 J_0 在 $X_1 = 1.0$ 和 $X_2 = 50.0$ 之间的根,在子过程 ZBRAK 中将该区间分为 100 个部分($N=100$). 执行子过程 ZBRAK,结果发现 J_0 在区间(1,50)上有 16 个根,并打印出这些根所在区间上两端点上的函数 J_0 的值,显然,函数 J_0 在这些区间两端点上的值的符号必定相反. 验证程序 D10R3 如下:

```

PROGRAM D10R3
  ! Driver for routine ZBRAK
  EXTERNAL BESSJ0
  ! USES BESSJ0
  PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
  DIMENSION XB1(NBMAX),XB2(NBMAX)
  NB=NBMAX
  CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
  WRITE(*, '(1X,A/)' ) 'Brackets for roots of BESSJ0:'
  WRITE(*, '(1X,T15,A,T27,A,T39,A,T51,A/)' ) 'Lower', &
    'upper', 'F(lower)', 'F(upper)'
  DO I=1,NB
    WRITE(*, '(1X,A,I2,2(1X,2F12.4))' ) 'Root', I, XB1(I), &
      XB2(I), BESSJ0(XB1(I)), BESSJ0(XB2(I))
  END DO
END

```

计算结果如下:

Brackets for roots of BESSJ0:

	Lower	upper	F(lower)	F(upper)
Root 1	1.9800	2.4700	0.2354	-0.0334
Root 2	5.4100	5.9000	-0.0378	0.1220
Root 3	8.3500	8.8400	0.0826	-0.0497
Root 4	11.7800	12.2700	-0.0027	0.1049
Root 5	14.7200	15.2100	0.0436	-0.0564
Root 6	17.6600	18.1500	-0.0759	0.0148
Root 7	21.0900	21.5800	0.0211	-0.0619
Root 8	24.0300	24.5200	-0.0516	0.0269
Root 9	27.4600	27.9500	0.0051	-0.0665
Root10	30.4000	30.8900	-0.0336	0.0363
Root11	33.3400	33.8300	0.0583	-0.0074
Root12	36.7700	37.2600	-0.0193	0.0440
Root13	39.7100	40.2000	0.0432	-0.0178
Root14	43.1400	43.6300	-0.0073	0.0504
Root15	46.0800	46.5700	0.0303	0.0265
Root16	49.0200	49.5100	-0.0509	0.0031

(3) 为了验证子过程 RTBIS, 在验证程序 D10R4 中采用的例子是寻找贝塞尔函数 J_0 在 $X_1=1.0$ 和 $X_2=50.0$ 之间的所有的根. 这里 XACC 取为 10^{-6} . 在 D10R4 中先调用子过程 ZBRAK 确定了根所在的小区间, 然后再调用函数子过程 RTBIS, 确定在区间 (X_1, X_2) 上所有的根 ROOT. 验证程序 D10R4 如下:

```

PROGRAM D10R4
! Driver for routine RTBIS
EXTERNAL BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)
NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '(1X,A)') 'Roots of BESSJ0:'
WRITE(*, '(1X,T17,A,T31,A/))' 'x', 'F(x)'
DO I=1,NB
    XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
    ROOT=RTBIS(BESSJ0,XB1(I),XB2(I),XACC)

```

```

WRITE(*,'(1X,A,I2,2X,F12.6,E16.4)')&
      'Root',I,ROOT,BESSJ0(ROOT)
END DO
END

```

计算结果如下:

Roots of BESSJ0:

	x	F(x)
Root 1	2.404827	-0.5610E-06
Root 2	5.520077	-0.4607E-06
Root 3	8.653729	-0.4568E-06
Root 4	11.791533	-0.2996E-06
Root 5	14.930930	-0.2501E-05
Root 6	18.071054	-0.1845E-05
Root 7	21.211639	-0.4253E-06
Root 8	24.352465	-0.1160E-05
Root 9	27.493486	-0.1058E-05
Root10	30.634581	-0.3774E-05
Root11	33.775833	-0.1991E-05
Root12	36.917088	-0.1210E-05
Root13	40.058434	-0.1179E-05
Root14	43.199783	-0.8256E-06
Root15	46.341213	-0.3099E-05
Root16	49.482594	-0.1676E-05

10.3 割线法和试位法

1. 功能

设函数 $f(x)$ 定义在区间 $[a, b]$ 上, $f(x)$ 连续且满足 $f(a)f(b) < 0$, 求函数 $f(x)$ 在 $[a, b]$ 中的一个根.

子过程 RTFLSP 用试位法求 $f(x)$ 的根; 子过程 RTSEC 用割线法求 $f(x)$ 在 $[a, b]$ 上的根.

试位法和割线法比二分法快, 对于充分光滑的函数, 割线法的收敛阶为 1.618, 而试位法常是超线性的, 比割线法稍慢, 但割线法可能不收敛.

2. 方法

迭代公式均为割线法的公式

$$x_{i+1} = x_i + \frac{x_{i-1} - x_i}{f(x_i) - f(x_{i-1})} f(x_i)$$

但代换规律有些不同.

试位法的代换规律是:

首先,检查区间端点值哪个小于零,将对应函数值小于零的点记为 x_i ,另一点记为 x_{i-1} ,即若 $f(a) < 0$, $f(b) > 0$,则 $x_i = a$, $x_{i-1} = b$.

用上面迭代公式计算出 x_{i+1} , $f(x_{i+1})$.

(1) 若 $f(x_i) f(x_{i+1}) > 0$, 则用 $(x_{i+1}, f(x_{i+1}))$, 代替 $(x_i, f(x_i))$, $(x_{i-1}, f(x_{i-1}))$ 仍为 $(x_{i-1}, f(x_{i-1}))$, $\text{DEL} \leftarrow x_i - x_{i+1}$.

(2) 若 $f(x_i) f(x_{i+1}) < 0$, 则用 $(x_{i+1}, f(x_{i+1}))$ 代替 $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$ 仍为 $(x_i, f(x_i))$, $\text{DEL} \leftarrow x_{i+1} - x_{i-1}$.

若 $|\text{DEL}| < \varepsilon$ 或在某次迭代过程中 $f(x_{i+1}) = 0$, 迭代停止; 否则重复上面迭代过程.

割线法的代换规律是:

首先用两端点函数值绝对值较大者的对应点作为 x_{i-1} , 较小者的对应点作为 x_i , 即若 $|f(a)| < |f(b)|$, 则 $x_i \leftarrow a$, $x_{i-1} \leftarrow b$.

用上面迭代公式得出 x_{i+1} , $f(x_{i+1})$.

记

$$DX = \frac{x_{i-1} - x_i}{f(x_i) - f(x_{i-1})} \cdot f(x_i)$$

若 $|DX| < \varepsilon$ 或 $f(x_{i+1}) = 0$ 则停止计算, 此时已找到了一个满足精度要求的根; 否则, 用 $(x_i, f(x_i))$ 代替 $(x_{i-1}, f(x_{i-1}))$, $(x_{i+1}, f(x_{i+1}))$ 代替 $(x_i, f(x_i))$, 用上面迭代公式继续进行迭代.

3. 使用说明

(1) RTFLSP(FUNC, X1, X2, XACC)

(2) RTSEC(FUNC, X1, X2, XACC)

X1, X2 实型变量, 输入参数, 存放求根区间端点值

XACC 实型变量, 输入参数, 表示收敛的精度

FUNC 函数子过程 FUNC(X), 由用户自编, 用于求方程 $f(x) = 0$ 的左端函数 $f(x)$ 的值, 其中 X 为实型变量, 输入参数

4. 过程

(1) 函数过程 RTFLSP.

```

FUNCTION rtflsp(func,x1,x2,xacc)
INTEGER MAXIT
REAL rtflsp,x1,x2,xacc,func
EXTERNAL func
PARAMETER (MAXIT=30)
INTEGER j
REAL del,dx,f,fh,fl,swap,xh,xl
fl=func(x1)
fh=func(x2)
if(fl*fh>0.) pause 'root must be bracketed in rtflsp'
if(fl<0.)then
    xl=x1
    xh=x2
else
    xl=x2
    xh=x1
    swap=fl
    fl=fh
    fh=swap
endif
dx=xh-xl
do j=1,MAXIT
    rtflsp=xl+dx*f/(f-fh)
    f=func(rtflsp)
    if(f<0.) then
        del=xl-rtflsp
        xl=rtflsp
        fl=f
    else
        del=xh-rtflsp
        xh=rtflsp
        fh=f
    endif
    dx=xh-xl
    if(abs(del)<xacc. or. f==0.)return
end do
pause 'rtflsp exceed maximum iterations'

```

```
END FUNCTION rtflsp
```

(2) 函数过程 RTSEC.

```
FUNCTION rtsec(func,x1,x2,xacc)
INTEGER MAXIT
REAL rtsec,x1,x2,xacc,func
EXTERNAL func
PARAMETER (MAXIT=30)
INTEGER j
REAL dx,f,fl,swap,xl
fl=func(x1)
f=func(x2)
if(abs(fl)<abs(f)) then
    rtsec=x1
    x1=x2
    swap=fl
    fl=f
    f=swap
else
    x1=x1
    rtsec=x2
endif
do j=1,MAXIT
    dx=(x1-rtsec)*f/(f-fl)
    x1=rtsec
    fl=f
    rtsec=rtsec+dx
    f=func(rtsec)
    if(abs(dx)<xacc.or.f==0.) return
end do
pause 'rtsec exceed maximum iterations'
END FUNCTION rtsec
```

5. 例子

(1) 为了验证试位法的函数过程 RTFLSP,我们采用的例子是零阶贝塞尔函数 $J_0(x)$.

验证程序 D10R5 如下:

```

PROGRAM D10R5
! Driver for routine RTFLSP
EXTERNAL BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)
NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '(/1X,A)') 'Roots of BESSJ0;'
WRITE(*, '(/1X,T18,A,T31,A/)' ) 'x','F(x)'
DO I=1,NB
    XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
    ROOT=RTFLSP(BESSJ0,XB1(I),XB2(I),XACC)
    WRITE(*, '(1X,A,I2,2X,F12.6,E16.4)' )&
        'Root',I,ROOT,BESSJ0(ROOT)
END DO
END

```

计算结果如下:

Roots of BESSJ0:

	x	F(x)
Root 1	2.404826	-0.6592E-07
Root 2	5.520078	0.2606E-07
Root 3	8.653728	0.6107E-07
Root 4	11.791534	-0.7786E-07
Root 5	14.930918	0.5948E-07
Root 6	18.071064	-0.5491E-07
Root 7	21.211637	-0.9482E-07
Root 8	24.352472	0.7354E-07
Root 9	27.493479	0.1030E-06
Root10	30.634607	0.7591E-07
Root11	33.775818	0.1037E-06
Root12	36.917095	-0.2078E-06
Root13	40.058426	-0.2176E-06
Root14	43.199791	0.1006E-06
Root15	46.341187	0.3056E-07
Root16	49.482609	0.5503E-07

(2) 为了验证割线法的函数过程 RTSEC, 所采用的例子仍为零阶贝塞尔函数 $J_0(x)$, 验证程序 D10R6 如下:

```

PROGRAM D10R6
! Driver for routine RTSEC
EXTERNAL BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)
NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '(1X,A)') 'Roots of BESSJ0;'
WRITE(*, '(1X,T18,A,T31,A/)' ) 'x','F(x)'
DO I=1,NB
  XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
  ROOT=RTSEC(BESSJ0,XB1(I),XB2(I),XACC)
  WRITE(*, '(1X,A,I2,2X,F12.6,E16.4)')&
    'Root',I,ROOT,BESSJ0(ROOT)
END DO
END

```

计算结果如下:

Roots of BESSJ0:

	x	F(x)
Root 1	2.404825	0.5786E-07
Root 2	5.520078	0.2606E-07
Root 3	8.653728	0.6107E-07
Root 4	11.791534	-0.7786E-07
Root 5	14.930918	0.5948E-07
Root 6	18.071064	-0.5491E-07
Root 7	21.211637	-0.9482E-07
Root 8	24.352472	0.7354E-07
Root 9	27.493479	0.1030E-06
Root10	30.634607	0.7591E-07
Root11	33.775818	0.1037E-06
Root12	36.917095	-0.2078E-06
Root13	40.058426	-0.2176E-06
Root14	43.199791	0.1006E-06

Root15	46.341187	0.3056E-07
Root16	49.482609	0.5503E-07

10.4 布伦特(Brent)方法

1. 功能

用布伦特方法求在区间 $[a, b]$ 上两端点函数值异号的实函数方程 $f(x)=0$ 在 $[a, b]$ 内的一个实根.

本算法兼有二分法和反二次插值的优点,只要函数 $f(x)$ 在方程的有根区间内可求值,则它的收敛速度比二分法快且即使对病态函数也总能保证收敛.

2. 方法

设 $[a, b]$ 为方程 $f(x)=0$ 的一个有根区间,即 $f(a)f(b)<0$,且不妨设 $|f(b)|\leq|f(a)|$. 本算法的计算步骤是:

(1) 取 $c=a, f(c)=f(a)$.

(2) 若 $\frac{1}{2}|c-b|<\epsilon$ 或 $f(b)=0$,则 b 点即为满足精度要求的根,过程结束;否则,执行(3).

(3) 若 $a=c$,则用二分法求一根的新近似值 x ;若 $a\neq c$,则用 $(a, f(a)), (b, f(b)), (c, f(c))$ 作反二次插值,且让 $y=0$,则得出一个根的新近似值:

$$x = b + P/Q$$

其中

$$P = S[T(R - T)(c - b) - (1 - R)(b - a)]$$

$$Q = (T - 1)(R - 1)(S - 1)$$

$$R \equiv f(b)/f(c), \quad S \equiv f(b)/f(a), \quad T \equiv f(a)/f(c)$$

用 x 代替原来的 b ,将原来的 b 作为新的 a .

在上述过程中, b 总是当前的根的最好近似值, P/Q 为对 b 的一个微小修正值.当修正值 P/Q 使新的根的近似值 x “跑”出了区间 $[c, b]$,以及当有根区间用反二次插值计算衰减很慢时,用二分法求根的近似值.

返回(2),重复执行上面步骤.

只有当找到了满足精度要求的根或迭代次数已达到给定的最大迭代次数时,上面过程停止.

3. 使用说明

ZBRENT(FUNC,X1,X2,TOL)

X1,X2 实型变量,输入参数,存放求根区间端点,要求 $f(X_1)f(X_2)<0$,即 $[X_1,X_2]$ 一定为有根区间

TOL 实型变量,输入参数,表示收敛的精度

FUNC 函数子过程 FUNC(X),由用户自编,用于计算方程左端函数的值,其中 X 为实型变量,输入参数

4. 过程

函数过程 ZBRENT.

```

FUNCTION zbrent(func,x1,x2,tol)
INTEGER ITMAX
REAL zbrent,tol,x1,x2,func,EPS
EXTERNAL func
PARAMETER (ITMAX=100,EPS=3.e-8)
INTEGER iter
REAL a,b,c,d,e,fa,fb,fc,p,q,r,s,tol1,xm
a=x1
b=x2
fa=func(a)
fb=func(b)
if((fa>0. .and. fb>0. ).or. (fa<0. .and. fb<0. ))&
    pause 'root must be bracketed for zbrent'
c=b
fc=fb
do iter=1,ITMAX
    if((fb>0. .and. fc>0. ).or. (fb<0. .and. fc<0. )) then
        c=a
        fc=fa
        d=b-a
        e=d
    endif
    if(abs(fc)<abs(fb)) then
        a=b
        b=c
    
```

```

c=a
fa=fb
fb=fc
fc=fa
endif
tol1=2. * EPS * abs(b)+0.5 * tol
xm=.5 * (c-b)
if(abs(xm)<=tol1.or.fb==0.)then
    zbrent=b
    return
endif
if(abs(e)>=tol1.and.abs(fa)>abs(fb)) then
    s=fb/fa
    if(a==c) then
        p=2. * xm * s
        q=1.-s
    else
        q=fa/fc
        r=fb/fc
        p=s*(2. * xm * q*(q-r)-(b-a)*(r-1.))
        q=(q-1.)*(r-1.)*(s-1.)
    endif
    if(p>0.) q=-q
    p=abs(p)
    if(2. * p<min(3. * xm * q-abs(tol1 * q),abs(c * q))) then
        e=d
        d=p/q
    else
        d=xm
        e=d
    endif
else
    d=xm
    e=d
endif
a=b
fa=fb

```

```

if(abs(d)>tol1) then
    b=b+d
else
    b=b+sign(tol1,xm)
endif
fb=func(b)
end do
pause 'zbrent exceeding maximum iterations'
zbrent=b
END FUNCTION zbrent

```

5. 例子

为了验证 ZBRENT, 我们采用的例子是零阶贝塞尔函数 $J_0(x)$. 先调用本章 10.2 节中的子过程 ZBRAK 确定所有根所在的区间, 然后再调用函数过程 ZBRENT, 确定全部根的精确位置. 验证程序 D10R7 如下:

```

PROGRAM D10R7
! Driver for routine ZBRENT
EXTERNAL BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)
NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '( /1X,A)') 'Roots of BESSJ0;'
WRITE(*, '( /1X,T18,A,T31,A/ )') 'x','F(x)'
DO I=1,NB
    TOL=(1.0E-6)*(XB1(I)+XB2(I))/2.0
    ROOT=ZBRENT(BESSJ0,XB1(I),XB2(I),TOL)
    WRITE(*, '(1X,A,I2,2X,F12.6,E16.4)') &
        'Root',I,ROOT,BESSJ0(ROOT)
END DO
END

```

计算结果如下:

```

Roots of BESSJ0:
           x           F(x)
Root 1      2.401826    -0.6592E-07

```

Root 2	5.520078	0.2606E-07
Root 3	8.653728	0.6107E-07
Root 4	11.791535	0.1438E-06
Root 5	14.930918	0.5948E-07
Root 6	18.071068	0.6613E-06
Root 7	21.211637	-0.9482E-07
Root 8	24.352474	0.3820E-06
Root 9	27.493481	-0.1873E-06
Root10	30.634600	-0.1024E-05
Root11	33.775822	-0.4201E-06
Root12	36.917091	-0.7087E-06
Root13	40.058430	-0.6985E-06
Root14	43.199791	0.1006E-06
Root15	46.341183	0.4777E-06
Root16	49.482609	0.5503E-07

10.5 牛顿-拉斐森(Newton-Raphson)法

1. 功能

本节有两个函数过程,它们的功能分别是:

(1) RTNEWT,用牛顿法求端点函数值异号的连续函数方程 $f(x)=0$ 在区间 $[a,b]$ 内的一个近似实根.对单根和较好的迭代初值,方法是二阶收敛的.若初值选择的不好,则可能不收敛而停止运算.

(2) RTSAFE,用牛顿法和二分法相结合的方法求连续函数方程 $f(x)=0$ 在有限区间 $[a,b]$ 内的一个近似实根.此方法具有防止故障的特性,若迭代次数足够大,则总能求出一个近似实根而不会出现迭代过程中未得出结果而中断计算的情况.

这两个函数子过程均要利用根附近的函数值和导数值的信息.

2. 方法

对 $f(x)=0$ 在 $x=x_i$,邻域作泰勒展开,略去二次和二次以上的项,得

$$f(x) \approx f(x_i) + f'(x_i)(x - x_i) = 0$$

若 $f'(x_i) \neq 0$,则得牛顿迭代公式

$$x_{i+1} = x_i - f(x_i)/f'(x_i), \quad i = 1, 2, \dots$$

在子过程 RTNEWT 中,取 $x_1 = (a+b)/2$, 其中 $[a, b]$ 为 $f(x)=0$ 的一个有根区间. 设已求得 x_i , 用上面迭代公式求 x_{i+1} . 若满足下面条件:

(1) $|x_{i+1} - x_i| < \epsilon$, 则停止迭代, x_{i+1} 即为 $f(x)=0$ 在 $[a, b]$ 中的满足精度要求的根.

(2) 若 x_{i+1} 已不在 $[a, b]$ 内或迭代次数超过给定的最大迭代次数, 则停止迭代, 此时迭代失败.

在子过程 RTSAFE 中, 取 $x_1 = (a+b)/2$, 其中 $[a, b]$ 为 $f(x)=0$ 的一个有根区间. 设已求得 x_i , 用牛顿迭代公式求 x_{i+1} , 但若 $x_{i+1} \notin [a, b]$ 或区间长度缩小得不够快时, 改用二分法求 x_{i+1} . 迭代一直进行到迭代次数超过给定的最大迭代次数或已求得满足精度要求的根为止.

3. 使用说明

(1) RTNEWT(FUNCD, X1, X2, XACC)

(2) RTSAFE(FUNCD, X1, X2, XACC)

X1, X2 实型变量, 输入参数, 有根区间的两个端点, 满足 $f(X_1) \cdot f(X_2) < 0$

XACC 实型变量, 输入参数, 表示收敛的精度

FUNCD 子过程 FUNCD(X, FN, DF), 由用户自编, 其功能是计算所给方程 $f(x)=0$ 左端函数在 x 处的值及其导数值, 并把它们分别放入 FN 和 DF 中, X 为实型变量, 输入参数

4. 过程

(1) 函数过程 RTNEWT.

```
FUNCTION rtnewt(funcd, x1, x2, xacc)
```

```
INTEGER JMAX
```

```
REAL rtnewt, x1, x2, xacc
```

```
EXTERNAL funcd
```

```
PARAMETER (JMAX=20)
```

```
INTEGER j
```

```
REAL df, dx, f
```

```
rtnewt = .5 * (x1 + x2)
```

```
do j=1, JMAX
```

```
    call funcd(rtnewt, f, df)
```

```
    dx = f/df
```

```
    rtnewt = rtnewt - dx
```

```

    if((x1-rtnewt)*(rtnewt-x2)<0.) pause
! rtnewt jumped out of brackets'
    if(abs(dx)<xacc) return
end do
pause 'rtnewt exceeded maximum iterations'
END FUNCTION rtnewt

```

(2) 函数过程 RTSAFE.

```

FUNCTION rtsafe(funcd,x1,x2,xacc)
INTEGER MAXIT
REAL rtsafe,x1,x2,xacc
EXTERNAL funcd
PARAMETER (MAXIT=100)
INTEGER j
REAL df,dx,dxold,f,fh,fl,temp,xh,xl
call funcd(x1,fl,df)
call funcd(x2,fh,df)
if((fl>0. .and. fh>0. ), or. (fl<0. .and. fh<0. ))&
    pause 'root must be bracketed in rtsafe'
if(fl==0. ) then
    rtsafe=x1
    return
else if(fh==0. ) then
    rtsafe=x2
    return
else if(fl<0. ) then
    xl=x1
    xh=x2
else
    xh=x1
    xl=x2
endif
rtsafe=.5*(x1+x2)
dxold=abs(x2-x1)
dx=dxold
call funcd(rtsafe,f,df)
do j=1,MAXIT

```



```

if(((rtsafe-xh)*df-f)*((rtsafe-xl)*df-f)>0.&
    .or.abs(2.*f)>abs(dxold*df)) then
    dxold=dx
    dx=0.5*(xh-xl)
    rtsafe=xl+dx
    if(xl==rtsafe) return
else
    dxold=dx
    dx=f/df
    temp=rtsafe
    rtsafe=rtsafe-dx
    if(temp==rtsafe) return
endif
if(abs(dx)<xacc) return
call funcd(rtsafe,f,df)
if(f<0.) then
    xl=rtsafe
else
    xh=rtsafe
endif
end do
pause 'rtsafe exceeding maximum iterations'
END FUNCTION rtsafe

```

5. 例子

验证程序 D10R8 和 D10R9 分别验证了函数过程 RTNEWT 和函数过程 RTSafe, 采用的例子为零阶贝塞尔函数. 函数过程 FUNCD 返回了给定 x 点上的函数值和导数值. 计算函数值 $J_0(x)$ 和其导数值 $-J_1(x)$ 的子过程都能很方便地从第 4 章查到.

(1) 验证程序 D10R8 如下:

```

PROGRAM D10R8
! Driver for routine RTNEWT
EXTERNAL FUNCD,BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)

```

```

NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '(1X,A)') 'Roots of BESSJ0;'
WRITE(*, '(1X,T18,A,T31,A/)') 'x', 'F(x)'
DO I=1,NB
    XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
    ROOT=RTNEWT(FUNCD,XB1(I),XB2(I),XACC)
    WRITE(*, '(1X,A,I2,2X,F12.6,E16.4)') &
        'Root',I,ROOT,BESSJ0(ROOT)
END DO
END PROGRAM
SUBROUTINE FUNCD(X,FN,DF)
! USES BESSJ0,BESSJ1
    FN=BESSJ0(X)
    DF=-BESSJ1(X)
END SUBROUTINE FUNCD

```

计算结果如下：

Roots of BESSJ0:

	x	F(x)
Root 1	2.404825	0.5786E-07
Root 2	5.520078	0.2606E-07
Root 3	8.653728	0.6107E-07
Root 4	11.791534	-0.7786E-07
Root 5	14.930918	0.5948E-07
Root 6	18.071064	-0.5491E-07
Root 7	21.211637	-0.9482E-07
Root 8	24.352472	0.7354E-07
Root 9	27.493479	0.1030E-06
Root10	30.634607	0.7591E-07
Root11	33.775818	0.1037E-06
Root12	36.917095	-0.2078E-06
Root13	40.058426	-0.2176E-06
Root14	43.199791	0.1006E-06
Root15	46.341187	0.3056E-07
Root16	49.482609	0.5503E-07

(2) 验证程序 D10R9 如下:

```

PROGRAM D10R9
! Driver for routine RTSAFE
EXTERNAL FUNCD,BESSJ0
! USES BESSJ0,ZBRAK
PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
DIMENSION XB1(NBMAX),XB2(NBMAX)
NB=NBMAX
CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
WRITE(*, '(1X,A)') 'Roots of BESSJ0;'
WRITE(*, '(1X,T18,A,T31,A/)') 'x','F(x)'
DO I=1,NB
    XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
    ROOT=RTSAFE(FUNCD,XB1(I),XB2(I),XACC)
    WRITE(*, '(1X,A,I2,2X,F12.6,E16.4)') &
        'Root',I,ROOT,BESSJ0(ROOT)
END DO
END PROGRAM
SUBROUTINE FUNCD(X,FN,DF)
! USES BESSJ0,BESSJ1
FN=BESSJ0(X)
DF=-BESSJ1(X)
END SUBROUTINE FUNCD

```

计算结果如下:

	x	F(x)
Roots of BESSJ0;		
Root 1	2.404825	0.5786E-07
Root 2	5.520078	0.2606E-07
Root 3	8.653728	0.6107E-07
Root 4	11.791534	-0.7786E-07
Root 5	14.930918	0.5948E-07
Root 6	18.071064	-0.5491E-07
Root 7	21.211637	-0.9482E-07
Root 8	24.352472	0.7354E-07
Root 9	27.493479	0.1030E-06
Root10	30.634607	0.7591E-07
Root11	33.775818	0.1037E-06

Root12	36.917095	-0.2078E-06
Root13	40.058426	-0.2176E-06
Root14	43.199791	0.1006E-06
Root15	46.341187	0.3056E-07
Root16	49.482609	0.5503E-07

10.6 求复系数多项式根的拉盖尔(Laguerre)方法

1. 功能

用拉盖尔方法求复系数多项式的根. 本节有两个子过程: 子过程 LAGUER 是对任何初始近似值, 可求出复系数多项式的一个复根; 子过程 ZROOTS 是一个驱动程序, 它通过连续调用子过程 LAGUER 用收缩技术求出多项式的全部复根.

2. 方法

(1) 拉盖尔方法.

设多项式为

$$P_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n) \quad (10-1)$$

其中 x_1, x_2, \dots, x_n 为 $P_n(x)$ 的几个根, 则有

$$\begin{aligned} \ln |P_n(x)| &= \ln |x - x_1| + \ln |x - x_2| + \cdots + \ln |x - x_n| \\ \frac{d \ln |P_n(x)|}{dx} &= \frac{1}{x - x_1} + \frac{1}{x - x_2} + \cdots + \frac{1}{x - x_n} \\ &= \frac{P'_n}{P_n} \equiv G \\ - \frac{d^2 \ln |P_n(x)|}{dx^2} &= \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2)^2} + \cdots + \frac{1}{(x - x_n)^2} \\ &= \left[\frac{P'_n}{P_n} \right]^2 - \frac{P''_n}{P_n} \equiv H \end{aligned}$$

假设我们先求根 x_1 , 我们对 x_1 的当前猜想值为 x , 且设

$$a = x - x_1, \quad b = x - x_i, \quad i = 2, 3, \dots, n$$

由上面的关系式可得

$$\frac{1}{a} + \frac{n-1}{b} = G, \quad \frac{1}{a^2} + \frac{n-1}{b^2} = H$$

由此得

$$a = n / [G \pm \sqrt{(n-1)(nH - G^2)}] \quad (10-2)$$

这里分母中的符号使分母的模最大,因为平方根内的因子可能是负的,所以 a 可以是复数.

计算步骤如下:

任选 x_1 的初始近似值 x ,由式(10-2)计算 a ,若 $a=0$,或 $|a| \leq \epsilon |x-a|$,则结束迭代, x 即为 x_1 的近似值;否则将 $x-a$ 作为新的初始近似值,进行下一次迭代.

本子过程中设立了一个逻辑变量 POLISH,正常使用,其值输入为 FALSE,按上述过程执行,当其值输入为 TRUE 时,执行过程为:

任选 x_1 的初始近似值 ξ_1 ,由 a 的表达式(10-2)求对 ξ_1 的修正值 a_1 ,这样就完成了一次迭代.

将 $\xi_2 = \xi_1 - a$ 作为下一次迭代的初始值,得 a_2 ,设这样已求得了 a_i ,若 $i > 9$ 且 $|a_i| \geq |a_{i+1}|$,则停止迭代, $x_1 = \xi_i$;否则将 $\xi_{i+1} = \xi_i - a_i$ 作为新的初始值,重复上面过程.

(2) 设多项式为

$$P(x) = (x - x_1)(x - x_2) \cdots (x - x_n)$$

要求 $P(x)$ 的全部复根 $x_i, i=1,2,\dots,n$,计算步骤如下:

① 取 $x=0+0i$ 为 x_n 的初始近似值,用拉盖尔方法求 a ,若 $a=0$ 则结束, $x_n=0$;否则将 $x-a$ 作为 x_n 的新的初始近似值,这样继续重复求 a 的过程,直到 $a=0$ 或 $|a| \leq \epsilon |x-a|$ 为止,最后求得 x_n .

② 求多项式 $Q(x) = P(x)/(x - x_n)$.

对 $Q(x)$ 重复步骤①得 $P(x)$ 的另一根 x_{n-1} .

③ 设已求得了 $P(x)$ 的根 $x_n, x_{n-1}, \dots, x_{n-i}$,对多项式

$$q(x) = P(x)/[(x - x_n)(x - x_{n-1}) \cdots (x - x_{n-i})]$$

重复步骤①,则可求得 $x_{n-i-1}, i=0,1,2,\dots,n-1$.

由步骤①、②、③求得了 $P(x)$ 的几个近似根.子过程 ZROOTS 中也设立了一个逻辑变量 POLISH,若 POLISH 的值输入为真,则通过调用子过程 LAGUER,对 $P(x)$ 以上面求得的几个根的近似值为初始近似值对 $P(x)$ 再一次进行修正,然后将修正后的几个根的近似值按其实部的大小从小到大排列为 $ROOTS(j), j=1,2,\dots,n$.若 POLISH 的值为假,则对 n 个根的近似值(由步骤①、②、③得到的)不再重新修正,而对其直接分类,即按其实部的大小将其由小到大进行排列为 $ROOTS(j), j=1,2,\dots,n$.

3. 使用说明

(1) LAGUER(A,M,X,EPS,POLISH)

- M 整型变量,输入参数,多项式的阶
- A 复型数组 $A(M+1)$,输入参数,存放多项式的系数,其中 $A(1)$ 是多项式的常数项, $A(M+1)$ 为多项式中 x 的次数最高项的系数
- X 复型变量,输入、输出参数,开始时存放根的初始近似值,结束时存放一个根的最后结果
- EPS 实型变量,输入参数,表示收敛的相对误差精度
- POLISH 逻辑型变量,输入参数,对于正常的使用其值输入为 FALSE,具体见方法中

(2) ZROOTS(A,M,ROOTS,POLISH)

- M 整型变量,输入参数,多项式的阶
- A 复型数组 $A(M+1)$,输入参数,存放多项式的系数,其中 $A(1)$ 为多项式的常数项, $A(M+1)$ 为多项式中 x 的次数最高项的系数
- ROOTS 复型数组 $ROOTS(M)$,输出参数,存放满足精度要求的多项式的 M 个根
- POLISH 逻辑型变量,输入参数,若其值为真,则对用劈因子法求得的 M 个根再修正一次;若其值为假,则不再重新修正
- 本子过程要调用子过程 LAGUER.

4. 过程

(1) 子过程 LAGUER.

```

SUBROUTINE laguer(a,m,x,eps,polish)
COMPLEX a,x,dx,x1,b,d,f,g,h,sq,gp,gm,g2,zero
1 dimension a(m+1)
DIMENSION a(*)
LOGICAL polish
PARAMETER (zero=(0.,0.),epss=6.e-8,maxit=100)
INTEGER iter,j
dxold=cabs(x)
do iter=1,maxit
    b=a(m+1)

```

```

err=cabs(b)
d=zero
f=zero
abx=cabs(x)
do j=m,1,-1
    f=x*f+d
    d=x*d+b
    b=x*b+a(j)
    err=cabs(b)+abx*err
end do
err=epss*err
if(cabs(b)<=err) then
    dx=zero
    return
else
    g=d/b
    g2=g*g
    h=g2-2.*f/b
    sq=csqrt((m-1)*(m*h-g2))
    gp=g+sq
    gm=g-sq
    if(cabs(gp)<cabs(gm)) gp=gm
    dx=m/gp
endif
x1=x-dx
if(x==x1) return
x=x1
cdx=cabs(dx)
if(iter>6.and.cdx>=dxold) return
dxold=cdx
if(.not.polish) then
    if(cabs(dx)<=eps*cabs(x)) return
endif
end do
PAUSE 'too many iterations'
END SUBROUTINE laguer

```

(2) 子过程 ZROOTS.

```

SUBROUTINE zroots(a,m,roots,polish)
INTEGER m,MAXM
REAL EPS
COMPLEX a(m+1),roots(m)
LOGICAL polish
PARAMETER (EPS=1.e-6,MAXM=101)
! USES laguer
INTEGER i,j,jj,its
COMPLEX ad(MAXM),x,b,c
do j=1,m+1
    ad(j)=a(j)
end do
do j=m,1,-1
    x=cmplx(0.,0.)
    call laguer(ad,j,x,eps,.false.)
    if(abs(aimag(x))<=2.*EPS**2*abs(real(x)))&
        x=cmplx(real(x),0.)
    roots(j)=x
    b=ad(j+1)
    do jj=j,1,-1
        c=ad(jj)
        ad(jj)=b
        b=x*b+c
    end do
end do
if (polish) then
    do j=1,m
        call laguer(a,m,roots(j),eps,.true.)
    end do
endif
do j=2,m
    x=roots(j)
    do i=j-1,1,-1
        if(real(roots(i))<=real(x)) exit
        roots(i+1)=roots(i)
    end do
    if(real(roots(i))>real(x)) i=0

```



```

    roots(i-1)=x
end do
END SUBROUTINE zroots

```

5. 例子

(1) 子过程 LAGUER 能求出 M 次复系数多项式的根, 该多项式的 $M+1$ 个系数在验证程序 D10R10 里存储在数组 A 中. 我们所采用的例子为

$$F(x) = x^4 - (1 + 2i)x^2 + 2i$$

这个多项式的四个根为: $x=1.0$, $x=-1.0$, $x=1+i$ 和 $x=-(1+i)$. 验证程序 D10R10 如下:

```

PROGRAM D10R10
! Driver for routine LAGUER
PARAMETER(M=4,MP1=M+1,NTRY=21,EPS=1.0E-6)
COMPLEX A(MP1),Y(NTRY),X
LOGICAL POLISH
DATA A/(0.0,2.0),(0.0,0.0),(-1.0,-2.0),(0.0,0.0),&
      (1.0,0.0)/
WRITE(*, '(1X,A)') 'Roots of polynomial x^4 - (1+2i) * x^2 + 2i'
WRITE(*, '(1X,T16,A,T29,A/)') 'Real', 'Complex'
N=0
POLISH=.FALSE.
DO I=1,NTRY
    X=CMPLX((I-11.0)/10.0,(I-11.)/10.0)
    CALL LAGUER(A,M,X,EPS,POLISH)
    IF(N=-0) THEN
        N=1
        Y(1)=X
        WRITE(*, '(1X,I5,2F15.6)') N,X
    ELSE
        IFLAG=0
        DO J=1,N
            IF(CABS(X-Y(J))<=EPS*CABS(X)) IFLAG=1
        END DO
        IF(IFLAG==0) THEN
            N=N+1
            Y(N)=X
        END IF
    END IF
END DO

```

```

        WRITE(*,'(1X,I5,2F15.6)') N,X
    ENDIF
ENDIF
END DO
END

```

计算结果如下:

Roots of polynomial $x^4 - (1+2i) * x^2 + 2i$

	Real	Complex
1	-1.000000	-1.000000
2	-1.000000	0.000000
3	1.000000	0.000000
4	1.000000	1.000000

(2) 验证子过程 ZROOTS 的程序 D10R11 中所采用的例子和上述验证 LAGUER 的例子相同. 首先采用子过程 ZROOTS 得到四个根. 然后将每个根乘以 1.01 使之粗糙化. 最后再一次采用 ZROOTS, 利用建立的逻辑参数 POLISH 达到 TRUE 而修正了粗糙的根. 验证程序 D10R11 如下:

```

PROGRAM D10R11
! Driver for routine ZROOTS
PARAMETER(M=4,M1=M+1)
COMPLEX A(M1),X,ROOTS(M)
LOGICAL POLISH
DATA A/(0.0,2.0),(0.0,0.0),(-1.0,-2.0),(0.0,0.0),&
      (1.0,0.0)/
WRITE(*,'(/1X,A)')&
      'Roots of polynomial  $x^4 - (1+2i) * x^2 + 2i$ '
POLISH=.FALSE.
CALL ZROOTS(A,M,ROOTS,POLISH)
WRITE(*,'(/1X,A)') 'Unpolished roots:'
WRITE(*,'(1X,T10,A,T25,A,T37,A)') 'Root #','Real',&
      'Imag.'
DO I=1,M
    WRITE(*,'(1X,I11,5X,2F12.6)') I,ROOTS(I)
END DO
WRITE(*,'(/1X,A)') 'Corrupted roots:'
DO I=1,M

```

```

      ROOTS(I)=ROOTS(I)*(1.0+0.01*I)
END DO
WRITE(*, '(1X,T10,A,T25,A,T37,A)' ) 'Roots #', 'Real', &
      'Imag.'
DO I=1,M
      WRITE(*, '(1X,I11,5X,2F12.6)' ) I,ROOTS(I)
END DO
POLISH=.TRUE.
CALL ZROOTS(A,M,ROOTS,POLISH)
WRITE(*, '(/1X,A)' ) 'Polished roots:'
WRITE(*, '(1X,T10,A,T25,A,T37,A)' ) 'Root #', 'Real', &
      'Imag.'
DO I=1,M
      WRITE(*, '(1X,I11,5X,2F12.6)' ) I,ROOTS(I)
END DO
END

```

计算结果如下:

Roots of polynomial $x^4 - (1+2i) * x^2 + 2i$

Unpolished roots:

Root #	Real	Imag.
1	-1.000000	-1.000000
2	-1.000000	0.000000
3	1.000000	1.000000
4	1.000000	0.000000

Corrupted roots:

Roots #	Real	Imag.
1	-1.010000	-1.010000
2	-1.020000	0.000000
3	1.030000	1.030000
4	1.040000	0.000000

Polished roots:

Root #	Real	Imag.
1	-1.000000	-1.000000
2	-1.000000	0.000000
3	1.000000	1.000000
4	1.000000	0.000000

10.7 求实系数多项式根的贝尔斯托(Bairstou)方法

1. 功能

给定 $n-1$ 阶实系数多项式 $P(x) = \sum_{k=1}^n p_k x^{k-1}$, 用贝尔斯托法求 $P(x)$ 的二次因子 $x^2 + bx + c$. 该方法是局部二次收敛的, 它常被用于改善已求得的复根.

对大多数非病态实多项式而言, 这个方法是较好的, 它仅用实运算, 但对于有多重二次因子或近于多重二次因子的实多项式而言, 本方法不是很好的.

2. 方法

(1) 设

$$P(x) = \sum_{k=1}^n p_k x^{k-1} \quad (10-3)$$

$$P(x) = (x^2 + bx + c)Q(x) + Rx + S \quad (10-4)$$

其中

$$R = R(b, c), \quad S = S(b, c)$$

$$Q(x) = \sum_{k=1}^{n-2} q_k x^{k-1} \quad (10-5)$$

由于当 $x^2 + bx + c$ 为 $P(x)$ 的二次因子时,

$$R(b, c) = 0 \quad (10-6)$$

$$S(b, c) = 0 \quad (10-7)$$

因此, 只需求出式(10-6), (10-7)的联立根. 贝尔斯托方法就是用牛顿法求解联立方程(10-6)与(10-7).

(2) 为用牛顿法, 需要计算方程组(10-6)和(10-7)的雅可比矩阵, 即要计算 $\partial R / \partial b, \partial R / \partial c, \partial S / \partial b, \partial S / \partial c$, 由

$$0 = \frac{\partial P(x)}{\partial b} = xQ(x) + (x^2 + bx + c) \frac{\partial Q(x)}{\partial b} + \frac{\partial R}{\partial b} x + \frac{\partial S}{\partial b}$$

$$0 = \frac{\partial P(x)}{\partial c} = Q(x) + (x^2 + bx + c) \frac{\partial Q(x)}{\partial c} + \frac{\partial R}{\partial c} x + \frac{\partial S}{\partial c}$$

有

$$-xQ(x) = (x^2 + bx + c) \frac{\partial Q(x)}{\partial b} + \frac{\partial R}{\partial b} x + \frac{\partial S}{\partial b} \quad (10-8)$$

$$-Q(x) = (x^2 + bx + c) \frac{\partial Q(x)}{\partial c} + \frac{\partial R}{\partial c} x + \frac{\partial S}{\partial c} \quad (10-9)$$

于是 $\frac{\partial R}{\partial b}, \frac{\partial R}{\partial c}$ 是 $-xQ(x)$ 除以二次因子 x^2+bx+c 的线性余式的两个系数, $\frac{\partial S}{\partial b}, \frac{\partial S}{\partial c}$ 是 $Q(x)$ 除以二次因子 x^2+bx+c 所得线性余式的两个系数.

(3) 算法.

① 给定初始近似二次因子 $x^2+b_0x+c_0, k=0$.

② 求 $n-3$ 阶多项式及 R_k, S_k , 使

$$P(x) = (x^2 + b_kx + c_k)Q_k(x) + R_kx + S_k$$

③ 求 $-xQ(x)$ 除以 $x^2+b_kx+c_k$ 的余式 Rb_kx+Rc_k 与 $Q_k(x)$ 除以 $x^2+b_kx+c_k$ 的余式 Sb_kx+S_k .

④ 解方程组

$$J_k\delta_k = -T_k$$

其中

$$J_k = \begin{bmatrix} Rb_k & Rc_k \\ Sb_k & S_k \end{bmatrix}, \quad \delta = \begin{bmatrix} \delta b_k \\ \delta c_k \end{bmatrix}, \quad T_k = \begin{bmatrix} b_k \\ c_k \end{bmatrix}$$

⑤ 如果 $\|\delta_k\| / \|T_k\| \leq \varepsilon$ (给定的精度), 则停止; 否则, $b_k \leftarrow b_k + \delta b_k, c_k \leftarrow c_k + \delta c_k, k \leftarrow k+1$. 转②

3. 使用说明

QROOT(P,N,B,C,EPS)

N 整型变量, 输入参数, 多项式的次数加 1

P N 个元素的一维实型数组, 输入参数, 存放多项式的系数, 其中 $P(1)$ 为多项式的常数项, $P(N)$ 为多项式的最高次项系数

EPS 实型变量, 输入参数, 要求的相对误差精度

B,C 均为实型变量, 输入、输出参数, 输入时存放多项式的二次因子 x^2+bx+c 中系数 b, c 的初始近似值, 输出时存放 b, c 的最后近似值

4. 过程

子过程 QROOT.

SUBROUTINE qroot(p,n,b,c,eps)

INTEGER n,NMAX,ITMAX

REAL b,c,eps,p(n),TINY

PARAMETER (NMAX=20,ITMAX=20,TINY=1.0e-6)

```

! USES poldiv
INTEGER iter
REAL delb,dele,div,r,rb,rc,s,sb,sc,d(3),q(NMAX),&
    qq(NMAX),rem(NMAX)
d(3)=1.
do iter=1,ITMAX
    d(2)=b
    d(1)=c
    call poldiv(p,n,d,3,q,rem)
    s=rem(1)
    r=rem(2)
    call poldiv(q,n-1,d,3,qq,rem)
    sc=-rem(1)
    rc=-rem(2)
    sb=-c * rc
    rb=sc-b * rc
    div=1. / (sb * rc-sc * rb)
    delb=(r * sc-s * rc) * div
    dele=(-r * sb+s * rb) * div
    b=b+delb
    c=c+dele
    if((abs(delb)<=eps * abs(b). or. abs(b)<TINY). and. &
        (abs(dele)<=eps * abs(c). or. abs(c)<TINY)) return
end do
pause 'too many iterations in qroot'
END SUBROUTINE qroot

```

5. 例子

为了验证子过程 QROOT,在验证程序 D10R12 中,我们采用的例子为

$$P(x) = x^6 - 6x^5 + 16x^4 - 24x^3 + 25x^2 - 18x + 10$$

调用子过程 QROOT 将得到三个二次因子中的 B 和 C , B 和 C 被打印出来,读者可将三个二次因子的积与上述多项式相比较. QROOT 中还需调用 POLDIV(见 5.2 节). 验证程序 D10R12 如下:

```

PROGRAM D10R12
! Driver for routine QROOT

```

```

PARAMETER(N=7, EPS=1.0E-6, NTRY=10, TINY=1.0E-5)
DIMENSION P(N), B(NTRY), C(NTRY)
DATA P/10.0, -18.0, 25.0, -24.0, 16.0, -6.0, 1.0/
WRITE(*, '(/1X, A)') &
    'P(x)=x^ 6-6x^ 5+16x^ 4-24x^ 3+25x^ 2-18x+10'
WRITE(*, '(1X, A)') &
    'Quadratic factors x^ 2+Bx+C'
WRITE(*, '(/1X, A, T15, A, T27, A/)') 'Factor', 'B', 'C'
NROOT=0
DO I=1, NTRY
    C(I)=0.5*I
    B(I)=-0.5*I
    CALL QROOT(P, N, B(I), C(I), EPS)
    IF(NROOT==0) THEN
        WRITE(*, '(1X, I3, 2X, 2F12.6)') &
            NROOT, B(I), C(I)
        NROOT=1
    ELSE
        NFLAG=0
        DO J=1, NROOT
            IF(ABS(B(I)-B(J))<TINY. &
                AND. ABS(C(I)-C(J))<TINY) NFLAG=1
        END DO
        IF(NFLAG==0) THEN
            WRITE(*, '(1X, I3, 2X, 2F12.6)') &
                NROOT, B(I), C(I)
            NROOT=NROOT+1
        ENDIF
    ENDIF
END DO
END

```

计算结果如下:

```

P(x)=x^ 6-6x^ 5+16x^ 4-24x^ 3+25x^ 2-18x+10
Quadratic factors x^ 2+Bx+C
Factor      B      C

```

0	0.000000	1.000000
1	-4.000000	5.000000
2	-1.999999	1.999999

10.8 非线性方程组的牛顿-拉斐森方法

1. 功能

用牛顿-拉斐森方法即牛顿法解非线性方程组

$$f_i(x_1, x_2, \dots, x_n) = 0, \quad i = 1, 2, \dots, N$$

对较好的迭代初值,它是二阶收敛的.

2. 方法

记 $\mathbf{x} = (x_1, \dots, x_N)^T$, 对 $f_i(\mathbf{x}), i = 1, 2, \dots, N$, 在 \mathbf{x} 的邻域作泰勒展开, 略去二次和二次以上的项得

$$f_i(\mathbf{x}^{(0)} + \delta \mathbf{x}^{(0)}) \approx f_i(\mathbf{x}^{(0)}) + \sum_{j=1}^N \frac{\partial f_i}{\partial x_j} \delta x_j^{(0)}$$

记

$$\alpha \equiv [\alpha_{ij}]_{N \times N}, \quad \beta \equiv (\beta_1, \beta_2, \dots, \beta_N)^T$$

$$\alpha_{ij} \equiv \partial f_i / \partial x_j, \quad \beta_i \equiv -f_i$$

若 $\det \alpha \neq 0$, 则得迭代公式

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i^{\text{old}} + \delta \mathbf{x}_i, \quad i = 1, 2, \dots, N$$

其中 $(\delta x_1, \delta x_2, \dots, \delta x_N)^T \equiv \delta \mathbf{x}$ 为线性代数方程组 $\alpha \cdot \delta \mathbf{x} = \beta$ 的解. 综上所述, 计算步骤如下:

(1) 给定根 \mathbf{x} 的初始近似 $\mathbf{x}^{(0)}$ (靠近 \mathbf{x}), 允许误差 ϵ_1, ϵ_2 , 并假定已得到第 k 次近似 $\mathbf{x}^{(k)}$.

(2) 计算

$$\alpha_j^{(k)} \equiv \partial f_i(\mathbf{x}^{(k)}) / \partial x_j, \quad i, j = 1, 2, \dots, N$$

$$\beta_i^{(k)} \equiv -f_i(\mathbf{x}^{(k)}), \quad i = 1, 2, \dots, N$$

得 $\alpha^{(k)} = [\alpha_{ij}^{(k)}]_{N \times N}$ 及 $\beta^{(k)} = (\beta_1^{(k)}, \beta_2^{(k)}, \dots, \beta_N^{(k)})^T$.

(3) 计算

$$S1 = |f_1(\mathbf{x}^{(k)})| + |f_2(\mathbf{x}^{(k)})| + \dots + |f_N(\mathbf{x}^{(k)})|$$

若 $S1 < \epsilon_1$ 则计算结束, $\mathbf{x}^{(k)}$ 作为满足精度要求的近似解; 否则, 执行(4).

(4) 用 LU 分解法求线性代数方程组

$$\alpha^{(k)} \cdot \delta x^{(k)} = \beta^{(k)}$$

得 $\delta x^{(k)} = (\delta x_1^{(k)}, \delta x_2^{(k)}, \dots, \delta x_N^{(k)})^T$.

(5) 计算 $x^{(k+1)} = x^{(k)} + \delta x^{(k)}$ 及

$$S2 = |\epsilon x_1^{(k)}| + |\delta x_2^{(k)}| + \dots + |\delta x_N^{(k)}|$$

若 $S2 < \epsilon_2$ 则计算结束, $x^{(k+1)}$ 作为满足精度要求的近似解; 否则, $k \leftarrow k+1$, 转向步骤(2)继续计算, 直到满足精度要求或迭代次数已达到给定的迭代次数为止.

3. 使用说明

MNEWT(NTRIAL, X, N, TOLX, TOLF)

NTRIAL 整型变量, 输入参数, 最大迭代次数, 应用中不要选得太大

N 整型变量, 输入参数, 方程个数

TOLX, TOLF 实型变量, 输入参数, 判别迭代收敛的正小量

X 含 N 个元素的一维实型数组, 输入、输出参数, 开始时存放根的初始近似值, 结束时存放根的最终近似值

本子过程要调用下面几个子过程:

(1) USRFUN(X, ALPHA, BETA)

由使用者自编, 其功能是计算

$$\alpha_{ij} \equiv \partial f_i(X) / \partial x_j, \quad i, j = 1, 2, \dots, N$$

$$\beta_i = -f_i(X), \quad i = 1, 2, \dots, N$$

且将它们放入 $\text{ALPHA} = [\alpha_{ij}]$ 和 $\text{BETA} = (\beta_1, \beta_2, \dots, \beta_N)^T$, 其中 X 为含 N 个元素的一维实型数组 $X = (x_1, x_2, \dots, x_N)^T$, 输入参数; ALPHA 为二维实型数组 $\text{ALPHA}(N, N)$, 输出参数; BETA 为一维实型数组 $\text{BETA}(N)$, 输出参数.

(2) LUDCMP(A, N, NP, INDX, D)

(3) LUBKSB(A, N, NP, INDX, B)

子过程(2)及(3)见 1.2 节, 其功能是 LU 分解求线性代数方程组 $Ax = B$ 的解并将其解放入 B 中.

4. 过程

子过程 MNEWT.

SUBROUTINE mnewt(ntrial, x, n, tolX, tolf)

INTEGER n, ntrial, NP

```

REAL tolf,tolx,x(n)
PARAMETER (NP=15)
! USES lubksb,ludcmp,usrfun
INTEGER i,k,indx(n)
REAL d,errf,errx,fjac(NP,NP),fvec(NP),p(NP)
do k=1,ntrial
  call usrfun(x,n,np,fjac,fvec)
  errf=0.
  do i=1,n
    errf=errf+abs(fvec(i))
  end do
  if(errf<=tolf) return
  call ludcmp(fjac,n,np,indx,d)
  call lubksb(fjac,n,np,indx,fvec)
  errx=0.
  do i=1,n
    errx=errx+abs(fvec(i))
    x(i)=x(i)+fvec(i)
  end do
  if(errx<=tolx) return
end do
END SUBROUTINE mnewt

```

5. 例子

为了验证 MNEWT,我们所取的例子为

$$\begin{aligned}
 -x_1^2 - x_2^2 - x_3^2 + x_4 &= 0 \\
 x_1^2 + x_2^2 + x_3^2 + x_4^2 - 1 &= 0 \\
 x_1 - x_2 &= 0 \\
 x_2 - x_3 &= 0
 \end{aligned}$$

在验证程序 D10R13 中,我们提供了一个子过程 USRFUN,其目的是为返回关于每个自变量的偏导数矩阵 ALPHA;还返回向量 BETA,表示方程组左边的负函数值.没有采用 MNEWT 前,读者可能会注意到 $x_1=x_2$ 和 $x_2=x_3$,计算结束后可将 MNEWT 的输出结果代入方程组进行检验.另外从 MNEWT 的输出结果可以看出一个好的开始猜测值是非常必要的,验证程序 D10R13 如下:

```

PROGRAM D10R13
1 Driver for routine MNEWT
PARAMETER(NTRIAL=5,TOLX=1.0E-6,N=4,&
          TOLF=1.0E-6,NP=15)
DIMENSION X(NP),ALPHA(NP,NP),BETA(NP)
DO KK=-1,1,2
  DO K=1,3
    XX=0.2001*K*KK
    WRITE(*, '(/1X,A,I2)')&
      'Starting vector number',K
    DO I=1,4
      X(I)=XX-0.2*I
      WRITE(*, '(1X,T5,A,I1,A,F5.2)')&
        'X(',I,')=',X(I)
    END DO
    DO J=1,NTRIAL
      CALL MNEWT(1,X,N,TOLX,TOLF)
      CALL USRFUN(X,N,NP,ALPHA,BETA)
      WRITE(*, '(/1X,T5,A,T14,A,T29,A/)' )&
        'I','X(I)','F'
      DO I=1,N
        WRITE(*, '(1X,I4,2E15.6)')&
          1,X(I),-BETA(I)
      END DO
      WRITE(*, '(/1X,A)')&
        'Press RETURN to END DO...'
      READ(*,*)
    END DO
  END DO
END DO
END PROGRAM

SUBROUTINE USRFUN(X,N,NP,ALPHA,BETA)
DIMENSION ALPHA(NP,NP),BETA(NP),X(NP)
ALPHA(1,1)=-2.0*X(1)
ALPHA(1,2)=-2.0*X(2)
ALPHA(1,3)=-2.0*X(3)

```

```

ALPHA(1,4)=1.0
ALPHA(2,1)=2.0 * X(1)
ALPHA(2,2)=2.0 * X(2)
ALPHA(2,3)=2.0 * X(3)
ALPHA(2,4)=2.0 * X(4)
ALPHA(3,1)=1.0
ALPHA(3,2)=-1.0
ALPHA(3,3)=0.0
ALPHA(3,4)=0.0
ALPHA(4,1)=0.0
ALPHA(4,2)=1.0
ALPHA(4,3)=-1.0
ALPHA(4,4)=0.0
BETA(1)=X(1) * * 2+X(2) * * 2+X(3) * * 2-X(4)
BETA(2)=-X(1) * * 2-X(2) * * 2-X(3) * * 2-X(4) * * 2+1.0
BETA(3)=-X(1)+X(2)
BETA(4)=-X(2)+X(3)
END SUBROUTINE USRFUN

```

计算结果如下:

Starting vector number 1

```

X(1)= 0.00
X(2)= 0.20
X(3)= 0.40
X(4)= 0.60

```

I	X(I)	F
1	0.682061E+00	-0.777436E+00
2	0.682061E+00	0.777771E+00
3	0.682061E+00	0.000000E+00
4	0.618183E+00	0.000000E+00

Press RETURN to END DO...

I	X(I)	F
1	0.492052E+00	-0.108310E+00
2	0.492052E+00	0.108310E+00
3	0.492052E+00	0.000000E+00
4	0.618034E+00	0.000000E+00

Press RETURN to END DO...

I	X(I)	F
1	0.455365E+00	-0.403769E-02
2	0.455365E+00	0.403772E-02
3	0.455365E+00	0.000000E+00
4	0.618034E+00	0.000000E+00

Press RETURN to END DO...

I	X(I)	F
1	0.453887E+00	-0.650706E-05
2	0.453887E+00	0.654376E-05
3	0.453887E+00	0.000000E+00
4	0.618034E+00	0.000000E+00

Press RETURN to END DO...

I	X(I)	F
1	0.453885E+00	-0.141683E-07
2	0.453885E+00	0.508738E-07
3	0.453885E+00	0.000000E+00
4	0.618034E+00	0.000000E+00

Press RETURN to END DO...

