

目 录

第一篇 数值计算

第1章 线性代数方程组的求解 2

- 1.1 全选主元高斯消去法 2
- 1.2 全选主元高斯-约当消去法 3
- 1.3 复系数方程组的全选主元高斯消去法 5
- 1.4 复系数方程组的全选主元高斯-约当消去法 7
- 1.5 求解三对角线方程组的追赶法 9
- 1.6 一般带型方程组的求解 11
- 1.7 求解对称方程组的分解法 15
- 1.8 求解对称正定方程组的平方根法 17
- 1.9 求解大型稀疏方程组的全选主元高斯-约当消去法 19
- 1.10 求解托伯利兹方程组的列文逊方法 20
- 1.11 高斯-赛德尔迭代法 24
- 1.12 求解对称正定方程组的共轭梯度法 25
- 1.13 求解线性最小二乘问题的豪斯荷尔德变换法 27
- 1.14 求解线性最小二乘问题的广义逆法 29
- 1.15 病态方程组的求解 31

第2章 矩阵运算 34

- 2.1 实矩阵相乘 34
- 2.2 复矩阵相乘 35
- 2.3 实矩阵求逆的全选主元高斯-约当法 37
- 2.4 复矩阵求逆的全选主元高斯-约当法 39
- 2.5 对称正定矩阵的求逆 42
- 2.6 托伯利兹矩阵求逆的特兰持方法 44
- 2.7 求行列式值的全选主元高斯消去法 47

第3章 矩阵特征值与特征向量的计算 64

- 3.1 约化对称矩阵为对称三对角阵的豪斯荷尔德变换法 64
- 3.2 实对称三对角阵的全部特征值与特征向量的计算 67
- 3.3 约化一般实矩阵为赫申伯格矩阵的初等相似变换法 69
- 3.4 求赫申伯格矩阵全部特征值的 QR 方法 70
- 3.5 求实对称矩阵特征值与特征向量的雅可比法 74
- 3.6 求实对称矩阵特征值与特征向量的雅可比过法 77

第4章 非线性方程与方程组的求解 79

- 4.1 求非线性方程实根的对分法 79
- 4.2 求非线性方程一个实根的牛顿法 80
- 4.3 求非线性方程一个实根的埃特金迭代法 82
- 4.4 求非线性方程一个实根的连分式解法 83
- 4.5 求实系数代数方程全部根的 QR 方法 85
- 4.6 求实系数代数方程全部根的牛顿-下山法 87
- 4.7 求复系数代数方程全部根的牛顿-下山法 89

4.8 求非线性方程组一组实根的梯度法 90

4.9 求非线性方程组一组实根的拟牛顿法 93

4.10 求非线性方程组最小二乘解的广义逆法 96

4.11 求非线性方程一个实根的蒙特卡洛法 100

4.12 求实函数或复函数方程一个复根的蒙特卡洛法 102

4.13 求非线性方程组一组实根的蒙特卡洛法 104

第5章 插值 107

- 5.1 一元全区间不等距插值 107
- 5.2 一元全区间等距插值 108
- 5.3 一元三点不等距插值 110
- 5.4 一元三点等距插值 111
- 5.5 连分式不等距插值 112
- 5.6 连分式等距插值 114
- 5.7 埃尔米特不等距插值 115
- 5.8 埃尔米特等距插值 117
- 5.9 埃特金不等距逐步插值 118
- 5.10 埃特金等距逐步插值 120
- 5.11 光滑不等距插值 121
- 5.12 光滑等距插值 123
- 5.13 第一种边界条件的三次样条函数插值、微商与积分 125
- 5.14 第二种边界条件的三次样条函数插值、微商与积分 128
- 5.15 第三种边界条件的三次样条函数插值、微商与积分 131
- 5.16 二元三点插值 135
- 5.17 二元全区间插值 136

第6章 数值积分 139

- 6.1 变步长梯形求积法 139
- 6.2 变步长辛卜生求积法 140
- 6.3 自适应梯形求积法 142
- 6.4 龙贝格求积法 143
- 6.5 计算一维积分的连分式法 145
- 6.6 高振荡函数求积法 147
- 6.7 勒让德-高斯求积法 150
- 6.8 拉盖尔-高斯求积法 152
- 6.9 埃尔米特-高斯求积法 154