Starting vector number 2

X(1) = -0.20

X(2) = 0.00

X(3) = 0.20

X(4) = 0.40

⋮ ⋮ ⋮

第 11 章 函数的极值和最优化

本章包括：一维极小化问题的搜索方法(见 11.1~11.3 节)；多维无约束最优化算法(见 11.4~11.7 节)；线性规划的单纯形方法(见 11.8 节)。

最优化方法有用导数的方法和不用导数的方法之分。一般地，用导数的方法比不用导数(仅用函数值)的方法稍微强一些，但这种好处不一定能补偿附加的导数计算。不过，计算时，如果可以计算导数，还是应采纳用导数的方法。

在一维极小化中，若选用不用导数的方法，可先用 11.1 节中的方法找出包含极小值点的一个区间，然后再采用 11.2 节中的布伦特方法求出极小值点。若函数没有直至二阶(或低阶)的连续导数时，应该用更简单的黄金分割法(见 11.1 节)。11.3 节中提供了用导数的布伦特方法的一个变形。

在多维极小化中，应该注意存储量的要求，11.4 节中的下山单纯形方法，对函数几乎没有作任何假定，这个方法可能极慢，但在有些应用中也可能是极稳妥的，它对于当极小化计算仅是某整个问题的一附属部分时非常有用，存储要求为 n^2 阶，是一个不用导数的方法。11.5 节涉及方向集方法中的包维尔方法，是不用导数的方法，存储量是 n^2 阶。

在多维极小化中，对于用导数的方法，其一维极小化的子算法本身可以用导数的也可以是不用导数的。11.6 节中的共轭梯度法属于用导数的方法，存储量是 n 阶的。11.7 节中介绍用导数的拟牛顿法(或称变尺度方法)的一个范例——BFGS 的秩 2 算法，其存储量为 n^2 阶。

11.1 黄金分割搜索法

1. 功能

本节有两个过程，它们的功能分别是：

(1) MNBRK, 利用黄金比率和二次插值确定函数的极小值点所在区间，即用尽可能少的计算量来确定一个区间，并保证函数的极小值点在这个区间内。

(2) GOLDEN, 用黄金分割法求一元函数的极小值，此时常假设目标函数是单峰函数，且已确定了极小值点所在的区间。

2. 方法

(1) 确定极小值点所在区间.

任选区间初始点 a 和 b , 按如下步骤计算:

① 确定下降方向. 设要求 $f(x)$ 的极小值, 计算 $f(a)$ 和 $f(b)$.

若 $f(a) > f(b)$, 则下降方向为从 a 到 b , 沿 $a \rightarrow b$ 的方向按黄金比率选取一点 c ;

若 $f(a) \leq f(b)$, 将点 a 和 b 位置进行交换, 则此时下降方向仍是从 a 到 b , 沿此方向按黄金比率选取 c , 并计算 $f(c)$.

② 若 $f(b) < f(c)$, 则 $[a, c]$ 即为所求区间, 计算结束;

若 $f(b) \geq f(a)$, 由 $(a, f(a)), (b, f(b)), (c, f(c))$ 进行二次插值, 求其极小值点 u 及 $ulim = b + 1.618034(c - b)$.

③ 若 u 在 b 和 c 之间, 计算 $f(u)$, 并作判断:

若 $f(u) \leq f(c)$, 则 $[b, c]$ 即为所求, 计算结束;

若 $f(u) > f(b)$, 则 $[a, u]$ 即为所求, 计算结束;

若上面两条件均不满足, 则用黄金比率重新选点 $u = c + 1.618034(c - b)$ 并计算 $f(u)$.

④ 当 u 在 c 和 $ulim$ 之间时, 若 $f(u) > f(c)$, 则极小值点所在区间即为 $[b, u]$, 结束; 若 $f(u) \leq f(c)$, 则去掉离对应的极小值点最远的点, 此时显然为 a , 将 c 和 u 看作新的 b, c , 计算 $u = c + 1.618034(c - b)$ 及 $f(u)$, 把此时的 b, c, u 记为新的 a, b, c , 转②. 若 u 已不在 c 和 $ulim$ 之间, 则将此时的 $b, c, ulim$ 看作新的一组 a, b, c , 转②.

(2) 黄金分割法求满足精度要求的极小值.

设已知初始三点 a, b, c , 且 $f(b) < f(a)$, $f(b) < f(c)$, b 在 a 和 c 之间, 不妨设 $a < b < c$.

① 在 (a, c) 中按黄金比率再选择一点 $d \neq b$, 且若 $b - a > c - b$, 则将点 d 选在 (a, b) 中, 否则选在 (b, c) 中, 此时为说明算法, 不妨设 $b - a > c - b$, 则 d 在 (a, b) 中, 计算 $f(b), f(d)$, 转②.

② 检验区间长度是否已很小, 即若

$$\left| \frac{c - a}{|b| + |d|} \right| < \epsilon,$$

则转④, 否则转③.

③ 若 $f(b) < f(d)$, 将 d, b, c 分别看作一组新的 a, b, c , 转①, 否则, 将 a, d, b 分别看作一组新的 a, b, c , 转①.

④ 若 $f(b) < f(d)$, 则将 $f(b)$ 看作极小值的近似值, b 为极小值点, 否则将 d 作为极小值点, $f(d)$ 作为极小值.

3. 使用说明

(1) MNBRAK(AX, BX, CX, FA, FB, FC, FUNC)

AX 实型变量, 输入、输出参数, 开始时存放初始区间端点, 调用后存放极小值点所在区间的一个端点

BX 实型变量, 输入、输出参数, 开始时存放初始区间端点, 调用后存放极小值点的初始近似

CX 实型变量, 输出参数, 存放极小值点所在区间的另一个端点

FA, FB, FC 实型变量, 输出参数, 分别存放目标函数 $f(x)$ 在点 AX, BX, CX 的值

FUNC 函数子过程 FUNC(X), 用户自编, 其功能是计算目标函数值, X 为实型变量

(2) GOLDEN(AX, BX, CX, TOL, XMIN)

AX 实型变量, 输入参数, 存放极小值点所在区间的一个端点

BX 实型变量, 输入参数, 存放极小值点的初始近似

CX 实型变量, 输入参数, 存放极小值点所在区间的另一个端点

TOL 实型变量, 输入参数, 一维搜索精度

XMIN 实型变量, 输出参数, 存放极小值点的横坐标

4. 过程

(1) 子过程 MNBRAK.

```
SUBROUTINE mnbrak(ax, bx, cx, fa, fb, fc, func)
```

```
REAL ax, bx, cx, fa, fb, fc, func, GOLD, GLIMIT, TINY
```

```
EXTERNAL func
```

```
PARAMETER (GOLD=1.618034, GLIMIT=100., TINY=1.e-20)
```

```
REAL dum, fu, q, r, u, ulim
```

```
LOGICAL DONE
```

```
fa=func(ax)
```

```
fb=func(bx)
```

```
if(fb>fa) then
```

```
    dum=ax
```

```
    ax=bx
```

```
    bx=dum
```



```

    dum=fb
    fb=fa
    fa=dum
endif
cx=bx+GOLD*(bx-ax)
fc=func(cx)
do
    if(fb<fc) exit
    done=-1
    r=(bx-ax)*(fb-fc)
    q=(bx-cx)*(fb-fa)
    u=bx-((bx-cx)*q-(bx-ax)*r)/(2.*sign(max8.
                                (abs(q-r),TINY),q-r))
    ulim=bx+GLIMIT*(cx-bx)
    if((bx-u)*(u-cx)>0.) then
        fu=func(u)
        if(fu<fc) then
            ax=bx
            fa=fb
            bx=u
            fb=fu
            return
        else if(fu>fb) then
            cx=u
            fc=fu
            return
        endif
    u=cx+GOLD*(cx-bx)
    fu=func(u)
    else if((cx-u)*(u-ulim)>0.) then
        fu=func(u)
        if(fu<fc) then
            bx=cx
            cx=u
            u=cx+GOLD*(cx-bx)
            fb=fc
            fc=fu

```

```

        fu=func(u)
    endif
else if ((u-ulim) * (ulim-cx) >= 0.) then
    u=ulim
    fu=func(u)
else
    u=cx+GOLD*(cx-bx)
    fu=func(u)
endif
if (done) then
    ax=bx
    bx=cx
    cx=u
    fa=fb
    fb=fc
    fc=fu
else
    done=.false.
end if
if (.not. done) exit
end do
END SUBROUTINE mnbrak

```

(2) 函数过程 GOLDEN.

```

FUNCTION golden(ax,bx,cx,f,tol,xmin)
REAL golden,ax,bx,cx,tol,xmin,f,R,C
EXTERNAL f
PARAMETER (R=.61803399,C=1.-R)
REAL f1,f2,x0,x1,x2,x3
x0=ax
x3=cx
if(abs(cx-bx)>abs(bx-ax)) then
    x1=bx
    x2=bx+C*(cx-bx)
else
    x2=bx
    x1=bx-C*(bx-ax)

```

```

endif
f1=f(x1)
f2=f(x2)
do
  if(abs(x3-x0)<=tol*(abs(x1)+abs(x2))) exit
  if(f2<f1) then
    x0=x1
    x1=x2
    x2=R*x1+C*x3
    f1=f2
    f2=f(x2)
  else
    x3=x2
    x2=x1
    x1=R*x2+C*x0
    f2=f1
    f1=f(x1)
  endif
end do
if(f1<f2) then
  golden=f1
  xmin=x1
else
  golden=f2
  xmin=x2
endif
END FUNCTION golden

```

5. 例子

(1) 子过程 MNBRAK 求出一给定函数极小值所在的区间. 当给定自变量坐标值 AX 和 BX 后, 子过程将能发现包括有极小值的三个新值 AX, BX, CX . FA, FB, FC 是这些点上的函数值. 验证程序 D11R1 中采用的例子是零阶贝塞尔函数. 选取一系列区间长度为 0.5 的值 AX, BX ; 随后 MNBRAK 将找出包含有 J_0 的极小值的区间. 验证程序 D11R1 如下:

```

PROGRAM D11R1
! Driver for routine MNBRAK

```

```

EXTERNAL BESSJ0
! USES BESSJ0
DO I=1,10
  AX=I*0.5
  BX=(I+1.0)*0.5
  CALL MNBRAK(AX,BX,CX,FA,FB,FC,BESSJ0)
  WRITE(*, '(1X,T13,A,T25,A,T37,A)') 'A','B','C'
  WRITE(*, '(1X,A3,T5,3F12.6)') 'X',AX,BX,CX
  WRITE(*, '(1X,A3,T5,3F12.6)') 'F',FA,FB,FC
END DO
END

```

计算结果如下:

	A	B	C
X	1.809017	3.118034	12.871834
F	0.334742	-0.297428	0.196191
	A	B	C
X	2.309017	3.618034	4.520578
F	0.050679	-0.393423	-0.315731
	A	B	C
X	2.000000	2.809017	7.472735
F	0.223891	-0.188717	0.269934
	A	B	C
X	3.309017	4.271136	5.827878
F	-0.346269	-0.365838	0.100316
	A	B	C
X	3.000000	3.809017	3.871458
F	-0.260052	-0.402656	-0.402442
	A	B	C
X	3.000000	3.500000	4.309017
F	-0.260052	-0.380128	-0.359448
	A	B	C
X	3.500000	4.000000	4.809017
F	-0.380128	-0.397150	-0.237726
	A	B	C
X	4.500000	4.000000	3.190983
F	-0.320543	-0.397150	-0.317815

	A	B	C
X	4.500000	3.761258	3.690983
F	-0.320543	-0.401754	-0.398728
	A	B	C
X	5.000000	4.190983	2.873393
F	-0.177597	-0.377793	-0.214198

(2) 函数过程 GOLDEN 是利用上面所取得的三个值 AX, BX, CX 继续求函数的极小值. 其方法是用黄金分割法分离出达到规定精度 TOL 的最小值. 验证程序 D11R2 再次以零阶贝塞尔函数为例子. 验证程序中以长度 1.0 为区间 (AX, BX) , 调用子过程 MNBRAK, 找出 $x=1.0$ 和 $x=101$ 之间包含最小值的区间. 然后, 调用 GOLDEN 找到该区间里最小值的横坐标值. 最后, 每找到一个最小值的横坐标都要和先前找到的进行比较, 如果不同, 那么就将 NMIN(最小值的个数) 增加, 并将最小值的横坐标值存放在数组 AMIN 中. 为了检验 GOLDEN, 验证程序 D11R2 打印出在最小值点上的 J_0 的值, 还打印出 J_1 的值 (J_0 的导数为 $-J_1$), 在这些 J_0 的极值点上, J_1 应该是零. 验证程序 D11R2 如下:

```

PROGRAM D11R2
! Driver for routine GOLDEN
EXTERNAL BESSJ0
PARAMETER (TOL=1.0E-6,EQL=1.E-3)
! USES BESSJ0,MNBRAK,BESSJ1
DIMENSION AMIN(20)
NMIN=0
WRITE(*, '(/1X,A)') 'Minima of the function BESSJ0'
WRITE(*, '(/1X,T6,A,T19,A,T27,A,T40,A/)' ) 'Min. #', &
      'X', 'BESSJ0(X)', 'BESSJ1(X)'
DO 1=1,100
  AX=1
  BX=1+1.0
  CALL MNBRAK(AX,BX,CX,FA,FB,FC,BESSJ0)
  G=GOLDEN(AX,BX,CX,BESSJ0,TOL,XMIN)
  IF(NMIN==0) THEN
    AMIN(1)=XMIN
    NMIN=1
    WRITE(*, '(1X,5X,I2,3X,3F12.6)') NMIN,&
      XMIN,BESSJ0(XMIN),BESSJ1(XMIN)
  ELSE

```

```

      IFLAG=0
      DO J=1,NMIN
        IF (ABS(XMIN-AMIN(J))<=EQL * XMIN)&
          IFLAG=1
      END DO
      IF (IFLAG==0) THEN
        NMIN=NMIN+1
        AMIN(NMIN)=XMIN
        WRITE(*,'(1X,5X,I2,3X,3F12.6)') NMIN,&
          XMIN,BESSJ0(XMIN),BESSJ1(XMIN)
      ENDIF
    ENDIF
  END DO
END

```

计算结果如下:

Minima of the function BESSJ0

Min. #	X	BESSJ0(X)	BESSJ1(X)
1	3.831542	-0.402759	0.000066
2	10.173387	-0.249705	0.000020
3	16.470192	-0.196465	0.000086
4	22.759737	-0.167185	0.000058
5	29.046865	-0.148011	-0.000005
6	35.332317	-0.134211	-0.000001
7	41.617130	-0.123668	-0.000005
8	47.901424	-0.115274	0.000004
9	54.185322	-0.108385	0.000025
10	60.469204	-0.102601	0.000026
11	66.752922	-0.097653	0.000030
12	73.036659	-0.093358	0.000022
13	79.320267	-0.089585	0.000020
14	85.603973	-0.086235	0.000004
15	91.887245	-0.083234	0.000021
16	98.170670	-0.080527	0.000023

11.2 不用导数的布伦特(Brent)法

1. 功能

联合使用反抛物内插法和黄金分割法求一维优化问题. 适用于函数有较好

解析性质的情形,它仅需用函数值且它兼有黄金分割和反二次插值的优点.

2. 方法

求一维优化问题, $\min_{x \in [a,b]} f(x)$, 且假设 $[a,b]$ 中有 $f(x)$ 的局部极小值点, 点 $x \in (a,b)$ 且 $f(x) < f(a), f(x) < f(b)$. 求极小值点的思想和黄金分割类似, 即通过取试探点使包含极小点的区间不断缩短, 当区间长度小到一定程度时, 区间上各点的函数值均接近极小值, 任意一点均可作为极小值点的近似. 具体计算步骤如下:

(1) $k=0, v=w=x, f(v)=f(w)=f(x)$.

(2) 计算当前区间的中点 $x_m = \frac{a+b}{2}$.

若 $|b-a| < \epsilon$, 则 x 可作为近似极小值点, 否则转(3).

(3) 分下面几种情况:

① 若点 x, w, v 共线, 转(4).

② 若新找的近似点 u 与当前函数的最小值点 x 之间满足 $|u-x| \leq \frac{1}{2} \max\{|a-x|, |b-x|\}$, 转(4).

③ 当 $|u-x| < \frac{1}{2} \max\{|a-x|, |b-x|\}$ 时, 用点 $(x, f(x)), (w, f(w)), (v, f(v))$ 作抛物插值, 求出其极小值点 u , 用它作为函数极小值点的新的近似, 但要排除下面情形: 若上面的极小值点 u 距当前包含极小值的区间的任意一个端点很近, 则上面求出的 u 无用, 将 x 作微小变化作为新的 u 而代替上面计算出的抛物插值的极小值点 u , 转(5).

(4) 按黄金分割选取点 u , 且 u 选在两区间 $[a, x]$ 和 $[x, b]$ 长度较大的一个之中. 若用黄金分割选取的 u 相对于 x 的改变量小于 $\epsilon|x|$, 则用 $\epsilon|x|$ 代替前面的改变量.

(5) 计算 $f(u)$, 由 u, x, a, b, w, v 及 $f(u), f(x), f(w), f(v)$ 将当前的 a, b, x, w, v 按其定义作相应改变, 得到一组新的 $a, b, x, w, v, k \leftarrow k+1$, 进行下一次迭代, 即转(2).

点 a, b, x, w, v, u 的定义如下:

(a, b) 是去掉对应的极小值点最远的点后得到的新的包含极小值点的区间;

x 其函数值 $f(x)$ 是当前求出的几个函数值中的最小值;

w 其函数值 $f(w)$ 是当前求出的几个函数值中的第二小函数值, 即 $f(w) > f(x)$, 但比其它算出的函数值小;

v 是前一次 w 的值;
 u 是最新求函数值的点.

上面方法的结束标志有两个,一是若已经可以确定的极小值点,一定在一个长度小于指定要求的区间之内,迭代停止;二是迭代次数超过指定的最大允许值时,停止迭代,表示迭代失败.

3. 使用说明

BRENT(AX,BX,CX,F,TOL,XMIN)

AX,CX 实型变量,输入参数,为含极小值点的区间端点

BX 实型变量,输入参数,它在 AX,CX 之间,且满足 $f(BX) < f(AX), f(BX) < f(CX)$

TOL 实型变量,输入参数,为变量 XMIN 的相对误差精度

XMIN 实型变量,输出参数,存放极小值点的近似值的最终结果

F 函数子过程,由使用者自编,其功能是计算目标函数 $f(x)$ 的值,其中 x 为实型变量

4. 过程

函数过程 BRENT.

```
FUNCTION brent(ax,bx,cx,f,tol,xmin)
```

```
INTEGER ITMAX
```

```
REAL brent,ax,bx,cx,tol,xmin,f,CGOLD,ZEPS
```

```
EXTERNAL f
```

```
PARAMETER (ITMAX=100,CGOLD=.3819660,ZEPS=1.0e-10)
```

```
INTEGER iter
```

```
REAL a,b,d,e,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,&
```

```
      v,w,x,xm
```

```
LOGICAL done
```

```
a=min(ax,cx)
```

```
b=max(ax,cx)
```

```
v=bx
```

```
w=v
```

```
x=v
```

```
e=0.
```

```
fx=f(x)
```

```
fv=fx
```



```

fw=fx
do iter=1,ITMAX
  xm=0.5*(a+b)
  tol1=tol*abs(x)+ZEPS
  tol2=2.*tol1
  if(abs(x-xm)<=(tol2-.5*(b-a))) exit
  done=-1
  if(abs(e)>tol1) then
    r=(x-w)*(fx-fv)
    q=(x-v)*(fx-fw)
    p=(x-v)*q-(x-w)*r
    q=2.*(q-r)
    if(q>0.) p=-p
    q=abs(q)
    etemp=e
    e=d
    dum=abs(.5*q*etemp)
    if(abs(p)>=dum.or.p<=q*(a-x).or.p>=q*(b-x)) then
      d=p/q
      u=x+d
      if(u-a<tol2.or.b-u<tol2) then
        d=sign(tol1,xm-x)
        end if
        done=0
      end if
    endif
    if(done) then
      if(x>=xm) then
        e=a-x
      else
        e=b-x
      endif
      d=CGOLD*e
    end if
    if(abs(d)>=tol1) then
      u=x+d
    else

```

```

      u = x + sign(tol1,d)
    endif
    fu = f(u)
    if(fu <= fx) then
      if(u >= x) then
        a = x
      else
        b = x
      endif
    endif
    v = w
    fv = fw
    w = x
    fw = fx
    x = u
    fx = fu
  else
    if(u < x) then
      a = u
    else
      b = u
    endif
    if(fu <= fw, or. w == x) then
      v = w
      fv = fw
      w = u
      fw = fu
    else if(fu <= fv, or. v == x, or. v == w) then
      v = u
      fv = fu
    endif
  end if
end do
if (iter > itmax) pause 'brent exceed maximum iterations'
xmin = x
brent = fx
END FUNCTION brent

```

5. 例子

函数过程 BRENT 和上节中的 GOLDEN 一样,也是要先调用 MNBRAK 确定出包含最小值点的三个数 AX, BX, CX . 验证程序 D11R3 基本上和验证 GOLDEN 的验证程序 D11R2 相同. 验证程序 D11R3 如下:

```

PROGRAM D11R3
! Driver for routine BRENT
EXTERNAL FUNC
PARAMETER (TOL=1.0E-6,EQL=1.E-4)
! USES MNBRAK,BESSJ1,BESSJ0
DIMENSION AMIN(20)
NMIN=0
WRITE(*, '(/1X,A)') 'Minima of the function BESSJ0'
WRITE(*, '(/1X,T6,A,T19,A,T28,A,T40,A/)' ) 'Min. #', &
      'X', 'BESSJ0(X)', 'BESSJ1(X)'
DO I=1,100
  AX=I
  BX=I+1.0
  CALL MNBRAK(AX,BX,CX,FA,FB,FC,FUNC)
  B=BRENT(AX,BX,CX,FUNC,TOL,XMIN)
  IF(NMIN==0) THEN
    AMIN(1)=XMIN
    NMIN=1
    WRITE(*, '(1X,5X,I2,3X,3F12.6)') NMIN, &
          XMIN, BESSJ0(XMIN), BESSJ1(XMIN)
  ELSE
    IFLAG=0
    DO J=1,NMIN
      IF(ABS(XMIN-AMIN(J))<=EQL*XMIN)&
        IFLAG=1
    END DO
    IF(IFLAG==0) THEN
      NMIN=NMIN+1
      AMIN(NMIN)=XMIN
      WRITE(*, '(1X,5X,I2,3X,3F12.6)') NMIN, &
            XMIN, BESSJ0(XMIN), BESSJ1(XMIN)
    
```

```

        ENDIF
    ENDIF
END DO
END PROGRAM
FUNCTION FUNC(X)
! USES BESSJ0
    FUNC=BESSJ0(X)
RETURN
END FUNCTION FUNC

```

计算结果如下:

Minima of the function BESSJ0

Min. #	X	BESSJ0(X)	BESSJ1(X)
1	3.831756	-0.402759	-0.000020
2	10.173570	-0.249705	-0.000025
3	16.470547	-0.196465	0.000016
4	22.760139	-0.167185	-0.000009
5	29.046837	-0.148011	-0.000001
6	35.332619	-0.134211	-0.000042
7	41.617237	-0.123668	-0.000018
8	47.901474	-0.115274	-0.000002
9	54.185696	-0.108385	-0.000015
10	60.469593	-0.102601	-0.000014
11	66.753098	-0.097653	0.000013
12	73.036911	-0.093358	-0.000002
13	79.320305	-0.089585	0.000016
14	85.604240	-0.086235	-0.000019
15	91.887634	-0.083234	-0.000011
16	98.170845	-0.080527	0.000008

11.3 用导数的布伦特(Brent)法

1. 功能

本子过程用牛顿法和二分法相结合的办法求一维优化问题. 要求目标函数有一阶导数.

2. 方法

设要求 $f(x)$ 的极小化问题, $[a, b]$ 为含极小值点的区间, $x \in (a, b)$ 且 $f(x) < f(a), f(x) < f(b)$. 求近似极小值点 x_{\min} 的步骤如下:

(1) $k=0, w=v=x$.

(2) 求 $x_m = \frac{a+b}{2}$.

若 $\frac{1}{2}|b-a| + |x-x_m| \leq 2\varepsilon|x| + 10^{-10}$, 则近似极小值点 $x_{\min}=x$, 计算结束, 否则转(3).

(3) 分下面几种情况:

① 若 $k=0$, 转(4).

② 当 $k \neq 0$ 时, 用牛顿(正割)法求一新近似极小值点 u :

$$u = x - \frac{f'(x)}{f''(x)}$$

其中 $f''(x)$ 用 $\frac{f'(x)-f'(w)}{x-w}$ 或 $\frac{f'(x)-f'(v)}{x-v}$ 来近似代替.

(a) 当 $f''(x) < 0$ 时, 则用这种方法所求点不是近似极小值点, 舍弃此时的 u , 转(4);

(b) 当 $f''(x) \geq 0$ 时, 若用牛顿法选取的 u 不落在前面选定的哪一段中, 转(4);

(c) 当 $f''(x) \geq 0$ 时, 若这样选取的 u 离当前的区间端点 a 或 b 很近, 则舍弃上面的 u , 重选 $u = x + \varepsilon|x|$ 或 $u = x - \varepsilon|x|$, 符号用来保证 u 和 x 不要偏向区间的一边, 转(5);

若(a), (b), (c)均不满足, 则用牛顿(正割)法选取的 u 可用, 转(5).

(4) 由 $f'(x)$ 的符号决定极小值点在区间 $[a, b]$ 中的哪一段 ($[a, x]$ 中或 $[x, b]$ 中), 在含极小值点的那一段中 ($[a, x]$ 中或 $[x, b]$ 中) 用二分法选取点 u .

(5) 计算 $f(u)$ 和 $f'(u)$, 将当前的 a, b, w, v, x, u 按其定义作相应改变, 得到一组新的 $a, b, w, v, x, k \leftarrow k+1$, 进行下一次迭代, 即转(2).

其中点 a, b, x, w, v, u 的定义如下:

(a, b) 是去掉对应的极小值点中最远的点后得到的新的包含极小值点的区间;

x 其函数值 $f(x)$ 是当前所求的几个函数值中最小的;

w 其函数值 $f(w)$ 是当前所求的几个函数值中第二小的函数值, 即 $f(w) \geq f(x)$;

v 是上一次迭代时的 w 的值;

u 是最新计算函数值的点.

结束标志有两个,一是若已经可以确定极小值点在一个长度小于指定要求的区间之内,迭代停止;二是迭代次数超过指定的最大允许值,停止迭代,表示迭代失败.

3. 使用说明

DBRENT(AX,BX,CX,F,DF,TOL,XMIN)

AX,CX 实型变量,输入参数,含极小值点的两区间端点

BX 实型变量,输入参数,它在 AX,CX 之间,且满足 $f(BX) < f(AX), f(BX) < f(CX)$

DF 函数子过程 DF(X),由用户自编,用来求目标函数的一阶导数值,其中 X 为实型变量,输入参数

TOL 实型变量,输入参数,一维搜索的精度

XMIN 实型变量,输出参数,存放极小值点近似值的最终结果

F 函数子过程,由用户自编,用来求目标函数的值,其中 x 为实型自变量,输入参数

4. 过程

函数过程 DBRENT.

```

FUNCTION dbrent(ax,bx,cx,f,df,tol,xmin)
INTEGER ITMAX
REAL dbrent,ax,bx,cx,tol,xmin,ZEPS
EXTERNAL df,f
PARAMETER (ITMAX=100,ZEPS=1.0e-10)
INTEGER iter
REAL a,b,d,d1,d2,du,dv,dw,dx,e,fu,fv,fw,fx,olde,&
    toll,tol2,u,u1,u2,v,w,x,xm
LOGICAL ok1,ok2,done
a=min(ax,cx)
b=max(ax,cx)
v=bx
w=v
x=v
e=0.
fx=f(x)

```

```

fv = fx
fw = fx
dx = df(x)
dv = dx
dw = dx
do iter = 1, ITMAX
    xm = 0.5 * (a + b)
    tol1 = tol * abs(x) + ZEPS
    tol2 = 2. * tol1
    if (abs(x - xm) <= (tol2 - .5 * (b - a))) then
        done = -1
        exit
    else
        done = 0
    end if
    if (abs(e) > tol1) then
        d1 = 2. * (b - a)
        d2 = d1
        if (dw /= dx) d1 = (w - x) * dx / (dx - dw)
        if (dv /= dx) d2 = (v - x) * dx / (dx - dv)
        u1 = x + d1
        u2 = x + d2
        ok1 = ((a - u1) * (u1 - b) > 0. .and. (dx * d1 <= 0.))
        ok2 = ((a - u2) * (u2 - b) > 0. .and. (dx * d2 <= 0.))
        olde = e
        e = d
        if (ok1 .or. ok2) then
            if (ok1 .and. ok2) then
                d = d1
            else
                d = d2
            endif
        else if (ok1) then
            d = d1
        else
            d = d2
        endif
    end if
end do

```

```

    if(abs(d)>abs(0.5 * olde)) then
        u=x+d
        if(u-a<tol2. or. b-u<tol2) d=sign(tol1,xm-x)
    end if
endif
if(dx>=0.) then
    e=a-x
else
    e=b-x
endif
d=0.5 * e
if(abs(d)>=tol1) then
    u=x+d
    fu=f(u)
else
    u=x+sign(tol1,d)
    fu=f(u)
    if(fu>fx) then
        done=-1
        exit
    else
        done=0
    end if
endif
du=df(u)
if(fu<=fx) then
    if(u>=x) then
        a=x
    else
        b=x
    endif
    v=w
    fv=fw
    dv=dw
    w=x
    fw=fx
    dw=dx

```



```

    x=u
    fx=fu
    dx=du
else
    if(u<x) then
        a=u
    else
        b=u
    endif
    if(fu<=fw, or, w==x) then
        v=w
        fv=fw
        dv=dw
        w=u
        fw=fu
        dw=du
    else if(fu<=fv, or, v==x, or, v==w) then
        v=u
        fv=fu
        dv=du
    endif
endif
end do
if (.not. done) then
    pause 'dbrent exceeded maximum iterations'
else
    xmin=x
    dbrent=fx
end if
END FUNCTION dbrent

```

5. 例子

函数过程 DBRENT 和上节介绍的函数过程 BRENT 非常相似,在验证程序 D11R4 中,先调用子过程 MNBRAK 确定包含最小值点的三个数 AX, BX, CX . 然后调用 DBRENT 即可找到最小值点的位置. 采用的例子仍是零阶贝塞尔函数,且在外部的函数子过程 DERIV 中定义了它的导数(—BESSJ1). 注意: DBRENT 只是在所求函数的导数能方便地被计算时才能采用. 验证程序 D11R4

如下:

```

PROGRAM D11R4
! Driver for routine DBRENT
EXTERNAL BESSJ0,DERIV
PARAMETER (TOL=1.0E-6,EQL=1.E-4)
! USES MNBRAK,BESSJ0
DIMENSION AMIN(20)
NMIN=0
WRITE(*, '(/1X,A)') 'Minima of the function BESSJ0'
WRITE(*, '(/1X,T6,A,T19,A,T27,A,T39,A,T53,A/)' )&
    'Min. #','X','BESSJ0(X)','BESSJ1(X)','DBRENT'
DO I=1,100
    AX=I
    BX=I+1.0
    CALL MNBRAK(AX,BX,CX,FA,FB,FC,BESSJ0)
    DBR=DBRENT(AX,BX,CX,BESSJ0,DERIV,TOL,XMIN)
    IF(NMIN==0) THEN
        AMIN(1)=XMIN
        NMIN=1
        WRITE(*, '(1X,5X,I2,3X,4F12.6)') NMIN,&
            XMIN,BESSJ0(XMIN),DERIV(XMIN),DBR
    ELSE
        IFLAG=0
        DO J=1,NMIN
            IF(ABS(XMIN-AMIN(J))<=EQL*XMIN)&
                IFLAG=1
        END DO
        IF(IFLAG==0) THEN
            NMIN=NMIN+1
            AMIN(NMIN)=XMIN
            WRITE(*, '(1X,5X,I2,3X,4F12.6)') NMIN,&
                XMIN,BESSJ0(XMIN),DERIV(XMIN),DBR
        ENDIF
    ENDIF
END DO
END PROGRAM
FUNCTION DERIV(X)

```

```

! USES BESSJ1
  DERIV = -BESSJ1(X)
RETURN
END FUNCTION DERIV

```

计算结果如下:

Minima of the function BESSJ0

Min. #	X	BESSJ0(X)	BESSJ1(X)	DBRENT
1	3.831704	-0.402759	-0.000001	-0.402759
2	10.173451	-0.249705	-0.000004	-0.249705
3	16.470623	-0.196465	-0.000001	-0.196465
4	22.760090	-0.167185	0.000001	-0.167185
5	29.046801	-0.148011	-0.000004	-0.148011
6	35.332275	-0.134211	-0.000004	-0.134211
7	41.617138	-0.123668	0.000005	-0.123668
8	47.901424	-0.115274	-0.000004	-0.115274
9	54.185535	-0.108385	-0.000002	-0.108385
10	60.469494	-0.102601	0.000004	-0.102601
11	66.753166	-0.097653	-0.000006	-0.097653
12	73.036926	-0.093358	0.000003	-0.093358
13	79.320557	-0.089585	0.000006	-0.089585
14	85.604103	-0.086235	0.000007	-0.086235
15	91.887596	-0.083234	0.000008	-0.083234
16	98.171059	-0.080527	0.000009	-0.080527

11.4 多元函数的下山单纯形法

1. 功能

本子过程可用来求解多元函数的极小值点. 子过程中只用到函数值, 不需计算目标函数的导数. 其优点是稳定性好, 准备时间短, 适用范围较广, 在搜索开始阶段效率更高一些, 但在试验点接近极小值点时速度会明显变慢, 因而它更常用于搜索的开始阶段. 对于问题的变量个数少、精度要求不高的情形, 这个方法常常是最好的方法. 使用时应注意到单纯形有可能退化到低维空间的情形, 此时, 可能会得不到问题的近似极小值点.

2. 方法

子过程中用的是 Nelder 和 Mead 的单纯形法. 具体做法如下:

(1) 选定一个初始单纯形, 它的 $N+1$ 个顶点 $\mathbf{x}^{(i)} (i=1, 2, \dots, N+1)$ 是这样确定的:

首先给定一个初始点 $\mathbf{x}^{(0)}$, 可定义其它点为

$$\mathbf{x}^{(i)} = \mathbf{x}^{(0)} + \lambda \mathbf{e}_i, \quad i = 1, 2, \dots, N$$

其中 \mathbf{e}_i 为 N 维单位向量, λ 是一个由问题的特性而猜想的常数 (也可以对每个方向 \mathbf{e}_i 选取不同的常数 λ_i).

(2) 计算 $f(x)$ 在这 $N+1$ 个顶点上的值 $f_i = f(\mathbf{x}^{(i)})$, $i=1, \dots, N+1$. 记

$$f_L = f(\mathbf{x}^{(L)}) = \min_{1 \leq i \leq N+1} \{f_i\}$$

$$f_H = f(\mathbf{x}^{(H)}) = \max_{1 \leq i \leq N+1} \{f_i\}$$

$$f_G = f(\mathbf{x}^{(G)}) = \max_{\substack{1 \leq i \leq N+1 \\ i \neq H}} \{f_i\}$$

对于求极小值点的问题来说, 显然认为 $\mathbf{x}^{(L)}$ 是最好的点, $\mathbf{x}^{(H)}$ 是最坏的点, $\mathbf{x}^{(G)}$ 是次坏的点.

(3) 构造一个新的单纯形, 它保留最好的点 $\mathbf{x}^{(L)}$, 另外我们要用一个比较好的点 $\mathbf{x}^{(N)}$ (至少应比 $\mathbf{x}^{(H)}$ 好) 去代替最坏的点 $\mathbf{x}^{(H)}$, 具体做法如下:

首先求除去最坏点 $\mathbf{x}^{(H)}$ 以后的 N 个点的重心:

$$\mathbf{x}^{(c)} = \frac{1}{N} \left(\sum_{i=1}^{N+1} \mathbf{x}^{(i)} - \mathbf{x}^{(H)} \right)$$

求 $\mathbf{x}^{(H)}$ 关于 $\mathbf{x}^{(c)}$ 的反射点

$$\mathbf{x}^{(R)} = (1 + \alpha)\mathbf{x}^{(c)} - \alpha\mathbf{x}^{(H)}$$

$\alpha > 0$ 为反射系数, 一般取 $\alpha=1$ (程序中即是). 计算 $f_R = f(\mathbf{x}^{(R)})$ 并与 f_L 进行比较.

① 若 $f_R \leq f_L$, 表明反射成功, 进行扩展, 即求

$$\mathbf{x}^{(E)} = \nu \mathbf{x}^{(R)} + (1 - \nu)\mathbf{x}^{(c)}$$

其中 $\nu > 1$ 为给定的扩展系数.

若 $f_E < f_L$, 就以 $\mathbf{x}^{(E)}$ 作为 $\mathbf{x}^{(N)}$, 否则让 $\mathbf{x}^{(R)}$ 作为 $\mathbf{x}^{(N)}$.

② 若 $f_R > f_L$, 此时如果 $f_R < f_G$, 即 $\mathbf{x}^{(R)}$ 点比次坏点 $\mathbf{x}^{(G)}$ 要好一些, 则取 $\mathbf{x}^{(N)} = \mathbf{x}^{(R)}$, 否则, 若 $f_R < f_H$, 则用 $\mathbf{x}^{(R)}$ 代替 $\mathbf{x}^{(H)}$. 令

$$\mathbf{x}^{(N)} = \beta \mathbf{x}^{(H)} + (1 - \beta)\mathbf{x}^{(c)}$$

总之, 我们得到了一个新的顶点 $\mathbf{x}^{(N)}$.

(4) 如果 $f_N < f_H$, 则 $\mathbf{x}^{(N)}$ 比 $\mathbf{x}^{(H)}$ 好, 用 $\mathbf{x}^{(N)}$ 代替 $\mathbf{x}^{(H)}$ 得到一个新的单纯形, 上述过程可重复进行下去, 即此时转 (2).

如果 $f_N \geq f_H$, 表明所取单纯形太大, 缩小原来的单纯形, 令

$$\mathbf{x}^{(i)} = \frac{1}{2}(\mathbf{x}^{(H)} + \mathbf{x}^{(L)}), \quad i = 1, 2, \dots, N+1$$

对缩小后的新单纯形继续迭代,即转(2).

(5) 终止条件.

① 若 $2|f(\mathbf{x}^{(H)}) - f(\mathbf{x}^{(L)})| / (|f(\mathbf{x}^{(H)})| + |f(\mathbf{x}^{(L)})|) < \epsilon$, 迭代停止.

② 若迭代次数超过指定的最大允许值, 停止迭代, 表示迭代失败.

3. 使用说明

AMOEBA(P, Y, MP, NP, NDIM, FTOL, FUNK, ITER)

NDIM 整型变量, 输入参数, 优化问题的维数, 即自变量的个数, 程序要求其不超过 20, 若超过 20 时用其它方法较好

MP, NP 整型变量, 输入参数, 存储数组 P 与 Y 的物理维数, $MP \geq NDIM + 1$, $NP \geq NDIM$

P $(NDIM + 1) \times NDIM$ 个元素的二维实型数组, 输入、输出参数, 输入时存放初始单纯形的 $NDIM + 1$ 个顶点 $(X^{(0)}, X^{(1)}, \dots, X^{(NDIM)})^T = P$, 输出时存放最终结果的单纯形的 $NDIM + 1$ 个顶点, 存储 P 的物理维数是 $MP \times NP$

Y $NDIM + 1$ 个元素的一维实型数组, 输入、输出参数, 输入时存放目标函数在初始单纯形的 $NDIM + 1$ 个顶点上的值, 输出时存放目标函数在最终单纯形上的 $NDIM + 1$ 个顶点上的近似极小值, 存储 Y 的物理维数是 MP

FTOL 实型变量, 输入参数, 存放在极小值所求的目标函数的相对误差精度

ITER 整型变量, 输出参数, 存放实际进行的迭代次数

FUNK 函数子过程, 调用目标函数, 调用形式是 FUNK(X), 其中 X 为含 NDIM 个元素的一维实型数组, 表示自变量, 该子过程用户自编

4. 过程

子过程 AMOEBA.

SUBROUTINE amoeba(p, y, mp, np, ndim, ftol, funk, iter)

PARAMETER (nmax=20, alpha=1.0, beta=0.5, gamma=2.0, &
itmax=500)

REAL p(mp, np), y(mp), pr(nmax), prr(nmax), pbar(nmax)

```

INTEGER i,j,ihl,inhl,ilo,mpts,iter
REAL, rtol
mpts=ndim+1
iter=0
do
    ilo=1
    if (y(1)>y(2)) then
        ihl=1
        inhl=2
    else
        ihl=2
        inhl=1
    endif
    do i=1,mpts
        if (y(i)<y(ilo)) ilo=i
        if (y(i)>y(ihl)) then
            inhl=ihl
            ihl=i
        else if (y(i)>y(inhl)) then
            if (i/=ihl) inhl=i
        endif
    end do
    rtol=2. * abs(y(ihl)-y(ilo))/(abs(y(ihl))+abs(y(ilo)))
    if (rtol<ftol) return
    if (iter==itmax)&
        pause 'amoeba exceeding maximum iterations.'
    iter=iter+1
    do j=1,ndim
        pbar(j)=0.
    end do
    do i=1,mpts
        if (i/=ihl) then
            do j=1,ndim
                pbar(j)=pbar(j)+p(i,j)
            end do
        endif
    end do

```

```

do j=1,ndim
  pbar(j)=pbar(j)/ndim
  pr(j)=(1.+alpha)*pbar(j)-alpha*p(ihi,j)
end do
ypr=funk(pr)
if(ypr<=y(ilo)) then
  do j=1,ndim
    prr(j)=gamma*pr(j)+(1.-gamma)*pbar(j)
  end do
  yprr=funk(prr)
  if (yprr<y(ilo)) then
    do j=1,ndim
      p(ihi,j)=prr(j)
    end do
    y(ihi)=yprr
  else
    do j=1,ndim
      p(ihi,j)=pr(j)
    end do
    y(ihi)=ypr
  endif
else if (ypr>=y(inhi)) then
  if (ypr<y(ihi)) then
    do j=1,ndim
      p(ihi,j)=pr(j)
    end do
    y(ihi)=ypr
  endif
do j=1,ndim
  prr(j)=beta*p(ihi,j)+(1.-beta)*pbar(j)
end do
yprr=funk(prr)
if(yprr<y(ihi)) then
  do j=1,ndim
    p(ihi,j)=prr(j)
  end do
  y(ihi)=yprr

```

```

      else
        do i=1,mpts
          if (i/=ilo) then
            do j=1,ndim
              pr(j)=0.5*(p(i,j)+p(ilo,j))
              p(i,j)=pr(j)
            end do
            y(i)=funk(pr)
          endif
        end do
      endif
    else
      do j=1,ndim
        p(ihi,j)=pr(j)
      end do
      y(ihi)=ypr
    endif
  end do
END SUBROUTINE amoeba

```

5. 例子

为了验证 AMOEBA, 我们所采用的例子为

$$\text{FAMOEB} = 0.6 - J_0[(x-0.5)^2 + (y-0.6)^2 - (z-0.7)^2]$$

该函数在 $(x, y, z) = (0.5, 0.6, 0.7)$ 有一极小值. 作为输入, AMOEBA 需要 N 维初始单纯形的 $N+1$ 个顶点的坐标和函数在这些顶点上的值 y , 我们用数组 P 确定的 $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$ 和 $(0, 0, 1)$ 作为开始单纯形的顶点. 为了得到计算的函数值 y , 接着我们设定了向量 $X(I)$. 这组数据和 $\text{FTOL} = 1.0\text{E}-6$ 一起被提供给 AMOEBA. 最终的单纯形顶点和对应的函数值被打印出来. 验证程序 D11R5 如下:

```

PROGRAM D11R5
  ! Driver for routine AMOEBA
  EXTERNAL FAMOEB
  PARAMETER (NP=3,MP=4,FTOL=1.0E-6)
  DIMENSION P(MP,NP),X(NP),Y(MP)
  DATA P/0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0/
  NDIM=NP

```



```

DO I=1,MP
  DO J=1,NP
    X(J)=P(I,J)
  END DO
  Y(I)=FAMOEB(X)
END DO
CALL AMOEBA(P,Y,MP,NP,NDIM,FTOL,FAMOEB,ITER)
WRITE(*, '(/1X,A,I3)' ) 'Iterations: ',ITER
WRITE(*, '(/1X,A)' ) 'Vertices of final 3-D simplex and'
WRITE(*, '(1X,A)' ) 'function values at the vertices:'
WRITE(*, '(/3X,A,T11,A,T23,A,T35,A,T45,A/)' ) 'I', &
      'X(I)', 'Y(I)', 'Z(I)', 'FUNCTION'
DO I=1,MP
  WRITE(*, '(1X,I3,4F12.6)' ) I, (P(I,J), J=1,NP), Y(I)
END DO
WRITE(*, '(/1X,A)' ) 'True minimum is at (0.5,0.6,0.7)'
END PROGRAM
FUNCTION FAMOEB(X)
! USES BESSJ0
DIMENSION X(3)
  FAMOEB=0.6-BESSJ0((X(1)-0.5)**2+(X(2)-0.6)**2+&
      (X(3)-0.7)**2)
END FUNCTION FAMOEB

```

计算结果如下:

Iterations: 20

Vertices of final 3- D simplex and

function values at the vertices:

I	X(I)	Y(I)	Z(I)	FUNCTION
1	0.490676	0.605329	0.717564	-0.400000
2	0.474908	0.597644	0.687566	-0.400000
3	0.530208	0.590894	0.684828	-0.400000
4	0.507350	0.633569	0.698853	-0.400000

True minimum is at (0.5,0.6,0.7)

11.5 多元函数的包维尔(Powell)法

1. 功能

本节有三个子过程,它们的功能分别是:

(1) POWELL, 用包维尔算法求多元函数的极小值点和极小值.

(2) LINMIN, 通过调用确定极小值点所在区间的子过程 MNBRAK 和求极小值点的一维搜索函数过程 BRENT, 求多元函数从起始点 p 沿某指定方向的极小值点, 它被子过程 POWELL 所调用.

(3) F1DIM, 求多元函数沿某指定方向上的点的函数值. 它被子过程 LINMIN 所调用.

包维尔算法又叫方向加速法, 它不需计算目标函数的导数, 使用起来准备时间少, 且具有较快的收敛速度. 在不依赖于目标函数的导数的所有直接法中, 它是最有效的一个方法.

2. 方法

(1) 选取起始点 $\mathbf{x}^{(0)}$ 和一组线性无关的方向 $\mathbf{e}^{(i)} (i=1, 2, \dots, N)$, N 为变量个数. 通常 $\mathbf{e}^{(i)}$ 取为 N 个坐标轴的方向, 即 $\mathbf{e}^{(i)} = (0, 0, \dots, 1, 0, \dots, 0)^T (i=1, \dots, N)$.

(2) 从 $\mathbf{x}^{(0)}$ 沿方向 $\mathbf{e}^{(i)} (i=1, 2, \dots, N)$ 依次进行 N 次一维搜索, 得到

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \lambda_i^* \mathbf{e}^{(i)}, \quad i = 1, 2, \dots, N$$

$$f(\mathbf{x}^{(i)}) = \min_{\lambda} f(\mathbf{x}^{(i-1)} + \lambda \mathbf{e}^{(i)}), \quad i = 1, 2, \dots, N$$

在完成了这 N 次一维搜索后, 得到 $\mathbf{x}^{(N)}$.

(3) 计算最速上升方向上函数的变化:

$$\text{DEL} = \max_{1 \leq i \leq N} |f(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(0)})| = |f(\mathbf{x}^{(BIG)}) - f(\mathbf{x}^{(0)})|$$

(4) 引进方向

$$\mathbf{e} \equiv \mathbf{x}^{(N)} - \mathbf{x}^{(0)}, \quad \mathbf{PTT} \equiv 2\mathbf{x}^{(N)} - \mathbf{x}^{(0)}$$

计算

$$f_E \equiv f(\mathbf{PTT}) = f(2\mathbf{x}^{(N)} - \mathbf{x}^{(0)})$$

(5) 若 $f_E \geq f_0$ 或 $f_E < f_0$ 且

$$2(f_0 - 2f_N + f_E)[(f_0 - f_N) - \text{DEL}]^2 \geq (f_0 - f_E)^2 \text{DEL}$$

则将 $\mathbf{x}^{(N)}$ 作为新的起始点, 沿上面的一组旧方向 $\mathbf{e}^{(i)} (i=1, 2, \dots, N)$ 重复上面步骤, 即转(2).

(6) 若(5)中条件均不满足, 沿方向 $\mathbf{e} = \mathbf{x}^{(N)} - \mathbf{x}^{(0)}$ 以 $\mathbf{x}^{(N)}$ 作为起始点进行探索得到目标函数在此方向上的极小值点 P . 将原来的方向 $\mathbf{e}^{(BIG)}$ 去掉而保留其余原有的 $N-1$ 个方向加上方向 \mathbf{e} 仍得到 N 个方向: $\mathbf{e}^{(1)}, \mathbf{e}^{(2)}, \dots, \mathbf{e}^{(N)}$, 以此时的 P 作为新起始点, 重复上面步骤, 即转(2).

(7) 结束标志有两个:

① 若 $\frac{2|f(\mathbf{x}^{(N)}) - f(\mathbf{x}^{(0)})|}{|f(\mathbf{x}^{(N)})| + |f(\mathbf{x}^{(0)})|} \leq \epsilon$, 则停止计算.

② 若上面过程进行到等于某选定的迭代最大次数时, 停止迭代, 表示迭代失败.

3. 使用说明

(1) POWELL(P, XI, N, NP, FTOL, ITER, FRET)

N 整型变量, 输入参数, 自变量个数

NP 整型变量, 输入参数, 存储 P 与 XI 的物理维数

P 含 N 个元素的一维实型数组, 输入、输出参数, 开始时存放极小值点的初始近似(由用户给定), 程序调用后存放求得的极小值点, 存储数组 P 的物理维数是 NP

XI 二维实型数组 $XI(N, N)$, 输入、输出参数, 开始时存放由用户选定的 N 个线性无关的向量, 常取为 $\mathbf{e}^{(i)} = (0, 0, \dots, 1, 0, \dots, 0)^T$ ($i = 1, 2, \dots, N$), 结束时存放当前的方向组, XI 存储在物理维数为 $NP \times NP$ 的一数组中

FTOL 实型变量, 输入参数, 存放对极小值点处的目标函数值的相对误差精度

ITER 整型变量, 输出参数, 存放程序已迭代的次数

FRET 实型变量, 输出参数, 存放目标函数在找到的近似极小值点 P 处的值

(2) LINMIN(P, XI, N, FRET)

N 整型变量, 输入参数, 存放自变量个数

P 含 N 个元素的一维实型数组, 输入、输出参数, 开始时存放初始点, 结束时存放从初始点沿某给定方向 XI 找到的目标函数的极小值点的近似点

XI 含 N 个元素的一维实型数组, 输入、输出参数, 开始时存放指定方向, 结束时存放找到的极小值点的实际位移向量

FRET 实型变量, 输出参数, 存放近似极小值 $f(p)$

子过程 LINMIN 被子过程 POWELL 调用. 而它本身还要调用过程 MN-BRAK, BRENT 和 F1DIM(均见本章).

(3) F1DIM(X)

X 实型变量, 输入参数

函数过程 F1DIM 必须伴随子过程 LINMIN.

4. 过程

(1) 子过程 POWELL.

```

SUBROUTINE powell(p,xi,n,np,ftol,iter,fret)
INTEGER iter,n,np,NMAX,ITMAX
REAL fret,ftol,p(np),xi(np,np),func
EXTERNAL func
PARAMETER (NMAX=20,ITMAX=200)
! USES func,linmin
INTEGER i,ibig,j
REAL del,fp,fptt,t,pt(NMAX),ptt(NMAX),xit(NMAX)
fret=func(p)
do j=1,n
    pt(j)=p(j)
end do
iter=0
do
    do
        do
            iter=iter+1
            fp=fret
            ibig=0
            del=0.
            do i=1,n
                do j=1,n
                    xit(j)=xi(j,i)
                end do
                fptt=fret
                call linmin(p,xit,n,fret)
                if(abs(fptt-fret)>del) then
                    del=abs(fptt-fret)
                    ibig=i
                endif
            end do
            end do
            if(2. * abs(fp-fret)<=ftol * (abs(fp)+abs(fret))) return
            if(iter==ITMAX) then
                pause 'powell exceeding maximum iterations'
            end if
        end do
    end do
end do

```

```

        return
    end if
    do j=1,n
        ptt(j)=2. * p(j)-pt(j)
        xit(j)=p(j)-pt(j)
        pt(j)=p(j)
    end do
    fptt=func(ptt)
    if(fptt>=fp) exit
end do
t=2. * (fp-2. * fret+fptt) * (fp-fret-del) * * 2-del * (fp-fptt) * * 2
if(t>=0.) exit
end do
call linmin(p,xit,n,fret)
do j=1,n
    xi(j,ibig)=xi(j,n)
    xi(j,n)=xit(j)
end do
end do
END SUBROUTINE powell

```

(2) 子过程 LINMIN.

```

SUBROUTINE linmin(p,xi,n,fret)
INTEGER n,NMAX
REAL fret,p(n),xi(n),tol
PARAMETER (nmax=50,TOL=1.e-4)
! USES brent,fldim,mnbrak,dfldim
INTEGER j,ncom
REAL ax,bx,fa,fb,fx,xmin,xx,pcom(nmax),&
    xicom(nmax),brent
COMMON /ficom/ pcom,xicom,ncom
EXTERNAL fldim
! EXTERNAL dfldim ! 共轭梯度法
ncom=n
do j=1,n
    pcom(j)=p(j)
    xicom(j)=xi(j)

```

```

end do
ax=0.
xx=1.
call mnbrak(ax,xx,bx,fa,fx,fb,fldim)
! fret=brent(ax,xx,bx,fldim,tol,xmin) ! Powell 法
fret=dbrent(ax,xx,bx,fldim,dfldim,tol,xmin) ! 共轭梯度法
do j=1,n
    xi(j)=xmin * xi(j)
    p(j)=p(j)+xi(j)
end do
END SUBROUTINE linmin

```

(3) 函数过程 F1DIM.

```

FUNCTION f1dim(x)
INTEGER NMAX
REAL f1dim,func,x
PARAMETER (NMAX=50)
! USES func
INTEGER j,ncom
REAL pcom(NMAX),xicom(NMAX),xt(NMAX)
COMMON /f1com/ pcom,xicom,ncom
do j=1,ncom
    xt(j)=pcom(j)+x * xicom(j)
end do
f1dim=func(xt)
END FUNCTION f1dim

```

5. 例子

(1) 为了验证 POWELL, 我们所取的例子为

$$\text{FUNC2}(x,y,z) = 0.5 - J_0[(x-1)^2 + (y-2)^2 + (z-3)^2]$$

在验证程序 D11R6 中该函数是由函数过程所定义的. 采用的初始点 P 为 (1.5, 1.5, 2.5), 一组初始方向为单位向量 (1,0,0), (0,1,0) 和 (0,0,1). POWELL 用下面即要验证的 LINMIN 来进行一维极小化. 验证程序 D11R6 如下:

```

PROGRAM D11R6
! Driver for routine POWELL
PARAMETER (NDIM=3,FTOL=1.0E-6)

```

```

DIMENSION P(NDIM),XI(NDIM,NDIM)
DATA XI/1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0/
DATA P/1.5,1.5,2.5/
NP=NDIM
CALL POWELL(P,XI,NDIM,NP,FTOL,ITER,FRET)
WRITE(*, '(/1X,A,I3)' ) 'Iterations: ',ITER
WRITE(*, '(/1X,A/1X,3F12.6)' ) &
    'Minimum found at: ',(P(I),I=1,NDIM)
WRITE(*, '(/1X,A,F12.6)' ) 'Minimum function value= ',&
    FRET
WRITE(*, '(/1X,A)' ) 'True minimum of function is at:'
WRITE(*, '(1X,3F12.6/)' ) 1.0,2.0,3.0
END
FUNCTION FUNC(X)
! USES BESSJ0
DIMENSION X(3)
FUNC=0.5-BESSJ0((X(1)-1.0)* * 2+(X(2)-2.0)* * 2+&
    (X(3)-3.0)* * 2)
END

```

计算结果如下:

```

Iterations:    2
Minimum found at:
    1.012993    1.987284    3.005544
Minimum function value=    -0.500000
True minimum of function is at:
    1.000000    2.000000    3.000000

```

(2) 上面我们已经提到的 LINMIN 是在 N 维空间沿着某一方向求出函数 FUNC 的极小值. 为了应用它, 我们要确定一个点 P 和一个 N 个元素的方向向量 XI . 我们采用的例子为

$$\text{FUNC2}(x,y,z) = (x-1)^2 + (y-1)^2 + (z-1)^2$$

该函数在 $(x,y,z)=(1,1,1)$ 有一极小值. 在验证程序 D11R7 中, 我们取 P 为原点 $(0,0,0)$, 并设定一组方向

$$\left(\sqrt{2} \cos\left(\frac{\pi}{2} \frac{I}{10.0}\right), \sqrt{2} \sin\left(\frac{\pi}{2} \frac{I}{10.0}\right), 1.0 \right), \quad I = 1, \dots, 10$$

每取一个方向时, 此时的最小值点和在这个最小值点上的函数值将被打印出来.

方向(1,1,1)在这组方向中. 当然, 沿这个方向极值点上的函数值应该是零. 验证程序 D11R7 如下:

```

PROGRAM D11R7
1 Driver for routine LINMIN
PARAMETER(NDIM=3,PIO2=1.5707963)
DIMENSION P(NDIM),XI(NDIM)
WRITE(*, '(1X,A)') &
    'Minimum of a 3-D quadratic centered'
WRITE(*, '(1X,A)') &
    'at (1.0,1.0,1.0). Minimum is found'
WRITE(*, '(1X,A)') 'along a series of radials.'
WRITE(*, '(1X,T10,A,T22,A,T34,A,T42,A/)') &
    'X','Y','Z','MINIMUM'
DO I=0,10
    X=PIO2*I/10.0
    SR2=SQRT(2.0)
    XI(1)=SR2*COS(X)
    XI(2)=SR2*SIN(X)
    XI(3)=1.0
    P(1)=0.0
    P(2)=0.0
    P(3)=0.0
    CALL LINMIN(P,XI,NDIM,FRET)
    WRITE(*, '(1X,4F12.6)') (P(J),J=1,3),FRET
END DO
END PROGRAM
FUNCTION FUNC(X)
DIMENSION X(3)
FUNC=0.0
DO I=1,3
    FUNC=FUNC+(X(I)-1.0)**2
END DO
END FUNCTION FUNC

```

计算结果如下:

```

Minimum of a 3-D quadratic centered
at (1.0,1.0,1.0). Minimum is found

```


along a series of radials.