6.10 切比雪夫求积法 156

6.11 计算一维积分的蒙特卡洛法 157

6.12 变步长辛卜生二重积分法 158

6.13 计算多重积分的高斯方法 161

6.14 计算二重积分的连分式法 163

6.15 计算多重积分的蒙特卡洛法 165

第7章 常微分方程(组)的求解 167

- 7.1 全区间积分的定步长欧拉方法 167
- 7.2 积分一步的变步长欧拉方法 169
- 7.3 全区间积分的定步长维梯方法 172
- 7.4 全区间积分的定步长龙格-库塔法 174
- 7.5 积分一步的变步长龙格-库塔法 177
- 7.6 积分一步的变步长基尔方法 179
- 7.7 全区间积分的变步长基尔方法 182
- 7.8 全区间积分的变步长默森方法 184
- 7.9 积分一步的连分式法 187
- 7.10 全区间积分的连分式法 190
- 7.11 全区间积分的双边法 192
- 7.12 全区间积分的阿当姆斯预报-校正法 195
- 7.13 全区间积分的哈明方法 197
- 7.14 积分一步的特雷纳方法 200
- 7.15 全区间积分的特雷纳方法 204
- 7.16 积分刚性方程组的吉尔方法 206
- 7.17 二阶微分方程边值问题的数值解法 213

第8章 拟合与逼近 217

- 8.1 最小二乘曲线拟合 217
- 8.2 切比雪夫曲线拟合 220
- 8.3 最佳一致逼近的里米兹方法 222
- 8.4 矩形域的最小二乘曲面拟合 225

第9章 数据处理与回归分析 229

- 9.1 随机样本分析 229
- 9.2 一元线性回归分析 231
- 9.3 多元线性回归分析 233
- 9.4 逐步回归分析 236
- 9.5 半对数数据相关 245
- 9.6 对数数据相关 247

第10章 极值问题 250

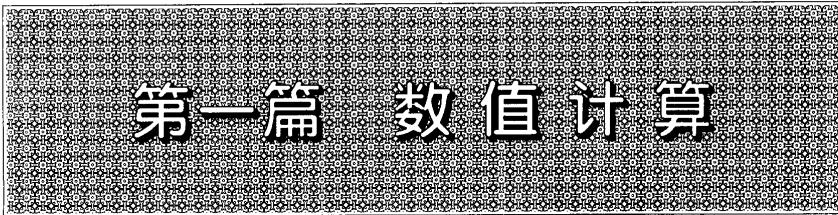
- 10.1 一维极值连分式法 250
- 10.2 n 维极值连分式法 252

10.3 不等式约束线性规划问题	254	机数	317	16.14 实数堆排序	367	18.9 TVGA 图形模式(5FH, 1024×768, 16 色)	420
10.4 求 n 维极值的单形调优法	258	13.2 0 到 1 之间均匀分布的随机数序列	318	16.15 字符堆排序	369	18.10 TVGA 图形模式(62H, 1024×768, 256 色)	421
10.5 求约束条件下 n 维极值的复形调优法	261	13.3 任意区间内均匀分布的一个随机整数	319	16.16 字符串堆排序	370	第 19 章 基本图形操作	424
第 11 章 数学变换与滤波	266	13.4 任意区间内均匀分布的随机整数序列	320	16.17 关键字成员为整数的结构排序	371	19.1 直线	424
11.1 傅里叶级数逼近	266	13.5 任意均值与方差的一个正态分布随机数	321	16.18 关键字成员为实数的结构排序	374	19.2 线段构成的图形	425
11.2 快速傅里叶变换	268	13.6 任意均值与方差的正态分布随机数序列	322	16.19 关键字成员为字符的结构排序	376	19.3 虚线	426
11.3 快速沃尔变换	273			16.20 关键字成员为字符串的结构排序	378	19.4 单点划线	428
11.4 五点三次平滑	275			16.21 磁盘文件排序	380	19.5 双点划线	430
11.5 离散随机线性系统的卡尔曼滤波	277			16.22 拓扑分类	383	19.6 坐标轴	431
11.6 $\alpha\beta\gamma$ 滤波	281					19.7 矩形及其填充	432
第 12 章 特殊函数	285					19.8 矩形域图形的清除	433
12.1 伽马函数	285	14.1 一维多项式求值	324			19.9 矩形域图形的复制	434
12.2 不完全伽马函数	287	14.2 一维多项式多组求值	325			19.10 矩形域图形的平移	436
12.3 误差函数	288	14.3 二维多项式求值	327			19.11 圆形域图形的复制	437
12.4 第一类整数阶贝塞耳函数	290	14.4 复系数多项式求值	329			19.12 圆形域图形的平移	438
12.5 第二类整数阶贝塞耳函数	293	14.5 多项式相乘	330			19.13 由中心、半轴(半径)以及起终点 夹角画椭圆(圆)弧	439
12.6 变型第一类整数阶贝塞耳函数	296	14.6 多项式相除	332			19.14 由中心、半轴(半径)以及起终点 夹角画扇形	441
12.7 变型第二类整数阶贝塞耳函数	299	14.7 复系数多项式相乘	333			19.15 由中心与半轴(半径)画椭圆 (圆)	442
12.8 不完全贝塔函数	301	14.8 复系数多项式相除	335			19.16 由中心、半轴(半径)以及起终点 夹角画扇形填充	443
12.9 正态分布函数	304	14.9 函数连分式的计算	336			19.17 由中心与半轴(半径)画椭圆(圆) 填充	444
12.10 t -分布函数	305					19.18 抛物线	445
12.11 χ^2 分布函数	307					19.19 双曲线	447
12.12 F 分布函数	308					19.20 三次多项式曲线	449
12.13 正弦积分	309					19.21 一般函数曲线	451
12.14 余弦积分	311					19.22 矩形域三维图形透视图	452
12.15 指数积分	312					19.23 矩形域三维图形平行投影	454
12.16 第一类椭圆积分	313					19.24 圆形域三维图形平行投影	456
12.17 第二类椭圆积分	315					第 20 章 汉字操作	458
第 13 章 随机数的产生	317					20.1 小汉字库的建立	458
13.1 0 到 1 之间均匀分布的一个随机数	317					20.2 ASCII 码字符图形库的建立	461
						20.3 一个汉字的显示	462
						20.4 一行汉字的显示	463
						20.5 ASCII 码字符串按图形显示	464
						20.6 汉字菜单的显示	466
						20.7 汉字菜单的选择	467