X	Y	Z	MINIMUM
1.137986	0.000000	0.804678	1.057191
1.219067	0.193081	0.872755	0.715299
1.247294	0.405270	0.927358	0.420135
1.218971	0.621097	0.967381	0.192580
1.134707	0.824413	0.991770	0.049045
1.000000	1.000000	1.000000	0.000000
0.824413	1.134707	0.991770	0.049045
0.621097	1.218971	0.967381	0.192580
0.405270	1.247293	0.927358	0.420134
0.193081	1.219067	0.872755	0.715299
0.000000	1.138268	0.804877	1.057191

11.6 多元函数的共轭梯度法

1. 功能

子过程 FRPRMN 用 Polyak-Polak-Ribiere 和 Fletcher-Reeves 和共轭梯度算法, 求解具有导数的多元函数的无约束极小值点.

2. 方法

共轭梯度法的基本思想是把共轭性与最速下降法相结合, 利用已知点处的梯度构造一组共轭方向, 并沿这组方向进行搜索, 求出目标函数的极小值点. 具体计算步骤如下:

(1) 选取 $\mathbf{x}^{(0)}$ 为起始点, 计算 $\mathbf{x}^{(0)}$ 处的梯度 $\nabla f(\mathbf{x}^{(0)})$. 令 $\mathbf{g}^{(0)} = -\nabla f(\mathbf{x}^{(0)})$, 若 $\mathbf{g}^{(0)} = 0$, 则 $\mathbf{x}^{(0)}$ 就是极值点; 否则令 $\mathbf{h}^{(1)} = \mathbf{g}^{(0)}$, 它为最速下降方向.

(2) 自 $\mathbf{x}^{(0)}$ 出发, 沿 $\mathbf{h}^{(1)}$ 方向作一维搜索, 得

$$\begin{aligned}\mathbf{x}^{(1)} &= \mathbf{x}^{(0)} + \lambda_1^* \mathbf{h}^{(1)}, \\ f(\mathbf{x}^{(1)}) &= \min_{\lambda} f(\mathbf{x}^{(0)} + \lambda \mathbf{h}^{(1)})\end{aligned}$$

(3) 计算 $\mathbf{x}^{(1)}$ 处的梯度 $\nabla f(\mathbf{x}^{(1)})$, 且令 $\mathbf{g}^{(1)} = -\nabla f(\mathbf{x}^{(1)})$. 设 $\mathbf{g}^{(1)} \neq 0$ (否则 $\mathbf{x}^{(1)}$ 就是极值点), 且令

$$\mathbf{h}^{(2)} = \mathbf{g}^{(1)} + \nu \mathbf{h}^{(1)}$$

其中

$$\nu = \frac{(\mathbf{g}^{(1)} - \mathbf{g}^{(0)})^T \mathbf{g}^{(1)}}{\mathbf{g}^{(0)T} \mathbf{g}^{(0)}} \quad \text{或} \quad \nu = \frac{\mathbf{g}^{(1)T} \mathbf{g}^{(1)}}{\mathbf{g}^{(0)T} \mathbf{g}^{(0)}}$$

它们分别为 Polak-Ribiere 形式和 Fletcher-Reeves 形式.

(4) 以 $\mathbf{x}^{(1)}$ 为新的起始点, 沿 $\mathbf{h}^{(2)}$ 方向作一维搜索, 重复上面步骤, 即令

$$\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{(1)}, \quad \mathbf{h}^{(1)} \leftarrow \mathbf{h}^{(2)}$$

转(2).

(5) 结束标志有三个:

① 若在某步其在某点的梯度为零, 迭代停止, 此时已找到了极值点.

② 若相邻两步的函数值的相对误差已很小, 即

$$\frac{|f(\mathbf{x}^{(1)}) - f(\mathbf{x}^{(0)})|}{|f(\mathbf{x}^{(1)})| + |f(\mathbf{x}^{(0)})|} < \frac{\varepsilon}{2}$$

则停止迭代, 此时已找到了满足精度要求的近似极值点.

③ 若迭代次数已达到指定的最大迭代次数, 则迭代停止, 表示迭代失败.

3. 使用说明

(1) FRPRMN(P, N, FTOL, ITER, FRET)

N 整型变量, 输入参数, 自变量的个数

P 含 N 个元素的一维实型数组, 输入、输出参数, 开始时存放极小值点的初始近似值, 结束时存放求得的目标函数极小值点

FTOL 实型变量, 输入参数, 存放对极小值点处的目标函数值的精度要求

ITER 整型变量, 输出参数, 存放执行的迭代次数

FRET 实型变量, 输出参数, 存放目标函数的近似极小值的最终结果

(2) DF1DIM(X)

X 实型变量, 输入参数

(3) 调用的子过程说明:

① FUNC(X), 求目标函数值的函数子过程, 由使用者自编, 其中 X 为含 N 个元素的一维实型数组.

② DFUNC(X, DF), 求目标函数在 P 点的梯度 $\nabla f(p)$ 且将其放入 DF 中的子过程, 由使用者自编, 其中 X 和 DF 均为含 N 个元素的一维实型数组.

③ LINMIN(P, XI, N, FRET), 用于作一维搜索, 具体说明及程序见上节(包维尔算法), 但将其中语句

FRET = BRENT(AX, XX, BX, F1DIM, TOL, XMIN)

换为

FRET = DBRENT(AX, XX, BX, F1DIM, DF1DIM, TOL, XMIN)

子过程 LINMIN 要调用子过程 MNBRAK(见 11.1 节)、DBRENT(见 11.3 节)和函数过程 F1DIM(见 11.5 节)、DE1DIM(见本节). 另外注意, EXTER-

NAL 语句也改为 DF1DIM.

④ 在子过程 FRPRMN 中若注释行为 $DGG = DGG + XI(J) ** 2$, 则程序为 Polak-Ribiere 形式的共轭梯度法, 若注释行改为 $DGG = DGG + (XI(J) + G(J)) * XI(J)$, 则程序为 Fletcher-Reeves 形式的共轭梯度法.

4. 过程

(1) 子过程 FRPRMN.

```

SUBROUTINE frprmn(p,n,ftol,iter,fret)
INTEGER iter,n,NMAX,ITMAX
REAL fret,ftol,p(n),EPS,func
EXTERNAL func
PARAMETER (NMAX=50,ITMAX=200,EPS=1.e-10)
! USES dfunc,func,linmin
INTEGER its,j
REAL dgg,fp,gam,gg,g(NMAX),h(NMAX),xi(NMAX)
fp=func(p)
call dfunc(p,xi)
do j=1,n
    g(j)=-xi(j)
    h(j)=g(j)
    xi(j)=h(j)
end do
do its=1,ITMAX
    iter=its
    call linmin(p,xi,n,fret)
    if(2.*abs(fret-fp)<=ftol*(abs(fret)+abs(fp)+EPS))&
        return
    fp=func(p)
    call dfunc(p,xi)
    gg=0.
    dgg=0.
    do j=1,n
        gg=gg+g(j)**2
        dgg=dgg+xi(j)**2 ! polak-ribiere 法
!       dgg=dgg+(xi(j)+g(j))*xi(j) ! Fletcher-Reeves 法
    end do
    if(gg==0.) return

```

```

    gam=dgg/gg
    do j=1,n
        g(j)=-xi(j)
        h(j)=g(j)+gam*h(j)
        xi(j)=h(j)
    end do
end do
pause 'frprmn maximum iterations exceeded'
END SUBROUTINE frprmn

```

(2) 函数过程.

```

FUNCTION df1dim(x)
PARAMETER (nmax=50)
REAL xicom(nmax),pcom(nmax)
COMMON /f1com/ pcom,xicom,ncom
REAL xt(nmax),df(nmax)
INTEGER j
do j=1,ncom
    xt(j)=pcom(j)+x*xicom(j)
end do
call dfunc(xt,df)
df1dim=0.
do j=1,ncom
    df1dim=df1dim+df(j)*xicom(j)
end do
END FUNCTION df1dim

```

5. 例子

为了验证子过程 FRPRMN, 在验证程序 D11R8 中, 我们采用的例子为

$$\text{FUNC2}(x, y, z) = 1.0 - J_0(x - 0.5)J_0(y - 0.5)J_0(z - 0.5)$$

和

$$\frac{\partial \text{FUNC2}}{\partial x} = J_1(x - 0.5)J_0(y - 0.5)J_0(z - 0.5)$$

一组试验初值被应用, 且每次 FRPRMN 总是在 (0.5, 0.5, 0.5) 处发现极值点. 验证程序 D11R8 如下:

```
PROGRAM D11R8
```

```

! Driver for routine FRPRMN
PARAMETER (NDIM=3,PIO2=1.5707963,FTOL=1.0E-6)
DIMENSION P(NDIM)
WRITE(*, '(1X,A)') &
    'Program finds the minimum of a function'
WRITE(*, '(1X,A)') 'with different trial starting vectors.'
WRITE(*, '(1X,A)') 'True minimum is (0.5,0.5,0.5)'
DO K=0,4
    ANGL=PIO2 * K/4.0
    P(1)=2.0 * COS(ANGL)
    P(2)=2.0 * SIN(ANGL)
    P(3)=0.0
    WRITE(*, '(1X,A,3(F6.4,A))') 'Starting vector: (' ,&
        P(1),',',P(2),',',P(3),')'
    CALL FRPRMN(P,NDIM,FTOL,ITER,FRET)
    WRITE(*, '(1X,A,I3)') 'Iterations:',ITER
    WRITE(*, '(1X,A,3(F6.4,A))') 'Solution vector: (' ,&
        P(1),',',P(2),',',P(3),')'
    WRITE(*, '(1X,A,E14.6)') 'Func. value at solution',FRET
END DO
END PROGRAM

FUNCTION FUNC(X)
DIMENSION X(3)
    FUNC=1.0-BESSJ0(X(1)-0.5) * BESSJ0(X(2)-0.5) * &
        BESSJ0(X(3)-0.5)
END FUNCTION FUNC

SUBROUTINE DFUNC(X,DF)
PARAMETER (NMAX=50)
! USES BESSJ0,BESSJ1
DIMENSION X(3),DF(NMAX)
DF(1)=BESSJ1(X(1)-0.5) * BESSJ0(X(2)-0.5) * BESSJ0(X(3)-0.5)
DF(2)=BESSJ0(X(1)-0.5) * BESSJ1(X(2)-0.5) * BESSJ0(X(3)-0.5)
DF(3)=BESSJ0(X(1)-0.5) * BESSJ0(X(2)-0.5) * BESSJ1(X(3)-0.5)
END SUBROUTINE DFUNC

```

计算结果如下：

Program finds the minimum of a function

```

with different trial starting vectors.
True minimum is (0.5,0.5,0.5)
Starting vector: (2.0000,0.0000,0.0000)
Iterations: 6
Solution vector: (0.4999,0.5000,0.5000)
Func. value at solution 0.000000E+00
Starting vector: (1.8478,0.7654,0.0000)
Iterations: 5
Solution vector: (0.4999,0.5000,0.5000)
Func. value at solution 0.000000E+00
Starting vector: (1.4142,1.4142,0.0000)
Iterations: 4
Solution vector: (0.5000,0.5000,0.4999)
Func. value at solution 0.000000E+00
Starting vector: (0.7654,1.8478,0.0000)
Iterations: 5
Solution vector: (0.5000,0.4999,0.5000)
Func. value at solution 0.000000E+00
Starting vector: (0.0000,2.0000,0.0000)
Iterations: 6
Solution vector: (0.5000,0.4999,0.5000)
Func. value at solution 0.000000E+00

```

11.7 多元函数的变尺度法

1. 功能

子过程 DFPMIN 用 B-F-G-S(Broyden-Fletcher-Goldfarb-Shanno) 秩 2 算法求具有导数的多元函数的无约束极小值点, 或具有导数的非线性方程组的近似解. 实践证明, 本算法是目前最成功的算法之一, 具有较好的数值稳定性.

2. 方法

计算步骤如下:

- (1) 给定初始点 $\mathbf{x}^{(1)} \in R^n$, 允许误差 FTOL.
- (2) 令 $\mathbf{H}_1 = \mathbf{I}_n$ (单位矩阵), 计算出在 $\mathbf{x}^{(1)}$ 处的梯度 $\mathbf{g}_1 = \nabla f(\mathbf{x}^{(1)})$, 并令 $k=1$.
- (3) 令 $\mathbf{d}^{(k)} = -\mathbf{H}_k \mathbf{g}_k$.

(4) 从 $\mathbf{x}^{(k)}$ 出发, 沿方向 $\mathbf{d}^{(k)}$ 作一维搜索, 使 $f(\mathbf{x}^{(k)} + \lambda_k \mathbf{d}^{(k)}) = \min_{\lambda \geq 0} f(\mathbf{x}^{(k)} + \lambda \mathbf{d}^{(k)})$, 令

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda_k \mathbf{d}^{(k)}$$

(5) 检验是否满足收敛准则, 若

$$\frac{|f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}^{(k)})|}{|f(\mathbf{x}^{(k+1)})| + |f(\mathbf{x}^{(k)})|} < \frac{\text{FTOL}}{2}$$

则停止迭代, 近似极小值点 (或非线性方程组的近似解) 即为 $\mathbf{x}^{(k+1)}$; 否则进行 (6).

(6) 令

$$\begin{aligned} \mathbf{g}_{k+1} &= \nabla f(\mathbf{x}^{(k+1)}), \quad \mathbf{p}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \mathbf{q}^{(k)} &= \mathbf{g}_{(k+1)} - \mathbf{g}_k \end{aligned}$$

利用 BFGS 公式计算 \mathbf{H}_{k+1} :

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{p}^{(k)} \mathbf{p}^{(k)T}}{\mathbf{p}^{(k)T} \mathbf{q}^{(k)}} - \frac{\mathbf{H}_k \mathbf{q}^{(k)} \mathbf{q}^{(k)T} \mathbf{H}_k^T}{\mathbf{q}^{(k)T} \mathbf{H}_k \mathbf{q}^{(k)}} + [\mathbf{q}^{(k)T} \mathbf{H}_k \mathbf{q}^{(k)}] \mathbf{u} \mathbf{u}^T$$

其中

$$\mathbf{u} \equiv \frac{\mathbf{p}^{(k)}}{\mathbf{p}^{(k)T} \mathbf{q}^{(k)}} - \frac{\mathbf{H}_k \mathbf{q}^{(k)}}{\mathbf{q}^{(k)T} \mathbf{H}_k \mathbf{q}^{(k)}}$$

让 $k \leftarrow k+1$, 若 k 已等于选定的最大迭代次数, 迭代停止, 表示迭代失败; 否则, 返回 (3).

3. 使用说明

(1) DFPMIN(P, N, FTOL, ITER, FRET)

N 整型变量, 输入参数, 自变量的个数

P 含 N 个元素的一维实型数组, 输入、输出参数: 开始时存放极小值点的初始近似值, 结束时存放近似极小值点的最后结果

FTOL 实型变量, 输入参数, 存放近似极小值所要满足的精度要求

ITER 整型变量, 输出参数, 存放完成的迭代次数

FRET 实型变量, 输出参数, 存放近似极小值的最后结果

(2) 要调用的子过程:

① LINMIN, 用于完成一维搜索 (见 11.5 节). 它要调用子过程 MNBRAK (见 11.1 节), 函数子过程 F1DIM (见 11.5 节).

② 函数子过程 FUNC(X), 由使用者自编, 用于求多元函数在给定点 X 处的值, 其中 X 为含 N 个元素的一维实型数组, 输入参数.

③ 子过程 DFUNC(X, DF), 由使用者自编, 用于求目标函数在 X 处的梯度

$\nabla f(X)$ 并将其放入 DF 中, 其中 X 和 DF 均为含 N 个元素的一维实型数组, X 为输入参数, DF 为输出参数.

4. 过程

子过程 DFPMIN.

```

SUBROUTINE dfpmin(p,n,ftol,iter,fret)
PARAMETER (nmax=50,itmax=200,eps=1.e-10)
! USES linmin
REAL p(n),hessin(nmax,nmax),xi(nmax),g(nmax),&
      dg(nmax),hdg(nmax)
INTEGER i,j,its
REAL fac,fae,fad,fp
fp=func(p)
call dfunc(p,g)
do i=1,n
  do j=1,n
    hessin(i,j)=0.
  end do
  hessin(i,i)=1.
  xi(i)=-g(i)
end do
do its=1,itmax
  iter=its
  call linmin(p,xi,n,fret)
  if(2.*abs(fret-fp)<=ftol*(abs(fret)+abs(fp)+eps))&
    return

  fp=fret
  do i=1,n
    dg(i)=g(i)
  end do
  fret=func(p)
  call dfunc(p,g)
  do i=1,n
    dg(i)=g(i)-dg(i)
  end do
  do i=1,n

```



```

      hdg(i)=0.
      do j=1,n
        hdg(i)=hdg(i)+hessin(i,j)*dg(j)
      end do
    end do
    fac=0.
    fae=0.
    do i=1,n
      fac=fac+dg(i)*xi(i)
      fae=fae+dg(i)*hdg(i)
    end do
    fac=1./fac
    fad=1./fae
    do i=1,n
      dg(i)=fac*xi(i)-fad*hdg(i)
    end do
    do i=1,n
      do j=1,n
        hessin(i,j)=hessin(i,j)+fac*xi(i)*xi(j)&
          -fad*hdg(i)*hdg(j)+fae*dg(i)*dg(j)
      end do
    end do
    do i=1,n
      xi(i)=0.
      do j=1,n
        xi(i)=xi(i)-hessin(i,j)*g(j)
      end do
    end do
  end do
  PAUSE 'too many iterations in DFPMIN'
END SUBROUTINE dfpmin

```

5. 例子

子过程 DFPMIN 比前面的子过程需要稍微多一点的中间存储,但是它是一种流行的方法. 验证子过程 DFPMIN 的程序 D11R9 的执行和 D11R8 一样,包括一个计算导数的子过程. 验证程序 D11R9 如下:

```

PROGRAM D11R9
! Driver for routine DFPMIN
PARAMETER (NDIM=3,PIO2=1.5707963,FTOL=1.0E-6)
DIMENSION P(NDIM)
WRITE(*, '( /1X,A)' )&
    'Program finds the minimum of a function'
WRITE(*, '(1X,A)') 'with different trial starting vectors.'
WRITE(*, '(1X,A)') 'True minimum is (0.5,0.5,0.5)'
DO K=0,4
    ANGL=PIO2 * K/4.0
    P(1)=2.0 * COS(ANGL)
    P(2)=2.0 * SIN(ANGL)
    P(3)=0.0
    WRITE(*, '( /1X,A,3(F6.4,A))' )&
        'Starting vector: (' ,P(1),',',P(2),',',P(3),')'
    CALL DFPMIN(P,NDIM,FTOL,ITER,FRET)
    WRITE(*, '(1X,A,I3)') 'Iterations:',ITER
    WRITE(*, '(1X,A,3(F6.4,A))' )&
        'Solution vector: (' ,P(1),',',P(2),',',P(3),')'
    WRITE(*, '(1X,A,E14.6)') 'Func. value at solution',FRET
END DO
END
FUNCTION FUNC(X)
! USES BESSJ0
DIMENSION X(3)
    FUNC=1.0-BESSJ0(X(1)-0.5) * BESSJ0(X(2)-0.5) * &
        BESSJ0(X(3)-0.5)
END FUNCTION FUNC
SUBROUTINE DFUNC(X,DF)
PARAMETER (NMAX=50)
! USES BESSJ0,BESSJ1
DIMENSION X(3),DF(NMAX)
    DF(1)=BESSJ1(X(1)-0.5) * BESSJ0(X(2)-0.5) * BESSJ0(X(3)-0.5)
    DF(2)=BESSJ0(X(1)-0.5) * BESSJ1(X(2)-0.5) * BESSJ0(X(3)-0.5)
    DF(3)=BESSJ0(X(1)-0.5) * BESSJ0(X(2)-0.5) * BESSJ1(X(3)-0.5)
END SUBROUTINE DFUNC

```

计算结果如下:

```

Program finds the minimum of a function
with different trial starting vectors.
True minimum is (0.5,0.5,0.5)
Starting vector: (2.0000,0.0000,0.0000)
Iterations: 4
Solution vector: (0.5000,0.4999,0.4999)
Func. value at solution 0.000000E+00
Starting vector: (1.8478,0.7654,0.0000)
Iterations: 4
Solution vector: (0.5000,0.4999,0.5000)
Func. value at solution 0.000000E+00
Starting vector: (1.4142,1.4142,0.0000)
Iterations: 4
Solution vector: (0.5000,0.5000,0.4999)
Func. value at solution 0.000000E+00
Starting vector: (0.7654,1.8478,0.0000)
Iterations: 4
Solution vector: (0.5000,0.5000,0.4999)
Func. value at solution 0.000000E+00
Starting vector: (0.0000,2.0000,0.0000)
Iterations: 4
Solution vector: (0.4999,0.5000,0.4999)
Func. value at solution 0.000000E+00

```

11.8 线性规划的单纯形法

1. 功能

用线性规划的二阶段单纯形方法求解下列线性规划问题:

$$\text{Max } z = a_{01}x_1 + a_{02}x_2 + \cdots + a_{0n}x_n \quad (11-1)$$

约束

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i, \quad i = 1, \cdots, m_1 \quad (11-2)$$

$$a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n \geq b_j, \quad j = m_1 + 1, \cdots, m_1 + m_2 \quad (11-3)$$

$$a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n = b_k, \quad k = m_1 + m_2 + 1, \cdots, m \quad (11-4)$$

$$x_j \geq 0, \quad j = 1, \cdots, n \quad (11-5)$$

$$b_i \geq 0, \quad i = 1, \cdots, m, \quad m = m_1 + m_2 + m_n < n \quad (11-6)$$

子过程 SIMPLX 为求解线性规划问题的二阶段单纯形方法;子过程 SIMP1

确定判别系数中的最大值(或绝对值最大者);子过程 SIMP2 用于确定单纯形方法中的主元;子过程 SIMP3 用于交换一个基变量与一个非基变量,使该非基变量进入基变量中,即执行转轴运算.

2. 方法

(1) 基本方法.

考虑标准形式的线性规划问题:

$$\text{Max } z = \mathbf{a}^T \mathbf{x} \quad (11-7)$$

约束

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq 0, \mathbf{b} \geq 0$$

其中 $\mathbf{A} \in R^{m \times n}$, $\mathbf{a} \in R^n$, $\mathbf{x} \in R^n$, $\mathbf{b} \in R^m$, $\text{rank}(\mathbf{A}) = m < n$.

单纯形法的基本步骤是:

设 $\mathbf{A} = (\mathbf{A}_1, \dots, \mathbf{A}_n)$, $\mathbf{B} = (\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_m})$ 为一初始基.

① 对基 \mathbf{B} 有基可行解 $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} = (x_{i_1}, \dots, x_{i_m})$, $x_N = 0$, x_N 的分量为非基变量,该基可行解对应的目标函数值为 $z = \mathbf{a}_B^T \mathbf{x}_B$, $\mathbf{a}_B = (a_{i_1}, \dots, a_{i_m})^T$;

② 设 $\mathbf{W}^T = \mathbf{a}_B^T \mathbf{B}^{-1}$, 对所有非基变量计算判别系数 $\lambda_j = \mathbf{W}^T \mathbf{A}_j - a_j$ (或 $\mu_j = -\lambda_j$), $j \in R$, R 表示非基变量的下标集,令 $\lambda_k = \min_{j \in R} \{\lambda_j\}$ (或 $u_k = \max_{j \in R} \{u_j\}$, 对极小化问题 $\lambda_k = \max_{j \in R} \{\lambda_j\}$). 这里当有多个 k 可取时,取最小的 k ,以避免退化时基的循环.

若 $\lambda_k \geq 0$ (或 $\mu_k \leq 0$, 对极小化问题, $\lambda_k \leq 0$), 则这时已得到最优基可行解,停止计算,否则转下一步.

③ 计算 $\mathbf{y}_k = \mathbf{B}^{-1} \mathbf{A}_k = (y_{1k}, \dots, y_{mk})^T$ (或 $\mathbf{z}_k = -\mathbf{y}_k$), 若 $y_{rk} \leq 0$ (或 $z_k \geq 0$), 则停止计算,问题不存在有限最优解,即目标函数无界,否则转下一步.

④ 计算 $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b} = (\bar{b}_1, \dots, \bar{b}_m)^T$, 确定指标 r 使

$$\bar{b}_r / y_{rk} = \min_{i \in E} \{\bar{b}_i / y_{ik} : y_{ik} > 0\}$$

(或 $\bar{b}_r / z_{rk} = \min_{i \in E} \{\bar{b}_i / z_{ik} : z_{ik} < 0\}$), 这里 E 表示基变量的指标集,于是 x_r 为离基变量, x_k 为进基变量,用 \mathbf{A}_k 替换 \mathbf{A}_r 后得新的基 \mathbf{B} , 返回①.

若设 $\mathbf{A} = (\mathbf{B}, \mathbf{N})$, $\mathbf{B} \in R^{m \times m}$ 可逆,记 $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix}$, $\mathbf{a} = \begin{bmatrix} \mathbf{a}_B \\ \mathbf{a}_N \end{bmatrix} \in R^n$ (可能经列调换), 则式(11-7)等价于:

$$\begin{aligned} & \text{Max } z \\ \text{约束 } & z = (\mathbf{a}_B^T \mathbf{B}^{-1} \mathbf{N} - \mathbf{a}_N^T) \mathbf{x}_N = \mathbf{a}_B^T \mathbf{B}^{-1} \mathbf{b} \\ & \mathbf{x}_B + \mathbf{B}^{-1} \mathbf{N} \mathbf{x}_N = \mathbf{B}^{-1} \mathbf{b} \end{aligned}$$

$$x_B \geq 0, x_N \geq 0 \quad (11-8)$$

把式(11-8)的约束方程的系数置于表中即得单纯形表:

	b	x_B	x_N
z	$a_B^T B^{-1} b$	0	$\lambda^T = a_N^T B^{-1} N - a_N^T$
x_B	$\tilde{b} = B^{-1} b$	I_m	$Y = B^{-1} N$

写成分量形式: ($l = n - m$)

	b	$x_{i_1} x_{i_2} \cdots x_{i_m}$	$x_{j_1} x_{j_2} \cdots x_{j_l}$
z	$a_B^T \tilde{b}$	0 0...0	$\lambda_{j_1} \lambda_{j_2} \cdots \lambda_{j_l}$
x_{i_1}	\tilde{b}_1	1 0...0	$y_{1j_1} y_{1j_2} \cdots y_{1j_l}$
x_{i_2}	\tilde{b}_2	0 1...0	$y_{2j_1} y_{2j_2} \cdots y_{2j_l}$
\vdots	\vdots	\cdots	\cdots
x_{i_m}	\tilde{b}_m	0 0...1	$y_{mj_1} y_{mj_2} \cdots y_{mj_l}$

于是单纯形法的基本步骤如下:

① $(x_B, x_N) = (B^{-1}b, 0)$ 即是一基可行解, 对应的目标函数为 $z = a^T \tilde{b}$, $\tilde{b} = B^{-1}b$.

② 若 $\lambda \geq 0$, 则现行基可行解即为最优解.

③ 否则设 $\lambda_k = \min\{\lambda_j\}$, 当 $y_k = B^{-1}A_k = (y_{1k}, \cdots, y_{mk})^T \leq 0$ 时, 停止计算. 问题无有限最优解.

④ 否则选主元, 进行转轴运算. 令

$$\bar{b}_r / y_{rk} = \min_{i \in E} \{\bar{b}_i / y_{ik} \mid y_{ik} > 0\}$$

则 y_{rk} 即为主元.

转轴运算: 对单纯形表中双线中的数构成的矩阵进行主元消去法 (即把主元列 y_k 变为 $e_k = (0, \cdots, 0, 1, 0, \cdots, 0)^T \in R^m$, λ_k 也变为 0), 且交换基变元 x_r 与非基变元 x_k 使 x_k 进入基变元, 得新的单纯形表, 重复之, 直至找到最优解或确定无最优解.

(2) 二阶段方法.

① 引入松弛变量, 把线性规划问题式(11-1)~(11-6)化为标准形式:

$$\text{Max } z = \sum_{i=1}^n a_{0i} x_i \quad (11-9)$$

$$\text{约束 } \sum_{i=1}^n a_{ii} x_i + y_i = b_i, \quad i = 1, \cdots, m_1 \quad (11-10)$$

$$\sum_{l=1}^n a_{jl}x_l - y_j = b_j, \quad j = m_1 + 1, \dots, m_1 + m_2 \quad (11-11)$$

$$\sum_{l=1}^n a_{kl}x_l = b_k, \quad k = m_1 + m_2 + 1, \dots, m \quad (11-12)$$

$$x_l \geq 0, \quad l = 1, \dots, n; \quad b_i \geq 0, \quad i = 1, \dots, m \quad (11-13)$$

$$y_i \geq 0, \quad i = 1, \dots, m_1 + m_2; \quad m = m_1 + m_2 + m_3 \quad (11-14)$$

其中 $y_i (i=1, \dots, m_1+m_2)$ 为松弛变量.

② 第一阶段. 引入人工变量, 解辅助问题, 求初始基可行解.

辅助问题:

$$\text{Max } z' = - \sum_{i=1}^m Z_i \quad (11-15)$$

$$\text{约束 } z_i = b_i - \sum_{l=1}^n a_{il}x_l - y_i, \quad i = 1, \dots, m_1 \quad (11-16)$$

$$z_j = b_j - \sum_{l=1}^n a_{jl}x_l + y_j, \quad j = m_1 + 1, \dots, m_1 + m_2 \quad (11-17)$$

$$z_k = b_k - \sum_{l=1}^n a_{kl}x_l, \quad k = m_1 + m_2 + 1, \dots, m \quad (11-18)$$

$$x_l \geq 0, \quad l = 1, \dots, n; \quad y_i \geq 0, \quad i = 1, \dots, m_1 + m_2 \quad (11-19)$$

$$z_i \geq 0, \quad b_i \geq 0, \quad i = 1, \dots, m; \quad m = m_1 + m_2 + m_3 \quad (11-20)$$

辅助问题显然有基可行解 $x_l=0, y_i=0, z_j=b_j (i=1, \dots, m_1+m_2, l=1, \dots, n, j=1, \dots, m)$ 对应的基 $B=I_m$, 用单纯形法解辅助问题, 如果 $\max z' < 0$, 则说明原问题无可行解存在, 即原问题的约束方程不相容, 停止计算; 否则, 如果 $\max z' = 0$, 则得到辅助问题的一最优解. 当最优解的基变元中有人工变量时, 先用消元法驱赶人工变量离基后得标准问题的一基可行解, 即可进入第二阶段.

③ 第二阶段. 从第一阶段得到的基可行解(对应的基 $B=I_m$)出发, 用单纯形法求标准问题的最优解.

3. 使用说明

(1) 先说明子过程中使用的二维数组 A:

按线性规划问题式(11-1)~(11-6)的排列次序, 初始 A 为

$$A = \begin{bmatrix} 0 & a_{01} & a_{02} & \cdots & a_{0n} \\ b_1 & -a_{11} & -a_{12} & \cdots & -a_{1n} \\ b_2 & -a_{21} & -a_{22} & \cdots & -a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_m & -a_{m1} & -a_{m2} & \cdots & -a_{mn} \\ -\sum_{l=1}^m b_l & \sum_{l=1}^m a_{l1} & \sum_{l=1}^m a_{l2} & \cdots & \sum_{l=1}^m a_{ln} \end{bmatrix}_{(m+2) \times (n+1)}$$

$$m = m_1 + m_2 + m_3$$

其中 A 的前 $m+1$ 行是输入、输出参数,输入时按上述方式输入值,输出时存放单纯形表的最终结果;而 A 的第 $m+2$ 行是工作单元,为在第一阶段解辅助问题而用. 在 A 中,前 $m+1$ 行是第二阶段解标准问题的单纯形表,后 $m+1$ 行是第一阶段解辅助问题的单纯形表.

(2) SIMPLX (A,M,N,MP,NP,M1,M2,M3,ICASE,IZROW,IPOSV)

SIMP1(A,MP,NP,MM,LL,NLL,IABF,KP,BMAX)

SIMP2(A,M,N,MP,NP,L2,NL2,IP,KP,Q1)

SIMP3(A,MP,NP,I1,K1,IP,KP)

- M1 整型变量,输入参数,存放“ \leq -约束”即形式(11-2)的约束条件的个数
- M2 整型变量,输入参数,存放“ \geq -约束”即形式(11-3)的约束条件的个数
- M3 整型变量,输入参数,存放等式约束即形式(11-4)的约束条件的个数
- M 整型变量,输入参数,约束条件的总个数, $M = M1 + M2 + M3$
- N 整型变量,输入参数,存放自变量的总个数
- MP,NP 整型变量,输入参数,存储数组的物理维数, $MP \geq M+2$, $NP \geq N+1$,见各数组的说明
- A $(M+2) \times (N+1)$ 个元素的二维实型数组,输入、输出参数,见“方法”中的说明,输出时最优基可行解(如果存在的话)存放在 A 的第一列中,它还由两个整型数组 IPOSV 与 IZROV 来说明,存储 A 的物理维数为 $MP \times NP$
- IPOSV M 个元素的一维整型数组,输出参数,用于说明 A 的输出结果,若 $IPOSV(J) = i$,则当 $i < N$ 时,说明原自变量 x_i 由 A 的第 $J+1$ 行表示,从而 x_i 是基变量,当 $i > N$ 时,则说明的变量是松弛变

- 量 y_i 而不是 $x_i, J=1, \dots, M$
- IZROV N 个元素的一维整型数组, 输出参数, 用于说明 A 的输出结果
若 $\text{IZROV}(J)=i$, 则当 $i < N$ 时, 说明原自变量 x_i 为非基变量, 它由 A 的第 $J+1$ 列表示, 从而在最优解中其值为 $x_i=0$. 当 $i > N$ 时, 则说明是松弛变量 y_i 而不是 x_i , 当 $i > N+M1+M2$ 时, 则说明是一人工变量, $J=1, \dots, N$
- ICASE 整型变量, 输出参数, 说明结果的标志量. 若 $\text{ICASE}=0$, 则表示求得一有限的最优解, 若 $\text{ICASE}=1$, 则表示目标函数是无界的, 即没有有限的最优解, 若 $\text{ICASE}=-1$, 则表示原问题无可行解
- MM 整型变量, 输入参数, 表示要在 A 的第 $MM+1$ 行中求最大元, $MM \leq M+1$
- NLL 整型变量, 输入参数, 需要比较元素的个数
- LL NLL 个元素的一维整型数组, 要比较的元素的指标集, 存储它的物理维数是 NP
- IABF 整型变量, 输入参数, 若输入的 $\text{IABF} \neq 0$, 则按绝对值求最大元, 若 $\text{IABF}=0$, 则直接求最大元
- KP 整型变量, 在子过程 SIMP1 中为输出参数, 存放最大元的指标, 在子过程 SIMP2 与 SIMP3 中为输入参数, 表示 A 的第 $KP+1$ 列为主列
- BMAX 实型变量, 输出参数, 存放最大元
- NL2 整型变量, 输入参数, 主列中求主元时所比较的元素的个数
- L2 $NL2$ 个元素的一维整型数组, 输入参数, 主列中求主元时所比较的元素的指标集, 存储它的物理维数为 MP
- IP 整型变量, 输出参数, 表示主元所在的行为第 $IP+1$ 行, 在子过程 SIMP3 中, 它为输入参数, 意义相同
- Q1 实型变量, 输出参数, 存放主行的常数项 \bar{b}_{IP+1} 与主元之比
- I1 整型变量, 输入参数, 转轴运算中的最大行号减 1
- K1 整型变量, 输入参数, 转轴运算中的最大列号减 1

4. 过程

(1) 子过程 SIMPLX.

```
SUBROUTINE simplx(a,m,n,np,np,m1,m2,m3,icase,izrov,&
    iposv)
```



```

INTEGER  icase,m,m1,m2,m3,mp,n,np,iposv(m),izrov(n),&
          MMAX,NMAX
REAL  a(mp,np),EPS
PARAMETER (MMAX=100,NMAX=100,EPS=1.e-6)
! USES simp1,simp2,simp3
INTEGER  i,ip,ir,is,k,kh,kp,m12,nl1,nl2,l1(NMAX),&
          l2(MMAX),l3(MMAX)
REAL  bmax,q1
if(m/=m1+m2+m3)&
    pause 'bad input constraint counts in simplex'
nl1=n
do k=1,n
    l1(k)=k
    izrov(k)=k
end do
nl2=m
do i=1,m
    if(a(i+1,1)<0.) pause 'bad input tableau in simplex'
    l2(i)=i
    iposv(i)=n+i
end do
do i=1,m2
    l3(i)=i
end do
ir=0
if(m2+m3==0) goto 3
ir=1
do k=1,n+1
    q1=0.
    do i=m1+1,m
        q1=q1+a(i+1,k)
    end do
    a(m+2,k)=-q1
end do
do
    call simp1(a,mp,np,m+1,l1,nl1,0,kp,bmax)
    if(bmax<=EPS. and. a(m+2,1)<-EPS) then

```

```

    icode = -1
    return
else if (bmax <= EPS. and. a(m+2,1) <= EPS) then
    m12 = m1 + m2 + 1
    do ip = m12, m
        if (iposv(ip) == ip + n) then
            call simp1(a, mp, np, ip, l1, n1, l, kp, bmax)
            if (bmax > 0.) goto 1
        endif
    end do
    ir = 0
    m12 = m12 - 1
    do i = m1 + 1, m12
        if (l3(i - m1) == 1) then
            do k = 1, n + 1
                a(i + 1, k) = -a(i + 1, k)
            end do
        endif
    end do
    exit
endif
call simp2(a, m, n, mp, np, l2, n12, ip, kp, q1)
if (ip == 0) then
    icode = -1
    return
endif
1 call simp3(a, mp, np, m + 1, n, ip, kp)
if (iposv(ip) >= n + m1 + m2 + 1) then
    do k = 1, n11
        if (l1(k) == kp) exit
    end do
    n11 = n11 - 1
    do is = k, n11
        l1(is) = l1(is + 1)
    end do
else
    if (iposv(ip) < n + m1 + 1) goto 2

```

```

      kh=iposv(ip)-m1--n
      if(l3(kh)==0) goto 2
      l3(kh)=0
    endif
    a(m+2,kp+1)=a(m+2,kp+1)+1.
    do i=1,m+2
      a(i,kp+1)=-a(i,kp+1)
    end do
2 is=izrov(kp)
  izrov(kp)=iposv(ip)
  iposv(ip)=is
  if(ir/=0) exit
end do
3 call simp1(a,mp,np,0,l1,nl1,0,kp,bmax)
  if(bmax<=0.) then
    icode=0
    return
  endif
  call simp2(a,m,n,mp,np,l2,nl2,ip,kp,q1)
  if(ip==0) then
    icode=1
    return
  endif
  call simp3(a,mp,np,m,n,ip,kp)
  goto 2
END SUBROUTINE simplx

```

(2) 子过程 SIMP1.

```

SUBROUTINE simp1(a,mp,np,mm,ll,nll,iabf,kp,bmax)
INTEGER iabf,kp,mm,mp,nll,np,ll(np)
REAL bmax,a(mp,np)
INTEGER k
REAL test
kp=ll(1)
bmax=a(mm+1,kp+1)
do k=2,nll
  if(iabf==0)then

```

```

      test=a(mm+1,ll(k)+1)-bmax
    else
      test=abs(a(mm+1,ll(k)+1))-abs(bmax)
    endif
    if(test>0.)then
      bmax=a(mm+1,ll(k)+1)
      kp=ll(k)
    endif
  end do
END SUBROUTINE simp1

```

(3) 子过程 SIMP2.

```

SUBROUTINE simp2(a,m,n,mp,np,l2,nl2,ip,kp,q1)
INTEGER ip,kp,m,mp,n,nl2,np,l2(mp)
REAL q1,a(mp,np),EPS
PARAMETER (EPS=1.e-6)
INTEGER i,ii,k
REAL q,q0,qp,flag
ip=0
flag=0
do i=1,nl2
  if(a(l2(i)+1,kp+1)<-EPS) flag=1
  if(flag==1) exit
end do
if(flag==0) return
q1=-a(l2(i)+1,1)/a(l2(i)+1,kp+1)
ip=l2(i)
do i=i+1,nl2
  ii=l2(i)
  if(a(ii+1,kp+1)<-EPS) then
    q=-a(ii+1,1)/a(ii+1,kp+1)
    if(q<q1) then
      ip=ii
      q1=q
    else if (q==q1) then
      do k=1,n
        qp=-a(ip+1,k+1)/a(ip+1,kp+1)

```

```

        q0 = -a(ii+1,k+1)/a(ii+1,kp+1)
        if(q0/=qp) exit
    end do
    if(q0<qp) ip=ii
endif
endif
end do
END SUBROUTINE simp2

```

(4) 子过程 SIMP3.

```

SUBROUTINE simp3(a,mp,np,i1,k1,ip,kp)
INTEGER i1,ip,k1,kp,mp,np
REAL a(mp,np)
INTEGER ii,kk
REAL piv
piv=1./a(ip+1,kp+1)
do ii=1,i1+1
    if(ii-1/=ip) then
        a(ii,kp+1)=a(ii,kp+1)*piv
        do kk=1,k1+1
            if(kk-1/=kp) then
                a(ii,kk)=a(ii,kk)-a(ip+1,kk)*a(ii,kp+1)
            endif
        end do
    endif
end do
end do
do kk=1,k1+1
    if(kk-1/=kp) a(ip+1,kk)=-a(ip+1,kk)*piv
end do
a(ip+1,kp+1)=piv
END SUBROUTINE simp3

```

5. 例子

子过程 SIMPLX 涉及到线性规划问题. 在这种问题中, 目标是求 N 个非负变量线性组合的极大值, 且这些变量还要满足一些其他的约束. 我们所采用的例子是

$$\text{Max } z = x_1 + x_2 + 3x_3 - 0.5x_4$$

且满足下列约束

$$x_1 + 2x_3 \leq 740$$

$$2x_2 - 7x_4 \leq 0$$

$$x_2 - x_3 + 2x_4 \geq 0.5$$

$$x_1 + x_2 + x_3 + x_4 = 9$$

当然,还有

$$x_1, x_2, x_3, x_4 \geq 0$$

这个例子的解为: $x_1=0$, $x_2=3.33$, $x_3=4.73$, $x_4=0.95$. 下面我们利用该节所介绍的方法采用子过程 SIMPLX 解这个问题.

首先写出调用子过程 SIMPLX 时的输入矩阵 A . 注意只需写出 A 的前 $m+1$ 行, 最后的第 $m+2$ 行可任意, 现在

$$m_1 = 2, \quad m_2 = 1, \quad m_3 = 1, \quad m = m_1 + m_2 + m_3 = 4, \quad n = 4$$

$$A = \begin{bmatrix} 0 & a_{01} & a_{02} & a_{03} & a_{04} \\ b_1 & -a_{11} & -a_{12} & -a_{13} & -a_{14} \\ b_2 & -a_{21} & -a_{22} & -a_{23} & -a_{24} \\ b_3 & -a_{31} & -a_{32} & -a_{33} & -a_{34} \\ b_4 & -a_{41} & -a_{42} & -a_{43} & -a_{44} \\ x & x & x & x & x \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 1 & 3 & -\frac{1}{2} \\ 740 & -1 & 0 & -2 & 0 \\ 0 & 0 & -2 & 0 & 7 \\ \frac{1}{2} & 0 & -1 & 1 & -2 \\ 9 & -1 & -1 & -1 & -1 \\ x & x & x & x & x \end{bmatrix}$$

调用子过程 SIMPLX, 或者无可行解, 或者目标函数无界, 或者得到一个有限的最优解. 在第三种情况下, 打印最终的单纯形表.

验证程序 D11R10 如下:

```
PROGRAM D11R10
```

```
! Driver for routine SIMPLX
```

```
! Incorporates examples discussed in text
```

```
PARAMETER(N=4,M=4,NP=5,MP=6,M1=2,M2=1,M3=1,NM1M2=N+M1  
+M2)
```

```
DIMENSION A(MP,NP),IZROV(N),IPOSV(M),ANUM(NP)
```

```

CHARACTER TXT(NM1M2)*2,ALPHA(NP)*2
DATA TXT/'x1','x2','x3','x4','y1','y2','y3'/
DATA a/0.0,740.0,0.0,0.5,9.0,0.0,1.0,-1.0,0.0,0.0,-1.0,&
      0.0,1.0,0.0,-2.0,-1.0,-1.0,0.0,3.0,-2.0,0.0,1.0,&
      -1.0,0.0,-0.5,0.0,7.0,-2.0,-1.0,0.0/
CALL SIMPLX(A,M,N,MP,NP,M1,M2,M3,ICASE,IZROV,IPOSV)
IF (ICASE==1) THEN
  WRITE(*,*) 'Unbounded objective function'
ELSE IF (ICASE==-1) THEN
  WRITE(*,*) 'No solutions satisfy constraints given'
ELSE
  JJ=1
  DO I=1,N
    IF (IZROV(I)<=(N+M1+M2)) THEN
      ALPHA(JJ)=TXT(IZROV(I))
      JJ=JJ+1
    ENDIF
  END DO
  JMAX=JJ-1
  WRITE(*, '(/3X,5A10)') ' ',(ALPHA(JJ),JJ=1,JMAX)
  DO I=1,M+1
    IF (I>1) THEN
      ALPHA(1)=TXT(IPOSV(I-1))
    ELSE
      ALPHA(1)=' '
    ENDIF
    ANUM(1)=A(1,1)
    JJ=2
    DO J=2,N+1
      IF (IZROV(J-1)<=(N+M1+M2)) THEN
        ANUM(JJ)=A(1,J)
        JJ=JJ+1
      ENDIF
    END DO
    JMAX=JJ-1
    WRITE(*, '(1X,A3,(5F10.2))') ALPHA(1),&
      (ANUM(JJ),JJ=1,JMAX)

```

```
END DO
ENDIF
END
```

计算结果如下:

		x1	y2	y3
	17.02	-0.95	-0.05	-1.05
y1	730.55	0.10	-0.10	0.90
x2	3.33	-0.35	-0.15	0.35
x4	0.95	-0.10	0.10	0.10
x3	4.73	-0.55	0.05	-0.45

第 12 章 傅里叶变换谱方法

本章有 10 个快速算法. 在 12.1 节中, 介绍了计算复数据离散傅里叶 (Fourier) 变换 (DFT) 的快速算法, 它结构直观清晰, 计算速度较快; 12.2 节和 12.3 节分别介绍了对两个实数序列作 DFT 和仅对一个实数序列计算其 DFT 的快速算法, 它们比直接调用复数据快速傅里叶变换 (FFT) 算法的计算效率高一倍; 12.4 节中介绍了快速正弦和余弦变换的算法; 在 12.5 节中, 介绍了用 FFT 算法计算卷积 (逆卷积) 的算法.

我们知道, 一个平稳过程可用相关函数、谱密度和平稳模型三种不同的方式进行分析和描述, 它们从不同方面说明了同一随机过程的统计性质. 在本章的 12.6 节中, 是计算相关 (自相关) 的快速算法. 在 12.7 节中, 我们还给出了多维复数据 DFT 的快速算法.

12.1 复数据快速傅里叶变换算法

1. 功能

计算离散傅里叶变换

$$X(j) = \sum_{k=0}^{N-1} X_0(k) W^{jk}, \quad j = 0, 1, \dots, N-1 \quad (12-1)$$

的值, 其中 $W = \exp(-2\pi i/N)$ 或 $W = \exp(2\pi i/N)$, $i = \sqrt{-1}$, $X_0(k)$ ($k=0, 1, \dots, N-1$) 是变换初值, 即采样点数据, 可取为复数; N 是采样点数, $X(j)$ 是变换结果, 为复数.

2. 方法

如果直接使用式 (12-1) 计算傅里叶变换, 大约需要 N^2 次复数乘加运算, 而采用快速傅里叶变换, 只需要 $N \log_2 N$ 次复数乘加运算.

本算法采用先反序、反变换的方法计算傅里叶变换, 有较高的运算速度. 现以 $N=2^3$ 说明算法的思想.

将下标 j 和 k 表示为二进制

$$j = (j_2, j_1, j_0) = j_2 2^2 + j_1 2 + j_0, \quad j_0, j_1, j_2 = 0, 1$$

$$k = (k_2, k_1, k_0) = k_2 2^2 + k_1 2 + k_0, \quad k_0, k_1, k_2 = 0, 1$$

由于

$$\begin{aligned} jk &= (j_2 2^2 + j_1 2 + j_0)(k_2 2^2 + k_1 2 + k_0) \\ &= k_2(j_0 2^2) + k_1(j_1 2^2 + j_0 2) + k_0(j_2 2^2 + j_1 2 + j_0) \\ &\quad + 2^3(k_2 j_2 2 + j_1 k_2 + k_1 j_2) \end{aligned}$$

故有

$$W^{jk} = W^{k_2(j_0, 0, 0)} W^{k_1(j_1, j_0, 0)} W^{k_0(j_2, j_1, j_0)}$$

令

$$X(j) = X(j_2, j_1, j_0), \quad X_0(k) = X_0(k_2, k_1, k_0)$$

于是

$$\begin{aligned} X(j_2, j_1, j_0) &= \sum_{k_0=0}^1 \sum_{k_1=0}^1 \sum_{k_2=0}^1 X_0(k_2, k_1, k_0) W^{k_2(j_0, 0, 0)} W^{k_1(j_1, j_0, 0)} W^{k_0(j_2, j_1, j_0)} \\ &= \sum_{k_0=0}^1 \left\{ \sum_{k_1=0}^1 \left[\sum_{k_2=0}^1 X_0(k_2, k_1, k_0) W^{k_2(j_0, 0, 0)} \right] W^{k_1(j_1, j_0, 0)} \right\} W^{k_0(j_2, j_1, j_0)} \end{aligned}$$

这样就导出了递推过程如下:

$$X^{(0)}(k_2, k_1, k_0) = X_0(k_2, k_1, k_0)$$

$$\begin{aligned} X^{(1)}(j_0, k_1, k_0) &= \sum_{k_2=0}^1 X^{(0)}(k_2, k_1, k_0) W^{k_2(j_0, 0, 0)} \\ &= X^{(0)}(0, k_1, k_0) + X^{(0)}(1, k_1, k_0) W^{(j_0, 0, 0)} \end{aligned}$$

$$\begin{aligned} X^{(2)}(j_0, j_1, k_0) &= \sum_{k_1=0}^1 X^{(1)}(j_0, k_1, k_0) W^{k_1(j_1, j_0, 0)} \\ &= X^{(1)}(j_0, 0, k_0) + X^{(1)}(j_0, 1, k_0) W^{(j_1, j_0, 0)} \end{aligned}$$

$$\begin{aligned} X^{(3)}(j_0, j_1, j_2) &= \sum_{k_0=0}^1 X^{(2)}(j_0, j_1, k_0) W^{k_0(j_2, j_1, j_0)} \\ &= X^{(2)}(j_0, j_1, 0) + X^{(2)}(j_0, j_1, 1) W^{(j_2, j_1, j_0)} \end{aligned}$$

$$X(j_2, j_1, j_0) = X^{(3)}(j_0, j_1, j_2)$$

本算法开始作按位反序,以后不再作反序,算法为

$$X^{(0)}(j_2, j_1, j_0) = X_0(j_0, j_1, j_2)$$

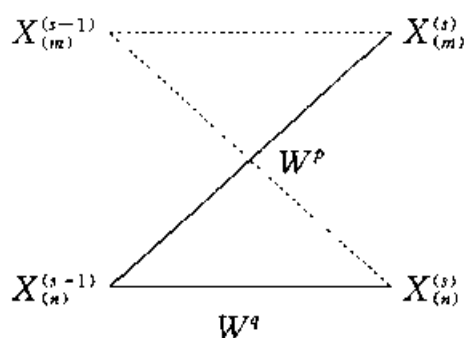
$$X^{(1)}(j_2, j_1, j_0) = X^{(0)}(j_2, j_1, 0) + X^{(0)}(j_2, j_1, 1) W^{(j_0, 0, 0)}$$

$$X^{(2)}(j_2, j_1, j_0) = X^{(1)}(j_2, 0, j_0) + X^{(1)}(j_2, 1, j_0) W^{(j_1, j_0, 0)}$$

$$X^{(3)}(j_2, j_1, j_0) = X^{(2)}(0, j_1, j_0) + X^{(2)}(1, j_1, j_0) W^{(j_2, j_1, j_0)}$$

$$X(j_2, j_1, j_0) = X^{(3)}(j_2, j_1, j_0)$$

这里,每步的 W 的幂次 $(j_0, 0, 0), (j_1, j_0, 0), (j_2, j_1, j_0)$ 由 (j_2, j_1, j_0) 分别左移 2, 1, 0 位而得. 这样,计算中每一步,都把变换的分量两两配对,每一对是一个“蝶形”计算:



其中虚线表示相加,实线表示乘以 W 的相当幂次后再相加,即

$$X_{(m)}^{(s)} = X_{(m)}^{(s-1)} + X_{(n)}^{(s-1)}W^p, \quad X_{(n)}^{(s)} = X_{(m)}^{(s-1)} + X_{(n)}^{(s-1)}W^q$$

上述显然可以推广到一般的 $N=2^M$ 的情况,这时,从 0 到 $N-1$ 的任意整数 j 可以表示为二进制:

$$j = (j_{M-1}, j_{M-2}, \dots, j_1, j_0) = j_{M-1}2^{M-1} + \dots + j_12 + j_01$$

$$j_{M-1}, \dots, j_1, j_0 = 0, 1; \quad 0 \leq j \leq N-1$$

于是有

$$X^{(0)}(j_{M-1}, \dots, j_1, j_0) = X_0(j_0, j_1, \dots, j_{M-1})$$

$$X^{(s)}(j_{M-1}, \dots, j_1, j_0) = X^{(s-1)}(j_{M-1}, \dots, j_s, 0, j_{s-2}, \dots, j_0) \\ + X^{(s-1)}(j_{M-1}, \dots, j_s, 1, j_{s-2}, \dots, j_0) \times W^{(j_{s-1}, \dots, j_0, 0, \dots, 0)} \\ (s = 1, 2, \dots, M)$$

$$X(j_{M-1}, \dots, j_1, j_0) = X^{(M)}(j_{M-1}, j_{M-2}, \dots, j_1, j_0) \\ j_{M-1}, \dots, j_1, j_0 = 0, 1$$

这是一个简单而有效的算法,实践表明,即使对于很大的 N ,计算过程也是稳定的.

另外, W^k 的计算采用递推公式进行:

$$\begin{cases} \sin((p+1)\theta) = \sin(p\theta) + s(p+1) \\ s(p+1) = s(p) + D_0 \sin(p\theta) \\ \cos((p+1)\theta) = \cos(p\theta) + c(p+1) \\ c(p+1) = c(p) + D_0 \cos(p\theta) \\ p = 0, 1, \dots, N-1 \end{cases}$$

其中

$$\begin{cases} D_0 = -4\sin^2(\theta/2) \\ s(0) = \sin\theta, \quad c(0) = 1 - \cos\theta \end{cases}$$

这里的 c 和 s 分别为 W^p 的实部和虚部.

3. 使用说明

FOUR1(NN,DATA,ISIGN)

NN 整型变量,输入参数,采样点数,要求 NN 为 2 的整数次幂,即 $NN=2^m$, m 为正整数

DATA 含 NN 个元素的复矩阵,或等价地说含 $2 \times NN$ 个元素的一组实型数组,输入、输出参数,开始时存放变换初值 $X_0(k)$ ($k=0, 1, \dots, N-1$),最后存放变换结果 $X(j)$ ($j=0, 1, \dots, N-1$)

ISIGN 整型变量,输入参数,其值为 ± 1 . 当输入 -1 时,本程序的计算结果乘以 $1/(NN)$ 则为离散傅里叶逆变换;若输入 $+1$,则是计算离散傅里叶变换

4. 过程

子过程 FOUR1.

SUBROUTINE four1(data1,nn,isign)

INTEGER isign,nn

REAL data1(2 * nn)

INTEGER i,istep,j,m,mmax,n

REAL tempi,tempr

DOUBLE PRECISION theta,wi,wpi,wpr,wr,wtemp

n=2 * nn

j=1

do i=1,n,2

if(j>i)then

tempr=data1(j)

tempi=data1(j+1)

data1(j)=data1(i)

data1(j+1)=data1(i+1)

data1(i)=tempr

data1(i+1)=tempi

endif

m=n/2

do while((m>=2). and. (j>m))

j=j-m

```

      m=m/2
    end do
    j=j+m
  end do
  mmax=2
  do while(n>mmax)
    istep=2 * mmax
    theta=6.28318530717959d0/(isign * mmax)
    wpr=-2.d0 * dsin(0.5d0 * theta) * * 2
    wpi=dsin(theta)
    wr=1.d0
    wi=0.d0
    do m=1,mmax,2
      do i=m,n,istep
        j=i+mmax
        tempr=sngl(wr) * data1(j)-sngl(wi) * data1(j+1)
        tempi=sngl(wr) * data1(j+1)+sngl(wi) * data1(j)
        data1(j)=data1(i)-tempr
        data1(j+1)=data1(i+1)-tempi
        data1(i)=data1(i)-tempr
        data1(i+1)=data1(i+1)+tempi
      end do
      wtemp=wr
      wr=wr * wpr-wi * wpi+wr
      wi=wi * wpr+wtemp * wpi+wi
    end do
    mmax=m+istep
  end do
  END SUBROUTINE four1

```

5. 例子

验证子过程 FOUR1 的程序 D12R1 中,对傅里叶变换做了五种检验.首先,它检验了下列四种对称性(其中 $h(t)$ 是数据, $H(n)$ 是变换):

- (1) 如果 $h(t)$ 是实的且为偶函数,那么 $H(n)=H(N-n)$ 且 H 是实的.
- (2) 如果 $h(t)$ 是虚的且为偶函数,那么 $H(n)=H(N-n)$ 且 H 是虚的.
- (3) 如果 $h(t)$ 是实的且为奇函数,那么 $H(n)=-H(N-n)$ 且 H 是虚的.
- (4) 如果 $h(t)$ 是虚的且为奇函数,那么 $H(n)=-H(N-n)$ 且 H 是实的.