第二篇 非数值计算

第 16 章 排序	350	16.7 字符快速排序	358
16.1 整数冒泡排序	350	16.8 字符串快速排序	359
16.2 实数冒泡排序	351	16.9 整数希尔排序	360
16.3 字符冒泡排序	353	16.10 实数希尔排序	362
16.4 字符串冒泡排序	354	16.11 字符希尔排序	364
16.5 整数快速排序	355	16.12 字符串希尔排序	365
16.6 实数快速排序	357	16.13 整数堆排序	366

第1章 线性代数方程组的求解



1.1 全选主元高斯消去法

一、功能

用全选主元高斯(Gauss)消去法求解 n 阶线性代数方程组 $AX = B$ 。其中

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0, n-1} \\ a_{10} & a_{11} & \cdots & a_{1, n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1, 0} & a_{n-1, 1} & \cdots & a_{n-1, n-1} \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, B = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

二、方法说明

为保证数值计算的稳定性,本函数采用全选主元。

全选主元高斯消去法分两步进行。

第一步 消去过程

对于 k 从 0 到 $n - 2$ 作以下三步:

(1) 从系数矩阵 A 的第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素,并通过行交换与列交换将它交换到主元素的位置上。

(2) 归一化

$$\begin{aligned} a_{kj}/a_{kk} &\Rightarrow a_{kj}, j = k + 1, \dots, n - 1 \\ b_k/a_{kk} &\Rightarrow b_k \end{aligned}$$

(3) 消去

$$a_{ij} - a_{ik}a_{kj} \Rightarrow a_{ij}, i, j = k + 1, \dots, n - 1$$

$$b_i - a_{ik}b_k \Rightarrow b_i, i = k + 1, \dots, n - 1$$

第二步 回代过程

$$(1) b_{n-1}/a_{n-1, n-1} \Rightarrow x_{n-1}$$

$$(2) b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j \Rightarrow x_i, i = n - 2, \dots, 1, 0$$

最后对解向量中的元素顺序进行调整。

三、函数语句

int agaus (a, b, n)

本函数返回一个整型标志值。若返回的标志值为 0, 则表示原方程组的系数矩阵奇异,

输出信息“fail”；若返回的标志值不为 0，则表示正常返回。

四、形参说明

a——双精度实型二维数组，体积为 $n \times n$ 。存放方程组的系数矩阵，返回时将被破坏。
b——双精度实型一维数组，长度为 n 。存放方程组右端的常数向量；返回方程组的解向量。
n——整型变量。存放方程组的阶数。

五、函数程序(文件名：agaus.c)

六、例

求解下列 4 阶方程组

$$\begin{cases} 0.2368x_0 + 0.2471x_1 + 0.2568x_2 + 1.2671x_3 = 1.8471 \\ 0.1968x_0 + 0.2071x_1 + 1.2168x_2 + 0.2271x_3 = 1.7471 \\ 0.1581x_0 + 1.1675x_1 + 0.1768x_2 + 0.1871x_3 = 1.6471 \\ 1.1161x_0 + 0.1254x_1 + 0.1397x_2 + 0.1490x_3 = 1.5471 \end{cases}$$

主函数程序(文件名：agaus.c)如下：

```
# include "stdio.h"
# include "agaus.c"
main()
| int i;
static double a[4][4] =
    {0.2368, 0.2471, 0.2568, 1.2671|,
     0.1968, 0.2071, 1.2168, 0.2271|,
     0.1581, 1.1675, 0.1768, 0.1871|,
     1.1161, 0.1254, 0.1397, 0.1490|};
static double b[4] = {1.8471, 1.7471, 1.6471, 1.5471};
if (agaus(a, b, 4) != 0)
    for (i=0; i<=3; i++)
        printf ("x(%d) = %e\n", i, b[i]);
|
```

运行结果为：

```
x(0) = 1.04058e+00
x(1) = 9.87051e-01
x(2) = 9.35040e-01
x(3) = 8.81282e-01
```

1.2 全选主元高斯-约当消去法

一、功能

用全选主元高斯-约当(Gauss-Jordan)消去法同时求解系数矩阵相同而右端具有 m 组常数向量的 n 阶线性代数方程组 $AX = B$ 。其中

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0, n-1} \\ a_{10} & a_{11} & \cdots & a_{1, n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1, 0} & a_{n-1, 1} & \cdots & a_{n-1, n-1} \end{bmatrix}, X = \begin{bmatrix} x_{00} & x_{01} & \cdots & x_{0, m-1} \\ x_{10} & x_{11} & \cdots & x_{1, m-1} \\ \vdots & \vdots & & \vdots \\ x_{n-1, 0} & x_{n-1, 1} & \cdots & x_{n-1, m-1} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0, m-1} \\ b_{10} & b_{11} & \cdots & b_{1, m-1} \\ \vdots & \vdots & & \vdots \\ b_{n-1, 0} & b_{n-1, 1} & \cdots & b_{n-1, m-1} \end{bmatrix}$$

二、方法说明

全选主元高斯-约当法的步骤如下。

对于 k 从 0 到 $n-1$ 作如下变换：

(1) 全选主元

从系数矩阵 A 的第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素，并通过行交换与列交换将它交换到主元素的位置上。

(2) 归一化

$$\begin{aligned} a_{kj}/a_{kk} &\Rightarrow a_{kj}, j = k+1, \dots, n-1 \\ b_{kj}/a_{kk} &\Rightarrow b_{kj}, j = 0, 1, \dots, m-1 \end{aligned}$$

(3) 消去

$$\begin{aligned} a_{ij} - a_{ik}a_{kj} &\Rightarrow a_{ij}, j = k+1, \dots, n-1 \\ i = 0, 1, \dots, n-1, i \neq k \\ b_{ij} - a_{ik}b_{kj} &\Rightarrow b_{ij}, j = 0, 1, \dots, m-1 \\ i = 0, 1, \dots, n-1, i \neq k \end{aligned}$$

最后对 B 进行恢复，则 B 中的每一列即为每一个方程组的解。

三、函数语句

```
int agjdn (a, b, n, m)
```

本函数返回一个整型标志值。若返回的标志值为 0，则表示系数矩阵为奇异，输出信息“fail”；若返回的标志值不为 0，则表示正常返回。

四、形参说明

a——双精度实型二维数组，体积为 $n \times n$ 。存放方程组的系数矩阵，在本函数中要被破坏。

b——双精度实型二维数组，体积为 $n \times m$ 。存放方程组右端的 m 组常数向量(一列为一组)；返回时存放 m 组解向量(一列为一组)。

n——整型变量。方程组的阶数。

m——整型变量。方程组右端常数向量的个数。

五、函数程序(文件名: agjdn.c)

六、例

求解 4 阶方程组 $AX = B$, 其中

$$A = \begin{bmatrix} 1 & 3 & 2 & 13 \\ 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -2 & 11 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 9 & 0 \\ 6 & 4 \\ 11 & 7 \\ -2 & -1 \end{bmatrix}$$