第五种检验是,如果一组数据连续做两次傅里叶变换,即第一次为变换接着做一次反变换,这时所得结果应为原来的数据.验证程序 D12R1 如下:

```

PROGRAM D12R1
  ! Driver for routine FOUR1
  PARAMETER (NN=32,NN2=2 * NN)
  DIMENSION DATA(NN2),DCMP(NN2)
  WRITE(*,*) 'h(t)=real-valued even-function'
  WRITE(*,*) 'H(n)=H(N-n) and real?'
  DO I=1,2 * NN-1,2
    DATA(I)=1.0/(((I-NN-1.0)/NN) * * 2+1.0)
    DATA(I+1)=0.0
  END DO
  ISIGN=1
  CALL FOUR1(DATA,NN,ISIGN)
  CALL PRNTFT(DATA,NN2)
  WRITE(*,*) 'h(t)=imaginary-valued even-function'
  WRITE(*,*) 'H(n)=H(N-n) and imaginary?'
  DO I=1,2 * NN-1,2
    DATA(I+1)=1.0/(((I-NN-1.0)/NN) * * 2+1.0)
    DATA(I)=0.0
  END DO
  ISIGN=1
  CALL FOUR1(DATA,NN,ISIGN)
  CALL PRNTFT(DATA,NN2)
  WRITE(*,*) 'h(t)=real-valued odd-function'
  WRITE(*,*) 'H(n)=H(N-n) and imaginary?'
  DO I=1,2 * NN-1,2
    DATA(I)=(I-NN-1.0)/NN/(((I-NN-1.0)/NN) * * 2+1.0)
    DATA(I+1)=0.0
  END DO
  DATA(1)=0.0
  ISIGN=1
  CALL FOUR1(DATA,NN,ISIGN)
  CALL PRNTFT(DATA,NN2)
  WRITE(*,*) 'h(t)=imaginary-valued odd-function'
  WRITE(*,*) 'H(n)=-H(N-n) and real?'

```

```

DO I=1,2*NN-1,2
  DATA(I+1)=(I-NN-1.0)/NN/(((I-NN-1.0)/NN)**2+1.0)
  DATA(I)=0.0
END DO
DATA(2)=0.0
ISIGN=1
CALL FOUR1(DATA,NN,ISIGN)
CALL PRNTFT(DATA,NN2)
! Transform, inverse-transform test
DO I=1,2*NN-1,2
  DATA(I)=1.0/((0.5*(I-NN-1)/NN)**2+1.0)
  DCMP(I)=DATA(I)
  DATA(I+1)=(0.25*(I-NN-1)/NN)*%
    EXP(-(0.5*(I-NN-1)/NN)**2)
  DCMP(I+1)=DATA(I+1)
END DO
ISIGN=1
CALL FOUR1(DATA,NN,ISIGN)
ISIGN=-1
CALL FOUR1(DATA,NN,ISIGN)
WRITE(*, '(/IX,T10,A,T44,A)')%
  'Double Fourier Transform:', 'Original Data:'
WRITE(*, '(/IX,T5,A,T11,A,T24,A,T41,A,T53,A)')%
  'k', 'Real h(k)', 'Imag h(k)', 'Real h(k)', 'Imag h(k)'
DO I=1,NN,2
  J=(I+1)/2
  WRITE(*, '(IX,I4,2X,2F12.6,5X,2F12.6)') J,DCMP(I),%
    DCMP(I+1),DATA(I)/NN,DATA(I+1)/NN
END DO
END PROGRAM

SUBROUTINE PRNTFT(DATA,NN2)
DIMENSION DATA(NN2)
WRITE(*, '(/IX,T5,A,T11,A,T23,A,T39,A,T52,A)') 'n',%
  'Real H(n)', 'Imag H(n)', 'Real H(N-n)', 'Imag H(N-n)'
WRITE(*, '(IX,I4,2X,2F12.6,5X,2F12.6)') 0,DATA(1),%
  DATA(2),DATA(1),DATA(2)
DO N=3,(NN2/2)+1,2

```

```

M=(N-1)/2
MM=NN2+2-N
WRITE(*,'(1X,I4,2X,2F12.6,5X,2F12.6)') M,DATA(N),&
DATA(N+1),DATA(MM),DATA(MM+1)
END DO
WRITE(*,'(/1X,A)') 'Press RETURN to continue ...'
READ(*,*)
END SUBROUTINE PRNTFT

```

计算结果如下:

$h(t)$ =real-valued even-function

$H(n)=H(N-n)$ and real?

n	Real H(n)	Imag H(n)	Real H(N-n)	Imag H(N-n)
0	25.127533	0.000000	25.127533	0.000000
1	-3.622967	0.000000	-3.622967	0.000000
2	-0.310545	0.000000	-0.310545	0.000000
3	-0.188846	0.000000	-0.188846	0.000000
	:	:	:	:
13	-0.017063	0.000000	-0.017063	0.000000
14	-0.016243	0.000000	-0.016243	0.000000
15	-0.015776	0.000000	-0.015776	0.000000
16	-0.015625	0.000000	-0.015625	0.000000

Press RETURN to continue ...

$h(t)$ =imaginary-valued even-function

$H(n)=H(N-n)$ and imaginary?

n	Real H(n)	Imag H(n)	Real H(N-n)	Imag H(N-n)
0	0.000000	25.127533	0.000000	25.127533
1	0.000000	-3.622967	0.000000	-3.622967
2	0.000000	-0.310545	0.000000	-0.310545
3	0.000000	-0.188846	0.000000	-0.188846
	:	:	:	:
13	0.000000	-0.017063	0.000000	-0.017063
14	0.000000	-0.016243	0.000000	-0.016243
15	0.000000	-0.015776	0.000000	-0.015776
16	0.000000	-0.015625	0.000000	-0.015625

Press RETURN to continue ...

$h(t)$ =real-valued odd-function

$H(n) = H(N-n)$ and imaginary?

n	Real H(n)	Imag H(n)	Real H(N-n)	Imag H(N-n)
0	0.000000	0.000000	0.000000	0.000000
1	0.000000	-7.495315	0.000000	7.495315
2	0.600000	-2.473274	0.000000	2.173271
3	0.000000	-1.670007	0.000000	1.670007
	⋮	⋮	⋮	⋮
13	0.000000	-0.151834	0.000000	0.151834
14	0.000000	-0.099557	0.000000	0.099557
15	0.000000	-0.049294	0.000000	0.049294
16	0.000000	0.000000	0.000000	0.000000

Press RETURN to continue ...

$h(t)$ = imaginary-valued odd-function

$H(n) = -H(N-n)$ and real?

n	Real H(n)	Imag H(n)	Real H(N-n)	Imag H(N-n)
0	0.000000	0.000000	0.000000	0.000000
1	7.495315	0.000000	-7.495315	0.000000
2	2.473274	0.000000	-2.473274	0.000000
3	1.670007	0.000000	-1.670007	0.000000
	⋮	⋮	⋮	⋮
13	0.151834	0.000000	-0.151834	0.000000
14	0.099557	0.000000	-0.099557	0.000000
15	0.049294	0.000000	-0.049294	0.000000
16	0.000000	0.000000	0.000000	0.000000

Press RETURN to continue ...

Double Fourier Transform;

Original Data;

k	Real h(k)	Imag h(k)	Real h(k)	Imag h(k)
1	0.800000	-0.194700	0.800000	-0.194700
2	0.819856	-0.188142	0.819856	-0.188142
3	0.839344	0.180643	0.839344	-0.180643
4	0.858340	-0.172222	0.858340	-0.172222
	⋮	⋮	⋮	⋮
13	0.984615	-0.061531	0.984615	-0.061531
14	0.991288	-0.046465	0.991288	-0.046465
15	0.996109	-0.031128	0.996109	-0.031128
16	0.999024	-0.015610	0.999024	-0.015610

12.2 实数据快速傅里叶变换算法(一)

1. 功能

在应用 FFT 时,经常碰到同时计算两个实时间函数而频率函数是复函数的情况.本算法采用双道组合法,考虑在采样值为实数据的条件下,将两组采样值分别作为变换的实部和虚部,从而同时算出两组实数据的傅里叶变换.

2. 方法

设 $\{x_1(j)\}$ 和 $\{x_2(j)\}$ 是各有 $N=2^m$ 个点的实数序列,并设它们的离散傅里叶变换分别为 $\{A_1(k)\}$ 和 $\{A_2(k)\}$,即

$$\{x_1(j)\} \leftrightarrow \{A_1(k)\}, \quad \{x_2(j)\} \leftrightarrow \{A_2(k)\}$$

则计算 $\{A_1(k)\}$ 和 $\{A_2(k)\}$ 的步骤如下:

(1) 用这两个实数序列构成一复数序列(即双道组合)

$$x(j) = x_1(j) + ix_2(j), \quad j = 0, 1, \dots, N-1$$

(2) 对复数序列 $\{x(j)\}$ 使用 FFT 程序计算它的离散傅里叶变换,并记为 $\{A(k)\}$.

因离散傅里叶变换具有线性,因此有

$$A(k) = A_1(k) + iA_2(k), \quad k = 0, 1, \dots, N-1 \quad (12-2)$$

用 $N-k$ 代替 k ,在式(12-2)两边取共轭,并利用关系

$$A(k) = A^*(N-k), \quad k = 0, 1, \dots, N-1 \quad (12-3)$$

得到

$$A^*(N-k) = A_1(k) - iA_2(k) \quad (12-4)$$

由式(12-2)和(12-4)得

$$\begin{cases} A_1(k) = 0.5[A(k) + A^*(N-k)] \\ A_2(k) = [A(k) - A^*(N-k)]/(2*i) \end{cases} \quad k = 0, 1, \dots, N/2 \quad (12-5)$$

(3) 利用上面的公式(12-5)计算 $\{A_1(k)\}$ 和 $\{A_2(k)\}$ ($k=0, 1, \dots, N/2$),由式(12-3)可得出 $\{A_1(k)\}$ 和 $\{A_2(k)\}$ ($k=N/2+1, \dots, N-1$).

3. 使用说明

(1) TWOFFT (DATA1,DATA2,FFT1,FFT2,N)

N 整型变量,输入参数,采样点的个数,要求 $N=2^m$, m 为正整数

- DATA1 实型数组 DATA1(N), 输入参数, 存放第一组实采样数据
 DATA2 实型数组 DATA2(N), 输入参数, 存放第二组实采样数据
 FFT1 复型数组 FFT1($N+1$), 输出参数, 其中 $\{\text{FFT1}(k)\} (k=1, 2, \dots, N)$ 为实数序列 $\{\text{DATA1}(j)\} (j=1, 2, \dots, N)$ 的离散傅里叶变换
 FFT2 复型数组 FFT2($N+1$), 输出参数, 其中 $\{\text{FFT2}(k)\} (k=1, 2, \dots, N)$ 为实数序列 $\{\text{DATA2}(j)\} (j=1, 2, \dots, N)$ 的离散傅里叶变换

(2) 要调用的子过程:

FOUR1(NN, DATA, 1)

各参数定义参见 12.1 节.

4. 过程

子过程 TWOFFT.

```
SUBROUTINE twofft(data1,data2,fft1,fft2,n)
INTEGER n
REAL data1(n),data2(n)
COMPLEX fft1(n),fft2(n)
! USES four1
INTEGER j,n2
COMPLEX h1,h2,c1,c2
c1=cmplx(0.5,0.0)
c2=cmplx(0.0,-0.5)
do j=1,n
  fft1(j)=cmplx(data1(j),data2(j))
end do
call four1(fft1,n,1)
fft2(1)=cmplx(aimag(fft1(1)),0.0)
fft1(1)=cmplx(real(fft1(1)),0.0)
n2=n+2
do j=2,n/2+1
  h1=c1*(fft1(j)+conjg(fft1(n2-j)))
  h2=c2*(fft1(j)-conjg(fft1(n2-j)))
  fft1(j)=h1
  fft1(n2-j)=conjg(h1)
```

```

      fft2(j)=h2
      fft2(n2-j)=conjg(h2)
    end do
  END SUBROUTINE twofft

```

5. 例子

为了验证子过程 TWOFFT,我们在验证程序 D12R2 中取了两组周期性数据,并且对这两组异相数据都进行了变换和逆变换.我们利用子过程 FOUR1 进行了逆变换,其计算结果应该乘以 $1/N(N=32)$,显然逆变换后即为原始数据.验证程序 D12R2 如下:

```

PROGRAM D12R2
  ! Driver for routine TWOFFT
  PARAMETER (N=32,N2=2 * N,PER=8.0,PI=3.14159)
  DIMENSION DATA1(N),DATA2(N),FFT1(N2),FFT2(N2)
  DO I=1,N
    X=2.0 * PI * I/PER
    DATA1(I)=NINT(COS(X))
    DATA2(I)=NINT(SIN(X))
  END DO
  CALL TWOFFT(DATA1,DATA2,FFT1,FFT2,N)
  WRITE(*,*) 'Fourier transform of first function:'
  CALL PRNTFT(FFT1,N2)
  WRITE(*,*) 'Fourier transform of second function:'
  CALL PRNTFT(FFT2,N2)
  ! Invert transform
  ISIGN=-1
  CALL FOUR1(FFT1,N,ISIGN)
  WRITE(*,*) 'Inverted transform -- first function:'
  CALL PRNTFT(FFT1,N2)
  CALL FOUR1(FFT2,N,ISIGN)
  WRITE(*,*) 'Inverted transform = second function:'
  CALL PRNTFT(FFT2,N2)
END PROGRAM

SUBROUTINE PRNTFT(DATA,N2)
  DIMENSION DATA(N2)
  WRITE(*, '(1X,T7,A,T13,A,T24,A,T35,A,T47,A)') &

```

```

      'n','Real(n)','Imag. (n)','Real(N-n)','Imag. (N-n)'
WRITE(*, '(1X,I6,4F12.6)') 0,DATA(1),DATA(2),&
      DATA(1),DATA(2)
DO I=3,(N2/2)+1,2
  M=(I-1)/2
  NN2=N2+2-I
  WRITE(*, '(1X,I6,4F12.6)') M,DATA(I),DATA(I+1),&
      DATA(NN2),DATA(NN2+1)
END DO
WRITE(*, '(/1X,A)') 'Press RETURN to continue ...'
READ(*,*)
END SUBROUTINE PRNTFT

```

计算结果如下:

Fourier transform of first function:

n	Real(n)	Imag. (n)	Real(N-n)	Imag. (N-n)
0	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000
4	13.656855	-13.656855	13.656855	13.656855
	⋮	⋮	⋮	⋮
12	2.343146	2.343146	2.343146	-2.343146
13	0.000000	0.000000	0.000000	0.000000
14	0.000000	0.000000	0.000000	0.000000
15	0.000000	0.000000	0.000000	0.000000
16	0.000000	0.000000	0.000000	0.000000

Press RETURN to continue ...

Fourier transform of second function:

n	Real(n)	Imag. (n)	Real(N-n)	Imag. (N-n)
0	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000
4	13.656855	13.656855	13.656855	-13.656855
	⋮	⋮	⋮	⋮
12	2.343146	-2.343146	2.343146	2.343146

13	0.000000	0.000000	0.000000	0.000000
14	0.000000	0.000000	0.000000	0.000000
15	0.000000	0.000000	0.000000	0.000000
16	0.000000	0.000000	0.000000	0.000000

Press RETURN to continue ...

Inverted transform = first function:

n	Real(n)	Imag. (n)	Real(N-n)	Imag. (N-n)
0	32.000000	0.000000	32.000000	0.000000
1	0.000000	0.000000	32.000000	0.000000
2	-32.000000	0.000000	32.000000	0.000000
3	-32.000000	0.000000	0.000000	0.000000
	⋮	⋮	⋮	⋮
13	0.000000	0.000000	-32.000000	0.000000
14	32.000000	0.000000	-32.000000	0.000000
15	32.000000	0.000000	0.000000	0.000000
16	32.000000	0.000000	32.000000	0.000000

Press RETURN to continue ...

Inverted transform = second function:

n	Real(n)	Imag. (n)	Real(N-n)	Imag. (N-n)
0	32.000000	0.000000	32.000000	0.000000
1	32.000000	0.000000	0.000000	0.000000
2	32.000000	0.000000	-32.000000	0.000000
3	0.000000	0.000000	-32.000000	0.000000
	⋮	⋮	⋮	⋮
13	-32.000000	0.000000	0.000000	0.000000
14	-32.000000	0.000000	32.000000	0.000000
15	0.000000	0.000000	32.000000	0.000000
16	32.000000	0.000000	32.000000	0.000000

Press RETURN to continue ...

12.3 实数据快速傅里叶变换算法(二)

1. 功能

本算法适用于仅计算一个实数序列的离散傅里叶变换. 它采用单道分离法考虑采样值为实数据的条件下, 将 $2N$ 个点采样值分离成两个 N 个点的采样值函数, 并利用对 N 个采样值的变换方法来得到 $2N$ 个采样值的变换结果. 另外,

子过程也可计算一个复数序列的逆变换,如果此时的复数序列是某一实数序列的傅里叶变换,在这种情况下,输出结果乘以 $1/N$ 后才是此复数序列的逆变换.

2. 方法

设 $\{x(j)\}$ 是具有 $2N$ 个点的实数序列,并设其傅里叶变换为 $\{A(k)\}$ ($k=0, 1, \dots, 2N-1$). 把 $\{x(j)\}$ 分成两个子序列:

$$\begin{cases} x_1(j) = x(2j) & (\text{偶数下标的序列}) \\ x_2(j) = x(2j+1) & (\text{奇数下标的序列}) \end{cases} \\ j = 0, 1, \dots, N-1$$

并记 $\{x_1(j)\}$ 和 $\{x_2(j)\}$ 的傅里叶变换分别为 $\{A_1(k)\}$ 和 $\{A_2(k)\}$. 由定义, $A_1(k)$ 和 $A_2(k)$ 可表示为

$$\begin{cases} A_1(k) = \sum_{j=0}^{N-1} x_1(j) \exp(2\pi i j k / N) \\ A_2(k) = \sum_{j=0}^{N-1} x_2(j) \exp(2\pi i j k / N) \end{cases} \\ k = 0, 1, \dots, N-1$$

这时, $A(k)$ 可表示为

$$\begin{aligned} A(K) &= \sum_{j=0}^{2N-1} x(j) \exp(2\pi i j 2k / (2N)) \\ &= \sum_{j=0}^{N-1} x(2j) \exp(2\pi i j 2k / (2N)) \\ &\quad + \sum_{j=0}^{N-1} x(2j+1) \exp(2\pi i k (2j+1) / (2N)) \\ &= A_1(k) + A_2(k) \exp(\pi i k / N) \end{aligned} \\ k = 0, 1, \dots, N-1 \quad (12-6)$$

为了有效地计算 $A_1(k)$ 和 $A_2(k)$, 使用 12.2 节中的方法同时算出 $\{x_1(j)\}$ 和 $\{x_2(j)\}$ 的 DFT $\{A_1(k)\}$ 和 $\{A_2(k)\}$, 即设 $z(j) = x_1(j) + ix_2(j)$, 使用 12.1 节的 FFT 程序可计算出复数序列 $\{z(j)\}$ 的 DFT, 记为 $\{x(k)\}$ ($k=0, 1, \dots, N-1$), 则

$$z(k) = A_1(k) + iA_2(k) = R(k) + iI(k) \quad (12-7)$$

其中 $R(k)$ 和 $I(k)$ 为 $z(k)$ 的实部和虚部. 因为

$$A_1(k) = A_1^*(N-k), \quad A_2(k) = A_2^*(N-k), \quad k = 0, 1, \dots, N/2$$

在式(12-7)中以 $N-k$ 代替 k , 并在等式两边取共轭得

$$z^*(N-k) = A_1(k) - iA_2(k) = R(N-k) - iI(N-k) \quad (12-8)$$

由式(12-7)和(12-8)解得

$$\left\{ \begin{aligned} A_1(k) &= [z(k) + z^*(N-k)]/2 \\ &= [R(k) + R(N-k)]/2 + i[I(k) - I(N-k)]/2 \\ &= R_e(k) + iI_o(k) \end{aligned} \right. \quad (12-9)$$

$$\left\{ \begin{aligned} A_2(k) &= [z(k) - z^*(N-k)]/(2i) \\ &= [I(k) + I(N-k)]/2 - i[R(k) - R(N-k)]/2 \\ &= I_e(k) - iR_o(k) \end{aligned} \right. \quad (12-10)$$

将式(12-9)和(12-10)代入式(12-6)得

$$\begin{aligned} A(k) &= A_1(k) + e^{i\pi k/N} A_2(k) \\ &= \left[R_e(k) + \cos\left(\frac{\pi k}{N}\right) I_e(k) + \sin\frac{\pi k}{N} R_o(k) \right] \\ &\quad + i \left[I_o(k) + \sin\left(\frac{\pi k}{N}\right) I_e(k) - \cos\frac{\pi k}{N} R_o(k) \right] \\ &= X_R(k) + iX_I(k) \end{aligned} \quad (12-11)$$

综上所述,对于具有 $2N$ 个点的实数序列 $\{x(j)\}$,其 DFT $\{A(k)\}$ 的计算步骤如下:

(1) 按下标的奇偶把 $\{x(j)\}$ 分为两个序列:

$$\begin{cases} x_1(j) = x(2j) \\ x_2(j) = x(2j+1) \end{cases} \\ j = 0, 1, \dots, N-1$$

(2) 使用 12.2 节中的方法同时算出 $\{x_1(j)\}$ 和 $\{x_2(j)\}$ 的 DFT $\{A_1(k)\}$ 和 $\{A_2(k)\}$, 这包括:

(a) 构成一个复数序列

$$z(j) = x_1(j) + ix_2(j), \quad j = 0, 1, \dots, N-1$$

(b) 用复数据的 FFT 计算 $\{z(j)\}$ 的 DFT, 记为 $\{z(k)\}$, $k = 0, 1, \dots, N-1$.

(c) 用上面的公式(12-9)和(12-10)计算 $A_1(k), A_2(k)$, $k = 0, 1, \dots, N/2$.

(3) 因

$$A^*(N-k) = A_1(k) - e^{i\pi k/N} A_2(k) \quad (12-12)$$

及 $A^*(2N-k) = A(k)$, 故只需由公式(12-11)和(12-12)计算 $\{A(k)\}$ ($k = 0, 1, \dots, N/2+1$) 即可.

3. 使用说明

(1) REALFT(DATA, N, ISIGN)

N 整型变量, 输入参数, 采样点数的一半, 它为 2 的整数幂, 即 $N = 2^m$, m 为正整数

- ISIGN 整型变量,输入参数,取值+1或-1.当输入+1时,是计算一个实数序列的 DFT,否则是计算一个复数序列逆傅里叶变换,此时要求此复数序列是某一实数序列的 DFT
- DATA 实型数组 DATA(2N+2),输入、输出参数,若 ISIGN 输入+1,则开始存放 2N 个点的实数序列 {DATA(j)} (j=1,2,...,2N),结束时存放此实数序列的 DFT;若 ISIGN 输入-1,则 {DATA(j)} (j=1,2,...,2N)开始时存放 N 点复数序列,结束时存放其逆 DFT,这时结果应乘以 1/N

(2) 要调用的子过程:

FOUT1(DATA,NN,ISIGN)

见 12.1 节.

4. 过程

子过程 REALFT.

```

SUBROUTINE realft(data1,n,isign)
REAL * 8 wr,wi,wpi,wpr,wtemp,theta
DIMENSION data1(2 * n)
1 USES four1
INTEGER n,isign,i1,i2,i3,i4,i
REAL c2,h2r,h2i,qrs,wis,h1r,h1i
theta=6.28318530717959d0/2.0d0/dfloat(n)
c1=0.5
if (isign==1) then
    c2=-0.5
    call four1(data1,n,+1)
else
    c2=0.5
    theta=-theta
endif
wpr=-2.0d0 * dsin(0.5d0 * theta) * * 2
wpi=dsin(theta)
wr=1.0d0+wpr
wi=wpi
n2p3=2 * n+3
do i=2,n/2+1

```

```

      i1=2*i-1
      i2=i1+1
      i3=n2p3-i2
      i4=i3+1
      wrs=sngl(wr)
      wis=sngl(wi)
      h1r=c1*(data1(i1)+data1(i3))
      h1i=c1*(data1(i2)-data1(i4))
      h2r=-c2*(data1(i2)+data1(i4))
      h2i=c2*(data1(i1)-data1(i3))
      data1(i1)=h1r+wrs*h2r-wis*h2i
      data1(i2)=h1i+wrs*h2i+wis*h2r
      data1(i3)=h1r-wrs*h2r+wis*h2i
      data1(i4)=-h1i+wrs*h2i+wis*h2r
      wtemp=wr
      wr=wr*wpr-wi*wri+wr
      wi=wi*wpr+wtemp*wpi+wi
    end do
    if (isign==1) then
      h1r=data1(1)
      data1(1)=h1r+data1(2)
      data1(2)=h1r-data1(2)
    else
      h1r=data1(1)
      data1(1)=c1*(h1r+data1(2))
      data1(2)=c1*(h1r-data1(2))
      call four1(data1,n,-1)
    endif
  END SUBROUTINE realft

```

5. 例子

为了验证子过程 REALFT, 在验证程序 D12R3 中, 我们所举的例子为余弦函数, 且允许选取周期. 在变换后, 以简单的图形表示变换后的每个元素的大小. 如果选取的周期是 2 的幂, 那么变换后仅在某一位上非零. 变换以后, 又作逆变换所得到的计算结果正是原来的函数值. 验证程序 D12R3 如下:

```
PROGRAM D12R3
```

```

! Driver for routine REALFT
PARAMETER (EPS=1.0E-3,NP=32,NPP2=NP+2,WIDTH=50.0,&
    PI=3.14159)
DIMENSION DATA1(NPP2),SIZE(NP)
N=NP/2
DO
WRITE(*,'(1X,A,I2,A)')'Period of sinusoid in&
    channels (2--',NP,', or 0 to stop) '
READ(*,*)PER
IF(PER<=0.)STOP
DO I=1,NP
    DATA1(I)=COS(2.0*PI*(I-1)/PER)
END DO
CALL REALFT(DATA1,N,+1)
BIG=-1.0E10
DO I=1,N
    SIZE(I)=SQRT(DATA1(2*I-1)**2+DATA1(2*I)**2)
    IF(I==1)SIZE(I)=DATA1(I)
    IF(SIZE(I)>BIG)BIG=SIZE(I)
END DO
SCAL=WIDTH/BIG
DO I=1,N
    NLIM=SCAL*SIZE(I)+EPS
    WRITE(*,'(1X,I4,1X,60A1)')I,(' '*J,J=1,NLIM+1)
END DO
WRITE(*,*)'press Enter continue'
READ(*,*)
CALL REALFT(DATA1,N,-1)
BIG=-1.0E10
SMALL=1.0E10
DO I=1,NP
    IF(DATA1(I)<SMALL)SMALL=DATA1(I)
    IF(DATA1(I)>BIG)BIG=DATA1(I)
END DO
SCAL=WIDTH/(BIG-SMALL)
DO I=1,NP
    NLIM=SCAL*(DATA1(I)-SMALL)+EPS

```

```

      WRITE(*,'(1X,I4,1X,60A1)') I, ('*',J=1,NLIM+1)
END DO
END DO
END

```

计算结果如下:

Period of sinusoid in channels (2—32, or 0 to stop)

```

32
1 *
2 *****
3 *
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
press Enter continue
1 *****
2 *****
3 *****
4 *****
5 *****
6 *****
7 *****
8 *****
9 *****
10 *****
11 *****
12 *****

```

```

13 *****
14 *****
15 **
16 *
17 *
18 *
19 **
20 *****
21 *****
22 *****
23 *****
24 *****
25 *****
26 *****
27 *****
28 *****
29 *****
30 *****
31 *****
32 *****
Period of sinusoid in channels (2-32, or 0 to stop)
0

```

12.4 快速正弦变换和余弦变换

1. 功能

本节有两个子过程, 它们的功能分别是:

(1) SINFT, 用快速算法计算实数序列 $\{f_j\}_{j=0}^{N-1}$ 的正弦变换

$$F_k = \sum_{j=0}^{N-1} f_j \sin(\pi jk/N)$$

(2) COSFT, 用快速算法计算 $\{f_j\}$ 的余弦变换

$$F_k = \sum_{j=0}^{N-1} f_j \cos(\pi jk/N)$$

和 $\{f_j\}$ 的逆余弦变换

$$\tilde{f}_e = \sum_{k=0}^{N-1} F_k \cos(\pi kl/N)$$

2. 方法

(1) 正弦变换的计算.

设 $\{f_j\} (j=0, 1, \dots, N-1)$ 为实数序列, 构造辅助实数序列 $\{y_i\}$ 如下:

$$\begin{cases} y_0 = 0 \\ y_j = \sin(j\pi/N)(f_j + f_{N-j}) + \frac{1}{2}(f_j - f_{N-j}) \end{cases} \quad (12-13)$$

$$j = 1, 2, \dots, N-1$$

用 12.3 节中计算一组实数序列的傅里叶变换的子过程 REALFT, 计算 $\{y_i\} (j=0, 1, \dots, N-1)$ 的 DFT 得 $\{Y_k\} (k=0, 1, \dots, N-1)$, 其中 $Y_k = R_k + iI_k$, R_k 和 I_k 分别为 Y_k 的实部和虚部, 满足:

$$\begin{aligned} R_k &= \sum_{j=0}^{N-1} y_j \cos(2\pi jk/N) \\ &= \sum_{j=1}^{N-1} (f_j + f_{N-j}) \sin(j\pi/N) \cos(2\pi jk/N) \\ &= \sum_{j=0}^{N-1} 2f_j \sin(j\pi/N) \cos(2\pi jk/N) \\ &= \sum_{j=0}^{N-1} f_j \left[\sin\left(\frac{2k+1}{N}j\pi\right) - \sin\left(\frac{2k-1}{N}j\pi\right) \right] \\ &= F_{2k+1} - F_{2k-1} \\ I_k &= \sum_{j=0}^{N-1} y_j \sin(2\pi jk/N) \\ &= \sum_{j=1}^{N-1} (f_j - f_{N-j}) \frac{1}{2} \sin(2\pi jk/N) \\ &= \sum_{j=0}^{N-1} f_j \sin(2\pi jk/N) \\ &= F_{2k} \end{aligned}$$

故正弦变换 $\{F_k\} (k=0, 1, \dots, N-1)$ 可由下式决定

$$\begin{cases} F_{2k} = I_k, & F_{2k+1} = F_{2k-1} + R_k, & k = 0, \dots, (N/2 - 1) \\ F_1 = \sum_{j=0}^{N-1} f_j \sin(j\pi/N) \end{cases} \quad (12-14)$$

综上所述, 计算步骤为:

- ① 由式(12-13)求得辅助序列 $\{y_i\} (j=0, 1, \dots, N-1)$.
- ② 用 REALFT 计算 $\{y_i\}$ 的 DFT $\{Y_k\} \equiv \{R_k + iI_k\}$.

③ 用公式(12.14)求正弦变换 $\{F_k\}$ ($k=0, 1, \dots, N-1$).

(2) 余弦变换的计算步骤.

① 构造辅助函数序列 $\{y_j\}$ ($j=0, 1, \dots, N-1$)

$$\begin{cases} y_0 = f_0 \\ y_j = \frac{1}{2}(f_j + f_{N-j}) - \sin(j\pi/N)(f_j - f_{N-j}) \end{cases}$$

② 用子过程 REALFT 计算 $\{y_j\}$ 的 DFT $\{Y_k\}$ ($k=0, 1, \dots, N-1$), 且设 R_k 和 I_k 为 Y_k 的实部和虚部.

③ 计算余弦变换 $\{F_k\}$ ($k=0, 1, \dots, N-1$):

$$\begin{cases} F_{2k} = R_k, & F_{2k+1} = F_{2k-1} + I_k, & k = 0, \dots, (N/2 - 1) \\ F_1 = \sum_{j=0}^{N-1} f_j \cos(j\pi/N) \end{cases}$$

(3) $\{f_j\}$ 的逆余弦变换的计算.

要计算 $\{f_j\}$ 的逆余弦变换 $\{\tilde{f}_l\}$, 即 $\{F_k\}$ 的余弦变换, 其中 $\{F_k\}$ 为 $\{f_j\}$ 的余弦变换, 由定义

$$\tilde{f}_l = \sum_{k=0}^{N-1} F_k \cos(\pi kl/N)$$

容易证明, $\{F_k\}$ 和 $\{\tilde{f}_l\}$ 有如下关系

$$\begin{cases} F_0 = N\tilde{f}_0 + \sum_{j \text{ 奇数}} \tilde{f}_j \\ F_l = \frac{N}{2}\tilde{f}_l + \sum_{j \text{ 偶数}} \tilde{f}_j, & l \text{ 为奇数} \\ F_l = \frac{N}{2}\tilde{f}_l + \sum_{j \text{ 奇数}} \tilde{f}_j, & l \text{ 为偶数且 } l \neq 0 \end{cases} \quad (12-15)$$

如记

$$\begin{cases} c_1 \equiv \sum_{l \text{ 偶}} F_l = \frac{N}{2} \left[\tilde{f}_0 + \sum_{j=0}^{N-1} \tilde{f}_j \right] \\ c_2 \equiv \sum_{l \text{ 奇}} F_l = \frac{N}{2} \sum_{j=0}^{N-1} \tilde{f}_j \end{cases} \quad (12-16)$$

则

$$\begin{cases} \frac{N}{2}\tilde{f}_0 = c_1 - c_2, & \sum_{j \text{ 奇}} \tilde{f}_j = F_0 - 2(c_1 - c_2) \\ \sum_{j \text{ 偶}} \tilde{f}_j = \frac{2}{N}c_2 - \sum_{j \text{ 奇}} \tilde{f}_j \end{cases} \quad (12-17)$$

故计算步骤如下:

- ① 用式(12-14)的方法计算 $\{y_j\}$ 的余弦变换 $\{F_k\}$.
- ② 用式(12-16)计算 c_1 和 c_2 ,再计算式(12-17).
- ③ 用式(12-15)计算 $\{f_j\}$ 的逆余弦变换 $\{\tilde{f}_j\}$.

3. 使用说明

(1) SINFT(Y,N)

- N 整型变量,输入参数,采样点个数,即给出的实数据的个数,要求 $N=2^m$, m 为正整数
- Y 实型数组 $Y(N)$,输入、输出参数,开始时存放给定的一组实数序列,结束时存放正弦变换值.若要求给定的一组实数序列的逆正弦变换,只需将输出结果乘以因子 $2/N$ 即可

(2) COSFT(Y,N,ISIGN)

- N 整型变量,输入参数,采样点个数,要求 N 为2的正整数次幂
- ISIGN 整型变量,输入参数,其值为+1或-1,当输入+1时,求一实数序列的余弦变换,当输入-1时,求此实数序列的逆余弦变换
- Y 实型数组 $Y(N)$,输入、输出参数,开始时存放 N 个点的采样数据,结束时,若ISIGN为+1,则存放此实数序列的余弦变换,否则存放此实数序列的逆余弦变换乘以 $N/2$ 后的值,即逆余弦变换为此时的输出结果乘以 $2/N$

(3) 这两个子过程均要调用子过程REALFT(见12.3节).

4. 过程

(1) 子过程 SINFT.

```
SUBROUTINE sinft(y,n)
  INTEGER n
  REAL y(n)
  ! USES realft
  INTEGER j
  REAL sum,y1,y2
  DOUBLE PRECISION theta,wi,wpi,wpr,wr,wtemp
  theta=3.141592653589793d0/dble(n)
  wr=1.0d0
  wi=0.0d0
  wpr=-2.0d0 * sin(0.5d0 * theta) * * 2
```



```

wpi=sin(theta)
y(1)=0.0
do j=1,n/2
    wtemp=wr
    wr=wr*wpr-wi*wpi+wr
    wi=wi*wpr+wtemp*wpi+wi
    y1=wi*(y(j+1)+y(n-j+1))
    y2=0.5*(y(j+1)-y(n-j+1))
    y(j+1)=y1+y2
    y(n-j+1)=y1-y2
end do
call realft(y,n/2,+1)
sum=0.0
y(1)=0.5*y(1)
y(2)=0.0
do j=1,n-1,2
    sum=sum+y(j)
    y(j)=y(j+1)
    y(j+1)=sum
end do
END

```

(2) 子过程 COSFT.

```

SUBROUTINE cosft(y,n,isign)
REAL * 8 wr,wi,wpr,wpi,wtemp,theta
! USES realft
REAL y(n),sum,odd,even,enf0,sum0,some
REAL y1,y2
INTEGER j,m,n,isign
theta=3.14159265358979d0/dfloat(n)
wr=1.0d0
wi=0.0d0
wpr=-2.0d0*dsin(0.5d0*theta)**2
wpi=dsin(theta)
sum=y(1)
m=n/2
do j=1,m-1

```

```

      wtemp=wr
      wr=wr*wpr-wi*wpi+wr
      wi=wi*wpr+wtemp*wpi+wi
      y1=0.5*(y(j+1)+y(n-j+1))
      y2=(y(j+1)-y(n-j+1))
      y(j+1)=y1-wi*y2
      y(n-j+1)=y1+wi*y2
      sum=sum+wr*y2
    end do
    call realft(y,m,+1)
    y(2)=sum
    do j=4,n,2
      sum=sum+y(j)
      y(j)=sum
    end do
    if (isign== -1) then
      even=y(1)
      odd=y(2)
      do i=3,n-1,2
        even=even+y(i)
        odd=odd+y(i+1)
      end do
      enf0=2.0*(even-odd)
      sum0=y(1)-enf0
      sume=(2.0*odd/float(n))-sum0
      y(1)=0.5*enf04
      y(2)=y(2)-sume
      do i=3,n-1,2
        y(i)=y(i)-sum0
        y(i+1)=y(i+1)-sume
      end do
    endif
  END SUBROUTINE cosft

```

5. 例子

(1) 验证子过程 SINFT 的 D12R4 基本上和前一节的验证程序相同. 注意: 在 SINFT 中, 变换和逆变换表面上没有区别, 它们是相同的. 验证程序 D12R4

如下:

```

PROGRAM D12R4
! Driver for routine SINFT
PARAMETER (EPS=1.0E-3,NP=16,WIDTH=30.0,PI=3.14159)
DIMENSION DATA(NP),SIZE(NP)
DO
  WRITE(*, '(1X,A,I2,A)') &
    'Period of sinusoid in channels&
    (2- ',NP,' or 0 to stop)'
  READ(*,*) PER
  IF(PER<=0.) STOP
  DO I=1,NP
    DATA(I)=SIN(2.0*PI*(I-1)/PER)
  END DO
  CALL SINFT(DATA,NP)
  BIG=-1.0E10
  SMALL=1.0E10
  DO I=1,NP
    IF(DATA(I)<SMALL) SMALL=DATA(I)
    IF(DATA(I)>BIG) BIG=DATA(I)
  END DO
  SCAL=WIDTH/(BIG-SMALL)
  DO I=1,NP
    NLIM=SCAL*(DATA(I)-SMALL)+EPS
    WRITE(*, '(1X,I4,1X,60A1)') I, ('*',J=1,NLIM+1)
  END DO
  WRITE(*,*) 'press Enter continue'
  READ(*,*)
  CALL SINFT(DATA,NP)
  BIG=-1.0E10
  SMALL=1.0E10
  DO I=1,NP
    IF(DATA(I)<SMALL) SMALL=DATA(I)
    IF(DATA(I)>BIG) BIG=DATA(I)
  END DO
  SCAL=WIDTH/(BIG-SMALL)

```

```

DO I=1,NP
  NLIM=SCAL*(DATA(I)-SMALL)+EPS
  WRITE(*,'(1X,I4,1X,60A1)') I, ('*',J=1,NLIM+1)
END DO
END DO
END

```

计算结果如下:

Period of sinusoid in channels(2—16 or 0 to stop)

16

```

1 *
2 *
3 * * * * *
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *

```

press Enter continue

```

1 * * * * *
2 * * * * *
3 * * * * *
4 * * * * *
5 * * * * *
6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
10 * * * * *

```

```

11 * * * * *
12 * *
13 *
14 * *
15 * * * * *
16 * * * * *
Period of sinusoid in channels (2-16 or 0 to stop)
0

```

(2) COSFT 是一个和 SINFT 配对的余弦变换子过程. 虽然验证程序 D12R5 和 D12R3 是相似的, 但可以注意到解的一些不同的地方. 具有 2 的整数幂的周期的余弦变换没有给出某一位为非零而其余为零的变换结果. 它在其它频率上有一些小值. 这是由于在 COSFT 中只选了一部分调用 REALFT. 验证程序 D12R5 在做了变换后, 紧接着又做了逆变换. 注意: 余弦变换和正弦变换不同, 余弦变换的逆变换不是它本身, 需令 $ISIGN = -1$. 验证程序 D12R5 如下:

```

PROGRAM D12R5
! Driver for routine COSFT
PARAMETER (EPS=1.0E-3,NP=16,WIDTH=30.0,PI=3.14159)
DIMENSION DATA(NP),SIZE(NP)
DO
  WRITE(*,'(1X,A,I2,A)')&
    'Period of cosine in channels (2-',NP,' or 0 to stop)'
  READ(*,*) PER
  IF(PER<=0.) STOP
  DO I=1,NP
    DATA(I)=COS(2.0*PI*(I-1)/PER)
  END DO
  CALL COSFT(DATA,NP,+1)
  BIG=-1.0E10
  SMALL=1.0E10
  DO I=1,NP
    IF(DATA(I)<SMALL) SMALL=DATA(I)
    IF(DATA(I)>BIG) BIG=DATA(I)
  END DO
  SCAL=WIDTH/(BIG-SMALL)
  DO I=1,NP

```

```

      NLIM=SCAL * (DATA(I)-SMALL)+EPS
      WRITE( *, '(1X,I2,F6.2,1X,60A1)' ) I,DATA(I),&
        (' * ',J=1,NLIM+1)
    END DO
    WRITE( *, * ) 'press Enter continue'
    READ( *, * )
    CALL COSFT(DATA,NP,-1)
    BIG=-1.0E10
    SMALL=1.0E10
    DO I=1,NP
      IF(DATA(I)<SMALL) SMALL=DATA(I)
      IF(DATA(I)>BIG) BIG=DATA(I)
    END DO
    SCAL=WIDTH/(BIG-SMALL)
    DO I=1,NP
      NLIM=SCAL * (DATA(I)-SMALL)+EPS
      WRITE( *, '(1X,I4,1X,60A1)' ) I, (' * ',J=1,NLIM+1)
    END DO
  END DO
END

```

计算结果如下:

Period of cosine in channels (2--16 or 0 to stop)

```

4
1  0.00 *
2  1.00 * * * *
3  0.00 *
4  1.00 * * * *
5  0.00 *
6  1.00 * * * *
7  0.00 *
8  1.00 * * * *
9  8.00 * * * * * * * * * * * * * * * * * * * * * *
10 1.00 * * * *
11 0.00 *
12 1.00 * * * *
13 0.00 *

```

```

14  1.00 * * * *
15  0.00 *
16  1.00 * * * *
press Enter continue
  1  * * * * * * * * * * * * * * * *
  2  * * * * * * * * * * * * * * * *
  3  *
  4  * * * * * * * * * * * * * * * *
  5  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  6  * * * * * * * * * * * * * * * *
  7  *
  8  * * * * * * * * * * * * * * * *
  9  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
10  * * * * * * * * * * * * * * * *
11  *
12  * * * * * * * * * * * * * * * *
13  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
14  * * * * * * * * * * * * * * * *
15  *
16  * * * * * * * * * * * * * * * *
Period of cosine in channels (2—16 or 0 to stop)
0

```

12.5 卷积和逆卷积的快速算法

1. 功能

计算实数序列 $\{s_j\}$ 和其响应函数 $\{r_k\}$ 的卷积和逆卷积。

2. 方法

离散卷积定理是:若信号 $\{s_j\}$ 的周期为 N ,其有限脉冲响应函数为 $\{r_k\}$, $\{s_j\} \leftrightarrow S_n, \{r_k\} \leftrightarrow R_n$,则其离散卷积为:

$$(r * s)_j \equiv \sum_{k=-N/2+1}^{N/2} s_{j-k} r_k \leftrightarrow S_n R_n$$

注意,离散卷积定理中有两个一般在实际中不满足的限制,一是它要求输入信号 $\{s_j\}$ 是周期的,二是它要求相应的持续时间与信号的周期相同,故要作一些

处理.

对于第二个限制处理是方便的,即若已知 $\{s_j\}$ 的周期为 N , $\{r_k\}$ 是其响应函数, $-M/2 < k \leq M/2$ 且 $M < N$,此时只需定义 $r_k = 0, M/2 < k \leq N/2$ 或 $-N/2 + 1 \leq k \leq -M/2$ 即可.对于第一个限制,处理方案是建立一个缓冲带(即在 $\{s_j\}$ 的末端用零填充).这个“缓冲器”中需要多少个零填充呢?记 k 为填零的数目,则 k 等于响应函数的最大正脉冲或最大负脉冲,即若 $r_{-3}, r_{-2}, r_{-1}, r_0, r_1, \dots, r_6$ 均非零,而 $r_{-4}, r_{-5}, \dots, r_7, r_8, \dots$ 均为零,则令 $s_{N-6} = s_{N-5} = \dots = s_{N-1} = 0$,此时 $k = \max\{|-3|, 6\} = 6$.

综上所述,若 $\{s_j\} (j=0, 1, \dots, p-1)$ 和 $\{r_j\} (j=0, 1, \dots, m)$ 是给定的两个实数序列,则计算其卷积和逆卷积的步骤为:

(1) 选择 $N \geq p+m-1$ 且 $N=2^l, l$ 为正整数.

(2) 若 $r_{m/2}$ 或 r_m 不为零,则 $k=m/2$;否则若 $r_{m/2-1}$ 或 r_{m-1} 不为零,则 $k=m/2-1, \dots$.在 $\{s_j\}$ 末端进行零填充,即令 $s_{N-k} = s_{N-k+1} = \dots = s_{N-1} = 0$.

(3) 将 $\{r_k\}$ 变为中间为零、两端非零(分别为正脉冲和负脉冲)、长度为 N 的序列 $\{\bar{r}_k\}$,即令

$$\bar{r}_{N-i} = r_{m-i+1}, \quad i = 1, 2, \dots, \frac{m}{2} \quad (m \text{ 为偶数})$$

$$\bar{r}_i = 0, \quad i = m/2 + 1, \dots, N - 1 - m/2$$

$$\bar{r}_i = r_i, \quad i = 0, 1, \dots, m/2$$

(4) 由卷积定理计算 $\{s_j\}$ 和 $\{\bar{r}_j\} (j=0, 1, \dots, N-1)$ 的卷积和逆卷积(注意,上面的 m 在子过程中为 $M-1$).

3. 使用说明

(1) CONVLEV(DATA, N, RESPNS, M, ISIGN, ANS)

N 整型变量,输入参数,设定的数据组的长度(或信号的周期),要求 $N=2^l (l \text{ 为正整数})$ 且 $N \geq P+M-1$,其中 P 为给定的信号取样点的个数

M 整型变量,输入参数,响应函数取值点的个数,要求 M 为奇整数

DATA 实型数组 DATA(N),输入参数,信号的样本值 $\{s_j\}_{j=1}^P$,输入时要作一些零填充,即令 $s_{N-k-1} = s_{N-k} = \dots = s_{N-1} = s_N = 0$,其中 $k = (M-1)/2$

RESPNS 实型数组 RESPNS(N),输入参数,其中前 M 个单元存放响应函数点列,后 $N-M$ 个单元为工作单元

- ISIGN 整型变量,输入参数,其值为 ± 1 .当输入 $+1$ 时,计算结果为 s_j 与 r_k 的卷积,当输入 -1 时,计算结果为逆卷积
- ANS 复型数组 $\text{ANS}(N+1)$,输出参数,当 ISIGN 为 $+1$ 时,其前 N 个单元的元素为离散卷积;当 ISIGN 为 -1 时,其前 N 个单元的元素为逆卷积.在调用程序中,用户必须令 ANS 为至少含有 $2N+2$ 个单元的数组,因为它也被作为工作单元

(2) 要调用的子过程:

TWOFIT(DATA1,DATA2,FFT1,FFT2,N)

同时对两个实数序列作离散傅里叶变换,见 12.2 节.

REALFT(DATA,N,ISIGN)

一个实数序列的离散傅里叶变换和逆变换,见 12.3 节.

4. 过程

子过程 CONVLV.

```
SUBROUTINE convlv(data1,n,respns,m,isign,ans)
```

```
PARAMETER(nmax=8192)
```

```
DIMENSION data1(n),respns(n)
```

```
COMPLEX fft(nmax),ans(n)
```

```
INTEGER i,m,isign,no2
```

```
do i=1,(m-1)/2
```

```
    respns(n+1-i)=respns(m+1-i)
```

```
end do
```

```
do i=(m+3)/2,n-(m-1)/2
```

```
    respns(i)=0.0
```

```
end do
```

```
call twofit(data1,respns,fft,ans,n)
```

```
no2=n/2
```

```
do i=1,no2+1
```

```
    if (isign==1) then
```

```
        ans(i)=fft(i)*ans(i)/no2
```

```
    else if (isign== -1) then
```

```
        if (cabs(ans(i))==0.0)&
```

```
            pause 'deconvolving at a response zero'
```

```
        ans(i)=fft(i)/ans(i)/no2
```

```
    else
```

```

        pause 'no meaning for isign'
    endif
end do
ans(1)=cmplx(real(ans(1)),real(ans(no2+1)))
call realft(ans,no2,-1)
END SUBROUTINE CONVLV

```

5. 例子

子过程 CONVLV 利用 FFT 计算了一组数据及其响应函数的卷积. 在验证程序 D12R6 中, 这两组数据只取值 0.0 和 1.0. 在数组 DATA(I) 中, 除 $I=6$ 和 $I=10$ 之间的值为 1.0 外, 其余均为零. 在响应函数值 RESPNS(I) 中, 除 $I=3$ 和 $I=6$ 之间的值为 1.0 外, 其余均为零. 卷积的值可以简单地确定, 将响应函数值首尾相接, 根据期望的移位将它们移向左边, 并计算出 RESPNS 的非零通道有多少落到 DATA 的非零通道上. 利用这种方法, 能够核实子过程 CONVLV 的计算结果. 验证程序 D12R6 如下:

```

PROGRAM D12R6
! Driver for routine CONVLV
PARAMETER (N=16,N2P2=34,M=9,PI=3.1415926)
DIMENSION DATA1(N),RESPNS(M),RESP(N),ANS(N2P2)
DO I=1,N
    DATA1(I)=0.0
    IF((I>=(N/2-N/8)).AND.(I<=(N/2+N/8)))&
        DATA1(I)=1.0
END DO
DO I=1,M
    RESPNS(I)=0.0
    IF(I>2.AND.I<7) RESPNS(I)=1.0
    RESP(I)=RESPNS(I)
END DO
ISIGN=1
CALL CONVLV(DATA1,N,RESP,M,ISIGN,ANS)
! Compare with a direct convolution
WRITE(*, '(/1X,T4,A,T13,A,T24,A)') 'I','CONVLV',&
    'Expected'
DO I=1,N
    CMP=0.0

```

```

DO J=1,M/2
  CMP=CMP+DATA1(MOD(I-J-1+N,N)+1)*RESPNS(J+1)
  CMP=CMP+DATA1(MOD(I+J-1,N)+1)*RESPNS(M-J+1)
END DO
CMP=CMP+DATA1(I)*RESPNS(1)
WRITE(*,'(1X,I3,3X,2F12.6)') I,ANS(I),CMP
END DO
END

```

计算结果如下:

I	CONVLV	Expected
1	0.091069	0.000000
2	1.068782	1.000000
3	0.908931	1.000000
4	1.036612	1.000000
5	1.091069	1.000000
6	1.077665	1.000000
7	-0.091069	0.000000
8	0.875000	1.000000
9	2.091069	2.000000
10	2.931218	3.000000
11	2.908931	3.000000
12	3.170495	3.000000
13	2.091069	2.000000
14	1.025888	1.000000
15	-0.091069	0.000000
16	-0.185660	0.000000

12.6 离散相关和自相关的快速算法

1. 功能

用 FFT 算法计算两个实数组的相关和自相关.

2. 方法

设 $\{g_k\}$ 和 $\{h_k\}$ 是周期为 N 的两个实样本函数, 其离散相关定义为

$$\text{Corr}(g, h)_j \equiv \sum_{k=0}^{N-1} g_{j+k} h_k$$

它满足离散相关定理,即:若 $\{g_j\} \leftrightarrow \{G_k\}$, $\{h_j\} \leftrightarrow \{H_k\}$, 则 $\text{Corr}(g, h)_j$ 等于 $G_k H_k^*$ 的逆 DFT, 其中星号表示复共轭.

我们可以这样计算两个样本函数的离散相关:

(1) 计算两样本函数的 DFT $\{G_k\}$ 和 $\{H_k\}$.

(2) 求 $Z_k = G_k H_k^*$.

(3) 求 Z_k 的逆 DFT 得 $\text{Corr}(g, h)_j$.

同在卷积中的情况一样,我们必须考虑末端的影响.另外,我们的数据一般并不是周期的而是有限的,故我们输入数据时要作一些处理.

设 $x(t)$ 和 $h(t)$ 是两个远离原点分别为 a 和 b 开始定义的有限长的函数, P 为 $x(t)$ 的取样点数, Q 为 $h(t)$ 的取样点数,即已知 $x(a+k\Delta)$ 和 $h(b+j\Delta)$ (Δ 为取样间隔), $k=0, 1, \dots, P-1$; $j=0, 1, \dots, Q-1$; 则输入方式有两种:

① 选择 $N \geq P+Q-1$ 且 $N=2^l$ (l 为正整数). 定义 $\{\bar{x}_k\}$ 和 $\{\bar{h}_k\}$ 为

$$x_k = x(a+k\Delta), \quad k=0, 1, \dots, P-1$$

$$\bar{x}_k = 0, \quad k=P, P+1, \dots, N-1$$

$$\bar{h}_k = h(b+k\Delta), \quad k=0, 1, \dots, Q-1$$

$$\bar{h}_k = 0, \quad k=Q, Q+1, \dots, N-1$$

$$x_k = \bar{x}(\pm N+k) = \bar{x}(\pm 2N+k) = \dots$$

$$\bar{h}_k = \bar{h}(\pm N+k) = \bar{h}(\pm 2N+k) = \dots$$

输入 $\{\bar{h}_k\}$ 和 $\{\bar{x}_k\}$, 按上面步骤求得 $\text{Corr}(x, k)_j$, 存放在数组 $\text{ANS}(N)$ 的前 N 个单元, 此时结果需作一些解释: 在负延迟上的相关存放在 $\text{ANS}(N)$, $\text{ANS}(N-1), \dots, \text{ANS}(N/2+1)$ 中 (延迟取 $-1, -2, \dots, -N/2+1$); 正延迟上的相关存放在 $\text{ANS}(1)$ (零延迟), $\text{ANS}(2)$ (延迟为 1), $\text{ANS}(3), \dots, \text{ANS}(N/2)$ (延迟为 $N/2+1$).

② 第二种输入方法是定义 $\{\bar{x}_k\}$ 和 $\{\bar{h}_k\}$ 为: 先取 $N \geq P+Q-1$ 且 $N=2^l$ (l 为正整数), 令

$$\bar{x}_k = 0, \quad k=0, 1, \dots, N-P-1$$

$$\bar{x}_{N-P+k} = x(a+k\Delta), \quad k=0, 1, \dots, P-1$$

$$\bar{h}_k = h(b+k\Delta), \quad k=0, 1, \dots, Q-1$$

$$\bar{h}_k = 0, \quad k=Q, Q+1, \dots, N-1$$

$$\bar{x}_k = \bar{x}(\pm N+k) = \bar{x}(\pm 2N+k) = \dots$$

$$h_k = h(\pm N + k) = \bar{h}(\pm 2N + k) = \dots$$

输入 $\{\bar{h}_k\}$ 和 $\{\bar{x}_k\}$, 按前面步骤计算其相关, 则相关将呈现一个正延迟的峰。

3. 使用说明

(1) CORREL(DATA1, DATA2, N, ANS)

N 整型变量, 输入参数, 取定的数组 DATA1 和 DATA2 的周期. 要求 $N \geq P + Q - 1$ 且 2^l (l 为正整数, P 和 Q 为两函数的采样点数)

DATA1, DATA2 实型数组 DATA1(N), DATA2(N), 输入参数, 输入方式有两种, 具体见“方法”

ANS 复型数组 ANS(N), 即含 $2N$ 个单元的实数组, 输出参数且兼工作单元, 前 N 个单元的元素是输出结果, 即求出的相关. 注意, 由于输入方式不同, 此时结果也不同. 对于第一种输入方式要作下面解释: 不断增加负延迟 $(-1, -2, -3, \dots, -N/2 + 1)$ 上的相关存放在 ANS 的第 N 个单元, 第 $N-1$ 个单元, \dots , 第 $N/2 + 1$ 个单元, 不断增加正延迟 $(0, 1, 2, \dots, N/2 + 1)$ 上的相关存放在 ANS 的第一个单元, 第二个单元, \dots , 第 $N/2$ 个单元

(2) 要调用的子过程:

TWOFFT(DATA1, DATA2, FFT1, FFT2, N)

同时对两个实数序列作离散傅里叶变换, 见 12.2 节.

REALFT(DATA, N, ISIGN)

仅对一个实数序列作 DFT 或逆 DFT, 见 12.3 节.