主函数程序(文件名: agjdn 0.c)如下:

```
#include "stdio.h"
#include "agjdn.c"

main()
{
    int i;
    static double a[4][4] = { {1.0, 3.0, 2.0, 13.0},
                             {7.0, 2.0, 1.0, -2.0},
                             {9.0, 15.0, 3.0, -2.0},
                             {-2.0, -2.0, 11.0, 5.0} };
    static double b[4][2] = { {9.0, 0.0}, {6.0, 4.0},
                            {11.0, 7.0}, {-2.0, -1.0} };
    if (agjdn(a, b, 4, 2) != 0)
        for (i = 0; i <= 3; i++)
            printf("x(%d) = %13.7e, %13.7e\n", i, b[i][0], b[i][1]);
}
```

运行结果为:

```
x(0) = 9.807447e-01,      4.979313e-01
x(1) = 2.679822e-01,      1.444940e-01
x(2) = -2.226289e-01,     6.285805e-02
x(3) = 5.892743e-01,      -8.131763e-02
```

1.3 复系数方程组的全选主元高斯消去法

一、功能

用全选主元高斯(Gauss)消去法求解 n 阶复系数线性代数方程组 $AX = B$ 。其中

$$\begin{aligned} A &= AR + jAI \\ B &= BR + jBI \\ X &= XR + jXI \end{aligned}$$

二、方法说明

同 1.1 节的说明, 但所有运算为复数运算。

为了计算两个复数的乘积 $e + jf = (a + jb)(c + jd)$, 本函数采用如下算法:

$$p = ac, q = bd, s = (a + b)(c + d)$$

$$e = p - q, f = s - p - q$$

为了计算两个复数的商 $e + jf = (c + jd)/(a + jb)$, 本函数采用如下算法:

$$p = ac, q = -bd, s = (a - b)(c + d), w = a^2 + b^2$$

$$e = (p - q)/w, f = (s - p - q)/w$$

三、函数语句

int acgas(ar, ai, n, br, bi)

本函数返回一个整型标志值。若返回的标志值为 0, 则表示系数矩阵奇异, 输出信息“err * * fail”; 若返回的标志值不为 0, 则表示正常返回。

四、形参说明

ar——双精度实型二维数组, 体积为 $n \times n$ 。存放复系数矩阵的实部, 在本函数中要被破坏。

ai——双精度实型二维数组, 体积为 $n \times n$ 。存放复系数矩阵的虚部, 在本函数中要被破坏。

n——整型变量。方程组的阶数。

br——双精度实型一维数组, 长度为 n 。存放方程组右端复常数向量的实部; 返回时存放方程组解向量的实部。

bi——双精度实型一维数组, 长度为 n 。存放方程组右端复常数向量的虚部; 返回时存放方程组解向量的虚部。

五、函数程序(文件名: acgas.c)

六、例

求解 4 阶复系数方程组 $AX = B$ 。其中

$$A = AR + jAI, \quad B = BR + jBI$$

$$AR = \begin{bmatrix} 1 & 3 & 2 & 13 \\ 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -2 & 11 & 5 \end{bmatrix}, \quad AI = \begin{bmatrix} 3 & -2 & 1 & 6 \\ -2 & 7 & 5 & 8 \\ 9 & -3 & 15 & 1 \\ -2 & -2 & 7 & 6 \end{bmatrix}$$

$$BR = (2, 7, 3, 9)^T, \quad BI = (1, 2, -2, 3)^T$$

主函数程序(文件名: acgas 0.c)如下:

```
#include "stdio.h"
#include "acgas.c"

main()
{
    int i;
    static double ar[4][4] = { {1.0, 3.0, 2.0, 13.0},
                             {7.0, 2.0, 1.0, -2.0},
                             {9.0, 15.0, 3.0, -2.0},
                             {-2.0, -2.0, 11.0, 5.0} };
    static double ai[4][4] = { {3.0, -2.0, 1.0, 6.0},
                             {-2.0, 7.0, 5.0, 8.0} };
}
```

```

    {9.0, -3.0, 15.0, 1.0},
    {-2.0, -2.0, 7.0, 6.0}};

static double br[4] = {2.0, 7.0, 3.0, 9.0};
static double bi[4] = {1.0, 2.0, -2.0, 3.0};
if (acgas(ar, ai, 4, br, bi) != 0)
    for (i=0; i<=3; i++)
        printf("b(%d) = %13.7e + j %13.7e\n", i, br[i], bi[i]);
}

```

运行结果为：

```

b(0) = 6.782330e-02 + j 7.078231e-02
b(1) = -1.623413e-01 + j -7.612936e-01
b(2) = 5.985239e-01 + j -4.371312e-01
b(3) = 2.464562e-01 + j 1.139959e-01

```

1.4 复系数方程组的全选主元高斯-约当消去法

一、功能

用全选主元高斯-约当(Gauss-Jordan)消去法求解右端具有 m 组常数向量的 n 阶复系数线性代数方程组 $AX = B$ 。其中

$$\begin{aligned} A &= AR + jAI \\ B &= BR + jBI \\ X &= XR + jXI \end{aligned}$$

二、方法说明

同 1.2 节的说明, 但所有运算均为复数运算。

为了计算两个复数的乘积 $e + jf = (a + jb)(c + jd)$, 本函数采用如下算法:

$$\begin{aligned} p &= ac, q = bd, s = (a + b)(c + d) \\ e &= p - q, f = s - p - q \end{aligned}$$

为了计算两个复数的商 $e + jf = (c + jd)/(a + jb)$, 本函数采用如下算法:

$$\begin{aligned} p &= ac, q = -bd, s = (a - b)(c + d), w = a^2 + b^2 \\ e &= (p - q)/w, f = (s - p - q)/w \end{aligned}$$

三、函数语句

```
int acjdn(ar, ai, br, bi, n, m)
```

本函数返回一个整型标志值。若返回的标志值为 0, 则表示系数矩阵奇异, 输出信息“err * * fail”; 若返回的标志值不为 0, 则表示正常返回。

四、形参说明

ar——双精度实型二维数组, 体积为 $n \times n$ 。存放复系数矩阵的实部, 在本函数中要被破坏。

ai——双精度实型二维数组, 体积为 $n \times n$ 。存放复系数矩阵的虚部, 在本函数中要被破坏。

br——双精度实型二维数组, 体积为 $n \times m$ 。存放方程组右端 m 组复常数向量的实部; 返回时存放 m 组解向量的实部。

bi——双精度实型二维数组, 体积为 $n \times m$ 。存放方程组右端 m 组复常数向量的虚部; 返回时存放 m 组解向量的虚部。

n——整型变量, 方程组的阶数。

m——整型变量。方程组右端复常数向量的个数。

五、函数程序(文件名: acjdn.c)

六、例

求解 4 阶复系数方程组 $AX = B$ 。其中

$$\begin{aligned} A &= AR + jAI, B = BR + jBI \\ AR &= \begin{bmatrix} 1 & 3 & 2 & 13 \\ 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -2 & 11 & 5 \end{bmatrix}, AI = \begin{bmatrix} 3 & -2 & 1 & 6 \\ -2 & 7 & 5 & 8 \\ 9 & -3 & 15 & 1 \\ -2 & -2 & 7 & 6 \end{bmatrix} \\ BR &= \begin{bmatrix} 2 & -2 \\ 7 & 3 \\ 3 & 2 \\ 9 & 1 \end{bmatrix}, BI = \begin{bmatrix} 1 & 3 \\ 2 & 7 \\ -2 & 9 \\ 3 & 2 \end{bmatrix} \end{aligned}$$

主函数程序(文件名: acjdn 0.c)如下:

```

#include "stdio.h"
#include "acjdn.c"
main()
{
    int i;
    static double ar[4][4] = { {1.0, 3.0, 2.0, 13.0},
                               {7.0, 2.0, 1.0, -2.0},
                               {9.0, 15.0, 3.0, -2.0},
                               {-2.0, -2.0, 11.0, 5.0} };
    static double ai[4][4] = { {3.0, -2.0, 1.0, 6.0},
                               {-2.0, 7.0, 5.0, 8.0},
                               {9.0, -3.0, 15.0, 1.0},
                               {-2.0, -2.0, 7.0, 6.0} };
    static double br[4][2] = { {2.0, -2.0}, {7.0, 3.0},
                               {3.0, 2.0}, {9.0, 1.0} };
    static double bi[4][2] = { {1.0, 3.0}, {2.0, 7.0},
                               {-2.0, 9.0}, {3.0, 2.0} };
    if (acjdn(ar, ai, br, bi, 4, 2) != 0)
        for (i=0; i<=3; i++)
            printf("x(%d) = %13.7e + j %13.7e, %13.7e + j %13.7e\n",
                   i, br[i][0], bi[i][0], br[i][1], bi[i][1]);
}