4. 过程

子过程 CORREL.

```
SUBROUTINE correl(data1, data2, n, ans)
```

```
PARAMETER (NMAX=8192)
```

```
INTEGER n, i
```

```
REAL data1(n), data2(n)
```

```
COMPLEX fft(NMAX), ans(n)
```

```
! USES realft, twofft
```

```
call twofft(data1, data2, fft, ans, n)
```

```
no2=float(n)/2.0
```

```
do i=1, no2+1
```

```

    ans(i) = fft(i) * conjg(ans(i)) / float(no2)
end do
ans(1) = cmplx(real(ans(1)), real(ans(no2+1)))
call realft(ans, n/2, -1)
END SUBROUTINE correl

```

5. 例子

为了验证子过程 CORREL, 在验证程序 D12R7 中, 我们定义 DATA1 为有 64 个元素的数组, 在这些元素中, 除 $I=25$ 到 $I=39$ 之间的元素的值为 1 外, 其余均为零. 数组 DATA2 和数组 DATA1 一样. 验证程序将采用 CORREL 所得的结果和直接计算的结果进行比较. 验证程序 D12R7 如下:

```

PROGRAM D12R7
! Driver for routine CORREL
PARAMETER (N=64, N2=128, PI=3.1415926)
DIMENSION DATA1(N), DATA2(N), ANS(N2)
DO I=1, N
    DATA1(I)=0.0
    IF((I>(N/2-N/8)).AND.(I<(N/2+N/8)))&
        DATA1(I)=1.0
    DATA2(I)=DATA1(I)
END DO
CALL CORREL(DATA1, DATA2, N, ANS)
! Caculate directly
WRITE(*, '(/1X, T4, A, T13, A, T25, A/)' ) 'n', 'CORREL', &
    'Direct Calc.'
DO I=0, 16
    CMP=0.0
    DO J=1, N
        CMP=CMP+ DATA1(MOD(I+J-1, N)+1) * DATA2(J)
    END DO
    WRITE(*, '(1X, I3, 3X, F12.6, F15.6)' ) I, ANS(I+1), CMP
END DO
END

```

计算结果如下:

n	CORREL	Direct Calc.
0	15.000001	15.000000
1	14.291568	14.000000
2	13.000001	13.000000
3	11.952835	12.000000
4	11.000000	11.000000
5	9.984779	10.000000
6	9.000001	9.000000
7	7.949881	8.000000
8	7.000001	7.000000
9	6.033556	6.000000
10	5.000001	5.000000
11	3.939079	4.000000
12	3.000000	3.000000
13	2.119030	2.000000
14	1.000000	1.000000
15	-0.197704	0.000000
16	0.000000	0.000000

12.7 多维快速傅里叶变换算法

1. 功能

用 FFT 算法计算多维(L 维)离散傅里叶变换

$$H(n_1, n_2, \dots, n_L) \equiv \sum_{k_L=0}^{N_L-1} \dots \sum_{k_1=0}^{N_1-1} \exp(2\pi i k_L n_L / N_L) \\ \times \dots \exp(2\pi i k_1 n_1 / N_1) h(k_1, k_2, \dots, k_L) \quad (10-18)$$

($n_1 = 0, 1, \dots, N_1 - 1$; $n_2 = 0, 1, \dots, N_2 - 1$; \dots , $n_L = 0, 1, \dots, N_L - 1$)
或与其相应的逆变换.

2. 方法

采用反复计算一维复数据快速傅里叶变换的方法计算多维离散傅里叶变换(一维复数据的快速傅里叶变换方法详见 12.1 节),即将多维傅里叶变换写成

$$H(n_1, n_2, \dots, n_L) \equiv \sum_{k_L=0}^{N_L-1} \dots \left(\sum_{k_1=0}^{N_1-1} \exp(2\pi i k_1 n_1 / N_1) h(k_1, k_2, \dots, k_L) \right) \\ \cdot \exp(2\pi i k_2 n_2 / N_2) \cdot \exp(2\pi i k_L n_L / N_L)$$

($n_1 = 0, 1, \dots, N_1 - 1$; $n_2 = 0, 1, \dots, N_2 - 1$; \dots ; $n_L = 0, 1, \dots, N_L - 1$)

进行计算。

3. 使用说明

FOURN(DATA, NN, NDIM, ISIGN)

ISIGN 整型变量, 输入参数, 其值为 ± 1 . 若输入 $+1$, 则是要计算多维离散傅里叶变换式(10-18); 若输入 -1 , 则是要计算式(10-18)的逆变换(指数中的 i 为 $-i$ 的情形, 且据逆变换的定义, 还有一个因子 $1/(N_1 N_2 \dots N_L)$, 故求逆变换时, 由本程序得出的计算结果还须乘以因子 $1/(N_1 N_2 \dots N_L)$ 才是逆变换的值)

NDIM 整型变量, 输入参数, 维的数目, 对于式(10-18), $NDIM = L$

NN NDIM 个元素的一维整型数组, 输入参数, 存放每维中向量的长度, 如对式(10-18) $NN = (N_1, N_2, \dots, N_L)^T$, 注意, 此时要求 N_1, N_2, \dots, N_L 均为 2 的正整数次幂, 且要求它们均为每个方向上复数据的数目

DATA 实型数组, 输入、输出参数, 开始时存放多维的采样数据, 如对式(10-18)共含有 $2N_1 N_2 \dots N_L$ 个元素, 每个复数据占用两个连续的单元, 虚部在实部后, 且存放顺序为“列优先顺序”: 数组元素按列向量排列, 第 $j+1$ 个列向量紧接在第 j 个列向量后面. 结束时, 存放其 DFT 或逆变换乘以因子 $(N_1, N_2, \dots, N_{DIM})$

4. 过程

子过程 FOURN.

SUBROUTINE fourn(data, nn, ndim, isign)

INTEGER isign, ndim, nn(ndim)

REAL data(*)

INTEGER i1, i2, i2rev, i3, i3rev, ibit, idim, ifp1, &

ifp2, ip1, ip2, ip3, k1, k2, n, nprev, nrem, ntot

REAL tempi, tempr

DOUBLE PRECISION theta, wi, wpi, wpr, wr, wtemp


```

ntot = 1
do idim = 1, ndim
    ntot = ntot * nn(idim)
end do
nprev = 1
do idim = 1, ndim
    n = nn(idim)
    nrem = ntot / (n * nprev)
    ip1 = 2 * nprev
    ip2 = ip1 * n
    ip3 = ip2 * nrem
    i2rev = 1
    do i2 = 1, ip2, ip1
        if (i2 < i2rev) then
            do i1 = i2, i2 + ip1 - 2, 2
                do i3 = i1, ip3, ip2
                    i3rev = i2rev + i3 - i2
                    tempr = data(i3)
                    tempi = data(i3 + 1)
                    data(i3) = data(i3rev)
                    data(i3 + 1) = data(i3rev + 1)
                    data(i3rev) = tempr
                    data(i3rev + 1) = tempi
                end do
            end do
        endif
        i2rev = ip2 / 2
        do while ((i2rev >= ip1). and. (i2rev > i2rev))
            i2rev = i2rev - i2rev
            i2rev = i2rev / 2
        end do
        i2rev = i2rev + i2rev
    end do
    ifp1 = ip1
    do while (ifp1 < ip2)
        ifp2 = 2 * ifp1
        theta = sign * 6.28318530717959d0 / (ifp2 / ip1)
    end do
end do

```

```

wpr= 2.d0 * sin(0.5d0 * theta) * * 2
wpi=sin(theta)
wr=1.d0
wi=0.d0
do i3=1,ifp1,ip1
  do i1=i3,i3+ip1-2,2
    do i2=i1,ip3,ifp2
      k1=i2
      k2=k1+ifp1
      tempr=sngl(wr) * data(k2)-sngl(wi) * data(k2+1)
      tempi=sngl(wr) * data(k2+1)+sngl(wi) * data(k2)
      data(k2)=data(k1)-tempr
      data(k2+1)=data(k1+1)-tempi
      data(k1)=data(k1)+tempr
      data(k1+1)=data(k1+1)+tempi
    end do
  end do
  wtemp=wr
  wr=wr * wpr-wi * wpi+wr
  wi=wi * wpr+wtemp * wpi+wi
end do
ifp1=ifp2
end do
nprev=n * nprev
end do
END SUBROUTINE fourn

```

5. 例子

为了验证子过程 FOURN, 在验证程序 D12R8 中, 将它应用于 $4 \times 8 \times 16$ 的三维复数据变换. 该解析函数是不容易想象的, 但它是很容易进行计算的. 我们所验证的过程是执行一个三维变换, 接着将它反演, 且与原始数据进行比较. 两者之比是为了比较方便而提供的. 验证程序 D12R8 如下:

```

PROGRAM D12R8
! Driver for routine FOURN
PARAMETER (NDIM=3,NDAT=1024)

```

```

DIMENSION NN(NDIM),DATA(NDAT)
DO I=1,NDIM
  NN(I)=2*(2*I-1)
END DO
DO I=1,NN(3)
  DO J=1,NN(2)
    DO K=1,NN(1)
      L=K+(J-1)*NN(1)+(I-1)*NN(2)*NN(1)
      LL=2*L-1
      DATA(LL)=FLOAT(LL)
      DATA(LL+1)=FLOAT(LL+1)
    END DO
  END DO
END DO
ISIGN=+1
CALL FOURN(DATA,NN,NDIM,ISIGN)
ISIGN=-1
WRITE(*, '(1X,A)') 'Double 3-dimensional Transform'
WRITE(*, '(1X,T10,A,T35,A,T63,A)') 'Double Trabsf.', &
  'Original Data', 'Ratio'
WRITE(*, '(1X,T8,A,T20,A,T33,A,T45,A,T57,A,T69,A/)') &
  'Real', 'Imag.', 'Real', 'Imag.', 'Real', 'Imag.'
CALL FOURN(DATA,NN,NDIM,ISIGN)
DO I=1,4
  J=2*I
  K=2*J
  L=K+(J-1)*NN(1)+(I-1)*NN(2)*NN(1)
  LL=2*L-1
  WRITE(*, '(1X,6F12.2)') DATA(LL),DATA(LL+1),&
    FLOAT(LL),FLOAT(LL+1),DATA(LL)/LL,DATA(LL+1)/(LL+1)
END DO
WRITE(*, '(1X,A,I4)') &
  'The product of transform lengths is:', &
  NN(1)*NN(2)*NN(3)
END

```

计算结果如下:

Double 3 dimensional Transform

Double Trabsf.		Original Data		Ratio	
Real	Imag.	Real	Imag.	Rcal	Imag.
7680.00	8192.00	15.00	16.00	512.00	512.00
52736.00	53247.98	103.00	104.00	512.00	512.00
97792.00	98304.00	191.00	192.00	512.00	512.00
142848.00	143360.00	279.00	280.00	512.00	512.00

The product of transform lengths is: 512

第 13 章 数据的统计描述

本章涉及数理统计的基本计算方法. 13.3 节、13.4 节与 13.7 节属于参数统计推断方法, 13.5 节属于非参数统计推断方法.

13.1 节由给定分布的样本值估计分布的各种矩; 13.2 节用分布的样本值估计分布的中位数, 对于分布的数学期望, 它是比均值更稳健的一种估计; 13.3 节由分布的样本值对分布的参数、均值与方差进行检验, 即由两组数据, 看作两个分布的样本值, 检验两个分布是否有相同的均值与方差. 更进一步, 在 13.4 节与 13.5 节中, 对给定的两组数据, 检验它们是否来自同一分布, 即把这两组数据看作两个随机变量的样本值, 由此检验这两个随机变量的分布是否相同.

13.1 分布的矩——均值、平均差、标准差、方差、斜差和峰态

1. 功能

给定容量为 n 的简单随机样本的样本值 (x_1, x_2, \dots, x_n) , 估计分布的均值 (mean)、平均差 (average deviation)、标准差 (standard deviation)、方差 (variance)、斜差 (skewness) 和峰态 (kurtosis).

2. 方法

$$\text{均值} \quad \bar{x} = \frac{1}{n} \sum_{j=1}^n x_j;$$

$$\text{平均差} \quad ADev(x_1, \dots, x_n) = \frac{1}{n} \sum_{j=1}^n |x_j - \bar{x}|;$$

$$\text{标准差} \quad \sigma(x_1, \dots, x_n) = \sqrt{Var(x_1, \dots, x_n)};$$

$$\text{方差} \quad Var(x_1, \dots, x_n) = \frac{1}{n-1} \sum_{j=1}^n (x_j - \bar{x})^2;$$

$$\text{斜差} \quad Skew(x_1, \dots, x_n) = \frac{1}{n} \sum_{j=1}^n \left[\frac{x_j - \bar{x}}{\sigma} \right]^3;$$

$$\text{峰态} \quad Kurt(x_1, \dots, x_n) = \frac{1}{n} \sum_{j=1}^n \left[\frac{x_j - \bar{x}}{\sigma} \right]^4 - 3.$$

3. 使用说明

MOMENT (DATA,N,AVE,ADEV,SDEV,VAR,SKEW,KURT)

N 整型变量,输入参数,样本容量
 DATA N 个元素的一维实型数组,输入参数,存放样本值
 AVE 实型变量,输出参数,存放均值
 ADEV 实型变量,输出参数,存放平均差
 SDEV 实型变量,输出参数,存放标准差
 VAR 实型变量,输出参数,存放方差
 SKEW 实型变量,输出参数,存放斜差
 KURT 实型变量,输出参数,存放峰态

4. 过程

子过程 MOMENT.

```
SUBROUTINE moment(data1,n,ave,adev,sdev,var,skew,kurt)
```

```
REAL data1(n)
```

```
INTEGER j,n
```

```
REAL ave,adev,var,skew,kurt,s,p
```

```
if(n<=1) pause 'n must be at least 2'
```

```
s=0.
```

```
do j=1,n
```

```
    s=s+data1(j)
```

```
end do
```

```
ave=s/n
```

```
adev=0.
```

```
var=0.
```

```
skew=0.
```

```
kurt=0.
```

```
do j=1,n
```

```
    s=data1(j)-ave
```

```
    adev=adev+abs(s)
```

```
    p=s*s
```

```
    var=var+p
```

```
    p=p*s
```

```
    skew=skew+p
```

```

    p=p * s
    kurt=kurt + p
end do
adev=adev/n
var=var/(n-1)
sdev=sqrt(var)
if(var/=0.)then
    skew=skew/(n * sdev * * 3)
    kurt=kurt/(n * var * * 2)-3.
else
    pause 'no skew or kurtosis when zero variance'
endif
END SUBROUTINE moment

```

5. 例子

验证子过程 D13R1 处理了一个正弦曲线分布值(由于是正弦的半个周期, 所以分布是一对称的峰). 我们已经算出这个分布的理论上的各种矩并记在程序中供比较, 程序 D13R1 如下:

```

PROGRAM D13R1
! Driver for routine MOMENT
PARAMETER (PI=3.14159265,NPTS=10000,NBIN=100,&
    NDAT=NPTS+NBIN)
DIMENSION DATA1(NDAT)
I=1
DO J=1,NBIN
    X=PI * J/NBIN
    NLIM=NINT(SIN(X) * PI/2.0 * NPTS/NBIN)
    DO K=1,NLIM
        DATA1(I)=X
        I=I+1
    END DO
END DO
WRITE(*, '(1X,A/)' ) 'Moments of a sinusoidal distribution'
CALL MOMENT(DATA1,NPTS,AVE,ADEV,SDEV,VAR,SKEW,CURT)
WRITE(*, '(1X,T29,A,T42,A/)' ) 'Calculated', 'Expected'
WRITE(*, '(1X,A,T25,2F12.4)' ) 'Mean :', AVE, PI/2.0

```

```

WRITE(*,'(1X,A,T25,2F12.4)') 'Average Deviation :', &
    ADEV, 0.570796
WRITE(*,'(1X,A,T25,2F12.4)') 'Standard Deviation :', &
    SDEV, 0.683667
WRITE(*,'(1X,A,T25,2F12.4)') 'Variance :', VAR, 0.467401
WRITE(*,'(1X,A,T25,2F12.4)') 'Skewness :', SKEW, 0.0
WRITE(*,'(1X,A,T25,2F12.4)') 'Kurtosis :', CURT, -0.806249
END

```

计算结果如下:

Moments of a sinusoidal distribution

	Calculated	Expected
Mean :	1.5706	1.5708
Average Deviation :	0.5709	0.5708
Standard Deviation :	0.6839	0.6837
Variance :	0.4678	0.4674
Skewness :	-0.0005	0.0000
Kurtosis :	-0.8063	-0.8062

13.2 中位数的搜索

1. 功能

给定容量为 n 的随机样本的样本值 (x_1, \dots, x_n) , 估计分布函数的中位数.

子过程 MDIAN1 用分类样本值估计分布函数的中位数, 适用于样本容量较小的情形; 子过程 MDIAN2 用迭代法估计分布函数的中位数, 适用于样本容量较大的情形.

2. 方法

概率分布函数 $p(x)$ 的中位数 x_{med} 满足下式:

$$\int_{-\infty}^{x_{\text{med}}} p(x) dx = \frac{1}{2} = \int_{x_{\text{med}}}^{\infty} p(x) dx$$

设样本值为 (x_1, \dots, x_n) , 则中位数是样本值中的某 x_{med} , 使 $x_i (i=1, \dots, n)$ 中大于 x_{med} 的个数等于小于 x_{med} 的个数.

(1) 分类法

如果假定样本值中的 $x_i (i=1, \dots, n)$ 为单调排序, 即 x_1, \dots, x_n 为单调增加

的或单调减小的,则

$$x_{\text{med}} = \begin{cases} x_{(n+1)/2} & \text{当 } n \text{ 为奇数时} \\ \frac{1}{2}(x_{n/2} + x_{n/2+1}) & \text{当 } n \text{ 为偶数时} \end{cases}$$

(2) 迭代法

不妨假定样本容量 n 为偶数,则中位数满足方程

$$\sum_{j=1}^n (x_j - x_{\text{med}}) / (|x_j - x_{\text{med}}|) = 0$$

即

$$x_{\text{med}} = \sum_{j=1}^n (x_j / |x_j - x_{\text{med}}|) / \sum_{j=1}^n (1 / |x_j - x_{\text{med}}|)$$

由此不动点形式即得迭代格式.

3. 使用说明

MDIAN1 (X,N,XMED)

MDIAN2 (X,N,XMED)

N 整型变量,输入参数,样本容量

X N 个元素的一维实型数组,输入、输出参数,输入存放样本值,输出时存放排序后的数组值

XMED 实型变量,输出参数,存放中位数的估计值

4. 过程

(1) 子过程 MDIAN1.

```
SUBROUTINE mdian1(x,n,xmed)
```

```
REAL x(n),xmed
```

```
INTEGER n2,n
```

```
! USES sort
```

```
call sort(n,x)
```

```
n2=n/2
```

```
if (2 * n2 == n) then
```

```
    xmed=0.5 * (x(n2) + x(n2+1))
```

```
else
```

```
    xmed = x(n2+1)
```

```
endif
```

```
END SUBROUTINE mdian1
```

(2) 子过程 MDIAN2.

```

SUBROUTINE mdian2(x,n,xmed)
REAL x(n)
PARAMETER (big=1.e30,afac=1.5,amp=1.5)
REAL a,eps,ap,am,xp,xm,xx,sum,sumx,aa,xmed,dum
INTEGER np,nm,n
a=0.5*(x(1)+x(n))
eps=abs(x(n)-x(1))
ap=big
am=-big
1 sum=0.
sumx=0.
np=0
nm=0
xp=big
xm=-big
do j=1,n
  xx=x(j)
  if(xx/=a) then
    if(xx>a) then
      np=np+1
      if(xx<xp) xp=xx
    else if(xx<a) then
      nm=nm+1
      if(xx>xm) xm=xx
    endif
    dum=1./(eps+abs(xx-a))
    sum=sum+dum
    sumx=sumx+xx*dum
  endif
end do
if(np-nm>=2) then
  am=a
  aa=xp+max(0.,sumx/sum-a)*amp
  if(aa>ap) aa=0.5*(a+ap)
  eps=afac*abs(aa-a)

```

```

a=aa
go to 1
else if(nm-np>=2) then
  ap=a
  aa=xm+min(0.,sumx/sum-a)*amp
  if(aa<am) aa=0.5*(a+am)
  eps=afac*abs(aa-a)
  a=aa
  go to 1
else
  if(mod(n,2) /= 0) then
    if(np==nm) then
      xmed=0.5*(xp+xm)
    else if(np>nm) then
      xmed=0.5*(a+xp)
    else
      xmed=0.5*(xm+a)
    endif
  else
    if(np==nm) then
      xmed=a
    else if(np>nm) then
      xmed=xp
    else
      xmed=xm
    endif
  endif
endif
endif
END SUBROUTINE mdian2

```

5. 例子

子过程 MDIAN1 和 MDIAN2 都将得到分布函数的中位数. 在验证程序 D13R2 和 D13R3 中, 我们利用函数过程 GASDEV 见(6.2)节产生了正态分布. 这个分布的均值为零, 方差为 1. 子过程 MDIAN1 还对数据进行排序. 因而, 验证程序 D13R2 打印出这些排了序的数据. 验证程序 D13R2 如下:

```

PROGRAM D13R2
! Driver for routine MDIAN1
PARAMETER (NPTS=50)
! USES GASDEV
DIMENSION DATA1(NPTS)
IDUM=-5
DO I=1,NPTS
    DATA1(I)=GASDEV(IDUM)
END DO
CALL MDIAN1(DATA1,NPTS,XMED)
WRITE(*,'(1X,A/)' )&
    'Gaussian distrib. , zero mean, unit variance'
WRITE(*,'(1X,A,F10.6/)' ) 'Median of DATA1 set is',XMED
WRITE(*,'(1X,A/,(5F12.6))' ) 'Sorted DATA1',&
    (DATA1(I),I=1,50)
END

```

计算结果如下:

Gaussian distrib. , zero mean, unit variance

Median of DATA1 set is -0.188822

Sorted DATA1

-2.334876	-2.217629	-2.099360	-1.924025	-1.872859
-1.524259	-1.418354	-1.347230	-1.329552	-1.120633
0.983745	-0.945489	-0.933141	-0.789795	-0.775194
-0.761360	-0.692489	-0.657364	-0.602143	-0.584651
-0.573552	-0.564484	-0.502163	-0.350424	-0.211478
-0.166166	-0.114347	0.075381	0.150349	0.155883
0.209088	0.298632	0.360088	0.399793	0.437967
0.474571	0.514807	0.521414	0.590622	0.630485
0.646524	0.683157	0.804758	0.845438	1.227279
1.268984	1.307562	1.401248	1.430002	2.485416

验证程序 D13R3 验证了子过程 MDIAN2, 打印出中位数, 并和 MDIAN 的结果进行了比较. 验证程序 D13R3 如下:

```

PROGRAM D13R3
! Driver for routine MDIAN2

```

```

PARAMETER (NPTS=50)
DIMENSION DATA1(NPTS)
! USES GASDEV,MDIAN1
IDUM=-5
DO I=1,NPTS
    DATA1(I)=GASDEV(IDUM)
END DO
CALL MDIAN2(DATA1,NPTS,XMED)
WRITE(*, '(1X,A/)' )&
    'Gaussian distrib. , zero mean, unit variance'
WRITE(*, '(1X,A,F12.6)' ) 'Median according to MDIAN2 is' ,&
    XMED
CALL MDIAN1(DATA1,NPTS,XMED)
WRITE(*, '(1X,A,F12.6/)' ) 'Median according to MDIAN1 is' ,&
    XMED
END

```

计算结果如下:

```

Gaussian distrib. , zero mean, unit variance
Median according to MDIAN2 is --0.188822
Median according to MDIAN1 is --0.188822

```

13.3 均值与方差的显著性检验

1. 功能

设 $\{x_1^{(1)}, \dots, x_{n_1}^{(1)}\}$ 为取自两个分布中第一个分布的简单随机样本, $\{x_1^{(2)}, \dots, x_{n_2}^{(2)}\}$ 为取自第二个分布的简单随机样本, 且两个样本独立, 检验两个分布的均值与方差是否有显著性差异.

子过程 TTEST, TUTEST, TPTEST 用 t 分布检验两个分布的均值是否有显著性差异, 其中子过程 TTEST 适用于有相同方差的情形, 子过程 TUTEST 适用于有不同方差的情形, 子过程 TPTEST 用于配对样本情形.

子过程 AVEVAR 用于计算一个样本的均值和方差.

子过程 FTEST 用 F 分布对两个分布的方差进行显著性检验.

2. 方法

(1) t 分布的概率分布函数

$$A(t|\nu) = \frac{1}{\nu^{1/2} B\left(\frac{1}{2}, \frac{\nu}{2}\right)} \int_{-t}^t \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}} dx$$

$$= 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right)$$

其中 t 为自变量, ν 为 t 分布的自由度, $B(Z, W)$ 为贝塔(beta)函数, $I_x(a, b)$ 为不完全贝塔函数(参考第 4 章).

(2) F 分布的概率分布函数

$$Q(F|\nu_1, \nu_2) = \frac{1}{B(\nu_2/2, \nu_1/2)} \int_F^\infty \frac{\nu_1}{\nu_2} \left(\frac{\nu_1}{\nu_2} x\right)^{\frac{\nu_1}{2}-1} \left(1 + \frac{\nu_1}{\nu_2} x\right)^{-\frac{\nu_1+\nu_2}{2}} dx$$

$$= I_{\nu_2/(\nu_2+\nu_1 F)}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

其中 F 为自变量, (ν_1, ν_2) 为 F 分布的自由度.

(3) 假定两个分布有相同方差, 但可能有不同均值.

零假设 H_0 : 两个分布有相同的均值;

检验统计量 $t = |\bar{x}^{(1)} - \bar{x}^{(2)}| / S_D$,

$$S_D = \sqrt{\frac{1/n_1 + 1/n_2}{n_1 + n_2 - 2}} \cdot \sqrt{\sum_{i=1}^{n_1} |x_i^{(1)} - \bar{x}^{(1)}|^2 + \sum_{i=1}^{n_2} |x_i^{(2)} - \bar{x}^{(2)}|^2}$$

若零假设 H_0 成立, 则认为统计量 t 服从自由度为 $\nu = n_1 + n_2 - 2$ 的 t 分布, 对于 t 的观察值 t_0 , 如果

$$\alpha_0 = P\{|t| > |t_0|\} = 1 - A(t_0|\nu) \leq \alpha$$

(比如 $\alpha = 0.05$ 或 0.01), 则认为均值有显著性差异.

(4) 假定两个分布的方差不相等.

零假设 H_0 : 两个分布的均值相等.

检验统计量 $t = |\bar{x}^{(1)} - \bar{x}^{(2)}| / [Var(x^{(1)})/n_1 + Var(x^{(2)})/n_2]^{1/2}$.

若零假设 H_0 成立, 则这时的统计量 t 当 n_1, n_2 很大时近似地服从 t 分布, 其自由度是

$$\nu = \frac{\left[\frac{Var(x^{(1)})}{n_1} + \frac{Var(x^{(2)})}{n_2}\right]^2}{d}$$

$$d = \frac{\left[\frac{Var(x^{(1)})}{n_1}\right]^2}{n_1 - 1} + \frac{\left[\frac{Var(x^{(2)})}{n_2}\right]^2}{n_2 - 1}$$

(5) 配对样本情形的 t 检验.

设两个样本的容量均为 n , 即 $n_1 = n_2 = n$.

零假设 H_0 : 两个分布的均值相等;

检验统计量 $t = \frac{\bar{x}^{(1)} - \bar{x}^{(2)}}{S_D}$,

$$S_D = \{[Var(x^{(1)}) + Var(x^{(2)}) - 2COV(x^{(1)}, x^{(2)})]/n\}^{1/2}$$

$$COV(x^{(1)}, x^{(2)}) = \frac{1}{n-1} \sum_{i=1}^n (x_i^{(1)} - \bar{x}^{(1)})(x_i^{(2)} - \bar{x}^{(2)})$$

若零假设 H_0 为真, 则认为统计量 t 服从自由度为 $\nu = n-1$ 的 t 分布.

(6) 方差的 F 检验.

零假设 H_0 : 两个分布的方差相等;

检验统计量 $F = Var(x^{(1)})/Var(x^{(2)})$.

若零假设 H_0 为真, 则认为统计量 F 服从自由度为 (n_1-1, n_2-1) 的 F 分布, 分布函数为

$$Q(F|n_1-1, n_2-1)$$

对统计量 F 的观察值 F_0 , 如果

$$Q(F_0|n_1-1, n_2-1) + 1 - Q(F_0|n_2-1, n_1-1)$$

为小值, 则认为方差有显著性差异.

3. 使用说明

(1) AVEVAR (DATA, N, AVE, VAR)

N 整型变量, 输入参数, 样本容量

DATA N 个元素的一维实型数组, 输入参数, 存入样本值

AVE 实型变量, 输出参数, 存放样本均值

VAR 实型变量, 输出参数, 存放样本方差

(2) TTEST (DATA1, N1, DATA2, N2, T, PROB)

TUTEST (DATA1, N1, DATA2, N2, T, PROB)

TPTEST (DATA1, DATA2, N, T, PROB)

FTEST (DATA1, N1, DATA2, N2, F, PROB)

N1 整型变量, 输入参数, 第一个样本的容量

N2 整型变量, 输入参数, 第二个样本的容量

N 整型变量, 输入参数, 配对情形两个样本的容量, $N = N1 = N2$

DATA1 $N1$ 个元素的一维实型数组, 输入参数, 存放第一个样本的样本值, 配对情形 $N1 = N$

DATA2	$N2$ 个元素的一维实型数组,输入参数,存放第二个样本的样本值,配对情形 $N2=N$
T	实型变量,输出参数,检验统计量 t 的观察值
F	实型变量,输出参数,检验统计量 F 的观测值
PROB	实型变量,输出参数,存放显著性水平,若它为小值,则认为所要检验的两个均值(子过程 TTEST, TUTEST, TPTEST 中)或方差(子过程 FTEST 中)有显著性差异

4. 过程

(1) 子过程 AVEVAR.

```

SUBROUTINE avevar(data,n,ave,var)
DIMENSION data(n)
REAL ave,var,s
INTEGER j,n
ave=0.0
var=0.0
do j=1,n
    ave=ave+data(j)
end do
ave=ave/n
do j=1,n
    s=data(j)-ave
    var=var+s*s
end do
var=var/(n-1)
END SUBROUTINE avevar

```

(2) 子过程 TTEST.

```

SUBROUTINE ttest(data1,n1,data2,n2,t,prob)
REAL data1(n1),data2(n2)
! USES avevar,betai
REAL df,var,prob,t,ave1,ave2,var1,var2
INTEGER n1,n2
call avevar(data1,n1,ave1,var1)
call avevar(data2,n2,ave2,var2)

```



```

df=n1+n2-2
var=((n1-1)*var1+(n2-1)*var2)/df
t=(ave1-ave2)/sqrt(var*(1./n1+1./n2))
prob=betai(0.5*df,0.5,df/(df+t**2))
END SUBROUTINE ttest

```

(3) 子过程 TUTEST.

```

SUBROUTINE tutest(data1,n1,data2,n2,t,prob)
DIMENSION data1(n1),data2(n2)
! USES avevar
REAL t,ave1,ave2,var1,var2,prob,df
INTEGER n1,n2
call avevar(data1,n1,ave1,var1)
call avevar(data2,n2,ave2,var2)
t=(ave1-ave2)/sqrt(var1/n1+var2/n2)
df=(var1/n1+var2/n2)**2/((var1/n1)**2/(n1-1)+&
    (var2/n2)**2/(n2-1))
prob=betai(0.5*df,0.5,df/(df+t**2))
END SUBROUTINE tutest

```

(4) 子过程 TPTEST.

```

SUBROUTINE tptest(data1,data2,n,t,prob)
DIMENSION data1(n),data2(n)
! USES avevar,betai
REAL cov,ave1,ave2,var1,var2,df,sd,t,prob
INTEGER j,n
call avevar(data1,n,ave1,var1)
call avevar(data2,n,ave2,var2)
cov=0.
do j=1,n
    cov=cov+(data1(j)-ave1)*(data2(j)-ave2)
end do
df=n-1
cov=cov/df
sd=sqrt((var1+var2-2.*cov)/n)
t=(ave1-ave2)/sd

```

```
prob=betai(0.5 * df,0.5,df/(df+t * * 2))
```

```
END SUBROUTINE tptest
```

(5) 子过程 FTEST.

```
SUBROUTINE ftest(data1,n1,data2,n2,f,prob)
```

```
DIMENSION data1(n1),data2(n2)
```

```
! USES avevar,betai
```

```
REAL ave1,avf2,var1,var2,prob,f,df1,df2
```

```
INTEGER n1,n2
```

```
call avevar(data1,n1,ave1,var1)
```

```
call avevar(data2,n2,ave2,var2)
```

```
if(var1>var2)then
```

```
    f=var1/var2
```

```
    df1=n1-1
```

```
    df2=n2-1
```

```
else
```

```
    f=var2/var1
```

```
    df1=n2-1
```

```
    df2=n1-1
```

```
endif
```

```
prob=betai(0.5 * df2,0.5 * df1,df2/(df2+df1 * f))&
```

```
    +(1.-betai(0.5 * df1,0.5 * df2,df1/(df1+df2/f)))
```

```
END SUBROUTINE ftest
```

5. 例子

(1) t 检验法是对两个有不同的均值的样本进行检验. 验证程序 D13R4 中采用了两组由函数过程 GASDEV(见 6.2 节)所形成的正态型数据 DATA1 和 DATA2. DATA2 是在 DATA1 的基础上再加上 (NSHFT/2.0) * EPS. 然后, DATA1 连续变化 NSHFT 次, 每次 DATA1 再加上 EPS, 利用子过程 TTEST 和 DATA2 进行比较. 注意, 这两个样本有着相同的方差(即 1.0), 而这正是 TTEST 所要求的. 验证程序 D13R4 如下:

```
PROGRAM D13R4
```

```
! Driver for routine TTEST
```

```
PARAMETER (NPTS=1024,MPTS=512,EPS=0.02,NSHFT=10)
```

```
! USES GASDEV
```

```

DIMENSION DATA1(NPTS),DATA2(MPTS)
! Generate Gaussian distributed data
IDUM=-5
DO I=1,NPTS
  DATA1(I)=GASDEV(IDUM)
END DO
DO I=1,MPTS
  DATA2(I)=(NSHFT/2.0)*EPS+GASDEV(IDUM)
END DO
WRITE(*, '(/IX,T4,A,T18,A,T25,A)')&.
      'Shift','T','Probability'
DO I=1,NSHFT+1
  CALL TTEST(DATA1,NPTS,DATA2,MPTS,T,PROB)
  SHIFT=(I-1)*EPS
  WRITE(*, '(IX,F6.2,2F12.2)') SHIFT,T,PROB
  DO J=1,NPTS
    DATA1(J)=DATA1(J)+EPS
  END DO
END DO
END

```

在子过程 TTEST 中需要调用 AVEVAR 和 BETAI,而 BETAI 中还要调用的 BETACF 和 GAMMLN. BETAI, BETACF 和 GAMMLN 来自于第 4 章 4.3 节和 4.1 节. 验证程序 D13R4 中调用的 GASDEV 见 6.2 节. 计算结果如下:

Shift	T	Probability
0.00	-2.83	0.00
0.02	-2.45	0.01
0.04	2.07	0.04
0.06	-1.70	0.09
0.08	-1.32	0.19
0.10	-0.94	0.35
0.12	-0.56	0.58
0.14	-0.18	0.86
0.16	0.20	0.84
0.18	0.58	0.56
0.20	0.96	0.34

(2) 子过程 AVEVAR 是子过程 TTEST 的一个辅助子过程. 它用于计算一个样本的均值和方差. 验证程序 D13R5 构造了一组正态分布 ($I=1, \dots, 11$), 且给出每次改变 $(I-1)$ EPS 和方差 I^2 . 这个级数使读者可以容易地用肉眼检查 AVEVAR 的计算结果. 验证程序 D13R5 如下:

```

PROGRAM D13R5
  ! Driver for routine AVEVAR
  PARAMETER (NPTS=1000,EPS=0.1)
  ! USES GASDEV
  DIMENSION DATA(NPTS)
  ! Generate Gaussian distributed data
  IDUM=-5
  WRITE(*, '(1X,T4,A,T14,A,T26,A)') &
    'Shift', 'Average', 'Variance'
  DO I=1,11
    SHIFT=(I-1)*EPS
    DO J=1,NPTS
      DATA(J)=SHIFT+I*GASDEV(IDUM)
    END DO
    CALL AVEVAR(DATA,NPTS,AVE,VAR)
    WRITE(*, '(1X,F6.2,2F12.2)') SHIFT,AVE,VAR
  END DO
END

```

计算结果如下:

Shift	Average	Variance
0.00	-0.05	0.97
0.10	0.16	3.55
0.20	0.19	9.26
0.30	0.25	16.25
0.40	0.44	26.04
0.50	0.21	34.35
0.60	0.55	45.35
0.70	0.93	65.83
0.80	0.75	80.14
0.90	1.01	95.87
1.00	1.13	114.88

(3) 子过程 TUTEST 也是用于 t 检验, 但它用于两个具有不同方差分布的均值的比较. 验证程序 D13R6A 中采用两个方差分别为 1.0 和 4.0 的分布 DATA1 和 DATA2. 验证程序 D13R6A 如下:

```

PROGRAM D13R6A
! Driver for routine TUTEST
PARAMETER (NPTS=5000,MPTS=1000,EPS=0.02,NSHFT=10,&
    VAR1=1.0,VAR2=4.0)
DIMENSION DATA1(NPTS),DATA2(MPTS)
! USES GASDEV
! Generate two Gaussian distributions of different variance
IDUM=-51773
FCTR1=SQRT(VAR1)
DO I=1,NPTS
    DATA1(I)=FCTR1 * GASDEV(IDUM)
END DO
FCTR2=SQRT(VAR2)
DO I=1,MPTS
    DATA2(I)=(NSHFT/2.0) * EPS+FCTR2 * GASDEV(IDUM)
END DO
WRITE(*, '(1X,A,F6.2)')&
    'Distribution #1 : variance =',VAR1
WRITE(*, '(1X,A,F6.2)')&
    'Distribution #2 : variance =',VAR2
WRITE(*, '(1X,T4,A,T18,A,T25,A)') 'Shift','T','Probability'
DO I=1,NSHFT+1
    CALL TUTEST(DATA1,NPTS,DATA2,MPTS,T,PROB)
    SHIFT=(I-1) * EPS
    WRITE(*, '(1X,F6.2,2F12.2)') SHIFT,T,PROB
    DO J=1,NPTS
        DATA1(J)=DATA1(J)+EPS
    END DO
END DO
END

```

子过程 TUTEST 和验证程序 D13R6A 还需要调用一些子过程, 这和子过程 TTEST 一样, 详见前面的解释. 计算结果如下:

Distribution #1 : variance = 1.00

Distribution #2 : variance = 4.00

Shift	T	Probability
0.00	-4.15	0.00
0.02	-3.83	0.00
0.04	-3.51	0.00
0.06	-3.19	0.00
0.08	-2.88	0.00
0.10	-2.56	0.01
0.12	-2.24	0.03
0.14	-1.92	0.05
0.16	-1.60	0.11
0.18	-1.29	0.20
0.20	-0.97	0.33

(4) 子过程 TPTEST 又前进了一步,不只是比较两个不同方差的分布,还可能有逐点相关. 验证程序 D13R6B 构造了两种情形,一种是相关的分布,一种为不相关的分布. 因而,这时有三个数组. DATA1 是具有零均值和方差为 1 的标准正态分布. DATA2 是 DATA1 加上较小振幅的正态型扰动. DATA3 类似于 DATA2,但由于是独立调用 GASDEV(见 6.2 节),使得 DATA3 的扰动必定和 DATA1 无关. 在分别调用了 AVEVAR 后,DATA1 又加上了一个补偿. 最后,在一系列的变化中,子过程 TPTEST 每步都被调用. 验证程序 D13R6B 如下:

```

PROGRAM D13R6B
! Driver for routine TPTEST
! Compare two correlated distributions vs. two
! uncorrelated distributions
PARAMETER (NPTS=500,EPS=0.01,NSHFT=10,ANOISE=0.3)
! USES GASDEV, AVEVAR
DIMENSION DATA1(NPTS),DATA2(NPTS),DATA3(NPTS)
IDUM=-5
WRITE(*, '(1X,T18,A,T46,A)') 'Correlated:', 'Uncorrelated:'
WRITE(*, '(1X,T4,A,T18,A,T25,A,T46,A,T53,A)') &
    'Shift', 'T', 'Probability', 'T', 'Probability'
OFFSET=(NSHFT/2)*EPS
DO J=1,NPTS

```

```

    GAUSS=GASDEV(IDUM)
    DATA1(J)=GAUSS
    DATA2(J)=GAUSS+ANOISE * GASDEV(IDUM)
    DATA3(J)=GASDEV(IDUM)+ANOISE * GASDEV(IDUM)
END DO
CALL AVEVAR(DATA1,NPTS,AVE1,VAR1)
CALL AVEVAR(DATA2,NPTS,AVE2,VAR2)
CALL AVEVAR(DATA3,NPTS,AVE3,VAR3)
DO J=1,NPTS
    DATA1(J)=DATA1(J)-AVE1+OFFSET
    DATA2(J)=DATA2(J)-AVE2
    DATA3(J)=DATA3(J)-AVE3
END DO
DO I=1,NSHFT+1
    SHIFT=I * EPS
    DO J=1,NPTS
        DATA2(J)=DATA2(J)+EPS
        DATA3(J)=DATA3(J)+EPS
    END DO
    CALL TPTEST(DATA1,DATA2,NPTS,T1,PROB1)
    CALL TPTEST(DATA1,DATA3,NPTS,T2,PROB2)
    WRITE(*, '(1X,F6.2,2X,2F12.4,4X,2F12.4)') &
        SHIFT,T1,PROB1,T2,PROB2
END DO
END

```

计算结果如下:

Shift	Correlated:		Uncorrelated:	
	T	Probability	T	Probability
0.01	3.1199	0.0019	0.6287	0.5298
0.02	2.3399	0.0197	0.4716	0.6374
0.03	1.5600	0.1194	0.3144	0.7534
0.04	0.7800	0.4357	0.1572	0.8751
0.05	0.0000	1.0000	0.0000	1.0000
0.06	-0.7800	0.4358	-0.1572	0.8751
0.07	-1.5600	0.1194	-0.3144	0.7534
0.08	-2.3399	0.0197	-0.4716	0.6374

0.09	-3.1199	0.0019	-0.6287	0.5298
0.10	-3.8999	0.0001	-0.7859	0.4323
0.11	-4.6799	0.0000	-0.9431	0.3460

(5) F 检验(子过程 FTEST)用于检验两个分布的方差是否相同. 验证程序 D13R7 建立了两个方差为 1 的正态分布 DATA1 和 DATA2. 然后, 第三个数组 DATA3 是给 DATA2 乘以变化的值 FACTR, FACTR 在(1.0, 1.1)之间变化. F 检验的效果能从概率 PROB 计算得到. 子过程 FTEST 中要调用 AVEVAR 和 BETAI(见 4.3 节), BETAI 中还需调用 BETACF 和 GAMMLN(见 4.3 节和 4.1 节), D13R7 中所调用的 GASDEV(见 6.2 节), 验证程序 D13R7 如下:

```

PROGRAM D13R7
! Driver for routine FTEST
PARAMETER (NPTS=1000,MPTS=500,EPS=0.01,NVAL=10)
DIMENSION DATA1(NPTS),DATA2(MPTS),DATA3(MPTS)
! USES GASDEV
! Generate two Gaussian distributions with
! different variances
IDUM=-13
DO J=1,NPTS
  DATA1(J)=GASDEV(IDUM)
END DO
DO J=1,MPTS
  DATA2(J)=GASDEV(IDUM)
END DO
WRITE(*, '(1X,T5,A,F5.2)') 'Variance 1 = ',1.0
WRITE(*, '(1X,T5,A,T21,A,T30,A)') &
  'Variance 2', 'Ratio', 'Probability'
DO I=1,NVAL+1
  VAR=1.0+(I-1)*EPS
  FACTOR=SQRT(VAR)
  DO J=1,MPTS
    DATA3(J)=FACTOR*DATA2(J)
  END DO
  CALL FTEST(DATA1,NPTS,DATA3,MPTS,F,PROB)
  WRITE(*, '(1X,F11.4,2X,2F12.4)') VAR,F,PROB
END DO
END

```


计算结果如下:

Variance 1	= 1.00	
Variance 2	Ratio	Probability
1.0000	1.0019	0.9874
1.0100	1.0081	0.9107
1.0200	1.0181	0.8108
1.0300	1.0280	0.7147
1.0400	1.0380	0.6237
1.0500	1.0480	0.5389
1.0600	1.0580	0.4608
1.0700	1.0680	0.3901
1.0800	1.0779	0.3270
1.0900	1.0879	0.2713
1.1000	1.0979	0.2229

13.4 分布拟合的 χ^2 检验

1. 功能

设 $(\zeta_1, \dots, \zeta_n)$ 是来自随机变量 ζ 的一个简单随机样本, 子过程 CHSONE 检验:

零假设 $H_0: (\zeta_1, \dots, \zeta_n)$ 来自已知分布 $F_0(x, \theta)$, 其中 $\theta = (\theta_1, \dots, \theta_k)$ 为未知参数.

再设 (η_1, \dots, η_m) 是来自随机变量 η 的随机样本, 并设这两个样本是相互独立的, 子过程 CHSTWO 检验:

零假设 $H_0: (\zeta_1, \dots, \zeta_n)$ 与 (η_1, \dots, η_m) 来自同一分布.

2. 方法

(1) χ^2 分布的概率分布函数

$$P(\chi^2 | \nu) = \left[2^{\nu/2} \Gamma\left(\frac{\nu}{2}\right) \right]^{-1} \int_0^{\chi^2} e^{-\frac{t}{2}} t^{\frac{\nu}{2}-1} dt = P\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

其中 χ^2 为自变量, ν 为 χ^2 分布随机变量的自由度, $P(a, x)$ 为不完全 Γ 函数, 记

$$Q(\chi^2 | \nu) = Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

其中 $Q(a, x) = 1 - P(a, x)$.

(2) 检验零假设 $H_0: (\zeta_1, \dots, \zeta_n)$ 来自已知分布 $F_0(x, \theta)$, 其中 $\theta = (\theta_1, \dots, \theta_k)$ 为未知参数.

把 ζ 的一切可能值的集合 X 进行分割:

$$X = \bigcup_{i=1}^r A_i, \quad A_i \cap A_j = \varnothing, \quad i \neq j$$

记 n_i 为样本 $(\zeta_1, \dots, \zeta_n)$ 的样本值中出现在 A_i 中的频数, E_i 为在 H_0 为真时样本值落入 A_i 的理论期望频数, 即 $E_i = np_i$, $p_i = p_i(\theta) = p\{\zeta \in A_i | H_0\}$.

$$\text{检验统计量 } \chi^2 = \sum_{i=1}^r \frac{(n_i - E_i)^2}{E_i}.$$

若 H_0 为真, E_i 中的未知参数 θ 用其最大似然估计 $\hat{\theta}$ 代换, 则统计量 χ^2 的极限分布是 χ^2 分布, 自由度为 $r - k - 1$.

(3) 检验零假设 $H_0: (\zeta_1, \dots, \zeta_n)$ 与 (η_1, \dots, η_n) 来自同一分布.

把 ζ 与 η 的一切可能值集合 X 进行分割:

$$X = \bigcup_{i=1}^r A_i, \quad A_i \cap A_j = \varnothing, \quad i \neq j$$

记 R_i 为样本 $(\zeta_1, \zeta_2, \dots, \zeta_n)$ 的样本值中落入 A_i 的个数, S_i 为样本 (η_1, \dots, η_n) 的样本值中落入 A_i 的个数.

$$\text{检验统计量 } \chi^2 = \sum_{i=1}^r \frac{(R_i - S_i)^2}{R_i + S_i}.$$

如果对样本或 R_i 与 $S_i (i=1, 2, \dots, r)$ 有 k 个附加约束

$$\varphi_j(R_1, \dots, R_r; S_1, \dots, S_r) = 0, \quad j = 1, \dots, k$$

则当 H_0 为真时, 检验统计量 χ^2 服从自由度为 $\nu = r - k$ 的 χ^2 分布.

3. 使用说明

(1) CHSONE(BINS, EBINS, NBINS, KNSTRN, DF, CHSQ, PROB)

NBINS	整型变量, 输入参数, ζ 的一切可能值集合的分割个数
BINS	NBINS 个元素的一维实型数组, 输入参数, 其中第 i 个元素存放样本值落入 A_i 的个数
EBINS	NBINS 个元素的一维实型数组, 输入参数, 其中第 i 个元素存放样本值落入 A_i 的期望频数
KNSTRN	整型变量, 输入参数, 存放已知分布所含未知参数的个数, 即对数组 EBINS 的约束条件数减 1
DF	实型变量, 输出参数, 统计量的自由度
CHSQ	实型变量, 输出参数, 统计量的观察值

PROB	实型变量,输出参数,显著性水平,若它为小值,则认为样本不是来自已知分布
(2) CHSTWO(BINS1,BINS2,NBINS,KNSTRN,DF,CHSQ,PROB)	
NBINS	整型变量,输入参数, ζ 与 η 两者所有可能取值集合的分割数
BINS1	NBINS 个元素的一维实型数组,输入参数,其第 i 个元素为样本 $(\zeta_1, \dots, \zeta_n)$ 的样本值中落入可能值集合的第 i 个分割 A_i 的数目
BINS2	NBINS 个元素的一维实型数组,输入参数,其第 i 个元素存放样本 (η_1, \dots, η_n) 的样本值中落入可能值集合的第 i 个分割 A_i 的数目
KNSTRN	整型变量,输入参数,样本的约束条件的个数减 1
DF	实型变量,输出参数,存放统计量的自由度
CHSQ	实型变量,输出参数,存放统计量的观测值
PROB	实型变量,输出参数,显著性水平,若它为小值,则认为两个样本不是来自同一分布

4. 过程

(1) 子过程 CHSONE.

```

SUBROUTINE chsone(bins,ebins,nbins,knstrn,df,&
                  chsq,prob)
  DIMENSION bins(nbins),ebins(nbins)
  REAL df,chsqa,prob
  INTEGER j,nbins,knstrn
  ! USES gammq
  df=nbins-1-knstrn
  chsq=0.
  do j=1,nbins
    if(ebins(j)<=0.) pause 'bad expected number'
    chsq=chsqa+(bins(j)-ebins(j)) * * 2/ebins(j)
  end do
  prob=gammq(0.5 * df,0.5 * chsq)
END SUBROUTINE chsone

```

(2) 子过程 CHSTWO.

```

SUBROUTINE chstwo(bins1,bins2,nbins,knstrn,df,chsqr,&
                  prob)
DIMENSION bins1(nbins),bins2(nbins)
! USES gammq
REAL chsq,prob,df
INTEGER j,nbins,knstrn
df=nbins-1-knstrn
chsqr=0.
do j=1,nbins
  if(bins1(j) /= 0. .and. bins2(j) /= 0. )then
    df=df-1.
  else
    chsq=chsqr+(bins1(j)-bins2(j)) ** 2/(bins1(j)+&
      bins2(j))
  endif
enddo
prob=gammq(0.5*df,0.5*chsqr)
END SUBROUTINE chstwo

```

5. 例子

(1) 子过程 CHSONE 和 CHSTWO 比较了两组基于 χ^2 检验的分布. 特别是 CHSONE 将一组分布数据和一组期望的分布进行了比较. 验证程序 D13R8 采用了子过程 EXPDEV 构造了一个指数分布 BINS(I). 然后, 又构造了一个数组 EBINS(I) 作为期望的结果 (一个无统计扰动的光滑指数衰减). 子过程 CHSONE 对 EBINS 和 BINS 进行了比较, 并给出了 χ^2 和概率. 验证程序 D13R8 如下:

```

PROGRAM D13R8
! Driver for routine CHSONE
PARAMETER (NPTS=2000,NBINS=10)
! USES EXPDEV
DIMENSION BINS(NBINS),EBINS(NBINS)
IDUM=-15
DO J=1,NBINS
  BINS(J)=0.0

```

```

END DO
DO I=1,NPTS
  X=EXPDEV(IDUM)
  IBIN=X * NBINS/3.0-1
  IF (IBIN.L.E. NBINS) BINS(IBIN)=BINS(IBIN)+1.0
END DO
DO I=1,NBINS
  EBINS(I)=3.0 * NPTS/NBINS * EXP(-3.0 * (I-0.5)/NBINS)
END DO
CALL CHSONE(BINS,EBINS,NBINS,-1,DF,CHSQ,PROB)
WRITE(*, '(1X,T10,A,T25,A)') 'Expected', 'Observed'
DO I=1,NBINS
  WRITE(*, '(1X,2F15.2)') EBINS(I),BINS(I)
END DO
WRITE(*, '(1X,T9,A,E12.4)') 'Chi-squared:',CHSQ
WRITE(*, '(1X,T9,A,E12.4)') 'Probability:',PROB
END

```

子过程 CHSONE 中调用了 GAMMQ(见 4.2 节),而 GAMMQ 中又调用了 GSER 和 GCF(见 4.2 节),GSER 和 GCF 中又调用了 GAMMLN(见 4.1 节). 验证程序 D13R8 中调用了 EXPDEV(见 6.2 节). 计算结果如下:

Expected	Observed
516.42	533.00
382.58	356.00
283.42	305.00
209.96	189.00
155.54	165.00
115.23	130.00
85.36	81.00
63.24	50.00
46.85	48.00
34.71	29.00
Chi-squared: 0.1254E+02	
Probability: 0.2503E+00	

(2) 子过程 CHSTWO 再次用 χ^2 检验比较了两个箱式分布 BINS1 和 BINS2. 验证程序 D13R9 以同样的方式提供了这两个分布. 每个分布是由取自

于指数分布的 2000 个随机数所组成,且将这些数放入 10 个箱内,然后,利用 CHSTWO 计算这两组数据的 χ^2 值和概率 PROB. 验证程序 D13R9 如下:

```

PROGRAM D13R9
  ! Driver for routine CHSTWO
  PARAMETER (NPTS=2000,NBINS=10)
  ! USES EXPDEV
  DIMENSION BINS1(NBINS),BINS2(NBINS)
  IDUM=-17
  DO J=1,NBINS
    BINS1(J)=0.0
    BINS2(J)=0.0
  END DO
  DO I=1,NPTS
    X=EXPDEV(IDUM)
    IBIN=X * NBINS/3.0+1
    IF (IBIN.LE.NBINS) BINS1(IBIN)=BINS1(IBIN)+1.0
    X=EXPDEV(IDUM)
    IBIN=X * NBINS/3.0+1
    IF (IBIN.LE.NBINS) BINS2(IBIN)=BINS2(IBIN)+1.0
  END DO
  CALL CHSTWO(BINS1,BINS2,NBINS,-1,DF,CHSQ,PROB)
  WRITE(*,'(1X,T10,A,T25,A)') 'Dataset 1','Dataset 2'
  DO I=1,NBINS
    WRITE(*,'(1X,2F15.2)') BINS1(I),BINS2(I)
  END DO
  WRITE(*,'(/1X,T10,A,E12.4)') 'Chi-squared:',CHSQ
  WRITE(*,'(1X,T10,A,E12.4)') 'Probability:',PROB
  END

```

子过程 CHSTWO 中调用了 GAMMQ,其详细解释见子过程 CHSONE 中的解释. 计算结果如下:

Dataset 1	Dataset 2
528.00	556.00
406.00	380.00
257.00	277.00
230.00	193.00

163.00	151.00
110.00	96.00
70.00	98.00
64.00	66.00
57.00	47.00
27.00	33.00
Chi-squared: 0.1324E+02	
Probability: 0.2107E+00	

13.5 分布拟合的 K-S 检验法

1. 功能

设已知一累积分布函数 $P(x)$, 随机样本 $(\zeta_1, \dots, \zeta_n)$ 来自分布 $F(x)$, 子过程 KSONE 用 K-S (柯尔莫戈洛夫-斯米尔诺夫, Kolmogorov-Smirnov) 检验法检验:

零假设 $H_0: F(x) = P(x)$.

再设随机样本 (η_1, \dots, η_m) 来自分布 $G(x)$, 子过程 KSTWO 用 K-S 检验法检验:

零假设 $H_0: F(x) = G(x)$.

函数子过程 PROBKS 用于计算 K-S 概率分布函数 $Q_k(\lambda)$ 的值.

2. 方法

(1) 检验零假设 $H_0: F(x) = P(x)$.

检验统计量 $D = \sup_{-\infty < x < \infty} |S_n(x) - P(x)|$, 其中 $S_n(x)$ 为对应于样本 $(\zeta_1, \dots, \zeta_n)$ 的经验分布函数.

定理 (柯尔莫戈洛夫) 设 $F(x)$ 是连续分布函数, $(\zeta_1, \dots, \zeta_n)$ 是取自分布 $F(x)$ 的简单随机样本, $S_n(x)$ 为对应的经验分布函数, 记

$$D_n = \sup_{-\infty < x < \infty} |S_n(x) - F(x)|$$

则

$$\lim_{n \rightarrow \infty} P\{\sqrt{n} D_n \leq \lambda\} = L(\lambda) \quad (\lambda > 0)$$

其中 $L(\lambda) = 1 - 2 \sum_{j=1}^{\infty} (-1)^{j-1} e^{-2j^2 \lambda^2}$.

由此, 如果 H_0 为真, 则当 $n \rightarrow \infty$ 时 $\sqrt{n} D$ 的分布函数将收敛到 $L(x)$, 设 D

的观测值为 d , 则当

$$\begin{aligned}\alpha &= P\{D > d\} = P\{\sqrt{n}D > \sqrt{n}d\} \\ &= 1 - L(\sqrt{n}d) \equiv Q_k(\sqrt{n}d)\end{aligned}$$

为小值时, 拒绝假设, 即认为样本 $(\zeta_1, \dots, \zeta_n)$ 不是来自 $P(x)$.

(2) 检验零假设 $H_0: F(x) = G(x)$.

检验统计量 $D = \sup_{-\infty < x < \infty} |R_n(x) - S_m(x)|$, 其中 $R_n(x)$ 为对应于样本 $(\zeta_1, \dots, \zeta_n)$ 的经验分布函数, $S_m(x)$ 为对应于样本 (η_1, \dots, η_m) 的经验分布函数.

定理 设 $F(x) = G(x)$ 为连续函数, 则有

$$\lim_{\substack{n \rightarrow \infty \\ m \rightarrow \infty}} P\left\{\sqrt{\frac{nm}{n+m}} D < \lambda\right\} = \begin{cases} L(\lambda) & (\lambda > 0) \\ 0 & (\lambda \leq 0) \end{cases}$$

由此定理, 统计量 $\sqrt{\frac{nm}{n+m}} D$ 的概率分布函数为 $L(x) = 1 - 2 \sum_{j=1}^{\infty} (-1)^{j-1} \cdot e^{-2j^2 x^2}$. 记 $Q_k(\lambda) = 1 - L(\lambda)$, 则对 D 的观测值 d , 当 $\alpha = P\{D > d\} = Q_k\left(\sqrt{\frac{nm}{n+m}} d\right)$ 为小值时, 即认为两个样本不是来自同一分布.

3. 使用说明

(1) KSONE(DATA, N, D, PROB)

N 整型变量, 输入参数, 样本容量
DATA N 个元素的一维实型数组, 输入参数, 存放样本观测值
D 实型变量, 输出参数, K-S 统计量的观测值
PROB 实型变量, 输出参数, 显著性水平, 若它为小值, 则认为样本不是来自已知的分布

(2) KSTWO(DATA1, N1, DATA2, N2, D, PROB)

N1 整型变量, 输入参数, 第一个样本 $(\zeta_1, \dots, \zeta_n)$ 的容量, $N1 = n$
N2 整型变量, 输入参数, 第二个样本 (η_1, \dots, η_m) 的容量, $N2 = m$
DATA1 $N1$ 个元素的一维实型数组, 输入参数, 存放第一个样本值
DATA2 $N2$ 个元素的一维实型数组, 输入参数, 存放第二个样本值
D 实型变量, 输出参数, 存放 K-S 统计量的观测值
PROB 实型变量, 输出参数, 显著性水平, 若它为小值, 则认为两个样本的经验分布函数具有显著性差异

(3) PROBK(SALAM)

ALAM 实型变量, 输入参数, 函数自变量

4. 过程

(1) 子过程 KSONE.

```

SUBROUTINE ksone(data1,n,func,d,prob)
DIMENSION data1(n)
! USES sort
REAL en,d,fo,fn,ff,dt,prob
INTEGER j,n
call sort(n,data1)
en=n
d=0.
fo=0.
do j=1,n
    fn=j/en
    ff=func(data1(j))
    dt=amax1(abs(fo-ff),abs(fn-ff))
    if(dt>d) d=dt
    fo=fn
end do
prob=probks(sqrt(en)*d)
END SUBROUTINE ksone

```

(2) 子过程 KSTWO.

```

SUBROUTINE kstwo(data1,n1,data2,n2,d,prob)
DIMENSION data1(n1),data2(n2)
! USES probks,sort
REAL en1,en2,fo1,fo2,fn1,dt
INTEGER j1,j2,n1,n2
call sort(n1,data1)
call sort(n2,data2)
en1=n1
en2=n2
j1=1
j2=1
fo1=0.
fo2=0.
d=0.

```

```

do while (j1 <= n1 .and. j2 <= n2)
  if (data1(j1) < data2(j2)) then
    fn1 = j1/en1
    dt = amax1(abs(fn1 - fo2), abs(fo1 - fo2))
    if (dt > d) d = dt
    fo1 = fn1
    j1 = j1 + 1
  else
    fn2 = j2/en2
    dt = amax1(abs(fn2 - fo1), abs(fo2 - fo1))
    if (dt > d) d = dt
    fo2 = fn2
    j2 = j2 + 1
  endif
end do
prob = probks(sqrt(en1 * en2 / (en1 + en2)) * d)
END SUBROUTINE kstwo

```

(3) 函数过程 PROBKS.

```

FUNCTION probks(alam)
  PARAMETER(eps1 = 0.001, eps2 = 1.e-8)
  REAL a2, fac, term, termbf, alam
  INTEGER j
  a2 = -2. * alam * * 2
  fac = 2.
  probks = 0.
  termbf = 0.
  do j = 1, 100
    term = fac * exp(a2 * j * * 2)
    probks = probks + term
    if (abs(term) < eps1 * termbf .or. abs(term) < &
      eps2 * probks) return
    fac = -fac
    termbf = abs(term)
  end do
  probks = 1.
END FUNCTION probks

```

5. 例子

(1) 柯尔莫戈洛夫-斯米尔诺夫(K-S)检验法,在 KSONE 和 KSTWO 里被应用到单个自变量的二项分布中, KSONE 是将 K-S 准则用于比较一组数据和另一已知的分布,而 KSTWO 是采用 K-S 准则去比较两组数据.验证子过程 KSONE 的程序 D13R10 构造了具有逐步增加方差的一组正态分布数据,且将这组累积分布函数和已知的结果进行比较.这个已知结果为误差函数(可调用 4.2 节中子过程 ERF).在经验分布中增加的方差应该是减少了它落入所描述的比较函数的分布中的可能性. KSONE 要调用本节中所介绍的子过程 PROBK 及 SORT(见 7.2 节).验证程序 D13R10 中调用了 GASDEV(见 6.2 节).程序 D13R10 如下:

```
PROGRAM D13R10
! Driver for routine KSONE
PARAMETER (NPTS=1000,EPS=0.1)
! USES GASDEV
DIMENSION DATA(NPTS)
EXTERNAL FUNC
IDUM= 5
WRITE(*, '(1X,T5,A,T24,A,T44,A/)' )8.
      'Variance Ratio','K-S Statistic','Probability'
DO J=1,11
  VAR= 1.0+(J-1)*EPS
  FACTR=SQRT(VAR)
  DO J=1,NPTS
    DATA(J)=FACTR*ABS(GASDEV(IDUM))
  END DO
  CALL KSONE(DATA,NPTS,FUNC,D,PROB)
  WRITE(*, '(1X,F14.6,F18.6,E20.4)' ) VAR,D,PROB
END DO
END PROGRAM
FUNCTION FUNC(X)
! USES ERF
  Y=X*SQRT(2.0)
  FUNC=ERF(Y)
END FUNCTION FUNC
```

计算结果如下:

Variance Ratio	K S Statistic	Probability
1.000000	0.027478	0.4370E+00
1.100000	0.028161	0.4060E+00
1.200000	0.050516	0.1215E-01
1.300000	0.090788	0.1386E-06
1.400000	0.105994	0.3489E-09
1.500000	0.093471	0.5156E-07
1.600000	0.105488	0.4322E-09
1.700000	0.135101	0.2801E-15
1.800000	0.145440	0.8470E-18
1.900000	0.160239	0.9971E-22
2.000000	0.164238	0.7442E-23

(2) 子过程 KSTWO 比较了两组非二项分布的数据 DATA1 和 DATA2. 在验证程序 D13R11 中, 它们二者是正态分布, 但 DATA2 具有逐步增加的方差. 另外, D13R11 类似于 D13R10. 验证程序 D13R11 如下:

```

PROGRAM D13R11
! Driver for routine KSTWO
PARAMETER (N1=2000,N2=1000,EPS=0.1)
! USES GASDEV
DIMENSION DATA1(N1),DATA2(N2)
IDUM=-1357
DO J=1,N1
  DATA1(J)=GASDEV(IDUM)
END DO
WRITE(*, '(1X,T6,A,T26,A,T46,A/)' ) &
  'Variance Ratio', 'K-S Statistic', 'Probability'
DO I=1,11
  VAR=1.0+(I-1)*EPS
  FACTR=SQRT(VAR)
  DO J=1,N2
    DATA2(J)=FACTR*GASDEV(IDUM)
  END DO
  CALL KSTWO(DATA1,N1,DATA2,N2,D,PROB)
  WRITE(*, '(1X,F15.6,F19.6,E20.4)' ) VAR,D,PROB

```

```
END DO
```

```
END
```

计算结果如下:

Variance Ratio	K-S Statistic	Probability
1.000000	0.047500	0.9874E-01
1.100000	0.032000	0.5021E+00
1.200000	0.046500	0.1119E+00
1.300000	0.063500	0.9249E-02
1.400000	0.083000	0.2051E-03
1.500000	0.072000	0.1992E-02
1.600000	0.057000	0.2628E-01
1.700000	0.083000	0.2051E-03
1.800000	0.087500	0.7372E-04
1.900000	0.079500	0.4378E-03
2.000000	0.091000	0.3205E-04

(3) PROBKS 是 KSONE 和 KSTWO 的一个辅助子过程. 在验证程序 D13R12 中所产生的函数我们选用图像来表示. 验证程序 D13R12 如下:

```
PROGRAM D13R12
! Driver for routine PROBKS
CHARACTER TEXT(50)*1
WRITE(*,*)&
  'Probability func. Kolmogorov-Smirnov statistic'
WRITE(*, '(/1X,T3,A,T15,A,T27,A)')&
  'Lambda;', 'Value;', 'Graph;'
NPTS=20
EPS=0.1
SCALE=40.0
DO I=1,NPTS
  ALAM=I*EPS
  VALUE=PROBKS(ALAM)
  TEXT(1)='*'
  DO J=1,50
    IF (J.LE. NINT(SCALE*VALUE)) THEN
      TEXT(J):='*'

```

```

      ELSE
        TEXT(J)=' '
      ENDIF
    END DO
    WRITE(*, '(1X,F9.6,F12.6,4X,50A1)') ALAM,VALUE,&
      (TEXT(J),J=1,50)
  END DO
END

```

计算结果如下:

Probability func. Kolmogorov-Smirnov statistic

Lambda: Value: Graph:

0.100000	1.000000	*****xxyxxxxx
0.200000	1.000000	*****
0.300000	0.999991	*****xy
0.400000	0.997192	*****
0.500000	0.963945	*****
0.600000	0.864283	*****
0.700000	0.711235	*****
0.800000	0.544142	*****
0.900000	0.392731	*****
1.000000	0.270000	*****
1.100000	0.177718	*****
1.200000	0.112250	****
1.300000	0.068092	***
1.400000	0.039682	**
1.500000	0.022218	*
1.600000	0.011952	
1.700000	0.006177	
1.800000	0.003068	
1.900000	0.001464	
2.000000	0.000671	

第 14 章 解常微分方程组

本章所涉及的常微分方程组的积分限于初值问题. 科学技术领域中的许多问题往往可归结为微分方程, 恰好遇到一个一阶微分方程的情形是少见的, 通常是多个未知函数, 如 (y_1, y_2, \dots, y_N) 多个一阶常微分方程

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_N), \quad i = 1, 2, \dots, N$$

具有初始条件

$$y_i(x_0) = y_{i0}, \quad i = 1, 2, \dots, N$$

一般高阶微分方程总可以改写成一阶常微分方程组. 本章介绍两种实用的方法: (1) 龙格-库塔方法 (RK4, RKDUMB 和 ODEINT); (2) 布里奇-斯托尔 (Bulirsch-Stoer) 外推法 (BSSTEP, MMID, RZEXTR 和 PZEXTR). 在一般情况下, 如果只希望方便而不要求高精度的话, 我们推荐带有步长控制的四阶龙格-库塔方法. 如果需要高精度的计算结果, 那么可采用外推法. 历史上曾被广泛使用过的预估-校正法由于在今天已不起重要作用, 因而其子过程没有被收入.

14.1 定步长四阶龙格-库塔(Runge-Kutta)法

1. 功能

解一阶常微分方程组的初值问题

$$\begin{cases} \frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_N) \\ y_i(x_0) = y_{i0}, \\ x > x_0, \quad y_i \in C'(a, \infty) \end{cases} \quad i = 1, 2, \dots, N$$

本节给出两个子过程. 子过程 RK4 由给定步长 h 和初始点上的值用四阶龙格-库塔法求解给定的初值问题. 调用它一次向前积分一步, 一般用于求解某一小区间中某几点的函数值. 子过程 RKDUKB 是连续调用子过程 RK4 计算出某一区间上的值, 此时该区间长度可较大, 但积分步长都较小. 一般来说, 在精度要求不高时, 可采用该方法.

2. 方法

四阶龙格-库塔法的计算公式为

$$\begin{aligned}
 Y^{(k+1)} &= (y_1^{(k+1)}, \dots, y_N^{(k+1)}) \\
 &= Y^{(k)} + \frac{1}{6} (K^{(1)} + 2K^{(2)} + 2K^{(3)} + K^{(4)})
 \end{aligned}$$

其中

$$\begin{aligned}
 K^{(1)} &= hF(x_k, Y^{(k)}) \\
 K^{(2)} &= hF(x_k + \frac{1}{2}h, Y^{(k)} + \frac{1}{2}K^{(1)}) \\
 K^{(3)} &= hF(x_k + \frac{1}{2}h, Y^{(k)} + \frac{1}{2}K^{(2)}) \\
 K^{(4)} &= hF(x_k + h, Y^{(k)} + K^{(3)}) \\
 F(x, y) &= (f_1(x, Y), \dots, f_N(x, Y))^T \\
 Y &= (y_1, y_2, \dots, y_N)^T
 \end{aligned}$$

3. 使用说明

(1) RK4(Y, DYDX, N, X, H, YOUT, DERIVS)

N	整型变量, 输入参数, 方程个数
H	实型变量, 输入参数, 积分步长
X	实型变量, 输入参数, 自变量初值
Y	含 N 个元素的一维实型数组, 输入参数, 调用前需把积分初值存放到本数组
DYDX	含 N 个元素的一维实型数组, 输入参数, 调用前存放方程组右端函数在初始点上的值
YOUT	含 N 个元素的一维实型数组, 输出参数, 存放积分结果
DERIVS	子过程 DERIVS(X, Y, DYDX), 由用户自编, 其功能是计算方程组右端的函数值, X 为实型变量, 函数值存放在数组 DYDX(N) 中

(2) RKDUMB(VSTART, NVAR, X1, X2, NSTEP, DERIVS)

VSTART	含 NVAR 个元素的一维实型数组, 输入参数, 存放积分初值
NVAR	整型变量, 输入参数, 方程个数
DERIVS	子过程 DERIVS(X, Y, DYDX), 由用户自编, 其功能是计算方程组右端的函数值, X 为实型变量, 函数值存放在数组 DYDX(N) 中
X1	实型变量, 输入参数, 自变量初值
X2	实型变量, 输入参数, 自变量终值
NSTEP	整型变量, 输入参数, 区间 [X1, X2] 上要积分的步数

(3) 公用块 PATH

X 一维实型数组, 输出自变量值

Y 二维实型数组, 输出该常微分方程组初值问题的解

4. 过程

(1) 子过程 RK4.

```

SUBROUTINE rk4(y,dydx,n,x,h,yout,derivs)
PARAMETER (nmax=10)
DIMENSION y(n),dydx(n),yout(n),yt(nmax),dym(nmax),&
           dym(nmax)
REAL hh,h6,xh,x,h
INTEGER i,n
hh=h*0.5
h6=h/6.
xh=x+hh
do i=1,n
    yt(i)=y(i)+hh*dydx(i)
end do
call derivs(xh,yt,dym)
do i=1,n
    yt(i)=y(i)+hh*dym(i)
end do
call derivs(xh,yt,dym)
do i=1,n
    yt(i)=y(i)+h*dym(i)
    dym(i)=dyt(i)+dym(i)
end do
call derivs(x+h,yt,dym)
do i=1,n
    yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.*dym(i))
end do
END SUBROUTINE rk4

```

(2) 子过程 RKDUMB.

```

SUBROUTINE rk dumb(vstart,nvar,x1,x2,nstep,derivs)
PARAMETER (nmax=10)
COMMON /path/ xx(200),y(10,200)

```

```

! USES rk4
DIMENSION vstart(nvar),v(nmax),dv(nmax)
REAL x,x1,x2,h
INTEGER i,nstep,k
EXTERNAL derivs
do i=1,nvar
    v(i)=vstart(i)
    y(i,1)=v(i)
end do
xx(1)=x1
x=x1
h=(x2-x1)/nstep
do k=1,nstep
    call derivs(x,v,dv)
    call rk4(v,dv,nvar,x,h,v,derivs)
    if(x+h==x)&
        pause 'stepsize not significant in rk4dumb.'
    x=x+h
    xx(k+1)=x
    do i=1,nvar
        y(i,k+1)=v(i)
    end do
end do
END SUBROUTINE rk4dumb

```

5. 例子

求解初值问题

$$\begin{cases}
 \frac{dy_1}{dx} = -y_2 \\
 \frac{dy_2}{dx} = y_1 - \frac{1}{x}y_2 \\
 \frac{dy_3}{dx} = y_2 - \frac{2}{x}y_3 \\
 \frac{dy_4}{dx} = y_3 - \frac{3}{x}y_4 \\
 y_1(x_1) = J_0(x_1), \quad y_2(x_1) = J_1(x_1) \\
 y_3(x_1) = J_2(x_1), \quad y_4(x_1) = J_3(x_1)
 \end{cases}$$

积分区间为 $[x_1, x_2]$, 精确解为

$$y_1 = J_0(x), \quad y_2 = J_1(x)$$

$$y_3 = J_2(x), \quad y_4 = J_3(x)$$

其中 J_0, J_1, J_2, J_3 为贝塞尔函数.

(1) 利用子过程 RK4 以等步长求解区间 $[0.2, 1.0]$ 中 5 个点的函数值. 验证程序 D14R1 如下:

```

PROGRAM D14R1
! Driver for routine RK4
EXTERNAL DERIVS1
PARAMETER(N=4)
! USES BESSJ0,BESSJ1,BESSJ
DIMENSION Y(N),DYDX(N),YOUT(N)
X=1.0
Y(1)=BESSJ0(X)
Y(2)=BESSJ1(X)
Y(3)=BESSJ(2,X)
Y(4)=BESSJ(3,X)
DYDX(1)= -Y(2)
DYDX(2)=Y(1)-Y(2)
DYDX(3)=Y(2)-2.0*Y(3)
DYDX(4)=Y(3)-3.0*Y(4)
WRITE(*, '(1X,A,T19,A,T31,A,T43,A,T55,A)') &
    'Bessel Function:', 'J0', 'J1', 'J2', 'J3'
DO I=1,5
    H=0.2*I
    CALL RK4(Y,DYDX,N,X,H,YOUT,DERIVS1)
    WRITE(*, '(1X,A,F6.2)') 'For a step size of:', H
    WRITE(*, '(1X,A10,4F12.6)') 'RK4:', (YOUT(J), J=1,4)
    WRITE(*, '(1X,A10,4F12.6)') 'Actual:', BESSJ0(X+H)&
        , BESSJ1(X+H), BESSJ(2,X+H), BESSJ(3,X+H)
END DO
END PROGRAM
SUBROUTINE DERIVS1(X,Y,DYDX)
DIMENSION Y(1),DYDX(1)
DYDX(1)=-Y(2)
DYDX(2)=Y(1)-(1.0/X)*Y(2)

```

```

DYDX(3)=Y(2)-(2.0/X)*Y(3)
DYDX(4)=Y(3)-(3.0/X)*Y(4)
END SUBROUTINE DERIVS1

```

在验证程序 D14R1 中为了计算贝塞尔函数需调用 BESSJ, BESSJ0 和 BESSJ1(见 4.4 节), 计算结果如下:

Bessel Function:	J0	J1	J2	J3
For a step size of: 0.20				
RK4:	0.671133	0.498290	0.159351	0.032869
Actual:	0.671133	0.498289	0.159349	0.032874
For a step size of: 0.40				
RK4:	0.566879	0.541971	0.207395	0.050358
Actual:	0.566855	0.541948	0.207356	0.050498
For a step size of: 0.60				
RK4:	0.455596	0.570049	0.257139	0.071574
Actual:	0.455402	0.569896	0.256968	0.072523
For a step size of: 0.80				
RK4:	0.340843	0.582087	0.306490	0.095195
Actual:	0.339986	0.581517	0.306144	0.098802
For a step size of: 1.00				
RK4:	0.226600	0.578296	0.353070	0.118961
Actual:	0.223891	0.576725	0.352834	0.128943

(2) 调用子过程 RKDUMB 以等步长求初值问题在区间[1.0, 20.0]上的解, 将区间划分为 150 个小区间(即步数 NSTEP=150). 验证程序 D14R2 如下:

```

PROGRAM D14R2
! Driver for routine RKDUMB
PARAMETER(NVAR=4,NSTEP=150)
DIMENSION VSTART(NVAR)
! USES BESSJ0,BESSJ1,BESSJ
COMMON /PATH/X(200),Y(10,200)
EXTERNAL DERIVS1
X1=1.0
VSTART(1)=BESSJ0(X1)
VSTART(2)=BESSJ1(X1)
VSTART(3)=BESSJ(2,X1)

```

```

VSTART(4)=BESSJ(3,X1)
X2=20.0
CALL RKDUMB(VSTART,NVAR,X1,X2,NSTEP,DERIVS1)
WRITE(*, '(1X,T9,A,T17,A,T31,A/)' ) 'X','Integrated', &
      'BESSJ3'
DO I=1,(NSTEP/10)
    J=10*I
    WRITE(*, '(1X,F10.4,2X,2F12.6)' ) X(J),Y(4,J), &
      BESSJ(3,X(J))
END DO
END PROGRAM
SUBROUTINE DERIVS1(X,Y,DYDX)
DIMENSION Y(*),DYDX(*)
    DYDX(1)=-Y(2)
    DYDX(2)=Y(1)-(1.0/X)*Y(2)
    DYDX(3)=Y(2)-(2.0/X)*Y(3)
    DYDX(4)=Y(3)-(3.0/X)*Y(4)
RETURN
END SUBROUTINE DERIVS1

```

验证程序 D14R2 还需调用计算贝塞尔函数的 BESSJ, BESSJ0 和 BESSJ1 (见 4.4 节). 计算结果中我们只列出所计算出的函数 y_4 和精确解 $J_3(x)$ 在区间 $[1, 20]$ 中每隔 10 步的结果. 计算结果如下:

X	Integrated	BESSJ3
2.1400	0.152015	0.152016
3.4067	0.374321	0.374321
4.6733	0.410068	0.410067
5.9400	0.132714	0.132712
7.2067	-0.211150	-0.211150
8.4733	-0.265609	-0.265606
9.7400	-0.005253	-0.005249
11.0067	0.227857	0.227857
12.2733	0.152674	0.152668
13.5400	-0.107471	-0.107477
14.8067	-0.206084	-0.206080
16.0733	-0.029527	-0.029518
17.3400	0.171944	0.171947

18.6067	0.132634	0.132625
19.8734	0.079679	-0.079688

14.2 自适应变步长的龙格-库塔法

1. 功能

本节用自适应变步长龙格-库塔法求解一阶常微分方程组的初值问题

$$\begin{cases} \frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_N), & i = 1, 2, \dots, N \\ y_i(x_0) = y_{i0} \end{cases}$$

子过程 RKQC 采用四阶龙格-库塔法计算出变步长结果,它根据截断误差调整步长和提高精度.子过程 ODEINT 利用子过程 RKQC 计算一积分区间 $[X1, X2]$ 上的积分值.

2. 方法

计算公式同 14.1 节.

自适应变步长龙格-库塔法的计算步骤如下:

(1) 用步长 $htry$ 积分一次,结果记为 $\{y_{i,h}^{k+1}\}_{i=1}^N$,再将步长缩小一半连续计算两步得到同一点的值记为 $\{y_{i,h/2}^{k+1}\}_{i=1}^N$.

(2) 记 $errmax = \max_i \left| \frac{y_{i,h/2}^{k+1} - y_{i,h}^{k+1}}{\epsilon * yscal(i)} \right|$. 计算 $errmax$, 其中 ϵ 为给定的误差限, $yscal(i) = |y_i^k| + |hf_i(x_k, Y^k)|$ ($i = 1, 2, \dots, N$).

(3) 将 $errmax$ 与 1 进行比较. 若 $errmax > 1$, 则缩小步长 $htry \leftarrow s * htry * errmax^{-0.25}$, 然后重新计算, 这里 s 为小于 1 的安全系数. 若 $errmax \leq 1$, 则表示已满足精度要求. 此步积分成功. 考虑到四阶 R-K 法的截断误差, 利用两次计算作外插, 得到 Y_{k+1} 的新近似值:

$$y_i^{k+1} = y_{i,h/2}^{k+1} + \frac{1}{15}(y_{i,h/2}^{k+1} - y_{i,h}^{k+1})$$

它对五阶是精确的. 为了选取下一步的步长, 算法中取定了一误差界 $errcon$. 若 $errmax > errcon$, 则步长取为 $s * htry * errmax^{-0.25}$, 否则取为 $4 * htry$.

3. 使用说明

(1) RKQC (Y, DYDX, N, X, HTRY, EPS, YSCAL, IIDID, HNEXT, DERIVS)

N	整型变量,输入参数,方程个数
EPS	实型变量,输入参数,给定的误差限
HTRY	实型变量,输入参数,预定的尝试积分步长
HDID	实型变量,输出参数,成功的积分步长
DYDX	含 N 个元素的一维实型数组,输入参数,存放方程组右端函数在初始点上的值
X	实型变量,输入、输出参数,开始时存放自变量初值,调用后存放下一步积分的自变量初值
Y	含 N 个元素的一维实型数组,输入、输出参数,开始时存放积分初值,最后存放积分结果
HNEXT	实型变量,输出参数,存放供下一步积分的初始尝试步长
YSCAL	含 N 个元素的一维实型数组,输入参数,为检验误差的一个量
DERIVS	子过程 DERIVS($X, Y, DYDX$)由用户自编,其功能是计算方程组右端函数值,并将其放入 $DYDX$ 中,其中 Y 和 $DYDX$ 为含 N 个元素的一维实型数组, X 为实型变量

(2) ODEINT (YSTART,NVAR,X1,X2,EPS,H1,HMIN,NOK,NBAD,DERIVS,RKQC)

NVAR	整型变量,输入参数,方程个数
EPS	实型变量,输入参数,给定的误差限
X1	实型变量,输入参数,自变量初值
X2	实型变量,输入参数,自变量终值
YSTART	含 NVAR 个元素的一维实型数组,输入参数,开始时存放积分初值 $Y(X1)$,最后存放 $Y(X2)$
H1	实型变量,输入参数,存放初始尝试步长
HMIN	实型变量,输入参数,积分时所允许的最小步长
NOK	整型变量,输出参数,成功的积分步数
NBAD	整型变量,输出参数,失败的积分步数
DERIVS	子过程 DERIVS($X, Y, DYDX$)由用户自编,其功能是计算方程组右端函数值,并将其放入 $DYDX$ 中,其中 Y 和 $DYDX$ 为含 N 个元素的一维实型数组, X 为实型变量
RKQC	自适应变步长龙格-库塔法子过程

(3) 关于使用子过程 ODEINT 的补充说明

使用 ODEINT 除先输入初始信息 $X1, Y1, H1$, 积分终值 $X2$ 及 EPS 外,还需输入:

① DXSAV, 它表示一小步长. 在子过程中只有比 DXSAV 大的积分区间上的中间值被储存, 否则不储存. 即若设积分步长为 h_1 , 当 $|h_1| > |DXSAV|$ 时, $X1 + h_1$ 处的信息被储存, 否则不储存. 设下步积分步长为 h_2 , 当 $|h_2| > |DXSAV|$ 时, $X1 + h_1 + h_2$ 点的信息被储存, 否则不储存.

② KMAX, 它表示打算储存的积分步的最大数目, 为一整型变量. 这里, 子过程中取值为 200, 即 $KMAX = 200$, 它表示在区间 $[X1, X2]$ 中最多储存 200 步的信息, KMAX 的值最小可取零. 若 $KMAX = 0$, 则中间信息不储存. 若定义的 KMAX 比实际要储存的中间步数小, 则存到第 KMAX 步, 后面的积分信息不再储存.

输入这些值后, 调用此子过程, 若从 $X1$ 到 $X2$ 每步的积分步长 h_i 均大于 $DXSAV$ ($i = 1, 2, \dots$), 则 $X1, Y1$ 分别储存在 $XP(1), YP(I, 1)$ ($I = 1, 2, \dots, NVAR$), $X1 + h_1, Y1(X1 + h_1)$ 分别储存在 $XP(2), YP(I, 2)$ ($I = 1, 2, \dots, NVAR$) \dots , 依次类推. 同时, 成功的积分步数和失败的步数也被储存. KOUNT 为实际存储的中间结果的个数.

4. 过程

(1) 子过程 RKQC.

```
SUBROUTINE rkqc(y,dydx,n,x,htry,eps,yscal,hdid,hnext,&
    derivs)
PARAMETER(nmax=10,one=1.,safety=0.9,errcon=6.e-4,&
    fcor=6.6667e-2)
! USES rk4
EXTERNAL derivs
DIMENSION y(n),dydx(n),yscal(n),ytemp(nmax),ysav(nmax),&
    dysav(nmax)
REAL h,xsav,pgrow,pshrnk,hh,x,errmax,hnext,hdid
INTEGER i,n
pgrow=-0.2
pshrnk=-0.25
xsav=x
do i=1,n
    ysav(i)=y(i)
    dysav(i)=dydx(i)
end do
h=htry
```



```

do
  hh=0.5 * h
  call rk4(ysav,dysav,n,xsav,hh,ytemp,derivs)
  x=xsav+hh
  call derivs(x,ytemp,dydx)
  call rk4(ytemp,dydx,n,x,hh,y,derivs)
  x=xsav+h
  if(x==xsav)&.
    pause 'stepsize not significant in rkqc.'
  call rk4(ysav,dysav,n,xsav,h,ytemp,derivs)
  errmax=0.
  do i=1,n
    ytemp(i)=y(i)-ytemp(i)
    errmax=max(errmax,abs(ytemp(i)/yscal(i)))
  end do
  errmax=errmax/eps
  if(errmax>one) then
    h=safety * h * (errmax ** pshrnk)
    flag=1.
  else
    hdid=h
    if(errmax>errcon) then
      hnext=safety * h * (errmax ** pgrow)
    else
      hnext=4. * h
    endif
    flag=0.
  endif
endif
if(flag/=1.) exit
end do
do i=1,n
  y(i)=y(i)+ytemp(i) * fcor
end do
END SUBROUTINE rkqc

```

(2) 子过程 ODEINT.

```

SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,&.

```

```

                                nbad,derivs,rkqc)
PARAMETER (maxstp=10000,nmax=10,two=2.0,zero=0.0,&
            tiny=1.e-30)
COMMON /path/ kmax,kount,dxsav,xp(200),yp(10,200)
! USES rkqc
DIMENSION ystart(nvar),yscal(nmax),y(nmax),dydx(nmax)
REAL x,x1,x2,eps,hmin,hdid,h,xsav
INTEGER nok,nbad,kount,nstp,i
EXTERNAL derivs
x=x1
h=sign(h1,x2-x1)
nok=0
nbad=0
kount=0
do i=1,nvar
    y(i)=ystart(i)
end do
xsav=x-dxsav*two
do nstp=1,maxstp
    call derivs(x,y,dydx)
    do i=1,nvar
        yscal(i)=abs(y(i))+abs(h*dydx(i))+tiny
    end do
    if(kmax>0)then
        if(abs(x-xsav)>abs(dxsav)) then
            if(kount<kmax-1) then
                kount=kount+1
                xp(kount)=x
                do i=1,nvar
                    yp(i,kount)=y(i)
                end do
                xsav=x
            endif
        endif
    endif
    if((x+h-x2)*(x+h-x1)>zero) h=x2-x
    call rkqc(y,dydx,nvar,x,h,eps,yscal,hdid,hnext,&

```

```

                                derivs)
if(hdid:=h) then
    nok=nok+1
else
    nbad=nbad+1
endif
if((x-x2)*(x2-x1)>=zero) then
    do i=1,nvar
        ystart(i)=y(i)
    end do
    if(kmax/=0) then
        kount=kount+1
        xp(kount)=x
        do i=1,nvar
            yp(i,kount)=y(i)
        end do
    endif
    return
endif
if(abs(hnext)<hmin)&
    pause 'stepsize smaller than minimum.'
h=hnext
end do
PAUSE 'Too many steps.'
END SUBROUTINE odeint

```

5. 例子

采用的例子和 14.1 节中定步长四阶龙格-库塔法中的例子相同。

(1) 以 $x_1=1.0$ 为初始点调用 RKQC 向前积分一步. 本例以步长 0.1 为预定的积分步长, 当给定不同的误差 EPS 时, 验证程序 D14R3 中将给出实际成功的积分步长 HDID 和下一步积分的初始尝试步长 HNEXT. 调用 RKQC 后, 积分出的函数值存放在一维数组 Y 中. 验证程序 D14R3 如下:

```

PROGRAM D14R3
! Driver for routine RKQC
EXTERNAL DERIVS1
PARAMETER(N=4)

```

```

! USES BESSJ,BESSJ0,BESSJ1
DIMENSION Y(N),DYDX(N),YSCAL(N)
X=1.0
Y(1)=BESSJ0(X)
Y(2)=BESSJ1(X)
Y(3)=BESSJ(2,X)
Y(4)=BESSJ(3,X)
DYDX(1)=-Y(2)
DYDX(2)=Y(1)-Y(2)
DYDX(3)=Y(2)-2.0*Y(3)
DYDX(4)=Y(3)-3.0*Y(4)
DO I=1,N
    YSCAL(I)=1.0
END DO
HTRY=0.1
WRITE(*, '(1X,T8,A,T19,A,T31,A,T43,A)') &
    'cps','htry','hdid','hnext'
DO I=1,15
    EPS=EXP(-FLOAT(I))
    CALL RKQC(Y,DYDX,N,X,HTRY,EPS,YSCAL,HDID,HNEXT,&
        DERIVS1)
    WRITE(*, '(2X,E12.4,F8.2,2X,2F12.6)') EPS,HTRY,&
        HDID,HNEXT
END DO
END PROGRAM
SUBROUTINE DERIVS1(X,Y,DYDX)
DIMENSION Y(1),DYDX(1)
    DYDX(1)=-Y(2)
    DYDX(2)=Y(1)-(1.0/X)*Y(2)
    DYDX(3)=Y(2)-(2.0/X)*Y(3)
    DYDX(4)=Y(3)-(3.0/X)*Y(4)
END SUBROUTINE DERIVS1

```

验证程序 D14R3 中还需调用计算贝塞尔函数的 BESSJ, BESSJ0 和 BESSJ1 (见 4.4 节), 计算结果如下:

eps	htry	hdid	hnext
0.3679E+00	0.10	0.100000	0.400000

0.1353E+00	0.10	0.100000	0.354954
0.4979E-01	0.10	0.100000	0.287823
0.1832E-01	0.10	0.100000	0.233879
0.6738E-02	0.10	0.100000	0.190423
0.2479E-02	0.10	0.100000	0.155293
0.9119E-03	0.10	0.100000	0.126883
0.3355E-03	0.10	0.100000	0.103845
0.1234E-03	0.10	0.073460	0.066323
0.4540E-04	0.10	0.034162	0.031216
0.1670E-04	0.10	0.028686	0.025915
0.6144E-05	0.10	0.011732	0.010733
0.2260E-05	0.10	0.010758	0.009784
0.8315E-06	0.10	0.004284	0.003911
0.3059E-06	0.10	0.004460	0.004035

(2) 子过程 ODEINT 充分调用子过程 RKQC 在一大的区间内用自适应变步长龙格-库塔法求解一阶常微分方程组的初值问题. 例子中采用的积分区间为 $[1.0, 10.0]$, 给定精度要求为 $EPS=1.0E-4$. 积分出的函数值存放在二维数组 Y 中. 验证程序 D14R4 如下:

```

PROGRAM D14R4
! Driver for routine ODEINT
PARAMETER(NVAR=4)
DIMENSION VSTART(NVAR)
! USES BESSJ,BESSJ0,BESSJ1
COMMON /PATH/KMAX,KOUNT,DXSAV,X(200),Y(10,200)
EXTERNAL DERIVS,RKQC
X1=1.0
X2=10.0
VSTART(1)=BESSJ0(X1)
VSTART(2)=BESSJ1(X1)
VSTART(3)=BESSJ(2,X1)
VSTART(4)=BESSJ(3,X1)
EPS=1.0E-4
H1=0.1
HMIN=0.0
KMAX=100
DXSAV=(X2-X1)/20.0

```

```

CALL ODEINT(VSTART,NVAR,X1,X2,EPS,H1,HMIN,NOK,NBAD,&
            DERIVS,RKQC)
WRITE(*,'(/1X,A,T30,I3)') 'Successful step;',NOK
WRITE(*,'(1X,A,T30,I3)') 'Bad step;',NBAD
WRITE(*,'(1X,A,T30,I3)')&
            'Stored intermediate values;',KOUNT
WRITE(*,'(/1X,T9,A,T20,A,T33,A)') 'X','Integral',&
            'BESSJ(3,X)'
DO I=1,KOUNT
    WRITE(*,'(1X,F10.4,2X,2F14.6)') X(I),Y(4,I),&
            BESSJ(3,X(I))
END DO
END PROGRAM
SUBROUTINE DERIVS(X,Y,DYDX)
DIMENSION Y(1),DYDX(1)
    DYDX(1)=-Y(2)
    DYDX(2)=Y(1)-(1.0/X)*Y(2)
    DYDX(3)=Y(2)-(2.0/X)*Y(3)
    DYDX(4)=Y(3)-(3.0/X)*Y(4)
END SUBROUTINE DERIVS

```

在验证程序 D14R4 中,调用了 RKQC,而 RKQC 还需调用 RK4(见 14.1 节),计算贝塞尔函数调用了 BESSJ,BESSJ0 和 BESSJ1(见 4.4 节),这里只列出部分计算结果:

Successful step;	30	
Bad step;	0	
Stored intermediate values;	15	
X	Integral	BESSJ(3,X)
1.0000	0.019563	0.019563
1.6331	0.076582	0.076582
2.1038	0.145910	0.145910
2.6813	0.250547	0.250546
3.3768	0.370096	0.370096
4.1039	0.433397	0.433396
4.7908	0.396407	0.396406
5.5141	0.252492	0.252492
6.1424	0.071763	0.071764

6.8412	-0.129085	-0.129083
7.4958	-0.257531	-0.257529
8.2114	-0.286386	-0.286384
8.9176	-0.197361	-0.197360
9.5856	-0.043955	-0.043956
10.0000	0.058381	0.058379

14.3 改进的中点法

1. 功能

对一阶常微分方程组初值问题

$$\begin{cases} \frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_N), & i = 1, 2, \dots, N \\ y_i(a) = y_{i0} \end{cases}$$

或改写为

$$\begin{aligned} \frac{dY}{dx} &= F(x, Y) \\ Y(a) &= C \end{aligned}$$

其中

$$\begin{aligned} Y &= (y_1, y_2, \dots, y_N)^T, \quad F = (f_1, f_2, \dots, f_N)^T \\ C &= (y_{1,0}, y_{2,0}, \dots, y_{N,0})^T \end{aligned}$$

由初值和选定步长积分到指定点. 此算法为二阶方法, 主要用于应用广泛的外推法中的求近似解序列.

2. 方法

一般来说, 将算法建立在一个较好的数值微分公式上, 是一个可行的办法, 例如, 采用对称的公式

$$Y'(x_n) = \frac{Y(x_{n+1}) - Y(x_{n-1}))}{2h} + O(h^2)$$

其中 $h=H/n$, H 为选定步长, n 为将选定步长划分的网格点数.

由上式产生的递推公式称为显式中点公式

$$Y_{n+1} = Y_{n-1} + 2hF(x_n, Y_n)$$

它是一种二步公式. 因为在计算 Y_{n+1} 时, 需要两个已计算好的值 Y_{n-1} 和 Y_n . 因此需要一个特殊的公式来计算 Y_1 . 将该方法运用于某些问题中时, 会出现由于初

值的扰动而引起解的摆动,摆动甚至可达 10%,这种现象称为弱不稳定性.

针对这种现象,W. B. Gragg 改进了这种算法,称为改进的中点法. 令

$$Y_0 = Y(a), \quad Y_1 = Y_0 + hF(x_0, Y_0)$$

$$Y_{n+1} = Y_{n-1} + 2hF(x_n, Y_n),$$

$$x_n = x_0 + nh, \quad n = 1, 2, \dots, m-1$$

对某些值来说,借助于下面的公式来抑制摆动误差的成分,其中 m 为偶数,

$$\hat{Y}_m = \frac{1}{2}(Y_m + Y_{m-1} + hF(x_m, Y_m))$$

3. 使用说明

MMID(Y,DYDX,NVAR,XS,HTOT,NSTEP,YOUT,DERIVS)

NVAR 整型变量,输入参数,方程个数

XS 实型变量,输入参数,自变量初值

HTOT 实型变量,输入参数,要积分的区间长度

NSTEP 整型变量,输入参数,积分区间上将要划分的网格点数

Y 含 NVAR 个元素的一维实型数组,输入参数,存放积分初值

DYDX 含 NVAR 个元素的一维实型数组,输入参数,存放方程组右端函数在初始点上的值

YOUT 含 NVAR 个元素的一维实型数组,输出参数,存放积分终值 $Y(XS+HTOT)$

DERIVS 子过程 DERIVS(X,Y,DYDX),由用户自编,其功能是计算方程组右端函数值,并将其放入 DYDX 中,Y 和 DYDX 均为含 NVAR 个元素的一维实型数组,X 为实型自变量

4. 过程

子过程 MMID.

```
SUBROUTINE mmid(y,dydx,nvar,xs,htot,nstep,yout,&
               derivs)
```

```
PARAMETER (nmax=10)
```

```
DIMENSION y(nvar),dydx(nvar),yout(nvar),ym(nmax),&
               yn(nmax)
```

```
REAL xs,htot,x,swap,h
```

```
INTEGER nstep,nvar,n,i
```

```
h=htot/nstep
```



```

do i=1,nvar
  ym(i)=y(i)
  yn(i)=y(i)+h * dydx(i)
end do
x=xs-h
call derivs(x,yn,yout)
h2=2. * h
do n=2,nstep
  do i=1,nvar
    swap=ym(i)+h2 * yout(i)
    ym(i)=yn(i)
    yn(i)=swap
  end do
  x=x+h
  call derivs(x,yn,yout)
end do
do i=1,nvar
  yout(i)=0.5 * (ym(i)+yn(i)+h * yout(i))
end do
END SUBROUTINE mmid

```

5. 例子

这里采用的例子和 14.1 节中定步长四阶龙格-库塔法中的例子相同。

我们取 $HTOT=0.5$, $X1=1.0$, 积分区间 $[1, 1.5]$ 上将要划分网格的点数为 $I=5, 10, 15, \dots, 50$, 使我们能观察到精度在不断地提高。验证程序 D14R5 如下:

```

PROGRAM D14R5
! Driver for routine MMID
EXTERNAL DERIVS1
PARAMETER(NVAR=4,X1=1.0,HTOT=0.5)
! USES BESSJ,BESSJ0,BESSJ1
DIMENSION Y(NVAR),YOUT(NVAR),DYDX(NVAR)
Y(1)=BESSJ0(X1)
Y(2)=BESSJ1(X1)
Y(3)=BESSJ(2,X1)
Y(4)=BESSJ(3,X1)
DYDX(1)=-Y(2)

```

```

DYDX(2)=Y(1)--Y(2)
DYDX(3)=Y(2)-2.0*Y(3)
DYDX(4)=Y(3)-3.0*Y(4)
XF=X1+HTOT
B1=BESSJ0(XF)
B2=BESSJ1(XF)
B3=BESSJ(2,XF)
B4=BESSJ(3,XF)
WRITE(*,'(/1X,A/)' ) 'First four Bessel functions'
DO I=5,50,5
  CALL MMID(Y,DYDX,NVAR,X1,HTOT,I,YOUT,DERIVS1)
  WRITE(*,'(1X,A,F6.4,A,F6.4,A,I2,A)' ) 'X=',X1,&
    ' to ',X1+HTOT,' in ',I,' steps'
  WRITE(*,'(1X,T5,A,T20,A)' ) 'Integration','BESSJ'
  WRITE(*,'(1X,2F12.6)' ) YOUT(1),B1
  WRITE(*,'(1X,2F12.6)' ) YOUT(2),B2
  WRITE(*,'(1X,2F12.6)' ) YOUT(3),B3
  WRITE(*,'(1X,2F12.6)' ) YOUT(4),B4
  WRITE(*,'(/1X,A)' ) 'press RETURN to continue...'
  READ(*,*)
END DO
END PROGRAM
SUBROUTINE DERIVS1(X,Y,DYDX)
DIMENSION Y(1),DYDX(1)
  DYDX(1)--Y(2)
  DYDX(2)=Y(1)-(1.0/X)*Y(2)
  DYDX(3)=Y(2)-(2.0/X)*Y(3)
  DYDX(4)=Y(3)-(3.0/X)*Y(4)
END SUBROUTINE DERIVS1

```

验证程序 D14R5 中计算贝塞尔函数需调用 BESSJ, BESSJ0 和 BESSJ1 (见 4.4 节). 计算结果还与精确解进行了比较. 计算结果如下:

```

First four Bessel functions
X=1.0000 to 1.5000 in 5 steps
  Integration    BESSJ
  0.511557      0.511828
  0.557758      0.557936

```

```

0.232423    0.232088
0.061172    0.060964
press RETURN to continue...
X=1.0000 to 1.5000 in 10 steps
  Integration    BESSJ
0.511760    0.511828
0.557891    0.557936
0.232170    0.232088
0.061019    0.060964
press RETURN to continue...
X=1.0000 to 1.5000 in 15 steps
  Integration    BESSJ
0.511798    0.511828
0.557916    0.557936
0.232124    0.232088
0.060988    0.060964
press RETURN to continue...
X=1.0000 to 1.5000 in 20 steps
  Integration    BESSJ
0.511811    0.511828
0.557925    0.557936
0.232108    0.232088
0.060978    0.060964
press RETURN to continue...
X=1.0000 to 1.5000 in 25 steps
  Integration    BESSJ
0.511817    0.511828
0.557929    0.557936
0.232101    0.232088
0.060973    0.060964
press RETURN to continue...
X=1.0000 to 1.5000 in 30 steps
  Integration    BESSJ
0.511820    0.511828
0.557931    0.557936
0.232097    0.232088
0.060970    0.060964

```

press RETURN to continue...

X=1.0000 to 1.5000 in 35 steps

Integration	BESSJ
0.511822	0.511828
0.557933	0.557936
0.232094	0.232088
0.060968	0.060964

press RETURN to continue...

X=1.0000 to 1.5000 in 40 steps

Integration	BESSJ
0.511823	0.511828
0.557934	0.557936
0.232093	0.232088
0.060967	0.060964

press RETURN to continue...

X=1.0000 to 1.5000 in 45 steps

Integration	BESSJ
0.511824	0.511828
0.557934	0.557936
0.232092	0.232088
0.060967	0.060964

press RETURN to continue...

X=1.0000 to 1.5000 in 50 steps

Integration	BESSJ
0.511825	0.511828
0.557935	0.557936
0.232091	0.232088
0.060966	0.060964

press RETURN to continue...

14.4 外 推 法

1. 功能

解一阶常微分方程组的初值问题

$$\begin{cases} \frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_N), & i = 1, 2, \dots, N \\ y_i(x_0) = y_{i,0} \end{cases}$$

本节给出三个子过程,子过程 BSSTEP 调用子过程 RZEXTR 时为有理函数外推;子过程 BSSTEP 调用子过程 PZEXTR 时为多项式外推. Bulirsch 和 Stoer 给出的这种外推法,由给定初始步长积分一步,同时自动给出下一步积分的步长.

2. 方法

(1) 外推法的一般思想.

设要求的值是 $y(x)$, 对趋于 0 的步长序列 $\{h\}$, 用某种离散方法求 $y(x)$ 的逼近序列是 $\{T(h)\}$, 即 $\lim_{h \rightarrow 0} T(h, x) = y(x)$. 为了加速收敛, 用某种满足 $\hat{T}_m(h_j, x) = T(h_j, x)$, $h_j \rightarrow 0 (j=0, 1, \dots, m)$ 的外推值序列 $\{\hat{T}_m(h_j, x)\}$ 来逼近 $T(0, x)$, 并把 $\hat{T}_m(0, x)$ 作为 $T(0, x)$ 的近似.

(2) 中点法求离散值.

对 $h \neq 0$, 设 $z(x, h)$ 是用中点法求得的 $y(x)$ 的适当近似离散值, 现在用修改的中点法求离散值 $T(h, x)$:

设 $x = x_0 + mh$, m 是整数. 计算

$$x_{i+1} = x_i + h, \quad i = 0, 1, \dots, m-1$$

$$z(x_1, h) = y_0 + hf(x_0, y_0)$$

$$z(x_{i+1}, h) = z(x_i, h) + 2hf(x_i, z(x_i, h)), \quad i = 0, 1, \dots, m-1$$

$$s(h, x) = \frac{1}{2}[z(x_m, h) + z(x_{m-1}, h) + hf(x_m, z(x_m, h))]$$

$$T(h, x) = s\left(\frac{h}{2}, x\right) = T(h)$$

在适当可微性假定下, $T(h, x)$ 的展开式具有形式:

$$T(h, x) = y(x) + \tau_1(x)h^2 + \tau_2(x)h^4 + \dots$$

所以可望 $T(h, x)$ 在 $h \rightarrow 0$ 时能较好地近似 $y(x)$.

(3) 有理外推法逼近 $T(0, x)$.

设 $\hat{T}_m^{(i)}(h)$ 表示有理函数

$$\hat{T}_m^{(i)}(h) = \frac{p_0^{(i)} + p_1^{(i)}h^2 + \dots + p_\mu^{(i)}h^{2\mu}}{q_0^{(i)} + q_1^{(i)}h^2 + \dots + q_\nu^{(i)}h^{2\nu}}$$

$$\mu = \left[\frac{m}{2} \right], \quad \nu = m - \mu$$

系数 $p_0^{(i)}, \dots, p_\mu^{(i)}$ 和 $q_0^{(i)}, \dots, q_\nu^{(i)}$ 要求满足的条件方程

$$\hat{T}_m^{(i)}(h_k) = \hat{T}(h_k), \quad k = i, i+1, \dots, i+m$$

决定. 这里 $\{h_k\}$ 是严格下降趋于零的, 外推值

$$T_m^{(i)} = \hat{T}_m^{(i)}(0, x) \approx T(0, x) = y(x)$$

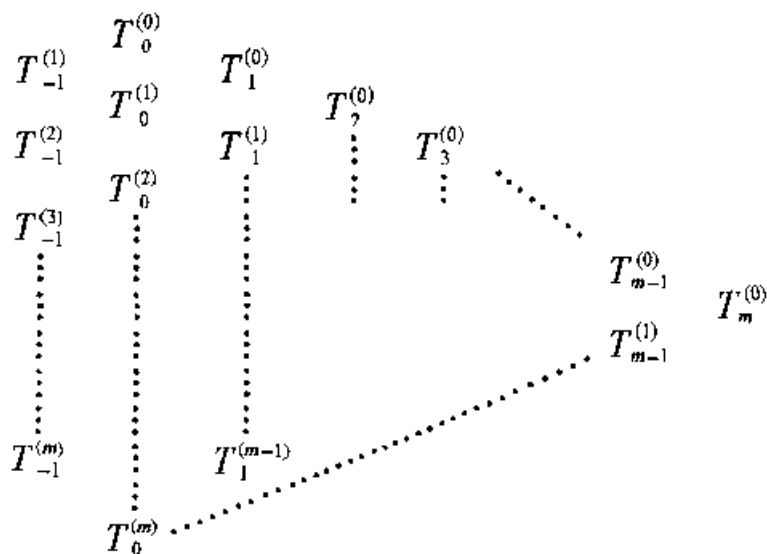
$T(h_i, x)$ 由下式计算:

$$T_{-1}^{(i)} = 0$$

$$T_0^{(i)} = T(h_i, x)$$

$$T_k^{(i)} = T_{k-1}^{(i+1)} + \frac{T_{k-1}^{(i+1)} - T_{k-1}^{(i)}}{\left(\frac{h_i}{h_{i+k}}\right)^2 \left[1 - \frac{T_{k-1}^{(i+1)} - T_{k-1}^{(i)}}{T_{k-1}^{(i+1)} - T_{k-2}^{(i)}}\right] - 1}, \quad k \geq 1$$

$T_{k-2}^{(i+1)}, T_{k-1}^{(i)}, T_{k-1}^{(i+1)}, T_k^{(i)}$ 的关系可用下面菱形表示:



为计算方便, 设

$$\Delta T_k^{(i)} = T_k^{(i)} - T_{k-1}^{(i+1)}, \quad C_k^{(i)} = T_k^{(i)} - T_{k-1}^{(i)}$$

则从上面 $T(h_i, x)$ 的计算公式可得到其等价公式:

对 $m=0, 1, 2, \dots$, 计算

$$\Delta T_0^{(m)} = T(h_m, x), \quad C_0^{(m)} = T(h_m, x), \quad W_0^{(m)} = T_0^{(m)} - T_0^{(m-1)}$$

$$\Delta T_k^{(m-k)} = \frac{C_{k-1}^{(m-k+1)} W_{k-1}^{(m-k+1)}}{\left[\frac{h_{m-k}}{h_m}\right]^2 \Delta T_{k-1}^{(m-k)} - C_{k-1}^{(m-k+1)}}, \quad k = 1, 2, \dots, m$$

$$C_k^{(m-k)} = \frac{\left[\frac{h_{m-k}}{h_m}\right]^2 \Delta T_{k-1}^{(m-k)} W_{k-1}^{(m-k+1)}}{\left[\frac{h_{m-k}}{h_m}\right]^2 \Delta T_{k-1}^{(m-k)} - C_{k-1}^{(m-k+1)}} = W_{k-1}^{(m-k+1)} + \Delta T_k^{(m-k)},$$

$$k = 1, 2, \dots, m$$

$$W_k^{(m-k)} = C_k^{(m-k)} - \Delta T_k^{(m-k-1)} \equiv T_k^{(m-k)} - T_k^{(m-k-1)}$$

$$T_m^{(0)} = \sum_{k=0}^m \Delta T_k^{(m-k)} \approx T(0, x) = y(x)$$

外推值和离散值之间的误差界是:

$$|\hat{T}_m(0) - T(0)| \leq h_0^2 \cdots h_m^2$$

(4) 多项式外推法逼近 $T(0, x)$.

当外推函数在所求值处不光滑时, 有理函数外推可能失败, 此时可用多项式外推法积分一步或两步. 多项式外推法在效率上不如有理函数外推法, 但其思想同有理外推法. 下面给出逼近 $T(0, x)$ 的计算公式(用多项式外推).

设 $R_m(x, h)$ 是关于 h 的 m 次多项式, 系数由下列方程确定:

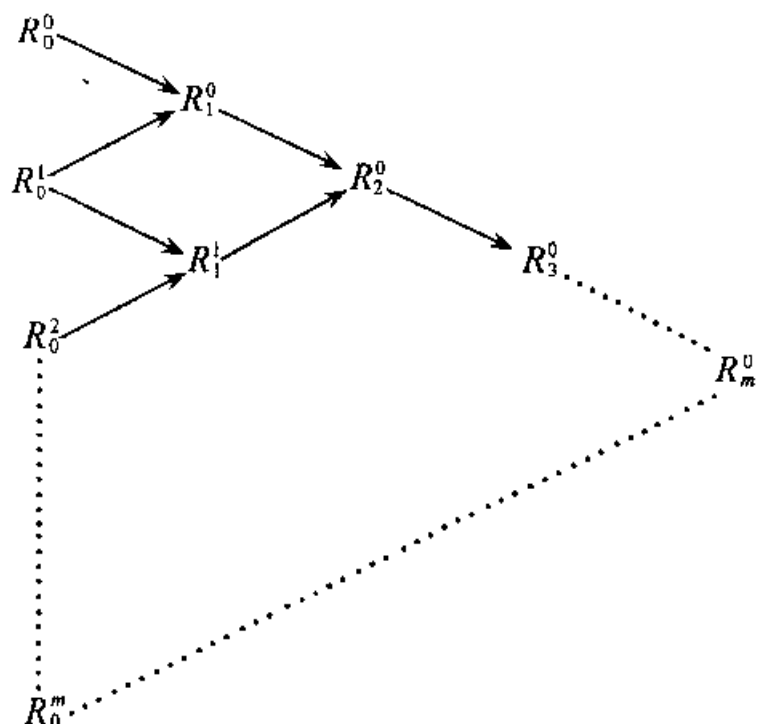
$$R_m(h_j, x) = T(h_j), \quad j = 0, 1, 2, \dots, m$$

则 $R_m(h_j, x)$ 满足:

$$R'_0(h) = T(h, x)$$

$$R'_k = R_{k-1}^{i+1} + \frac{R_{k-1}^{i+1} - R_{k-1}^i}{\left[\frac{h_i}{h_{i+k}}\right]^2 - 1}$$

它们的关系与有理函数外推类似, 也有菱形表:



(5) 步长的自动修改.

过程中用于外推(包括有理外推和多项式外推)的步长序列是

$$H_{RF} = \left\{ h_0', \frac{h_0'}{2}, \frac{h_0'}{4}, \frac{h_0'}{6}, \frac{h_0'}{8}, \dots, \frac{h_0'}{96} \right\}$$

用于修改的中点法的步长序列是

$$H_M = \left\{ \frac{h_0}{2}, \frac{h_0}{4}, \frac{h_0}{6}, \dots, \frac{h_0}{96} \right\}$$

由于有理外推法对舍入误差的敏感性,需要适当限制菱形表中求值的列数.过程中只计算 $k \leq 7$ 的 $T_k^{(i)}$,且取最大的 m 为 11,并把菱形表中 $m > 7$ 的 $T_i^{(m-7)}$ 作为 $T_m^{(0)}$ 的近似值.

为了自动调节步长,在外推 7 次才满足精度时,把步长因子取为 0.95,即取 $h_{next} = 0.95h_0$,在外推 6 次满足精度时,取 $h_{next} = 1.2h_0$,在外推第 i 次满足精度时 ($i \neq 6, 7$),取 $h_{next} = h_0 \frac{16}{\text{NSEQ}(i)}$,其中 $\text{NSEQ}(i)$ 为

$$i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$$

$$\text{NSEQ}(i) = 2, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96$$

在外推 11 次仍不能满足所需精度时,将步长取为 $\frac{h_0}{16}$ 后重新开始外推.

(6) 判断近似值是否满足要求的准则.