```

运行结果为：

```
x(0) = -6.782330e-02 + j 7.078231e-02, 2.511789e-01 + j 5.810123e-01
x(1) = -1.623413e-01 + j -7.612936e-01, 4.024521e-01 + j -1.436012e-01
x(2) = 5.985239e-01 + j -4.371312e-01, 3.356590e-01 + j 1.034377e-01
x(3) = 2.464562e-01 + j 1.139959e-01, -5.755388e-02 + j 2.079959e-01
```

1.5 求解三对角线方程组的追赶法

一、功能

用追赶法求解 n 阶三对角线方程组 $AX = D$ 。其中

$$A = \begin{bmatrix} a_{00} & a_{01} & & & \\ a_{10} & a_{11} & a_{12} & & 0 \\ & a_{21} & a_{22} & a_{23} & \\ & \ddots & \ddots & \ddots & \\ 0 & a_{n-2, n-3} & a_{n-2, n-2} & a_{n-2, n-1} & \\ & & a_{n-1, n-2} & a_{n-1, n-1} & \end{bmatrix}$$
$$X = (x_0, x_1, x_2, \dots, x_{n-2}, x_{n-1})^T$$
$$D = (d_0, d_1, d_2, \dots, d_{n-2}, d_{n-1})^T$$

二、方法说明

追赶法的本质是没有选主元的高斯(Gauss)消去法，只是在计算过程中考虑了三对角线矩阵的特点，对于绝大部分的零元素不再作处理。其步骤如下。

(1) 对于 $k = 0, 1, \dots, n-2$ 作归一化及消元处理。即

$$\begin{aligned} a_{k, k+1} / a_{kk} &\Rightarrow a_{k, k+1} \\ d_k / a_{kk} &\Rightarrow d_k \\ a_{k+1, k+1} - a_{k+1, k} a_{k, k+1} &\Rightarrow a_{k+1, k+1} \\ d_{k+1} - a_{k+1, k} d_k &\Rightarrow d_{k+1} \end{aligned}$$

(2) 进行回代。即

$$\begin{aligned} d_{n-1} / a_{n-1, n-1} &\Rightarrow x_{n-1} \\ d_k - a_{k, k+1} x_{k+1} &\Rightarrow x_k, k = n-2, \dots, 1, 0 \end{aligned}$$

在本函数中，为便于三对角线矩阵的输入，用一个一维数组 $B(0:3n-3)$ 以行为主存放三对角线矩阵 A 中的三条对角线上的非零元素。其中 n 为矩阵 A 的阶数。显然，三对角线矩阵 A 与一维数组 B 之间有如下关系：

$$A(i, j) = \begin{cases} B[2i + j], & i-1 \leq j \leq i+1 \\ 0, & \text{其它} \end{cases}$$

其中 $i, j = 0, 1, \dots, n-1$ 。

根据以上关系，可以得到相应的计算过程为：

(1) 对于 $k = 0, 1, \dots, n-2$ 作归一化及消元处理

$$B(3k+1)/B(3k) \Rightarrow B(3k+1)$$

$$d_k/B(3k) \Rightarrow d_k$$

$$B(3k+3) - B(3k+2)B(3k+1) \Rightarrow B(3k+3)$$

$$d_{k+1} - B(3k+2)d_k \Rightarrow d_{k+1}$$

(2) 进行回代

$$d_{n-1}/B(3n-3) \Rightarrow d_{n-1}$$

$$d_k - B(3k+1)d_{k+1} \Rightarrow d_k, k = n-2, \dots, 1, 0$$

最后的计算结果仍放在一维数组 D 中。

由于追赶法本质上是没有选主元的高斯消去法，因此，只有当三对角矩阵满足下列条件：

$$|a_{00}| > |a_{01}|$$

$$|a_{ii}| \geq |a_{i, i-1}| + |a_{i, i+1}|, i = 1, 2, \dots, n-2$$

$$|a_{n-1, n-1}| > |a_{n-1, n-2}|$$

时，追赶法的计算过程才不会出现中间结果数量级的巨大增长和舍入误差的严重积累。

三、函数语句

```
int atrde(b, n, m, d)
```

本函数返回一个整型标志值。若返回的标志值小于 0，则表示 m 的值不正确，输出信息“err”；若返回的标志值为 0，则表示程序工作失败（如主元素的绝对值太小或为 0），在本函数中还输出信息“fail”；若返回的标志值大于 0，则表示正常返回。