设 ε 为给定精度,如果对所有 i , $y(x_0+h)$ 前后相邻两个近似值之差的绝对值都小于 $\varepsilon \cdot yscal(i)$ 时,计算结束.这里

$$yscal(i) = \max_{\xi \in [x_0, x_0+h_0]} \{ |y_i(\xi)|, |y_i(x_0)| + h |f_i(x_0, y(x_0))| \}$$

3. 使用说明

(1) BSSTEP (Y, DYDX, NV, X, HTRY, EPS, YSCAL, HDID, HNEXT, DERIVS)

NV	整型变量,输入参数,方程个数
X	实型变量,输入输出参数,自变量初值,调用后存放下一步积分的自变量初值
Y	NV 个元素的一维实型数组,输入、输出参数,开始时存放函数 y 的初值,最后存放积分的结果
DYDX	NV 个元素的一维实型数组,输入参数,存放方程组右端函数在初始点上的值
HTRY	实型变量,输入参数,初始步长
EPS	实型变量,输入参数,给定的误差限
YSCAL	含 NV 个元素的一维实型数组,输入参数,为精度控制参数,习惯上取: $YSCAL(i) = y_i(x) + h_{try} \cdot f_i(x, y) , i = 1, \dots, NV$
HDID	实型变量,输出参数,成功的积分步长
HNEXT	实型变量,输出参数,下步积分的初始尝试步长

DERIVS 子过程 DERIVS(X,Y,DYDX),由用户自编,其功能为计算方程组右端函数值并将其存放在 DYDX 中,Y 和 DYDX 均为含 NV 个元素的一维实型数组,X 为实自变量

(2) RZEXTR(IEST,XEST,YEST,YZ,DY,NV,NUSE)

PZEXTR(IEST,XEST,YEST,YZ,DY,NV,NUSE)

NV 整型变量,输入参数,方程个数

NUSE 整型变量,输入参数,外推的期望最大次数,两子过程均取值为 7

IEST 整型变量,输入参数,外推调用次数

XEST 实型变量,输入参数,存放为外推计算服务的量 $\left[\frac{h_{xy}}{NSEQ(I)}\right]^2$

YEST 含 NV 个元素的一维实型数组,存放 $y(x)$ 的近似离散值 $T(h_i, x)$,工作单元

YZ 含 NV 个元素的一维实型数组,输出参数,存放外推函数值

DY 含 NV 个元素的一维实型数组,输出参数,存放外推估计误差

4. 过程

(1) 子过程 BSSTEP.

```
SUBROUTINE bsstep(y,dydx,nv,x,htry,eps,yscal,hdid,&
    hnext,derivs)
```

```
PARAMETER(nmax=10,imax=11,nuse=7,one=1.e0,&
    shrink=0.95e0,grow=1.2e0)
```

```
DIMENSION y(nv),dydx(nv),yscal(nv),yerr(nmax),&
    ysav(nmax),dysav(nmax),yseq(nmax),nseq(imax)
```

```
REAL h,htry,x,xsav,hnext
```

```
INTEGER i,j,k,m1
```

```
! USES mmid,rzextr
```

```
DATA nseq /2,4,6,8,12,16,24,32,48,64,96/
```

```
h=htry
```

```
xsav=x
```

```
do i=1,nv
```

```
    ysav(i)=y(i)
```

```
    dysav(i)=dydx(i)
```

```
end do
```

```
do
```

```
    do i=1,imax
```

```

call mmid(ysav,dysav,nv,xsav,h,nseq(i),yseq,&
          derivs)
xest=(h/nseq(i))*2
call rzextr(i,xest,yseq,y,yerr,nv,nuse)
errmax=0.
do j=1,nv
    errmax=max(errmax,abs(yerr(j)/yscal(j)))
end do
errmax=errmax/cps
if(errmax<one) then
    x=x+h
    hdid=h
    if(i==nuse) then
        hnext=h*shrink
    else if(i==nuse-1) then
        hnext=h*grow
    else
        hnext=(h*nseq(nuse-1))/nseq(i)
    endif
    return
endif
end do
h=0.25*h/2*((imax-nuse)/2)
if(x+h==x) pause 'step size underflow'
end do
END SUBROUTINE bsstep

```

(2) 子过程 RZEXTR.

```

SUBROUTINE rzextr(iest,xest,yest,yz,dy,nv,nuse)
PARAMETER (imax=11,nmax=10,ncol=7)
DIMENSION x(imax),yest(nv),yz(nv),dy(nv),&
          d(nmax,ncol),fx(ncol)
REAL yy,v,c,b,b1,ddy
INTEGER j,nv,k,m1
x(iest)=xest
if(iest==1) then
    do j=1,nv

```

```

        yz(j)=yest(j)
        d(j,1)=yest(j)
        dy(j)=yest(j)
    end do
else
    m1=min(iest,nuse)
    do k=1,m1-1
        fx(k+1)=x(iest-k)/xest
    end do
    do j=1,nv
        yy=yest(j)
        v=d(j,1)
        c=yy
        d(j,1)=yy
        do k=2,m1
            b1=fx(k)*v
            b=b1+c
            if(b/=0.) then
                b=(c-v)/b
                ddy=c*b
                c=b1*b
            else
                ddy=v
            endif
            v=d(j,k)
            d(j,k)=ddy
            yy=yy+ddy
        end do
        dy(j)=ddy
        yz(j)=yy
    end do
endif
END SUBROUTINE rzextr

```

(3) 子过程 PZEXTR.

```

SUBROUTINE pzextr(iest,xest,yest,yz,dy,nv,nuse)
PARAMETER (imax=11,ncol=7,nmax=10)

```

```

DIMENSION x(imax),yest(nv),yz(nv),dy(nv),&
           qcol(nmax,ncol),d(nmax)
REAL xest,f1,delta,f2,q
INTEGER nv,nuse,j,k1,m1,itest
x(iest)=xest
do j=1,nv
    dy(j)=yest(j)
    yz(j)=yest(j)
end do
if(iest==1)then
    do j=1,nv
        qcol(j,1)=yest(j)
    end do
else
    m1=min(iest,nuse)
    do j=1,nv
        d(j)=yest(j)
    end do
    do k1=1,m1-1
        delta=1.0/(x(iest-k1)-xest)
        f1=xest*delta
        f2=x(iest-k1)*delta
        do j=1,nv
            q=qcol(j,k1)
            qcol(j,k1)=dy(j)
            delta=d(j)-q
            dy(j)=f1*delta
            d(j)=f2*delta
            yz(j)=yz(j)+dy(j)
        end do
    end do
    do j=1,nv
        qcol(j,m1)=dy(j)
    end do
endif
END SUBROUTINE pzextr

```

5. 例子

求解一阶常微分方程组初值问题

$$\begin{cases} \frac{dy_1}{dx} = -y_2 \\ \frac{dy_2}{dx} = y_1 - \frac{1}{x}y_2 \\ \frac{dy_3}{dx} = y_2 - \frac{2}{x}y_3 \\ \frac{dy_4}{dx} = y_3 - \frac{3}{x}y_4 \\ y_1(x_1) = J_0(x_1), \quad y_2(x_1) = J_1(x_1) \\ y_3(x_1) = J_2(x_1), \quad y_4(x_1) = J_3(x_1) \end{cases}$$

积分区间为 $[x_1, x_2]$, 精确解为

$$y_1(x) = J_0(x), \quad y_2(x) = J_1(x)$$

$$y_3(x) = J_2(x), \quad y_4(x) = J_3(x)$$

其中 J_0, J_1, J_2, J_3 为贝塞尔函数.

(1) 子过程 BSSTEP 适用于精度要求高而对方法稳定性要求不高的问题. 它在运行中控制局部截断误差和调整积分步长, 以保证所得结果的误差低于预先给定的误差限 EPS. 在验证程序 D14R6 中, 在给定不同的误差 EPS 时, 将给出实际成功的积分步长 HDID 和下一步积分的初始尝试步长 HNEXT, 积分出的函数值存放在一维数组 Y 中. 本例从给定的初始点 $x_1 = 1.0$ 开始以步长 HTRY = 0.1 为尝试步长, 向前积分一步. 验证程序 D14R6 如下:

```
PROGRAM D14R6
  ! Driver for routine BSSTEP
  EXTERNAL DERIVS1
  PARAMETER(N=4)
  ! USES BESSJ, BESSJ0, BESSJ1
  DIMENSION Y(N), DYDX(N), YSCAL(N)
  X=1.0
  Y(1)=BESSJ0(X)
  Y(2)=BESSJ1(X)
  Y(3)=BESSJ(2,X)
  Y(4)=BESSJ(3,X)
  DYDX(1)=-Y(2)
```

```

DYDX(2)=Y(1)-Y(2)
DYDX(3)=Y(2)-2.0*Y(3)
DYDX(4)=Y(3)-3.0*Y(4)
DO I=1,N
    YSCAL(I)=1.0
END DO
HTRY=0.1
WRITE(*, '(/1X,T8,A,T19,A,T31,A,T43,A)') &
    'eps','htry','hdid','hnext'
DO I=1,15
    EPS=EXP(-FLOAT(I))
    CALL BSSSTEP(Y,DYDX,N,X,HTRY,EPS,YSCAL,HDID,&
        HNEXT,DERIVS1)
    WRITE(*, '(2X,E12.4,F8.2,2X,2F12.6)') EPS,HTRY,&
        HDID,HNEXT
END DO
END PROGRAM
SUBROUTINE DERIVS1(X,Y,DYDX)
DIMENSION Y(1),DYDX(1)
    DYDX(1)=-Y(2)
    DYDX(2)=Y(1)-(1.0/X)*Y(2)
    DYDX(3)=Y(2)-(2.0/X)*Y(3)
    DYDX(4)=Y(3)-(3.0/X)*Y(4)
END SUBROUTINE DERIVS1

```

在验证程序 D14R6 中, 计算贝塞尔函数可调用 BESSJ, BESSJ0, BESSJ1 (见 4.4 节). 计算结果如下:

eps	htry	hdid	hnext
0.3679E+00	0.10	0.100000	0.400000
0.1353E+00	0.10	0.100000	0.400000
0.4979E-01	0.10	0.100000	0.400000
0.1832E-01	0.10	0.100000	0.400000
0.6738E-02	0.10	0.100000	0.400000
0.2479E-02	0.10	0.100000	0.400000
0.9119E-03	0.10	0.100000	0.266667
0.3355E-03	0.10	0.100000	0.266667
0.1234E-03	0.10	0.100000	0.200000

0.4540E-04	0.10	0.100000	0.133333
0.1670E-04	0.10	0.100000	0.095000
0.6144E-05	0.10	0.100000	0.033333
0.2260E-05	0.10	0.006250	0.007500
0.8315E-06	0.10	0.006250	0.003125
0.3059E-06	0.10	0.006250	0.002083

(2) 为了验证子过程 RZEXTR, 采用以下例子:

$$F_n = \frac{1-x+x^3}{(x+1)^n}, \quad n=1, \dots, 4$$

调用子过程 RZEXTR 后, 将产生外推值 YZ 和估计误差, 最后可将计算结果和精确解进行比较. 验证程序 D14R7 如下:

```

PROGRAM D14R7
! Driver for routine RZEXTR
! Feed values from a rational function
! Fn(x)=(1-x+x**3)/(x+1)**n
PARAMETER(NV=4,NUSE=5)
DIMENSION YEST(NV),YZ(NV),DY(NV)
DO I=1,10
  IEST=I
  XEST=1.0/FLOAT(I)
  DUM=1.0-XEST+XEST**3
  DO J=1,NV
    DUM=DUM/(XEST+1.0)
    YEST(J)=DUM
  END DO
  CALL RZEXTR(IEST,XEST,YEST,YZ,DY,NV,NUSE)
  WRITE(*, '(1X,A,I2,A,F8.4)') 'IEST= ',I,&
    ' XEST=',XEST
  WRITE(*, '(1X,A,4F12.6)') 'Extrap. Function: ',&
    (YZ(J), J=1,NV)
  WRITE(*, '(1X,A,4F12.6)') 'Estimated Error: ',&
    (DY(J), J=1,NV)
END DO
WRITE(*, '(1X,A,4F12.6)') 'Actual Values: ',&
  1.0,1.0,1.0,1.0
END

```

计算结果如下:

IEST=1 XEST=1.0000				
Extrap. Function:	0.500000	0.250000	0.125000	0.062500
Estimated Error:	0.500000	0.250000	0.125000	0.062500
IEST=2 XEST=0.5000				
Extrap. Function:	0.357143	0.312500	0.357143	4.999979
Estimated Error:	-0.059524	0.034722	0.171958	4.876522
	⋮	⋮	⋮	
IEST=9 XEST=0.1111				
Extrap. Function:	0.999977	0.999974	0.999991	0.999886
Estimated Error:	0.001162	0.001472	0.002203	0.003574
IEST=10 XEST=0.1000				
Extrap. Function:	1.000018	1.000087	1.000088	0.999953
Estimated Error:	0.000620	0.000854	0.001275	0.002041
Actual Values:	1.000000	1.000000	1.000000	1.000000

(3) 子过程 PZEXTR 被子过程 BSSTEP 所调用, 执行多项式外推. 它的精度要比有理函数外推差一些, 主要是作为 RZEXTR 的备用品, 当一些问题中采用 RZEXTR 出现问题时可采用 PZEXTR. 这里用于验证 PZEXTR 的例子与验证 RZEXTR 的例子相同. 验证程序 D14R8 如下:

```

PROGRAM D14R8
! Driver for routine PZEXTR
! Feed values from a rational function
!  $F_n(x) = (1 - x + x * * 3) / (x + 1) * * n$ 
PARAMETER(NV=4,NUSE=5)
DIMENSION YEST(NV),YZ(NV),DY(NV)
DO I=1,10
  IEST=I
  XEST=1.0/FLOAT(I)
  DUM=1.0-XEST+XEST * * 3
  DO J=1,NV
    DUM=DUM/(XEST+1.0)
    YEST(J)=DUM
  END DO
  CALL PZEXTR(IEST,XEST,YEST,YZ,DY,NV,NUSE)

```



```

WRITE(*, '(1X,A,I2)') 'I=', I
WRITE(*, '(1X,A,4F12.6)') 'Extrap. Function: ', &
    (YZ(J), J=1,NV)
WRITE(*, '(1X,A,4F12.6)') 'Estimated Error: ', &
    (DY(J), J=1,NV)
END DO
WRITE(*, '(1X,A,4F12.6)') 'Actual Values: ', &
    1.0, 1.0, 1.0, 1.0
END

```

计算结果如下:

I= 1

Extrap. Function:	0.500000	0.250000	0.125000	0.062500
Estimated Error:	0.500000	0.250000	0.125000	0.062500

I= 2

Extrap. Function:	0.416667	0.277778	0.185185	0.123457
Estimated Error:	0.000000	0.000000	0.000000	0.000000

⋮

⋮

⋮

I= 8

Extrap. Function:	0.999939	0.999521	0.998433	0.996346
Estimated Error:	0.000269	0.001655	0.004329	0.008341

I= 9

Extrap. Function:	0.999958	0.999734	0.999187	0.998088
Estimated Error:	0.000160	0.001038	0.002824	0.005602

I=10

Extrap. Function:	1.000010	0.999928	0.999594	0.998898
Estimated Error:	0.000127	0.000739	0.001939	0.003847

Actual Values:	1.000000	1.000000	1.000000	1.000000
----------------	----------	----------	----------	----------

第 15 章 两点边值问题的解法

本章主要介绍两点边值问题的迭代解. 所谓两点边值问题是 N 个常微分方程耦合的方程组, 求有限区间 $[a, b]$ 上不恒等于零的解, 且该解在一个边界点 a 上要满足 n_1 个边值条件, 在另一个边界点 b 上还要满足 $n_2 = N - n_1$ 个边值条件. 本章介绍两种一般常见的数值解法.

第一种方法是打靶法(SHOOT 和 SHOOTF). 这种方法是把边值问题化为初值问题来解, 强制在一个边界点 a 上满足 n_1 个边值条件, 假设另一个边界点 b 上 n_2 个边值条件自由, 然后在区间试求解一次, 得到该解在另一边界点 b 上与 n_2 个边值条件的差异. 通过应用牛顿-拉斐森法调整参数变量, 最终使这个差异减少到零.

第二种方法是松弛法. 在松弛法中, 常微分方程将被积分区间内网格点上的有限差分方程所代替. 子过程 SOLVDE 实现了这个方法. 本章 15.3 节验证程序 SFROID 调用了子过程 SOLVDE 计算了球调和函数方程的本征值.

15.1 打靶法 (一)

1. 功能

用打靶法求解常微分方程组的两点边值问题

$$\frac{dy_i}{dx} = g_i(x, y_1(x), \dots, y_n(x)), \quad i = 1, \dots, N, \quad x \in (x_1, x_2) \quad (15-1)$$

$$B_{1j}(x_1, y_1(x_1), \dots, y_n(x_1)) = 0, \quad j = 1, \dots, n_1 \quad (15-2)$$

$$B_{2k}(x_2, y_1(x_2), \dots, y_n(x_2)) = 0, \quad k = 1, \dots, n_2 \quad (15-3)$$

其中 $n_1 + n_2 = N$.

打靶法又名试射法. 子过程 SHOOT 是打靶法的一步迭代, 即调用它一次仅试射一次.

2. 方法

(1) 打靶法的基本思想. 记

$$y(x) = (y_1(x), \dots, y_n(x))^T$$

$$\mathbf{g}(x, y) = (g_1(x, y), \dots, g_n(x, y))^T$$

$$\mathbf{B}_1(x, y) = (B_{11}(x, y), \dots, B_{1n_1}(x, y))^T$$

$$\mathbf{B}_2(x, y) = (B_{21}(x, y), \dots, B_{2n_2}(x, y))^T$$

则式(15-1)~(15-3)即为

$$\frac{dy}{dx} = \mathbf{g}(x, y), \quad x \in (x_1, x_2) \quad (15-4)$$

$$\mathbf{B}_1(x_1, y(x_1)) = 0 \quad (15-5)$$

$$\mathbf{B}_2(x_2, y(x_2)) = 0 \quad (15-6)$$

考虑带参数 $\mathbf{V} \in R_{n_2}$ 的初值问题

$$\frac{dy}{dx} = \mathbf{g}(x, y), \quad x \in (x_1, x_2) \quad (15-7)$$

$$\mathbf{B}_1(x_1, y(x_1, \mathbf{V})) = 0 \quad (15-8)$$

$$\tilde{\mathbf{B}}_1(x_1, y(x_1, \mathbf{V})\mathbf{V}) = 0, \quad \tilde{\mathbf{B}}_1(x, y, \mathbf{V}) \in R_{n_2} \quad (15-9)$$

记其解为

$$\mathbf{y} = \mathbf{y}(x, \mathbf{V}) \quad (15-10)$$

而记

$$\mathbf{F}(x_2, \mathbf{V}) = \mathbf{B}_2(x_2, \mathbf{y}(x_2, \mathbf{V})) \quad (15-11)$$

\mathbf{F} 表示差异向量.

于是求解

$$\mathbf{F}(x_2, \mathbf{V}) = 0 \quad (15-12)$$

得 \mathbf{V} 代入式(15-10)即得边值问题式(15-4)~(15-6)的解.

(2) 算法:

1) 给定一初始参数 $\mathbf{V}^{(0)}$ 及其一个修正增量 $\delta\mathbf{V}^{(0)}$, 在 $\mathbf{V} = \mathbf{V}^{(0)}$ 处解初值问题式(15-7)~(15-8), 得 $\mathbf{y}^{(0)} = \mathbf{y}(x, \mathbf{V}^{(0)})$, 并计算差异向量 $\mathbf{F}^{(0)} = \mathbf{F}(x_2, \mathbf{V}^{(0)}) = \mathbf{B}_2(x_2, \mathbf{y}^{(0)})$. 置 $k := 0$.

2) 计算 $\mathbf{V}^{(k+1)} = \mathbf{V}^{(k)} + \delta\mathbf{V}^{(k)}$.

3) 在 $\mathbf{V} = \mathbf{V}^{(k+1)}$ 处求解初值问题式(15-7)~(15-9), 得 $\mathbf{y}^{(k+1)} = \mathbf{y}(x, \mathbf{V}^{(k+1)})$.

4) 计算差异向量 $\mathbf{F}^{(k+1)} = \mathbf{F}(x_2, \mathbf{V}^{(k+1)}) = \mathbf{B}_2(x_2, \mathbf{y}^{(k+1)})$.

5) 若 $\|\mathbf{F}^{(k+1)}\| < \epsilon$, 则停止计算, 这时, $\mathbf{y}^{(k+1)}$ 即是要要求的近似解, 否则进行下一步.

6) 在 $\mathbf{V} = \mathbf{V}^{(k+1)}$ 处对方程式(15-12)作一步牛顿-拉斐森迭代, 即求解线性方程组

$$[\alpha] \delta\mathbf{V}^{(k+1)} = -\mathbf{F}^{(k+1)}$$

得 $\delta\mathbf{V}^{(k+1)}$, 其中

$$[\alpha]_{ij} = \frac{\partial F_i(x_2, V^{(k+1)})}{\partial V_j} \approx \frac{F_i^{(k+1)} - F_i^{(k)}}{\delta V_j^{(k+1)}}$$

置 $k := k + 1$, 转 2).

3. 使用说明

(1) SHOOT (NVAR, V, DELV, N2, X1, X2, EPS, H1, HMIN, F, DV)

- NVAR 整型变量, 输入参数, 方程个数
- V 含 $N2$ 个元素的一维实型数组, 输入、输出参数, 开始时存放试射参数值, 调用后存放修改了的参数值
- DELV 含 $N2$ 个元素的一维实型数组, 输入参数, 存放猜想的数组 V 的一个微小增量
- N2 整型变量, 输入参数, 右边界条件个数
- X1 实型变量, 输入参数, 积分区间左端点
- X2 实型变量, 输入参数, 积分区间右端点
- EPS 实型变量, 输入参数, 取定的误差限
- H1 实型变量, 输入参数, 初始尝试积分步长
- HMIN 实型变量, 输入参数, 最小积分步长
- F 含 $N2$ 个元素的一维实型数组, 输出参数, 存放右端点上的差异向量
- DV 含 $N2$ 个元素的一维实型数组, 输出参数, 存放对试射参数值 V 修改的增向量

(2) 调用的子过程说明.

- LOAD 子过程 LOAD(X1, V, Y), 由使用者自编, 其功能是对给定的 x_1 与 $V \in R^{n_2}$ 由式(15-8)和(15-9)解出

$$y_i(x_1) = y_i(x_1; v_1, v_2, \dots, v_{n_2}) \quad (i = 1, 2, \dots, N)$$

- SCORE 子过程 SCORE(X2, Y, F), 由使用者自编, 其功能是若已知 $y_i(x_2) (i=1, 2, \dots, N)$ 由式(15-3)计算差异向量

$$F = B(x_2, Y(x_2, V))$$

- DERIVS 子过程 DERIVS(X, Y, DYDX), 由使用者自编, 其功能是计算右函数值 $g_i(x, y)$ 并将其放入 DYDX 中, 其中 Y 和 DYDX 均为含 NVAR 个元素的一维实型数组, X 为实自变量
- ODEINT 为在区间 $[x_1, x_2]$ 上用变步长 R-K 法解初值问题的子过程, 见第 14 章
- RKQC 为用 R-K 法积分一步初值问题的子过程, 见第 14 章

LUDCMP 为对线性方程组 $AX=b$ 的系数矩阵作 LU 分解的子过程, 见第 1 章

LUBKSB 为对线性方程组用 LU 分解作回代求解的子过程, 见第 1 章

4. 过程

子过程 SHOOT.

```

SUBROUTINE shoot(nvar,v,delv,n2,x1,x2,eps,h1,hmin,&
                f,dv)
EXTERNAL derivs,rkqc
PARAMETER (np=20)
! USES ideint,score,load,ludcmp,lubksb
DIMENSION v(n2),delv(n2),f(n2),dv(n2),y(np),&
          dfdv(np,np),indx(np)
REAL sav,x1,x2,cps,h1,hmin
INTEGER n2,iv,i
call load(x1,v,y)
call odeint(y,nvar,x1,x2,eps,h1,hmin,nok,nbad,&
          derivs,rkqc)
call score(x2,y,f)
do iv=1,n2
  sav=v(iv)
  v(iv)=v(iv)+delv(iv)
  call load(x1,v,y)
  call odeint(y,nvar,x1,x2,eps,h1,hmin,nok,nbad,&
          derivs,rkqc)
  call score(x2,y,dv)
  do i=1,n2
    dfdv(i,iv)=(dv(i)-f(i))/delv(iv)
  end do
  v(iv)=sav
end do
do iv=1,n2
  dv(iv)=-f(iv)
end do
call ludcmp(dfdv,n2,np,indx,det)
call lubksb(dfdv,n2,np,indx,dv)

```

```

do iv=1,n2
  v(iv)=v(iv)+dv(iv)
end do
END SUBROUTINE shoot

```

5. 例子

考虑球调和函数 s , 它满足微分方程

$$\frac{d}{dx} \left[(1-x^2) \frac{ds}{dx} \right] + \left(\lambda - c^2 x^2 - \frac{m^2}{1-x^2} \right) s = 0$$

$$x \in [-1, 1] \quad (15-13)$$

其中 m 是一整数, c 是扁率参数, λ 是特征值. 不考虑符号, c^2 可以是正数, 也可以是负数, 当 $c^2 > 0$ 时函数称为长球调和函数, 当 $c^2 < 0$ 时函数称为扁球调和函数. 由于 $x = \pm 1$ 是方程的奇异点, 为求式(15-13)在 $[-1, 1]$ 上的正则非零解, 需在 $x = \pm 1$ 给定正则性条件, 设

$$s(-1) = s_-, \quad s(1) = s_+ \quad (15-14)$$

于是需要求解两点边值问题式(15-13)和(15-14), 它仅对一定的参数 λ 的值, 特征值有非零解.

首先考虑球面情形, 这时 $c=0$, 对每个固定的 m 有特征值 $\lambda_{mn} = n(n+1)$, $n=m, m+1, \dots$, 每个 λ_{mn} 对应的特征函数是连带勒让德函数 $P_n^m(x)$, 当 $n=m$ 时 $P_n^m(x)$ 在 $(-1, 1)$ 上无节点, 当 $n=m+1$ 时 $P_n^m(x)$ 在 $(-1, 1)$ 上有一个节点, \dots , 依此类推.

其次, 对于 $c^2 \neq 0$ 的一般情形, 记式(15-13)的特征值为 $\lambda_{mn}(c)$, 对应的特征函数为 $s_{mn}(x, c)$. 用传统方法计算 $\lambda_{mn}(c)$ 和 $s_{mn}(x, c)$ 是相当困难的, 下面说明通过求解微分方程计算它们是可行的. 由式(15-13)关于 m 的对称性不妨设 $m \geq 0$, 令

$$s = (1-x^2)^{m/2} y \quad (15-15)$$

则 y 满足方程

$$(1-x^2) \frac{d^2 y}{dx^2} - 2(m+1)x \frac{dy}{dx} + (\mu - c^2 x^2)y = 0 \quad (15-16)$$

其中

$$\mu \equiv \lambda - m(m+1) \quad (15-17)$$

方程(15-13)与(15-16)是线性的且在变换 $x \rightarrow -x$ 下是不变的, 从而 s 与 y 除可能差一个整体比例因子外也是不变的. 因为解要求是规范化的, 因此这个比例因子只可能是 ± 1 . 若 $n-m$ 是奇数, 则函数在 $(-1, 1)$ 上有奇数个零点, 因而必须选择反对称解 $y(-x) = -y(x)$, 它在 $x=0$ 有一零点, 相反如 $n-m$ 为偶

数,则必须选择对称解,因此

$$y_{mn}(-x) = (-1)^{n-m} y_{mn}(x) \quad (15-18)$$

对于 s_{mn} 是类似的.

由于要求 y 在 $x = \pm 1$ 处正则,即 y 有形式

$$y = a_0 + a_1(1 - x^2) + a_2(1 - x^2)^2 + \dots$$

代入方程得

$$a_1 = -\frac{\mu - c^2}{4(m+1)} a_0$$

因此

$$\frac{d}{dx} y(\pm 1) = \pm \frac{\mu - c^2}{2(m+1)} y(1) \quad (15-19)$$

由对称性式(15-18),我们代替 $[-1, 1]$ 上的积分为在 $(-1, 0)$ 或 $(0, 1)$ 上积分. 对 $y(\pm 1)$, 我们规范化 $s_{mn}(x, c)$ 使

$$\lim_{x \rightarrow 1} (1 - x^2)^{-m/2} s_{mn}(x, c) = \lim_{x \rightarrow 1} (1 - x^2)^{-m/2} P_n^m(x)$$

因而

$$y(1) = \lim_{x \rightarrow 1} (1 - x^2)^{-m/2} P_n^m(x) = \frac{(-1)^m (n+m)!}{2^m m! (n-m)!} \equiv \nu \quad (15-20)$$

而

$$y(-1) = y(1) \quad (15-21)$$

由式(15-18)得

$$\begin{aligned} y(0) &= 0, \quad \text{当 } n-m \text{ 为奇数时} \\ \frac{d}{dx} y(0) &= 0, \quad \text{当 } n-m \text{ 为偶数时} \end{aligned} \quad (15-22)$$

下面通过求解两点边值问题来求长球调和函数与扁球调和函数及其对应的特征值. 对于长球调和函数, 令

$$y_1 = y, \quad y_2 = \frac{d}{dx} y, \quad y_3 = \mu$$

则由式(15-16)得

$$\frac{d}{dx} y_1 = y_2 \quad (15-23)$$

$$\frac{d}{dx} y_2 = \frac{1}{1-x^2} [2x(m+1)y_2 - (y_3 - c^2 x^2)y_1] \quad (15-24)$$

$$\frac{d}{dx} y_3 = 0 \quad (15-25)$$

同样对扁球调和函数有

$$\frac{d}{dx}y_4 = y_5 \quad (15-26)$$

$$\frac{d}{dx}y_5 = \frac{1}{1-x^2}[2x(m+1)y_5 - (y_6 + c^2x^2)y_4] \quad (15-27)$$

$$\frac{d}{dx}y_6 = 0 \quad (15-28)$$

其中 y_1 与 y_4 对应于两个球调和函数(因为 $c^2 > 0$), y_3 和 y_6 对应于要求的特征值.

由于 $\frac{d}{dx}y_2$ 与 $\frac{d}{dx}y_5$ 在 $x = \pm 1$ 处不能计算, 因此这里取 $x_1 = -1 + \Delta x$, $\Delta x = 10^{-4}$, $x_2 = 0$, 即在 $[-1 + \Delta x, 0]$ 上来求解这两个方程组. 边界条件如下给定:

在 $x_1 = -1 + \Delta x$ 处

$$y_1(x_1) = \nu + y_2(x_1)\Delta x \quad (15-29)$$

$$y_2(x_1) = \frac{-1}{2(m+1)}[y_3(x_1) - c^2]\nu \quad (15-30)$$

$$y_4(x_1) = \nu + y_5(x_1)\Delta x \quad (15-31)$$

$$y_5(x_1) = \frac{-1}{2(m+1)}[y_6(x_1) + c^2]\nu \quad (15-32)$$

在 $x_2 = 0$ 处

$$y_1(x_2) = 0, \quad y_4(x_2) = 0, \quad \text{当 } n-m \text{ 为奇数时} \quad (15-33)$$

$$y_2(x_2) = 0, \quad y_5(x_2) = 0, \quad \text{当 } n-m \text{ 为偶数时} \quad (15-34)$$

下面说明如何调用本节的子过程.

根据算法首先须给出在 x_1 处的另外两个带参数 $V = (V_1, V_2)^T$ 的条件, 以及 V 的初始值 $V^{(0)}$, 初始修正增量 $\Delta V^{(0)}$, 自然设

$$y_3(x_1) = V_1, \quad y_6(x_1) = V_2 \quad (15-35)$$

再取

$$V_i^{(0)} = n(n+1) - m(m+1) - (-1)^i \frac{c^2}{2}, \quad i = 1, 2$$

$$\Delta V_1^{(0)} = \Delta V_2^{(0)} = V_1 \times 10^{-3}$$

子过程 DERIVS(X, Y, DYDX)的作用是在 $x = X$ 处由 $Y(x) = (y_1(x), \dots, y_6(x))^T$ 计算出方程式(15-23)~(15-28)的右端后存放在 DYDX 中. 子过程 LOAD(X1, V, Y)的作用是对给定的参数值 $V = (V_1, V_2)^T$ 由式(15-29)~(15-32)与式(15-35)计算出 Y 在 $x = x_1$ 处的值存放在 Y 中. 子过程 SCORE(X2, Y, F)的作用就是由 Y 在 $x = x_2$ 处的值计算差异向量 $F = (F_1, F_2)^T$, 根据式(15-33)

与(15-34)

$$F = \begin{cases} (Y(1), Y(4))^T, & n - m \text{ 为奇数} \\ (Y(2), Y(5))^T, & n - m \text{ 为偶数} \end{cases}$$

经过上面的一些解释,现在我们若给定 M, N 和 C^2 , 验证程序 D15R1 将建立估计值 $V(1)$ 和 $V(2)$ 和子过程 SHOOT 的反复循环,一直到 V 的变化增量小于给定的 EPS 和 V 相乘的积为止. 球调和函数的一些精确本征值在 15.3 节中给出,读者可以将这里的计算结果进行比较. 注意,验证程序中给出的结果是 μ (见式(15-17)), 15.3 节中给出的是 λ 的精确解. 验证程序 D15R1 如下:

```

PROGRAM D15R1
! Driver for routine SHOOT
! Prolate and Oblate case are handled simultaneously,
! leading to six first-order equations. Unknown to
! SHOOT, these are actually two independent sets of
! three coupled equations, one set with  $c^2$  positive
! and the other with  $c^2$  negative.
PARAMETER(NVAR=6,N2=2,DELTA=1.0E-3,EPS=1.0E-6,&
           DX=1.0E-4)
DIMENSION V(2),DELV(2),F(2),DV(2)
COMMON C2,M,N,FACTR
DO
  WRITE(*,*) 'Input M,N,C-Squared (999 to end)'
  READ(*,*) M,N,C2
  IF(C2==999.) STOP
  IF((N>=M).OR.(M>=0).OR.(N>=0)) EXIT
END DO
FACTR=1.0
IF (M/=0) THEN
  Q1=N
  DO I=1,M
    FACTR=-0.5*FACTR*(N+1)*(Q1/I)
    Q1=Q1-1.0
  END DO
ENDIF
V(1)=N*(N+1)-M*(M+1)+C2/2.0
V(2)=N*(N+1)-M*(M+1)-C2/2.0

```

```

DELV(1)=DELTA * V(1)
DELV(2)=DELV(1)
H1=0.1
HMIN=0.0
X1=-1.0+DX
X2=0.0
WRITE(*, '(1X,T12,A,T36,A)') 'Prolate','Oblate'
WRITE(*, '(1X,T6,A,T17,A,T30,A,T41,A)') 'Mu(M,N)',&
      'Error Est.','Mu(M,N)','Error Est.'
DO
  CALL SHOOT(NVAR,V,DELV,N2,X1,X2,EPS,H1,HMIN,F,DV)
  WRITE(*, '(1X,4F12.6)') V(1),DV(1),V(2),DV(2)
  IF ((ABS(DV(1))<=ABS(EPS * V(1))).OR. ((DV(2))<=&
      ABS(EPS * V(2)))) EXIT
END DO
END PROGRAM
SUBROUTINE LOAD(X1,V,Y)
COMMON C2,M,N,FACTR
DIMENSION V(2),Y(6)
  Y(3)=V(1)
  Y(2)=-(Y(3)-C2) * FACTR/2.0/(M+1.0)
  Y(1)=FACTR+Y(2) * DX
  Y(6)=V(2)
  Y(5)=-(Y(6)+C2) * FACTR/2.0/(M+1.0)
  Y(4)=FACTR+Y(5) * DX
END SUBROUTINE LOAD
SUBROUTINE SCORE(X2,Y,F)
COMMON C2,M,N
DIMENSION Y(6),F(2)
  IF(MOD((N-M),2)==0) THEN
    F(1)=Y(2)
    F(2)=Y(5)
  ELSE
    F(1)=Y(1)
    F(2)=Y(4)
  ENDIF
END SUBROUTINE SCORE

```

```

SUBROUTINE DERIVS(X,Y,DYDX)
COMMON C2,M,N
DIMENSION Y(6),DYDX(6)
  DYDX(1)=Y(2)
  DYDX(3)=0.0
  DYDX(2)=(2.0*X*(M+1.0)*Y(2)-(Y(3)-C2*X*X)*Y(1))/&
    (1.0-X*X)
  DYDX(4)=Y(5)
  DYDX(6)=0.0
  DYDX(5)=(2.0*X*(M+1.0)*Y(5)-(Y(6)+C2*X*X)*Y(4))/&
    (1.0-X*X)
END SUBROUTINE DERIVS

```

计算结果如下:

Input M,N,C-Squared (999 to end)

Prolate		Oblate	
Mu(M,N)	Error Est.	Mu(M,N)	Error Est.
0.014046	-0.035954	-0.014524	0.035476
0.014266	0.000220	-0.014305	0.000219
0.014266	0.000000	-0.014305	0.000000

最后要注意,D15R1 中要调用 SHOOT,而 SHOOT 还要调用 ODEINT, RKQC(见 14.2 节),LUDCMP,LUBKSB(见 1.2 节).

15.2 打靶法 (二)

1. 功能

用打靶法求解两点边值问题式(15-1)~(15-3). 当初始条件给得很不好或终点是常微分方程组的奇异点时,用打靶法(一)(15.1 节)有时会遇到麻烦. 本节子过程 SHOOTF 是解决这些困难的一种方法,它以远离奇异点的方向积分. 调用它一次分别从区间两端点向区间中某一拟合点 x_f 试射一次,且得出修改了的参数 \tilde{V}_1 和 \tilde{V}_2 ,可连续调用使两边向拟合点的试射吻合而得到理想的参数 \tilde{V}_1 和 \tilde{V}_2 ,最后由理想的参数 \tilde{V}_1 和 \tilde{V}_2 选择一种合适的求解初值问题的方法(如第 14 章中的变步长 R-K 法),在两个区间 $[x_1, x_f]$ 和 $[x_f, x_2]$ 积分初值问题而得到两点边值问题式(15-1)~(15-3)满足精度要求的解.

2. 方法

(1) 设定 n_2 个自由参数, 由式(15-2)解出

$$y_i(x_1) = y_i(x_1, V_{(1)1}, V_{(1)2}, \dots, V_{(1)n_2}), \quad i = 1, 2, \dots, N$$

(2) 选取一组参数值 $V_{(1)} = (V_{(1)1}, V_{(1)2}, \dots, V_{(1)n_2})$, 选定区间 $[x_1, x_2]$ 中一合适点 x_f , 在 $[x_1, x_f]$ 上求解

$$\frac{dy_i(x)}{dx} = g_i(x, y_1, y_2, \dots, y_N) \quad (i = 1, 2, \dots, N)$$

$$y_i(x_1) = y_i(x_1; V_{(1)})$$

(3) 构造含 N 个分量的函数 F , 它依赖于 $Y = (y_1, y_2, \dots, y_N)^T$ (大多数情况下设 $F_i = y_i$), 计算

$$F1 = F[Y(x_f, V_{(1)})]$$

(4) 设定 n_1 个自由参数 $V_{(2)} = (V_{(2)1}, V_{(2)2}, \dots, V_{(2)n_1})$, 由式(15-3)解出 $y_i(x_2) = y_i(x_2; V_{(2)}) (i = 1, 2, \dots, N)$.

(5) 从点 x_2 向 x_f 积分下面的初值问题:

$$\frac{dy_i(x)}{dx} = g_i(x, y_1, y_2, \dots, y_N), \quad (i = 1, 2, \dots, N)$$

$$y_i(x_2) = y_i(x_2; V_{(2)})$$

得到 $y_i(x_f; V_{(2)}) (i = 1, 2, \dots, N)$.

(6) 计算 $F2 = F[Y(x_f; V_{(2)})]$.

(7) 现在问题转化为求 $\bar{V} = (\bar{V}_{(1)}, \bar{V}_{(2)})$, 使

$$W_{(0)} = F[Y(x_f; \bar{V}_{(1)})] - F[Y(x_f; \bar{V}_{(2)})] = 0$$

用 Newton-Raphson 法求上面的非线性方程组即求解线性方程组

$$[\alpha] \delta V = -W_{(0)}$$

其中

$$\delta V = (\delta V_{(1)1}, \delta V_{(1)2}, \dots, \delta V_{(1)n_2}, \delta V_{(2)1}, \dots, \delta V_{(2)n_1})^T$$

$$W_{(0)} = F1 - F2$$

$$[\alpha] = \left[\frac{\partial W}{\partial V} \right] \Big|_{x=x_f} = \left[\frac{\partial F}{\partial V_{(1)}}, \frac{\partial F}{\partial V_{(2)}} \right]$$

$$\approx \left[\frac{F(\bar{V}_{(1)}) - F(V_{(1)})}{\Delta V_1}, \frac{F(\bar{V}_{(2)}) - F(V_{(2)})}{\Delta V_2} \right]$$

为 $N \times N$ 阶方阵.

(8) 记

$$\delta V = (\delta V_1, \delta V_2)^T, \delta V_1 = (\delta V_{(1)1}, \delta V_{(1)2}, \dots, \delta V_{(1)n_2})^T$$

$$\delta V_2 = (\delta V_{(2)1}, \delta V_{(2)2}, \dots, \delta V_{(2)n_2)})^T$$

则 $V_1 = V_{(1)} + \delta V_1$ 为 $V_{(1)}$ 的一修正值, $V_2 = V_{(2)} + \delta V_{(2)}$ 的一修正值. 此时一般有 $\|F[Y(x_f; V_1)] - F[Y(x_f; V_2)]\| \leq \|F[Y(x_f; V_{(1)})] - F[Y(x_f; V_{(2)})]\|$.

使用者可连续调用子过程 SHOOTF, 使最终得到的参数 V_1 和 V_2 满足 $\|F_1(V_1) - F_2(V_2)\| < \epsilon_1$, 此时上次求得的两个初值问题的解 YP_1 和 YP_2 即为两点边值问题式(15-1)~(15-3)的解.

3. 使用说明

(1) SHOOTF (NVAR, V1, V2, DELV1, DELV2, N1, N2, X1, X2, XF, EPS, H1, HMIN, F, DV1, DV2)

NVAR	整型变量, 输入参数, 方程个数
N1	整型变量, 输入参数, 左边界条件个数
N2	整型变量, 输入参数, 右边界条件个数
V1	含 $N2$ 个元素的一维实型数组, 输入、输出参数, 开始时存放从 $X1$ 点试射的初始猜测参数值, 调用后存放修改了的参数值
V2	含 $N1$ 个元素的一维实型数组, 输入、输出参数, 开始时存放从 $X2$ 点试射的初始猜测参数值, 调用后存放对 $V2$ 修改了的参数值
DELV1	含 $N2$ 个元素的一维实型数组, 输入参数, 存放猜想的数组 $V1$ 的一个微小增量
DELV2	含 $N1$ 个元素的一维实型数组, 输入参数, 存放猜想的数组 $V2$ 的一个微小增量
X1	实型变量, 输入参数, 积分区间左端点
X2	实型变量, 输入参数, 积分区间右端点
XF	实型变量, 输入参数, 区间 $[X1, X2]$ 中的一合适点
EPS	实型变量, 输入参数, 预先取定的误差限
H1	实型变量, 输入参数, 解初值问题时选取的初始尝试步长
HMIN	实型变量, 输入参数, 最小积分步长
F	含 NVAR 个元素的一维实型数组, 输出参数, 存放在 x_f 点的差异向量
DV1	含 $N2$ 个元素的一维实型数组, 输出参数, 存放试射参数值 $V1$ 修改的增量, 即经修改后, $V1$ 的值变为 $V1 + DV1$
DV2	含 $N1$ 个元素的一维实型数组, 输出参数, 存放对试射参数值 $V2$ 修正的增量

(2) 调用的子过程说明

- LOAD1 子过程 LOAD1($X1, V1, Y$), 由使用者自编, 其功能是由式(15-2)设定 n_2 个自由参数解出
- $$y_i(x_1) = Y_i(x_1; V_{(1)}) \quad (i = 1, 2, \dots, N)$$
- LOAD2 子过程 LOAD2($X2, V2, Y$), 由使用者自编, 其功能是由式(15-3)设定 n_1 个自由参数解出 $y_i(x_2) = y_i(x_2; V_{(2)}) (i = 1, 2, \dots, N)$
- SCORE 子过程 SCORE(XF, Y, F), 由使用者自编, 其功能是建立一个依赖于 $Y = (y_1, y_2, \dots, y_N)^T$ 的函数 $F = (F_1, F_2, \dots, F_N)^T$, 并计算 F_i . 在大多数情况下, 常令 $F_i[y(x_f, V)] = y_i(x_f, V)$
- DERIVS 子过程 DERIVS($X, Y, DYDX$), 由使用者自编, 其功能是计算右函数值 $\{g_i(x, y)\} (i = 1, \dots, N)$ 并将其放入 $DYDX$ 中
- ODEINT 为在区间上用变步长 R-K 法解初值问题的子过程, 见第 14 章
- RKQC 为用变步长 R-K 法积分一步初值问题的子过程, 它被 ODEINT 调用, 见第 14 章

4. 过程

过程 SHOOTF.

```

SUBROUTINE shootf(nvar,v1,v2,delv1,delv2,n1,n2,x1,&
    x2,xf,eps,h1,hmin,f,dv1,dv2)
EXTERNAL derivs,rkqc
PARAMETER (np=20)
! USES load1,odeint,score,load2,ludcmp,lubksb
DIMENSION v1(n2),delv1(n2),v2(n1),delv2(n1),f(nvar),&
    dv1(n2),dv2(n1),y(np),f1(np),f2(np),dfdvd(np,np)&
    ,indx(np)
REAL sav,x1,h1,eps,xf,hmin
INTEGER iv,j,n2,i,nvar
call load1(x1,v1,y)
call odeint(y,nvar,x1,xf,eps,h1,hmin,nok,nbad,&
    derivs,rkqc)
call score(xf,y,f1)
call load2(x2,v2,y)
call odeint(y,nvar,x2,xf,eps,h1,hmin,nok,nbad,&
    derivs,rkqc)
call score(xf,y,f2)

```

```

j=0
do iv=1,n2
  j=j+1
  sav=v1(iv)
  v1(iv)=v1(iv)+delv1(iv)
  call load1(x1,v1,y)
  call odeint(y,nvar,x1,xf,eps,h1,hmin,nok,nbad,&
              derivs,rkqc)
  call score(xf,y,f)
  do i=1,nvar
    dfdv(i,j)=(f(i)-f1(i))/delv1(iv)
  end do
  v1(iv)=sav
end do
do iv=1,n1
  j=j+1
  sav=v2(iv)
  v2(iv)=v2(iv)+delv2(iv)
  call load2(x2,v2,y)
  call odeint(y,nvar,x2,xf,eps,h1,hmin,nok,nbad,&
              derivs,rkqc)
  call score(xf,y,f)
  do i=1,nvar
    dfdv(i,j)=(f2(i)-f(i))/delv2(iv)
  end do
  v2(iv)=sav
end do
do i=1,nvar
  f(i)=f1(i)-f2(i)
  f1(i)=-f(i)
end do
call ludcmp(dfdv,nvar,np,indx,det)
call lubksb(dfdv,nvar,np,indx,f1)
j=0
do iv=1,n2
  j=j+1
  v1(iv)=v1(iv)+f1(j)

```

```

      dv1(iv)=f1(j)
    end do
    do iv=1,n1
      j=j+1
      v2(iv)=v2(iv)+f1(j)
      dv2(iv)=f1(j)
    end do
  END SUBROUTINE shootf

```

5. 例子

在验证程序 D15R2 中,我们所取的例子和 15.1 节中的例子一样,即球调和函数. 这里我们取两个端点为 $-1.0+DX$ 和 $1.0-DX$, 拟合点为 0.0. 为了清楚起见,我们只考虑长的球调和函数(即 $c^2 > 0$). 计算是和 15.1 节类似的,其中不同的地方为:

(1) 在子过程 DERIVS 中,只有三个一阶微分方程,因为这里只限于长球调和函数(若为扁球调和函数我们只需输入 c^2 为一负数).

(2) 有两个输入子过程 LOAD1 和 LOAD2,它们分别设定两个边界端点的值. 在第一个边界点上, $Y(3)$ 被赋以初值 $V_1(1)$,而 $V_1(1)$ 被初始设定为本征值的一个粗糙的猜测值. $Y(1)$ 即球调和函数的值被设定为 $FACTR + (dy_1/dx) \Delta x$. $Y(2)$ 和 15.1 节中所说的一样,是变化二阶微分方程到耦合一阶微分方程组的中间变量. 它在边界点上的值由式(15-19)确定. 在第二个边界点上 $Y(3)$ 和 $Y(1)$ 被分别赋以本征值的猜测值和 $y(1-\Delta x)$ 的猜测值. 我们在处理第二个边界点上本征值的猜测值时,和第一个边界点上不同,尽管它们应该收敛于同一个值. 因而在第二边界点上,本征值初始猜测值相差了 1.0(即 $V_2(2) = V_1(1) + 1.0$).

由 15.1 节我们有 $y_3 = \mu$, 而 $\mu = \lambda - m(m+1)$ (见式(15-17)). 由 15.3 节知该例中 $\lambda = 6.01427$. 本例中令 $m = 2$, 因而 $\mu = 0.01427$. 于是在左、右边界点上有 $V_1(1) = 0.01427$, $V_2(2) = 0.01427$. 另外,由于 $V_2(1) = y(1-\Delta x)$, 由 15.1 节中式(15-20), 可得 $y(1-\Delta x) = \frac{(-1)^m(n+m)!}{2^m m! (n-m)!} = 3$. 在验证程序 D15R2 中,调用子过程 SHOOTF 后,我们可将数值结果与上述结果相比较. 验证程序 D15R2 如下:

```

PROGRAM D15R2
! Driver for routine SHOOTF

```



```

PARAMETER(NVAR=3,N1=2,N2=1,DELTA=1.0E-3,EPS=1.0E-6,&
           DXX=1.0E-4)
DIMENSION V1(N2),DELV1(N2),V2(N1),DELV2(N1),&
           DV1(N2),DV2(N1),F(NVAR)
COMMON C2,M,N,FACTR,DX
DO
  WRITE(*,*) 'Input M,N,C-Squared (999 to end)'
  READ(*,*) M,N,C2
  IF(C2.EQ.999.) STOP
  IF((N>=M).OR.(M>=0)) THEN
    WRITE(*,*) 'Improper arguments'
    EXIT
  ENDIF
END DO
FACTR=1.0
IF (M/=0) THEN
  Q1=N
  DO I=1,M
    FACTR=-0.5*FACTR*(N+I)*(Q1/I)
    Q1=Q1-1.0
  END DO
ENDIF
DX=DXX
V1(1)=N*(N+1)-M*(M+1)+C2/2.0
IF(MOD(N-M,2)==0) THEN
  V2(1)=SIGN(1.0,FACTR)
ELSE
  V2(1)=-SIGN(1.0,FACTR)
ENDIF
V2(2)=V1(1)+1.0
DELV1(1)=DELTA*V1(1)
DELV2(1)=DELTA*FACTR
DELV2(2)=DELV1(1)
H1=0.1
HMIN=0.0
X1=-1.0+DX
X2=1.0-DX

```

```

XF=0.0
WRITE(*, '(/1X,T20,A,T40,A,T60,A)') &
      'Mu(-1)', 'Y(1-dx)', 'Mu(+1)'
DO
  CALL SHOOTF(NVAR,V1,V2,DELV1,DELV2,N1,N2,X1,X2,&
      XF,EPS,H1,HMIN,F,DV1,DV2)
  WRITE(*, '(/4X,A,3F20.6)') 'V', V1(1), V2(1), V2(2)
  WRITE(*, '(4X,A,3F20.6)') 'DV', DV1(1), DV2(1), DV2(2)
  IF (ABS(DV1(1)) <= ABS(EPS * V1(1))) EXIT
END DO
END PROGRAM
SUBROUTINE LOAD1(X1,V1,Y)
COMMON C2,M,N,FACTR,DX
DIMENSION V1(1),Y(3)
  Y(3)=V1(1)
  Y(2)=- (Y(3)-C2) * FACTR/2.0/(M+1.0)
  Y(1)=FACTR+Y(2) * DX
END SUBROUTINE LOAD1
SUBROUTINE LOAD2(X2,V2,Y)
COMMON C2,M,N
DIMENSION V2(2),Y(3)
  Y(3)=V2(2)
  Y(2)=(Y(3)-C2) * Y(1)/2.0/(M+1.0)
  Y(1)=V2(1)
END SUBROUTINE LOAD2
SUBROUTINE SCORE(XF,Y,F)
COMMON C2,M,N
DIMENSION Y(3),F(3)
  DO I=1,3
    F(I)=Y(I)
  END DO
END SUBROUTINE SCORE
SUBROUTINE DERIVS(X,Y,DYDX)
COMMON C2,M,N
DIMENSION Y(3),DYDX(3)
  DYDX(1)=Y(2)
  DYDX(3)=0.0

```

```

      DYDX(2)=(2.0*X*(M+1.0)*Y(2)-(Y(3)-C2*X*X)*Y(1))/&
      (1.0-X*X)
END SUBROUTINE DERIVS

```

计算结果如下:

Input M,N,C-Squared (999 to end)

Improper arguments

	Mu(-1)	Y(1-dx)	Mu(+1)
V	-0.868401	5.204582	-0.870388
DV	-0.918401	4.204582	-1.920388
V	-0.271325	3.055688	-0.271325
DV	0.597076	-2.148894	0.599062
V	-0.000912	3.010636	-0.000912
DV	0.270413	-0.045053	0.270414
V	0.014202	3.000006	0.014202
DV	0.015113	-0.010630	0.015113
V	0.014266	3.000003	0.014266
DV	0.000065	-0.000002	0.000065
V	0.014266	3.000005	0.014266
DV	0.000000	0.000002	0.000000

15.3 松 弛 法

1. 功能

本节用松弛法求解两点边值问题

$$\frac{dy}{dx} = g(x, y)$$

$$B_1(a, y(a)) = 0$$

$$B_2(b, y(b)) = 0$$

其中

$$y: R \rightarrow R^n, \quad g: R \times R^n \rightarrow R^n, \quad B_1: R \times R^n \rightarrow R^{n_1}$$

$$B_2: R \times R^n \rightarrow R^{n_2}, \quad n_1 + n_2 = n$$

当边界条件比较复杂或比较敏感时,松弛法比打靶法好. 松弛法最适用于问题的解光滑且不是高度振荡的情形.

本节子过程采用稀疏矩阵技术,分块运算,用隐式主元消去法求解线性方程组. 子过程 SOLVDE 是用松弛法解两点边值问题的子过程,它用牛顿法求解两点边值问题的有限差分方程. 子过程 BKSUB, PINVS, RED 均是被 SOLVDE 调用的子过程,其功能分别是:子过程 BKSUB 用于在每次迭代过程中解线性方程组的回代过程;子过程 PINVS 用主元消去法把每个子方块矩阵(n_1 阶,或 n_2 阶,或 n 阶)化为单位矩阵;子过程 RED 把每个子方块的左边的 n_1 个列(对第二个子方块与最后的子方块的)或 n 个列(对第三个及其以后的子方块)用该子方块上面的行化简为零,当然,该子方块也随之变化.

2. 方法

松弛法就是在积分区域的一组网格点上用有限差分方程代替常微分方程,再用迭代法求解有限差分方程得到常微分方程在网格点上的近似值.

(1) 有限差分方程.

1) 在积分区域上选取一组网格点 $\{x_j\}$, $j=1, 2, \dots, M$, 使

$$a = x_1 < x_2 < \dots < x_M \equiv x_{M+1} = b$$

记

$$Y^{(k)} = (y_1(x_k), \dots, y_n(x_k))^T = (y_1^{(k)}, \dots, y_n^{(k)})^T, \quad k = 1, \dots, M$$

2) 在每个网格点 K 处用下述形式的逼近格式逼近微分方程

$$0 = E^{(k)} = y^{(k)} - y^{(k-1)} - (x_k - x_{k-1})g^{(k)}(x_{1c}, x_{k-1}, y^{(k)}, y^{(k-1)}), \quad k = 2, 3, \dots, M$$

在第一个边界点 $x=a$ 处有

$$0 = E^{(1)} = B(x_1, y^{(1)})$$

在第二个边界点 $x=b$ 处有

$$0 = E^{(M+1)} = C(x_{m+1}, y^{(M+1)}), \quad y^{(M+1)} \equiv y^{(M)}.$$

这里每个 $E^{(k)} = E^{(k)}(y^{(k)}, y^{(k-1)})$, $k=2, \dots, m$, $E^{(1)} = E^{(1)}(y^{(1)})$, $E^{(M+1)} = E^{(M+1)}(y^{(M+1)})$, $E^{(1)}$ 和 B 中仅有 n_1 个非零分量, $E^{(M+1)}$ 和 C 中仅有 n_2 个非零分量,不妨设

$$E_j^{(1)} \neq 0, \quad j = n_2 + 1, \dots, n; \quad E_j^{(M+1)} \neq 0, \quad j = 1, \dots, n_2$$

$$E^{(1)} = (E_{n_2+1}^{(1)}, \dots, E_n^{(1)})^T, \quad E^{(2)} = (E_1^{(2)}, \dots, E_{n_2}^{(2)})^T$$

3) 记

$$y = (y^{(1)T}, \dots, y^{(m)T})^T \in R^{(M \times n)}$$

$$E(y) = (E^{(1)}(y^{(1)T}), E^{(2)}(y^{(2)T}, y^{(1)T}), \dots, E^{(M)}(y^{(M)T}, y^{(M-1)T}), E^{(M+1)}(y^{(M)T})^T)$$

则问题即为求 $y \in R^{(M \times n)}$ 使

$$E(y) = 0$$

这里 $E: R^{(M \times n)} \rightarrow R^{(M \times n)}$. 此即是所求解的有限差分方程. 实质上是一代数方程组.

(2) 用牛顿迭代法解有限差分方程.

基本迭代格式是:

- 1) 给定 y 的初始值 y^0 , $k := 0$.
- 2) 解线性方程组

$$E'(y^k) \Delta y^k = -E(y^k)$$

3) 若 $\|\Delta y^k\| < \epsilon$ (预设精度), 则停止计算, 此时的 y^k 即为所求的近似解, 否则转下一步.

- 4) $y^k := y^k + \Delta y^k$, $k := k + 1$, 转 2).

(3) 牛顿迭代过程中的 $E'(y)$.

记 $S = E'(y)$, 则 S 为

$$S = \begin{bmatrix} \left(\frac{\partial E^{(1)}}{\partial y^{(1)}} \right)_{n_1 \times n} & & & & \\ \left(\frac{\partial E^{(2)}}{\partial y^{(1)}} \right)_{n \times n} & \left(\frac{\partial E^{(2)}}{\partial y^{(2)}} \right)_{n \times n} & & & \\ & \dots & & & \\ & & \left(\frac{\partial E^{(M)}}{\partial y^{(M-1)}} \right)_{n \times n} & \left(\frac{\partial E^{(M)}}{\partial y^{(M)}} \right)_{n \times n} & \\ & & & \left(\frac{\partial E^{(M+1)}}{\partial y^{(M)}} \right)_{n_2 \times n} \end{bmatrix}_{(M \times n) \times (M \times n)}$$

其中,

$$\frac{\partial E^{(k)}}{\partial y^{(k-1)}} = \left(\frac{\partial E_i^{(k)}}{\partial y_j^{(k-1)}} \right)_{n \times n}, \quad i, j = 1, \dots, n, \quad k = 2, \dots, M$$

$$\frac{\partial E^{(1)}}{\partial y^{(1)}} = \left(\frac{\partial E_i^{(1)}}{\partial y_j^{(1)}} \right)_{n_1 \times n}, \quad i = 1, \dots, n_1, \quad j = 1, \dots, n$$

$$\frac{\partial E^{(M+1)}}{\partial y^{(M)}} = \left(\frac{\partial E_i^{(M+1)}}{\partial y_j^{(M)}} \right)_{n_2 \times n}, \quad i = 1, \dots, n_2, \quad j = 1, \dots, n$$

取

$$\|\Delta y\| = \frac{1}{Mn} \sum_{k=1}^M \sum_{j=1}^n |\Delta y_j^{(k)} / c_j|$$

其中 c_j 是一适当的加权比例因子. 习惯上常取为: $c_j = |y_j^{(k)}| + |hg_j(x_k, g^{(k)})|$.

另外再取一校正控制参数 L . 若 $\|\Delta y\| < \varepsilon$, 则表示方法收敛, $y + \Delta y$ 即为所求, 否则, 将 $y_j^{(k)}$ 换为 $y_j^{(k)} + \frac{L}{\max(L, \|\Delta y\|)} \Delta y_j^{(k)}$.

3. 使用说明

(1) SOLVDE (ITMAX, CONV, SLOWC, SCALV, INDEXV, NE, NB, M, Y, NYJ, NYK, C, NCI, NCJ, NCK, S, NSI, NSJ)

- | | |
|--------|--|
| NE | 整型变量, 输入参数, 微分方程个数 |
| ITMAX | 整型变量, 输入参数, 迭代的最大次数 |
| CONV | 实型变量, 输入参数, 给定的收敛尺度 |
| SLOWC | 实型变量, 输入参数, 每次迭代的校正控制参数 |
| SCALV | 含 NE 个元素的一维实型数组, 输入参数, 是加权比例因子, 由用户提供 |
| INDEXV | 含 NE 个元素的一维整型数组, 输入参数, 存放导数矩阵 S 的列序与方程中的相关变量 y_i 之间的对应关系 |
| NB | 整型变量, 输入参数, 左边界条件的个数 n_1 |
| M | 整型变量, 输入参数, 网格点个数 |
| NYJ | 整型变量, 输入参数, 和 NE 相等 |
| NYK | 整型变量, 输入参数, 和 M 相等 |
| Y | 实型数组 $Y(NYJ, NYK)$, 输入、输出参数, 开始时存放在每个网格点上对 Y 的初始猜想值, 每次迭代后存放 Y 的校正值 |
| C | 实型数组 $C(NCI, NCJ, NCK)$, 工作单元 |
| NCI | 整型变量, 输入参数, 存放工作单元 C 第 K 块的行数, 且满足 $NCI \geq NE$ |
| NCJ | 整型变量, 输入参数, 存放工作单元 C 第 K 块的列数, 满足 $NCJ \geq NE - NB + 1$ |
| NCK | 整型变量, 输入参数, 存放工作单元 C 的块数, $NCK \geq M + 1$ |
| S | 实型数组 $S(NSI, NSJ)$, 输入参数 (由用户提供的子程序 DIFEQ 形成并输入), 存放每一网格点上形成的方程组的系数矩阵 |
| NSI | 整型变量, 输入参数, 存放 S 矩阵的行数, 满足 $NSI \geq NE$ |
| NSJ | 整型变量, 输入参数, 存放 S 矩阵的列数, 满足 $NSJ \geq 2 * NE + 1$ |

(2) BKSUB(NE,NB,JF,K1,K2,C,NCI,NCJ,NCK)

- NE 整型变量,输入参数,方程个数
- NB 整型变量,输入参数,左边界条件个数
- JF 整型变量,输入参数,为 $NE - NB + 1$
- K1 整型变量,输入参数, $K1 = 1$
- K2 整型变量,输入参数,为网格点个数
- C 实型数组 $C(NCI,NCJ,NCK)$,在消元过程中形成工作单元
- NCI 整型变量,输入参数,存放工作单元 C 中第 K 块的行数,满足 $NCI \geq NE$
- NCJ 整型变量,输入参数,存放工作单元 C 中第 K 块的列数,满足 $NCJ \geq NE - NB + 1$
- NCK 整型变量,输入参数,存放 C 的块数,即 $NCK = M + 1$

(3) PINVS (IE1,IE2,JE1,JSF,JC1,K,C,NCI,NCJ,NCK,S,NSI,NSJ)

- IE1 整型变量,输入参数,矩阵 S 的第一行的行号
- IE2 整型变量,输入参数,矩阵 S 的最后行的行号
- JE1 整型变量,输入参数,矩阵 S 中小正方块的第一列的列号
- JSF 整型变量,输入参数,方程右端项在 S 矩阵中的列号
- JC1 整型变量,输入参数,第 K 个 S 矩阵的第一列的列号
- K 整型变量,输入参数,当前的网格点号数
- NCI 整型变量,输入参数,工作单元 C 中第 K 块的行数,满足 $NCI \geq NE$
- NCJ 整型变量,输入参数,工作单元 C 中第 K 块的列数,满足 $NCJ \geq NE - NB + 1$
- NCK 整型变量,输入参数,工作单元 C 的块数
- C 实型数组 $C(NCI,NCJ,NCK)$,输出参数,存放消元后的系数,工作单元
- NSI 整型变量,输入参数,第 K 个 S 矩阵的行数
- NSJ 整型变量,输入参数,第 K 个 S 矩阵的列数
- S 实型矩阵 $S(NSI,NSJ)$,输入参数(由子程序 DIFEQ 输入),存放 $\frac{\partial E}{\partial y_n}$

(4) RED (JZ1,JZ2,JZ1,JZ2,JM1,JM2,JMF,IC1,JC1,JCF,KC,C,NCI,NCJ,NCK,S,NSI,NSJ)

- IZ1 整型变量,输入参数,存放第 K 个 S 矩阵的首行号

- IZ2 整型变量,输入参数,存放第 K 个 S 矩阵的末行号
- JZ1 整型变量,输入参数,存放第 K 个 S 矩阵第一列的列号
- JZ2 整型变量,输入参数, S 矩阵中第 $K-1$ 个小正方阵的最后一列的列号
- JM1 整型变量,输入参数,系数矩阵中第 K 块消元后第一个非零列的列号
- JM2 整型变量,输入参数,系数矩阵中第 $K-1$ 块非零的最后一列在整个系数矩阵中的列号;
- JMF 整型变量,输入参数,方程右端项在 S 矩阵中的列号
- IC1 整型变量,输入参数, $IC1=NE-NB+1$
- JC1 整型变量,输入参数, $JC1=1$
- JCF 整型变量,输入参数, $JCF=NE-NB+1$
- KC 整型变量,输入参数,存放系数矩阵上一块的块号,即若当前块为第 K 块,则此时 $KC=K-1$
- C 实型数组 $C(NCI,NCJ,NCK)$,在消元过程中形成,在本子过程中为输入参数
- NCI 整型变量,输入参数,数组 C 中第 K 块的行数,满足 $NCI \geq NE$
- NCJ 整型变量,输入参数,数组 C 中第 K 块的列数,满足 $NCJ \geq NE-NB+1$
- NCK 整型变量,输入参数,数组 C 的块数,满足 $NCK=M+1$
- S 实矩阵 $S(NSI,NSJ)$,由子程序 DIFEQ 形成,在本子过程中为输入参数
- NSI 整型变量,输入参数,矩阵 S 的行数, $NSI=NE$
- NSJ 整型变量,输入参数,矩阵 S 的列数,且 $NSJ=Z * NE+1$
- (5) DIFEQ (K, K1, K2, JSF, IS1, ISF, INDEXV, NE, S, NSI, NSJ, Y, NYJ, NYK)

子过程 DIFEQ 由使用者自编,其功能是形成矩阵 $S(I, J) (0 \leq I \leq NE, 1 \leq J \leq NE \text{ 或 } 1 \leq J \leq 2 \times NE)$. $S(I, J)$ 的具体形式见“方法”中. 使用说明如下:

- K 整型变量,输入参数,表示当前形成的矩阵 S 在整个系数矩阵中的行块的号数
- K1 整型变量,输入参数, $K1=1$
- K2 整型变量,输入参数, $K2=M$
- JSF 整型变量,输入参数,存放方程右端项在 S 矩阵的列号
- IS1 整型变量,输入参数,存放当前的 S 矩阵中第一行的行号

ISF	整型变量,输入参数,存放当前的 S 矩阵中最后一行的行号
INDEXV	含 NE 个元素的一组整型数组,输入参数,存放导数矩阵 S 的列序与变量 $Y(J)$ 的对应关系,以便在以后的消元过程中主元不为零
NE	整型变量,输入参数,方程个数
S	实型数组 $S(NSI, NSJ)$, 输出参数,由本子过程形成,存放 $\frac{\partial E}{\partial y}$
NSI	整型变量,输入参数,存放当前的 S 矩阵的行数, $NSI = ISF - IS1 = NE$
NSJ	整型变量,输入参数,存放当前的 S 矩阵的列数, $NSJ = 2 * NE + 1$
Y	实数组 $Y(NYJ, NYK)$, 输入参数,存放对 Y 的初始猜想
NYJ	整型变量,输入参数, $NYJ = NE$
NYK	整型变量,输入参数,等于网格点个数

4. 过程

(1) 子过程 SOLVDE.

```

SUBROUTINE solvde(itmax,conv,slowc,scalv,indexv,ne,&
    nb,m,y,nyj,nyk,c,nci,ncj,nck,s,nsi,nsj)
PARAMETER (nmax=10)
DIMENSION y(nyj,nyk),c(nci,ncj,nck),s(nsi,nsj),&
    scalv(nyj),ermax(nmax),kmax(nmax),indexv(nyj)
REAL errj,vmax,vz,err,fac,conv
INTEGER k1,k2,nvars,j1,j2,j3,j4,j5,j6,j7,j8,j9
INTEGER ic1,ic2,m,ne,nb,it,k,jv,km,kp
k1=1
k2=m
nvars=ne*m
j1=1
j2=nb
j3=nb+1
j4=ne
j5=j4+j1
j6=j4+j2
j7=j4+j3

```

```

j8=j4+j4
j9=j8+j1
ic1=1
ic2=ne-nb
ic3=ic2+1
ic4=ne
jc1=1
jcf=ic3
do it=1,itmax
  k=k1
  call difeq(k,k1,k2,j9,ic3,ic4,indexv,ne,s,nsi,&
             nsj,y,nyj,nyk)
  call pinvs(ic3,ic4,j5,j9,jc1,k1,c,nci,ncj,nck,s,&
             nsi,nsj)
  do k=k1+1,k2
    kp=k-1
    call difeq(k,k1,k2,j9,ic1,ic4,indexv,ne,s,&
             nsi,nsj,y,nyj,nyk)
    call red(ic1,ic4,j1,j2,j3,j4,j9,ic3,jc1,jcf,&
             kp,c,nci,ncj,nck,s,nsi,nsj)
    call pinvs(ic1,ic4,j3,j9,jc1,k,c,nci,ncj,nck&
             ,s,nsi,nsj)
  end do
  k=k2+1
  call difeq(k,k1,k2,j9,ic1,ic2,indexv,ne,s,nsi,&
             nsj,y,nyj,nyk)
  call red(ic1,ic2,j5,j6,j7,j8,j9,ic3,jc1,jcf,k2,&
             c,nci,ncj,nck,s,nsi,nsj)
  call pinvs(ic1,ic2,j7,j9,jcf,k2+1,c,nci,ncj,nck,&
             s,nsi,nsj)
  call bksub(ne,nb,jcf,k1,k2,c,nci,ncj,nck)
  err=0.
  do j=1,ne
    jv=indexv(j)
    ermax(j)=0.
    errj=0.
    kmax(j)=0

```

```

vmax=0.
do k=k1,k2
  vz=abs(c(j,1,k))
  if(vz>vmax) then
    vmax=vz
    km=k
  endif
  errj=errj+vz
end do
err=err+errj/scalv(jv)
ermax(j)=c(j,1,km)/scalv(jv)
kmax(j)=km
end do
err=err/nvars
fac=slowc/max(slowc,err)
do jv=1,ne
  j=indexv(jv)
  do k=k1,k2
    y(j,k)=y(j,k)-fac*c(jv,1,k)
  end do
end do
write(*, '(1x,i4,2f12.6,(/5x,i5,f12.6))') &
      it,err,fac,(kmax(j),ermax(j),j=1,ne)
if(err<conv) return
end do
pause 'itmax exceeded'
END SUBROUTINE solvde

```

(2) 子过程 BKSUB.

```

SUBROUTINE bksub(ne,nb,jf,k1,k2,c,nci,ncj,nck)
DIMENSION c(nci,ncj,nck)
REAL xx
INTEGER k,k2,k1,j,i,kp,nb,ne
nbf=ne-nb
do k=k2,k1,-1
  kp=k+1
  do j=1,nbf

```

```

      xx=c(j,jf,kp)
      do i=1,ne
        c(i,jf,k)=c(i,jf,k)-c(i,j,k)*xx
      end do
    end do
  end do
do k=k1,k2
  kp=k+1
  do i=1,nb
    c(i,1,k)=c(i+nbj,jf,k)
  end do
  do i=1,nbf
    c(i+nb,1,k)=c(i,jf,kp)
  end do
end do
END SUBROUTINE bksub

```

(3) 子过程 PINVS.

```

SUBROUTINE pinvs(ie1,ie2,je1,jf,jc1,k,c,nci,ncj,nck,&
  s,nsi,nsj)
PARAMETER (zero=0.,one=1.,nmax=10)
DIMENSION c(nci,ncj,nck),s(nsi,nsj),pscl(nmax),&
  indxr(nmax)
INTEGER ie1,ie2,je1,jf,jc1,k,nci,ncj,nck,nsi,nsj
INTEGER je2,js1,i,id,jp,ipiv,jpiv
REAL big,piv,pivinv,c
je2=je1+ie2-ie1
js1=je2+1
do i=ie1,ie2
  big=zero
  do j=je1,je2
    if(abs(s(i,j))>big) big=abs(s(i,j))
  end do
  if(big==zero)pause 'singular matrix, row all 0'
  pscl(i)=one/big
  indxr(i)=0
end do

```

```

do id=ie1,ie2
  piv=zero
  do i=ie1,ie2
    if(indxr(i)==0) then
      big=zero
      do j=je1,je2
        if(abs(s(i,j))>big) then
          jp=j
          big=abs(s(i,j))
        endif
      end do
      if(big * pscl(i)>piv) then
        ipiv=i
        jpiv=jp
        piv=big * pscl(i)
      endif
    endif
  end do
  if(s(ipiv,jpiv)==zero) pause 'singular matrix'
  indxr(ipiv)=jpiv
  pivinv=one/s(ipiv,jpiv)
  do j=je1,jpf
    s(ipiv,j)=s(ipiv,j) * pivinv
  end do
  s(ipiv,jpiv)=one
  do i=ie1,ie2
    if(indxr(i)/=jpiv) then
      if(s(i,jpiv)/=zero) then
        dum=s(i,jpiv)
        do j=je1,jpf
          s(i,j)=s(i,j)-dum * s(ipiv,j)
        end do
        s(i,jpiv)=zero
      endif
    endif
  end do
end do
end do

```

```

jcoff=jc1-js1
icoff=ie1-je1
do i=ie1,ie2
    irow=indxr(i)+icoff
    do j=js1,jsf
        c(irow,j+jcoff,k)=s(i,j)
    end do
end do
END SUBROUTINE pinvs

```

(4) 子过程 RED.

```

SUBROUTINE red(iz1,iz2,jz1,jz2,jm1,jm2,jmf,ic1,jc1,&
    jcf,kc,c,nci,ncj,nck,s,nsi,nsj)
    DIMENSION c(nci,ncj,nck),s(nsi,nsj)
    REAL vx
    INTEGER iz1,iz2,jz1,jz2,jm1,jm2,jmf,ic1,jc1,jcf,kc,nci
    INTEGER ncj,nck,nsi,nsj,loff,ic,l,i
    loff=jc1-jm1
    ic=ic1
    do j=jz1,jz2
        do l=jm1,jm2
            vx=c(ic,l+loff,kc)
            do i=iz1,iz2
                s(i,l)=s(i,l)-s(i,j)*vx
            end do
        end do
        vx=c(ic,jcf,kc)
        do i=iz1,iz2
            s(i,jmf)=s(i,jmf)-s(i,j)*vx
        end do
        ic=ic+1
    end do
END SUBROUTINE red

```

5. 例子

本节所用的例子与 15.1 节中的例子相同,即球调和函数 $S(x,c) = (1-x^2)^{m/2}y(x,c)$,其中 y 所满足的一阶微分方程组为

$$\frac{dy_1}{dx} = y_2 \quad (15-23)$$

$$\frac{dy_2}{dx} = \frac{1}{1-x^2} [2x(m+1)y_2 - (y_3 - c^2 x^2)y_1] \quad (15-24)$$

$$\frac{dy_3}{dx} = 0 \quad (15-25)$$

其中 y_1 即为所求的函数, y_3 对应于特征值. 当 c^2 为正数时对应于长球调和函数, 当 c^2 为负数时对应于扁球调和函数. 同 15.1 节相似, 这里我们在 $[0, 1]$ 上求解.

由式(15-19)~(15-22)得边界条件: 在 $x=0$ 时, $y_1=0$, $n-m$ 为奇数, $y_2=0$, $n-m$ 为偶数.

在 $x=1$ 时,

$$y_1 = \nu, \quad y_2 = \frac{1}{2(m+1)} [y_3 - c^2] y_1$$

如果我们只要求几个孤立的 λ 或 S 的值, 打靶法是最快的. 但是如果我们要求一系列 C 都求其值时, 松弛法是一种好方法. 它能给出一个具有快速收敛的初始猜测值, 且当 C 微小变化时对应于前一个 C 的解将是变化后 C 的解的一个好的初始猜测.

为简单起见, 我们采用 $[0, 1]$ 区间上的均匀网格, 总网格点数为 M . 这时

$$h = \frac{1}{M-1}, \quad x_k = (k-1)h, \quad k = 1, 2, \dots, M$$

在内点 $k=2, 3, \dots, M$, 在微分方程中以

$$\begin{aligned} (x_k + x_{k-1})/2 & \quad \text{代替 } x_k \\ (y^{(k)} + y^{(k-1)})/2 & \quad \text{代替 } y^{(k)} = y(x_k) \\ (y^{(k)} - y^{(k-1)})/(x_k - x_{k-1}) & \quad \text{代替 } \frac{dy(x_k)}{dx} \end{aligned}$$

则得相应的一有限差分方程. 式(15-23)给出

$$E_1^{(k)} = y_1^{(k)} - y_1^{(k-1)} - \frac{h}{2} (y_2^{(k)} - y_2^{(k-1)}) \quad (15-36)$$

方程式(15-24)给出

$$\begin{aligned} E_2^{(k)} = & y_2^{(k)} - y_2^{(k-1)} - \beta_k \\ & \times \left[\frac{(x_k + x_{k-1})(m+1)(y_2^{(k)} + y_2^{(k-1)})}{2} \right. \\ & \left. - \alpha_k \frac{y_1^{(k)} + y_1^{(k-1)}}{2} \right] \end{aligned} \quad (15-37)$$

其中

$$\alpha_k = \frac{y_3^{(k)} + y_3^{(k-1)}}{2} - \frac{c^2(x_k + x_{k-1})^2}{4} \quad (15-38)$$

$$\beta_k = h / \left[1 - \frac{1}{4}(x_k + x_{k-1})^2 \right] \quad (15-39)$$

方程式(15-25)给出

$$E_3^{(k)} = y_3^{(k)} - y_3^{(k-1)} \quad (15-40)$$

在本例中 $n=3, n_1=1, n_2=2$.

在第一边界点 $x=0$ 处有

$$E_3^{(1)} = \begin{cases} y_1^{(1)} - 0 = y_1^{(1)}, & \text{当 } n-m \text{ 为奇数时} \\ y_2^{(1)} - 0 = y_2^{(1)}, & \text{当 } n-m \text{ 为偶数时} \end{cases} \quad (15-41)$$

在第二边界点 $x=1$ 处有

$$E_1^{(M+1)} = y_2^{(M)} - \frac{1}{2(m+1)}[y_3^{(M)} - c^2]y_1^{(M)} \quad (15-42)$$

$$E_2^{(M+1)} = y_1^{(M)} - y \quad (15-43)$$

下面计算出 $E(y)$ 的导数矩阵 S (注意, 这是本节“方法”中的 S , 非表示球调和函数). 记

$$S^{(k)} = \left[\frac{\partial E^{(k)}}{\partial y^{(k-1)}}, \frac{\partial E^{(k)}}{\partial y^{(k)}} \right] = [s_{ij}]_{n \times (2n)}, \quad k = 1, \dots, M+1$$

其中

$$S^{(1)} \equiv \left[0, \frac{\partial E^{(1)}}{\partial y^{(1)}} \right]$$

$$S^{(M+1)} \equiv \left[0, \frac{\partial E^{(M+1)}}{\partial y^{(M+1)}} \right], \quad y^{(M+1)} \equiv y^{(M)}$$

于是式(15-36)给出

$$S_{11}^{(k)} = -1, \quad S_{12}^{(k)} = -\frac{h}{2}, \quad S_{13}^{(k)} = 0 \quad (15-44)$$

$$S_{14}^{(k)} = 1, \quad S_{15}^{(k)} = -\frac{h}{2}, \quad S_{16}^{(k)} = 0$$

方程式(15-37)给出

$$S_{21}^{(k)} = \alpha_k \beta_k / 2, \quad S_{22}^{(k)} = -1 - \beta_k(x_k + x_{k-1})(m+1)/2 \quad (15-45)$$

$$S_{23}^{(k)} = \beta_k(y_1^{(k)} + y_1^{(k-1)})/4, \quad S_{24}^{(k)} = S_{21}^{(k)}$$

$$S_{25}^{(k)} = 2 + S_{22}^{(k)}, \quad S_{26}^{(k)} = S_{23}^{(k)}$$

而由方程式(15-40)得

$$S_{31}^{(k)} = 0, \quad S_{32}^{(k)} = 0, \quad S_{33}^{(k)} = -1 \quad (15-46)$$

$$S_{34}^{(k)} = 0, \quad S_{35}^{(k)} = 0, \quad S_{36}^{(k)} = 1$$

第一边界条件给出

$$S_{34}^{(1)} = 1, \quad \text{当 } n - m \text{ 为奇数时} \quad (15-47)$$

$$S_{35}^{(1)} = 1, \quad \text{当 } n - m \text{ 为偶数时}$$

第二边界条件式(15-42), (15-43)给出

$$S_{14}^{(M+1)} = -\frac{y_3^{(M)} - c^2}{2(m+1)}, \quad S_{15}^{(M+1)} = 1,$$

$$S_{16}^{(M+1)} = -\frac{y_1^{(M)}}{2(M+1)} \quad (15-48)$$

$$S_{24}^{(M+1)} = 1, \quad S_{25}^{(M+1)} = S_{26}^{(M+1)} = 0 \quad (15-49)$$

例子中的子过程 DIFEQ 实现上述 $S_i^{(k)}$ 的计算, 其中选取网格点数 $M=41$ 的等距网格, 即 $h=0.025$.

应该注意, 在子过程 SOLUDE 中, 在每个子块中选列主元时, 若主元为零, 则方法失败. 为使程序正常运行, 必须交换 S 的列, 使子块中的主元不为零. 这时需记录交换后变量的次序. 子程序的整型数组 INDEXV 即是记录交换 S 的列时, 变量 $Y(J)$ 与矩阵 S 的列的对应性.

例子的主程序是 SFROID, 从它的计算结果我们将会看到该程序给出高精度的特征值. 由于当 $n-m$ 是偶数时, 在 $x=0$ 的边界条件中不包含 y_1 , 因此在初始的第一个子块中就会出现零主元, 因而我们必须使用数组 INDEXV, 交换 S 中 y_1 和 y_2 的列, 使 S 中主元不为零.

主程序 SFROID 已为用户输入 m 和 n . 然后, 以连带勒让德函数 P_n^m 为基础为 y 计算初值(计算勒让德函数的函数子过程 PLGNDR 见第 4 章)为了解出 y , 下一个提示是输入 C^2 的值, 然后, 每次计算 y 都以上次计算的 y 为初值, 如此不断地进行计算直至达到要求.

形成矩阵 S 的子过程附在主程序 SFROID 的后面.

主程序 SFROID 如下:

```
PROGRAM SFROID
  ! PARAMETER (NE=3,M=41,NB=1,NCI=NE,NCJ=NE-NB+1&
  !           NCK=M+1,NSI=NE,NSJ=2*NE+1,NYJ=NE,NYK=M)
  PARAMETER (NE=3,M=41,NB=1,NCI=3,NCJ=3,NCK=42,NSI=3,&
             NSJ=7,NYJ=3,NYK=41)
  ! USES PLGNDR
  COMMON X(M),H,MM,N,C2,ANORM
  DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,&
             NCK),S(NSI,NSJ)
```

```

ITMAX=100
CONV=5.E-6
SLOWC=1.
H=1./(M-1)
C2=0.0
WRITE(*,*)'ENTER M,N'
READ(*,*)MM,N
IF(MOD(N+MM,2) /= 1) THEN
  ! No interchange necessary
    INDEXV(1)=1
    INDEXV(2)=2
    INDEXV(3)=3
ELSE
  ! Interchange y1 and y2
    INDEXV(1)=2
    INDEXV(2)=1
    INDEXV(3)=3
ENDIF
ANORM=1.
IF(MM /= 0) THEN
  Q1=N
  DO I=1,MM
    ANORM=-.5 * ANORM * (N+I) * (Q1/I)
    Q1=Q1-1.
  END DO
ENDIF
! Initial guess
DO K=1,M-1
  X(K)=(K-1) * H
  FAC1=1.-X(K) * * 2
  FAC2=FAC1 * * (-MM/2.)
  ! PLGNDR computes Legendre function
  Y(1,K)=PLGNDR(N,MM,X(K)) * FAC2
  DERIV=-((N-MM+1) * PLGNDR(N+1,MM,X(K))-(N+1) * &.
    X(K) * PLGNDR(N,MM,X(K)))/FAC1
  Y(2,K)=MM * X(K) * Y(1,K)/FAC1+DERIV * FAC2

```

```

      Y(3,K)=N*(N+1)-MM*(MM+1)
END DO
! Initial guess at x=1 done separately
X(M)=1.
Y(1,M)=ANORM
Y(3,M)=N*(N+1)-MM*(MM+1)
Y(2,M)=(Y(3,M)-C2)*Y(1,M)/(2.*(MM+1.))
SCALV(1)=ABS(ANORM)
SCALV(2)=MAX(ABS(ANORM),Y(2,M))
SCALV(3)=MAX(1.,Y(3,M))
DO
  WRITE(*,*)'ENTER C * * 2 OR 999 TO END'
  READ(*,*)C2
  IF(C2==999.)STOP
  CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,&
    NB,M,Y,NYJ,NYK,C,NCI,NCJ,NCK,S,NSI,NSJ)
  WRITE(*, '(1X,A,I3,A,I3,A,F8.4,A,F11.7)') 'M =',MM,&
    ' N =',N,' C * * 2 =',C2,' LAMBDA =',Y(3,1)+MM*(MM+1)
END DO
END

```

子过程 DIFEQ.

```

SUBROUTINE difeq(k,k1,k2,jsf,is1,isf,indexv,ne,s,nsi&
  ,nsj,y,nyj,nyk)
PARAMETER(m=41)
COMMON x(m),h,mm,n,c2,anorm
DIMENSION y(nyj,nyk),s(nsi,nsj),indexv(nyj)
INTEGER k,k1,k2,jsf,is1,isf,indexv,ne,nsi,nsj,nyj,nyk
if(k==k1)then
! boundary condition at first point
  if(mod(n+mm,2)==1) then
    s(3,3+indexv(1))=1. ! equation (15.37)
    s(3,3+indexv(2))=0.
    s(3,3+indexv(3))=0.
    s(3,jsf)=y(1,1) ! equation (15.31)
  else

```

```

    s(3,3+indexv(1))=0.    ! equation (15. 37)
    s(3,3+indexv(2))=1.
    s(3,3+indexv(3))=0.
    s(3,jsf)=y(2,1)        ! equation (15. 31)
endif
else if(k>k2) then
! boundary conditions at last point
! equation (15. 38)
s(1,3+indexv(1))=-(y(3,m)-c2)/(2. * (mm+1. ))
s(1,3+indexv(2))=1.
s(1,3+indexv(3))=-y(1,m)/(2. * (mm+1. ))
! equation (15. 32)
s(1,jsf)=y(2,m)-(y(3,m)-c2)*y(1,m)/(2. * (mm+1. ))
s(2,3+indexv(1))=1.    ! equation (15. 39)
s(2,3+indexv(2))=0.
s(2,3+indexv(3))=0.
s(2,jsf)=y(1,m)-anorm ! equation (15. 33)
else
! interior point
s(1,indexv(1))=-1.    ! equation (15. 34)
s(1,indexv(2))=-0.5*h
s(1,indexv(3))=0.
s(1,3+indexv(1))=1.
s(1,3+indexv(2))=-0.5*h
s(1,3+indexv(3))=0.
temp=h/(1. -(x(k)+x(k-1))*2*.25)
temp2=.5*(y(3,k)+y(3,k-1))-c2*.25*(x(k)+x(k-1))*2
s(2,indexv(1))=temp*temp2*.5    ! equation (15. 35)
s(2,indexv(2))=-1. -.5*temp*(mm+1.)*(x(k)+x(k-1))
s(2,indexv(3))=.25*temp*(y(1,k)+y(1,k-1))
s(2,3+indexv(1))=s(2,indexv(1))
s(2,3+indexv(2))=2.+s(2,indexv(2))
s(2,3+indexv(3))=s(2,indexv(3))
s(3,indexv(1))=0.    ! equation (15. 36)
s(3,indexv(2))=0.
s(3,indexv(3))=-1.

```

```

s(3,3+indexv(1))=0.
s(3,3+indexv(2))=0.
s(3,3+indexv(3))=1.
! equation (15.26)
s(1,jsf)=y(1,k)-y(1,k-1)-.5*h*(y(2,k)+y(2,k-1))
! equation (15.27)
s(2,jsf)=y(2,k)-y(2,k-1)-temp*((x(k)+x(k-1))&
      *.5*(mm+1.)*(y(2,k)+y(2,k-1))-temp2*&
      .5*(y(1,k)+y(1,k-1)))
s(3,jsf)=y(3,k)-y(3,k-1) ! equation (15.30)
endif
END SUBROUTINE difeq

```

计算结果如下:

```

ENTER M,N
ENTER C * * 2 OR 999 TO END
2,2
  1      0.008750      1.000000
        41      0.014215
        1      -0.007157
        28      -0.014354
  2      0.000034      1.000000
        41      0.000075
        2      -0.000007
        28      0.000087
  3      0.000000      1.000000
        41      -0.000001
        13      0.000000
        28      0.000000

M=2  N=2  C * * 2=0.1000  LAMBDA=6.0142665
ENTER C * * 2 OR 999 TO END
1.0
  1      0.080969      1.000000
        41      0.127276
        1      -0.065830
        1      -0.133780

```

```

2      0.002709      1.000000
      41      0.001729
      1      -0.000615
      15      0.007099
3      0.000003      1.000000
      41      -0.000119
      40      0.000001
      38      -0.000004
M= 2  N= 2  C * * 2= 1.0000  LAMBDA= 6.1409512
ENTER  C * * 2  OR 999 TO END
4.0
1      0.291697      1.000000
      41      0.422934
      1      -0.241126
      14      -0.479138
2      0.030153      1.000000
      40      0.013822
      1      -0.007647
      16      0.077959
3      0.000165      1.000000
      40      0.000864
      1      0.000040
      11      -0.000397
4      0.000000      1.000000
      41      -0.000006
      2      0.000000
      35      0.000000
M= 2  N= 2  C * * 2= 4.0000  LAMBDA= 6.5425267
ENTER C * * 2 OR 999 TO END
999

```

为验证以上计算结果,可以与参考文献中介绍的 Abramowitz 的 Handbook of Mathematical Function 一书中表 21.1 所示的精确结果进行比较. 在一般情况下,经过三次迭代即可收敛. 下面给出少量结果进行比较,如表 15.1 所示.

表 15.1 计算结果比较

SFROID 的部分结果比较				
m	n	c^2	λ_{exact}	λ_{SFROID}
2	2	0.1	6.01427	6.01427
		1.0	6.14095	6.14095
		4.0	6.54250	6.54253
2	5	1.0	30.4361	30.4372
		16.0	36.9963	37.0135
4	11	-1.0	131.560	131.554

第 16 章 偏微分方程的解法

自从数字电子计算机问世以来,偏微分方程的数值解法得到很大发展,因此,可以说偏微分方程数值解法本身就是一个庞大的题目.本章只介绍两种解椭圆型方程时常用而有效的算法:SOR 和 ADI.

16.1 解边值问题的松弛法

1. 功能

子过程 SOR 采用 Chebyshev 加速超松弛法求解离散后的偏微分方程.

$$a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} = f_{j,l} \quad (16-1)$$

2. 方法

(1) 边值问题的离散化.

求解椭圆型边值问题

$$\begin{cases} a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial^2 u}{\partial y^2} + c \frac{\partial u}{\partial x} + d \frac{\partial u}{\partial y} + gu = f, & (x, y) \in D \\ u = \varphi(x, y), & (x, y) \in \partial D \end{cases}$$

为简便起见,选用五点差分格式,则得形式为式(16-1)的有限差分方程

$$\begin{cases} a_{ij}^R u_{i+1,j} + a_{ij}^L u_{i-1,j} + a_{ij}^T u_{i,j+1} + a_{ij}^B u_{i,j-1} + a_{ij}^O u_{ij} = f_{ij}, & (x_i, y_j) \in D_h \\ u_{ij} = \varphi_{ij}, & (x_i, y_j) \in \partial D_h \end{cases}$$

其中

$$a_{ij}^L = \frac{a_{ij}}{h^2} - \frac{c_{ij}}{2h}, \quad a_{ij}^B = \frac{b_{ij}}{k^2} - \frac{d_{ij}}{2k}$$

$$a_{ij}^R = \frac{a_{ij}}{h^2} + \frac{c_{ij}}{2h}, \quad a_{ij}^T = \frac{b_{ij}}{k^2} + \frac{d_{ij}}{2k}$$

$$a_{ij}^O = a_{ij}^L + a_{ij}^R + a_{ij}^B + a_{ij}^T - g_{ij}$$

$$a_{ij} = a(x_i, y_j), \quad b_{ij} = b(x_i, y_j), \quad c_{ij} = c(x_i, y_j)$$

$$d_{ij} = d(x_i, y_j), \quad g_{ij} = g(x_i, y_j), \quad f_{ij} = f(x_i, y_j)$$

(2) 用超松弛法求解式(16-1).

将式(16-1)改写为 $Ax=b$, 将 A 分解为

$$A = L + D + U$$

其中 D 为 A 的对角线部分组成的对角矩阵, L 为由 A 的下三角部分且对角线上为零组成的下三角矩阵, U 为由 A 的上三角部分组成的上三角矩阵(对角线上为零), 则求解式(16-1)的超松弛迭代格式为

$$x^{(n)} = x^{(n-1)} - \omega(L + D)^{-1} \cdot \zeta^{(n-1)}$$

其中 ω 为超松弛因子,

$$\zeta^{(n-1)} = Ax^{(n-1)} - b$$

写成分量形式即为

$$u_{j,l}^{\text{new}} = u_{j,l}^{\text{old}} - \omega \frac{\zeta_{j,l}}{e_{j,l}}$$

其中

$$\zeta_{j,l} = a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} - f_{j,l}$$

为残量.

在 Chebyshev 加速超松弛法中, 采用奇偶次序, 即由 $j+l$ 为偶数或奇数将网格点分成偶数点或奇数点, 先计算奇数点上的值, 再利用刚算出的奇数点上的值计算偶数点上的值, 且松弛因子按以下方式改变:

$$\omega^{(0)} = 1$$

$$\omega^{(1/2)} = 1/(1 - \rho_{\text{jacobi}}^2/2)$$

$$\omega^{(n+1/2)} = 1/(1 - \rho_{\text{jacobi}}^2 \omega^{(n)}/4), \quad n = 1/2, 1, \dots$$

$$\omega^{(\infty)} \rightarrow \omega_{\text{optimal}}$$

这样选取的 ω 使得每次迭代的误差总在减小, 当迭代到总残量与初始残量的比很小时, 即当

$$\frac{\sum_{j=2}^{JMAX} \sum_{l=2}^{JMAX} |\zeta_{j,l}^{(N)}|}{\sum_{j=2}^{JMAX} \sum_{l=2}^{JMAX} |f_{j,l}|} < \varepsilon$$

时, 迭代停止, 此时的 $u_{j,l}$ 即为 $u(x, y)$ 在 (x_j, y_l) 点的近似值.

3. 使用说明

SOR(A,B,C,D,E,F,U,JMAX,RJAC)

JMAX 整型变量, 输入参数, 求解区域的最大网格点数(包含边界点, 在一个方向上)

A,B,C,D,E,F 均为含 JMAX×JMAX 个元素的实型数组, 输入参数, 存放方程组(16-1)的系数

- U 实型数组 $U(JMAX, JMAX)$, 输入、输出参数, 开始时存放对 U 的初始猜想值(通常赋以零), 调用后存放 $U(x, y)$ 在 D_h 上的近似值
- RJAC 实型变量, 输入参数, 存放雅可比迭代矩阵的谱半径或其估计值

4. 过程

子过程 SOR.

```

SUBROUTINE sor(a,b,c,d,e,f,u,jmax,rjac)
IMPLICIT real * 8(a-h,o-z)
DIMENSION a(jmax,jmax),b(jmax,jmax),c(jmax,jmax),&
d(jmax,jmax),e(jmax,jmax),f(jmax,jmax),u(jmax,jmax)
PARAMETER (maxits=1000,eps=1.d-5,zero=0.d0,half=.5d0,&
           qtr=0.25d0,one=1.d0)
REAL anormf,anorm,omega,rjac
INTEGER j,l,n,jmax
anormf=zero
do j=2,jmax-1
  do l=2,jmax-1
    anormf=anormf+abs(f(j,l))
  end do
end do
omega=one
do n=1,maxits
  anorm=zero
  do j=2,jmax-1
    do l=2,jmax-1
      if(mod(j+l,2)==mod(n,2)) then
        resid=a(j,l)*u(j+1,l)+b(j,l)*u(j-1,&
          l)+c(j,l)*u(j,l+1)+d(j,l)*u(j,&
          l-1)+e(j,l)*u(j,l)-f(j,l)
        anorm=anorm+abs(resid)
        u(j,l)=u(j,l)-omega*resid/e(j,l)
      endif
    end do
  end do
  if(n==1) then
    omega=one/(one-half*rjac**2)
  end if
end do

```

```

else
    omega=one/(one-qtr*rjac**2*omega)
endif
if((n>1).and.(anorm<eps*anormf)) return
end do
pause 'maxits exceeded'
END SUBROUTINE sor

```

5. 例子

验证程序 D16R1 中所采用的例子是模型问题

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho$$

该方程可以被处理为松弛问题

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \rho$$

采用 FTCS 差分格式, 得到

$$u_{j,l}^{n+1} = u_{j,l}^n + \frac{\Delta t}{\Delta^2} (u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^n) - \rho_{j,l} \Delta t$$

在一维坐标空间时, FTCS 差分格式的稳定性条件为 $\Delta t / \Delta^2 \leq 1/2$. 在二维的情况下, 就成为 $\Delta t / \Delta^2 \leq 1/4$. 那么, 假定将时间步长 Δt 取为可能的最大值, 即 $\Delta t = \Delta^2/4$, 那么上述方程变为

$$u_{j,l}^{n+1} = \frac{1}{4} (u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n) - \frac{\Delta^2}{4} \rho_{j,l}$$

或

$$u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^{n+1} = \rho_{j,l} \Delta^2$$

这是一般差分方程的一种简单形式. 应用于 SOR 中时, 对所有的 j 和 l 有

$$A_{jl} = B_{jl} = C_{jl} = D_{jl} = 1.0 \text{ 且 } E_{jl} = -4.0$$

u 的开始猜测对所有的 j, l 有 $u_{jl} = 0.0$. 方程式 (16-1) 右端的源汇项函数 F_{jl} , 除了在自变量 x, y 的区域中心定为 $F(\text{MIDL}, \text{MIDL}) = 2.0$ 外, 其余所有点取为 $F_{jl} = 0.0$. 谱半径 ρ_{Jacobi} 的值 (程序中称为 RJAC) 由下式确定

$$\rho_{\text{Jacobi}} = \frac{\cos \frac{\pi}{J} + \left(\frac{\Delta x}{\Delta y} \right)^2 \cos \frac{\pi}{L}}{1 + \left(\frac{\Delta x}{\Delta y} \right)^2}$$

在我们这个例子里, $J = L = \text{JMAX}$ 且 $\Delta x = \Delta y$, 因而 $\text{RJAC} = \cos(\pi/\text{JMAX})$. 经过调用 SOR 后, 其解被打印在计算结果中. 为了验证这个解, 验证程序 D16R1

将结果代回差分方程并计算

$$F_{j,l} = u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^{n+1}$$

计算结果应该是除了区域的中心点为 2.0 外,其余都是零.验证程序 D16R1 如下:

```

PROGRAM D16R1
  1 Driver for routine SOR
  PARAMETER (JMAX=11,PI=3.1415926)
  IMPLICIT REAL * 8(A-H,O-Z)
  DIMENSION A(JMAX,JMAX),B(JMAX,JMAX),C(JMAX,JMAX),&
  D(JMAX,JMAX),E(JMAX,JMAX),F(JMAX,JMAX),U(JMAX,JMAX)
  DO I=1,JMAX
    DO J=1,JMAX
      A(I,J)=1.0
      B(I,J)=1.0
      C(I,J)=1.0
      D(I,J)=1.0
      E(I,J)=-4.0
      F(I,J)=0.0
      U(I,J)=0.0
    END DO
  END DO
  MIDL=JMAX/2+1
  F(MIDL,MIDL)=2.0
  RJAC=COS(PI/JMAX)
  CALL SOR(A,B,C,D,E,F,U,JMAX,RJAC)
  WRITE(*,'(1X,A)') 'SOR Solution:'
  DO I=1,JMAX
    WRITE(*,'(1X,11F6.2)') (U(I,J), J=1,JMAX)
  END DO
  WRITE(*,'(/1X,A)') &
  'Test that solution satisfies Difference Eqns:'
  DO I=2,JMAX-1
    DO J=2,JMAX-1
      F(I,J)=U(I+1,J)+U(I-1,J)+U(I,J+1)&
      +U(I,J-1)-4.0*U(I,J)
    END DO
  END DO

```

```

      WRITE(*,'(7X,11F6.2)')(F(I,J), J=2,JMAX-1)
    END DO
  END

```

计算结果如下:

SOR Solution:

```

0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00 -0.02 -0.04 -0.06 -0.08 -0.09 -0.08 -0.06 -0.04 -0.02  0.00
0.00 -0.04 -0.09 -0.13 -0.17 -0.19 -0.17 -0.13 -0.09 -0.04  0.00
0.00 -0.06 -0.13 -0.20 -0.28 -0.32 -0.28 -0.20 -0.13 -0.06  0.00
0.00 -0.08 -0.17 -0.28 -0.41 -0.55 -0.41 -0.28 -0.17 -0.08  0.00
0.00 -0.09 -0.19 -0.32 -0.55 -1.05 -0.55 -0.32 -0.19 -0.09  0.00
0.00 -0.08 -0.17 -0.28 -0.41 -0.55 -0.41 -0.28 -0.17 -0.08  0.00
0.00 -0.06 -0.13 -0.20 -0.28 -0.32 -0.28 -0.20 -0.13 -0.06  0.00
0.00 -0.04 -0.09 -0.13 -0.17 -0.19 -0.17 -0.13 -0.09 -0.04  0.00
0.00 -0.02 -0.04 -0.06 -0.08 -0.09 -0.08 -0.06 -0.04 -0.02  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00

```

Test that solution satisfies Difference Eqns:

```

0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  2.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00

```

16.2 交替方向隐式方法(ADI)

1. 功能

本算法用于求解离散的椭圆型边值问题,选用可变的参数,在大多数情况下,它比超松弛法收敛快,也更有效.在区域的形状和边界条件允许的情况下,优于各种松弛法.尽管公认其程序编制稍困难且有时不收敛,但它仍被推荐为第一试用算法.

2. 方法

(1) 离散微分方程, 形成迭代格式.

如对 Poisson 方程 $L_u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$, 因为离散椭圆型方程的迭代格式可看成相应抛物型方程的差分格式, 故我们可对偏微分方程 $\frac{\partial u}{\partial x} = L_u - \rho$ 构造稳定的差分格式而得到 $L_u = \rho$ 的隐式迭代格式:

$$(L_x + rI)u^{n+1/2} = (rI - L_y)u^n - \Delta^2 \rho \quad (16-2)$$

$$(L_y + rI)u^{n+1} = (rI - L_x)u^{n+1/2} - \Delta^2 \rho \quad (16-3)$$

其中 n 为迭代步数, r 为加速参数, 矩阵 L_x 和 L_y 均为三对角矩阵. 对泊松方程

$$(L_x u)_{j,l} = 2u_{j,l} - u_{j+1,l} - u_{j-1,l} \quad (16-4)$$

$$(L_y u)_{j,l} = 2u_{j,l} - u_{j,l+1} - u_{j,l-1} \quad (16-5)$$

通常情况下为

$$(L_x u)_{j,l} = a_{j,l}u_{j-1,l} + b_{j,l}u_{j,l} + c_{j,l}u_{j+1,l} \quad (16-6)$$

$$(L_y u)_{j,l} = d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} + f_{j,l}u_{j,l+1} \quad (16-7)$$

ADI 的计算过程相当于先用 $u_{j,l}^n$ 沿水平方向逐线解三对角方程组式(16-2)得 $u_{j,l}^{n+1/2}$, 然后用 $u_{j,l}^{n+1/2}$ 沿垂直方向逐线解三对角方程组式(16-3)得 $u_{j,l}^{n+1}$, 两步合起来才算完成一次迭代.

(2) 加速参数 r 的选取.

我们选用 N 个 r_n 的序列进行迭代, 即每次迭代顺序选的参数 r 为

$$r_1, r_2, r_3, \dots, r_N, r_1, r_2, \dots, r_N, r_1, r_2, \dots$$

而 N 个不同的参数 $r_i (1 \leq i \leq N)$ 按如下方式计算:

设 L_x 及 L_y 的特征值的取值范围为 $[\alpha, \beta]$, 则

① 令 $\alpha_0 = \alpha, \beta_0 = \beta$.

② 计算 $\alpha_{j+1} = \sqrt{\alpha_j \beta_j}, \beta_{j+1} = \frac{\alpha_j + \beta_j}{2}, j = 0, 1, \dots, k-1$.

③ 令 $S_1^{(0)} = \sqrt{\alpha_k \beta_k}$.

对每个 $j, j = 0, 1, 2, \dots, k-1$, 重复计算 $S_n^{(j+1)} (n = 1, 2, \dots, 2^{j+1})$, 而使其为下面二次方程的解

$$S_n^{(j)} = \frac{1}{2} \left(x + \frac{\alpha_{k-1-j} \beta_{k-1-j}}{x} \right), \quad n = 1, 2, \dots, 2^j$$

④ 令 $r_n = s_n^{(k)}, n = 1, 2, \dots, N = 2^k$.

(3) 综合求解式(16-2)和(16-3)的步骤.

- ① 计算初始向量 $\psi; \psi = (rI - L_y)u^0$.
- ② 对 $n=0, 1, 2, \dots$ 解 $(L_x + rI)u^{n+1/2} = \psi - \rho\Delta^2$, 得到 $u^{n+1/2}$.
- ③ 令 $\phi = -\psi + 2ru^{n+1/2}$.
- ④ 解 $(L_x + rI)u^{n+1} = \phi$, 得到 u^{n+1} .

这样就完成了一次迭代,若此时解 u^{n+1} 已满足精度要求,迭代结束,否则转到第(2)步继续迭代.

3. 使用说明

ADI (A,B,C,D,E,F,G,U,JMAX,K,ALPHA,BETA,EPS)

JMAX 整型变量,输入参数,一个方向上网格点的最大数目

A,B,C,D,E,F 均为含 JMAX×JMAX 个元素的二维实型数组,输入参数,存放差分方程的系数(与式(16-6)和式(16-7)对应)

G 实型数组 G(JMAX,JMAX),输入参数,存放差分方程右端项 ρ_i ,

U 实型数组 U(JMAX,JMAX),输入、输出参数,开始时存放对 U 的初始猜想(一般常取零),调用后存放满足精度要求的解

K 整型变量,输入参数, $N(=2^k)$ 是用户选定的不同加速参数的个数

ALPHA 实型变量,输入参数,存放 L_x 和 L_y 的特征值范围的下限

BETA 实型变量,输入参数,存放 L_x 和 L_y 的特征值范围的上限

EPS 实型变量,输入参数,预先指定的误差限

4. 过程

子过程 ADI.

```
SUBROUTINE adi(a,b,c,d,e,f,g,u,jmax,k,alpha,beta,eps)
```

```
IMPLICIT REAL * 8(a-h,o-z)
```

```
PARAMETER (jj=100, kk=6, nrr=2 * (kk-1), maxits=100, &
```

```
zero=0. d0, two=2. d0, half=0. 5d0)
```

```
DIMENSION a(jmax,jmax), b(jmax,jmax), c(jmax,jmax), d(jmax,
```

```
ax,jmax), e(jmax,jmax), f(jmax,jmax), g(jmax,jmax), &
```

```
u(jmax,jmax), aa(jj), bb(jj), cc(jj), rr(jj), uu(jj), &
```

```
psi(jj,jj), alph(kk), bet(kk), r(nrr), s(nrr, kk)
```

```
INTEGER j, k1, k, n, kits, l, nr, nits, next, jmax
```

```
if(jmax>jj) pause 'increase jj'
```

```
if(k>kk-1) pause 'increase kk'
```



```

k1=k+1
nr=2 * * k
alph(1)=alpha
bet(1)=beta
do j=1,k
    alph(j+1)=sqrt(alph(j) * bet(j))
    bet(j+1)=half * (alph(j)+bet(j))
end do
s(1,1)=sqrt(alph(k1) * bet(k1))
do j=1,k
    ab=alph(k1-j) * bet(k1-j)
    do n=1,2 * * (j-1)
        disc=sqrt(s(n,j) * * 2-ab)
        s(2 * n,j+1)=s(n,j)+disc
        s(2 * n-1,j+1)=ab/s(2 * n,j+1)
    end do
end do
do n=1,nr
    r(n)=s(n,k1)
end do
anormg=zero
do j=2,jmax-1
    do l=2,jmax-1
        anormg=anormg+abs(g(j,l))
        psi(j,l)=-d(j,l) * u(j,l-1)+(r(1)-e(j,l)) * u(j,l)&
            -f(j,l) * u(j,l+1)
    end do
end do
nits=maxits/nr
do kits=1,nits
    do n=1,nr
        if(n==nr) then
            next=1
        else
            next=n+1
        endif
        rfact=r(n)+r(next)
    end do
end do

```

```

do l=2,jmax-1
  do j=2,jmax-1
    aa(j-1)=a(j,l)
    bb(j-1)=b(j,l)+r(n)
    cc(j-1)=c(j,l)
    rr(j-1)=psi(j,l)-g(j,l)
  end do
  call tridag(aa,bb,cc,rr,uu,jmax-2)
  do j=2,jmax-1
    psi(j,l)=-psi(j,l)+two*r(n)*uu(j-1)
  end do
end do
do j=2,jmax-1
  do l=2,jmax-1
    aa(l-1)=d(j,l)
    bb(l-1)=e(j,l)+r(n)
    cc(l-1)=f(j,l)
    rr(l-1)=psi(j,l)
  end do
  call tridag(aa,bb,cc,rr,uu,jmax-2)
  do l=2,jmax-1
    u(j,l)=uu(l-1)
    psi(j,l)=-psi(j,l)+rfact*uu(l-1)
  end do
end do
end do
anorm=zero
do j=2,jmax-1
  do l=2,jmax-1
    resid=a(j,l)*u(j-1,l)+(b(j,l)+e(j,l))*u(j,&
      l)+c(j,l)*u(j+1,l)+d(j,l)*u(j,l-1)&
      *u(j,l-1)+f(j,l)*u(j,l+1)+g(j,l)
    anorm=anorm+abs(resid)
  end do
end do
if(anorm<eps*anormg) return
end do

```

```

      pause 'maxits exceeded'
      END SUBROUTINE adi

```

5. 例子

验证程序 D16R2 采用 16.1 节中同样的模型问题. 当模型方程被分裂成方程式(16-6)和(16-7)的形式时, 系数数组成为(见方程式(16-4)和(16-5))

$$A_{jt} = C_{jt} = D_{jt} = F_{jt} = -1.0$$

$$B_{jt} = E_{jt} = 2.0$$

试验解在每一网格点仍被设定为零, 且模型方程右端的源汇项仍设定为零(除了在区域中心点以外). 对模型方程(在一正方形区域上具有狄利克雷边界条件的泊松方程), 其特征值的界限为

$$\text{ALPHA} = 2 \left[1 - \cos \left(\frac{\pi}{\text{JMAX}} \right) \right]$$

$$\text{BETA} = 2 \left[1 - \cos \left(\frac{(\text{JMAX} - 1)\pi}{\text{JMAX}} \right) \right]$$

其中 $\text{JMAX} \times \text{JMAX}$ 是区域网格数. 迭代的次数 2^k 被缩到最小, 选定为 $\ln(4\text{JMAX}/\pi)$. 验证程序 D16R2 中调用 ADI 后, 即打印出计算结果. 可以和 16.1 节中调用 SOR 后的计算结果进行比较. 为了验证结果, 我们还可以将计算出来的解代入差分方程

$$G_{j,l} = u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^{n+1}$$

其结果应该是在所有网格点上均为零(除区域中心点的值为 2.0 外). 注意: ADI 要调用追赶法 TRIDAG 的双精度版本, 为此我们将双精度的 TRIDAG 附在验证程序 D16R2 的后面. 验证程序 D16R2 如下:

```

PROGRAM D16R2
  ! Driver for routine ADI
  PARAMETER (JMAX=11,PI=3.1415926)
  IMPLICIT REAL * 8(A-H,O-Z)
  DIMENSION A(JMAX,JMAX),B(JMAX,JMAX),C(JMAX,JMAX),&
    D(JMAX,JMAX),E(JMAX,JMAX),F(JMAX,JMAX),&
    G(JMAX,JMAX),U(JMAX,JMAX)
  DO I=1,JMAX
    DO J=1,JMAX
      A(I,J)=-1.0
      B(I,J)=2.0
      C(I,J)=-1.0

```

```

      D(I,J)=-1.0
      E(I,J)=2.0
      F(I,J)=-1.0
      G(I,J)=0.0
      U(I,J)=0.0
    END DO
  END DO
  MID=JMAX/2+1
  G(MID,MID)=2.0
  ALPHA=2.0*(1.0-COS(PI/JMAX))
  BETA=2.0*(1.0-COS((JMAX-1)*PI/JMAX))
  ALIM=LOG(4.0*JMAX/PI)
  K=0
  DO
    K=K+1
    IF(2**K>=ALIM) EXIT
  END DO
  EPS=1.0E-4
  CALL ADI(A,B,C,D,E,F,G,U,JMAX,K,ALPHA,BETA,EPS)
  WRITE(*, '(1X,A)') 'ADI Solution:'
  DO I=1,JMAX
    WRITE(*, '(1X,11F6.2)') (U(I,J), J=1,JMAX)
  END DO
  WRITE(*, '(/1X,A)') &
    'Test that solution satisfies Difference Eqns:'
  DO I=2,JMAX-1
    DO J=2,JMAX-1
      G(I,J)=-4.0*U(I,J)+U(I+1,J)&
        +U(I-1,J)+U(I,J-1)+U(I,J+1)
    END DO
    WRITE(*, '(2X,9F7.2)') (G(I,J), J=2,JMAX-1)
  END DO
END PROGRAM

SUBROUTINE TRIDAG(A,B,C,R,U,N)
! This is a double precision version for use with ADI
IMPLICIT REAL * 8(A-H,O-Z)
PARAMETER (NMAX=100)

```

```

DIMENSION GAM(NMAX),A(N),B(N),C(N),R(N),U(N)
IF(B(1)==0.) PAUSE
BET=B(1)
U(1)=R(1)/BET
DO J=2,N
    GAM(J)=C(J-1)/BET
    BET=B(J)-A(J)*GAM(J)
    IF(BET==0.) PAUSE
    U(J)=(R(J)-A(J)*U(J-1))/BET
END DO
DO J=N-1,1,-1
    U(J)=U(J)-GAM(J+1)*U(J+1)
END DO
END SUBROUTINE TRIDAG

```

计算结果如下:

ADI Solution:

[illegible]

Test that solution satisfies Difference Eqns:

[illegible]