四、形参说明

b ——双精度实型一维数组，长度为 m 。以行为主存放三对角线矩阵中三对角线上的元素，即在 b 中依次存放元素：

$$a_{00}, a_{01}, a_{10}, a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, \dots, a_{n-1, n-2}, a_{n-1, n-1}$$

此数组在本函数中要被破坏。

n ——整型变量。方程组的阶数。

m ——整型变量。 n 阶三对角矩阵三条对角线上的元素个数，也是一维数组 b 的长度。它的值应为 $m = 3n - 2$ 。在本函数中要对此值作检查。

d ——双精度实型一维数组，长度为 n 。存放方程组右端的常数向量；返回时存放方程组的解。

五、函数程序(文件名:atrde.c)

六、例

求解 5 阶三对角线方程组

$$\begin{bmatrix} 13 & 12 & & \\ 11 & 10 & 9 & \\ 8 & 7 & 6 & \\ 5 & 4 & 3 & \\ 2 & 1 & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ -2 \\ 6 \\ 8 \end{bmatrix}$$

其中 $n = 5, m = 13, b = (13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)$ 。

主函数程序(文件名: atrde 0.c)如下:

```
# include "stdio.h"
# include "atrde.c"
main()
| int i;
static double b[13] = {13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0,
6.0, 5.0, 4.0, 3.0, 2.0, 1.0};
static double d[5] = {3.0, 0.0, -2.0, 6.0, 8.0};
if (atrde(b, 5, 13, d) > 0)
for (i=0; i<=4; i++)
printf("x(%d) = %13.7e\n", i, d[i]);
|
```

运行结果为:

```
x(0) = 5.718367e+00
x(1) = -5.944898e+00
x(2) = -3.836735e-01
x(3) = 8.040816e+00
x(4) = -8.081633e+00
```

1.6 一般带型方程组的求解

一、功能

用列选主元(Gauss)消去法求解右端具有 m 组常数向量的 n 阶一般带型方程组 $AX = D$ 。其中 A 为 n 阶带型矩阵, 其元素满足

$$a_{ij} \begin{cases} \neq 0, i-l \leq j \leq i+l \\ = 0, \text{ 其它} \end{cases}$$

即

$$A = \left\{ \begin{array}{c|ccccc}
& \overbrace{a_{00} \dots a_{0l}}^{l+1} & & & & 0 \\
\hline
\begin{array}{c} \vdots \\ a_{l0} \dots a_{ll} \end{array} & \cdots & a_{l,2l} & & & \\
& \ddots & \ddots & \ddots & & \\
& a_{n-l-1,n-2l-1} & \cdots & a_{n-l-1,n-l-1} & \cdots & a_{n-l-1,n-1} \\
& \ddots & & \vdots & & \vdots \\
0 & & a_{n-1,n-l-1} & \cdots & a_{n-1,n-1} &
\end{array} \right.$$

l 称为半带宽, 带宽为 $il = 2l + 1$ 。

$$D = \begin{bmatrix} d_{00} & \cdots & d_{0,m-1} \\ d_{10} & \cdots & d_{1,m-1} \\ \vdots & & \vdots \\ d_{n-1,0} & \cdots & d_{n-1,m-1} \end{bmatrix}$$

二、方法说明

对此方程组的求解采用列选主元高斯消去法。由于系数矩阵中非零元素是带状的, 带区外均为零元素, 因此, 在求解过程中可以只考虑带区内的非零元素, 这样可以减少许多计算工作量。同时, 对于带型矩阵也只需存储带区内的非零元素, 可以用一个 n 行、 $(2l + 1)$ 列的二维数组 B 来表示。为了在消去过程中不在带区外产生新的非零元素, 对于带型矩阵 A 中前 $l + 1$ 行与最后 $l + 1$ 行的元素均以左边对齐顺序放在数组 B 的相应行, 而最右边的空余部分均填入零。即带型矩阵 A 用二维数组 B 表示的格式为

$$B = \begin{bmatrix} b_{00} & \cdots & b_{0l} & \cdots & b_{0,2l} \\ \vdots & & \vdots & & \vdots \\ b_{l0} & \cdots & b_{ll} & \cdots & b_{l,2l} \\ \vdots & & \vdots & & \vdots \\ b_{n-l-1,0} & \cdots & b_{n-l-1,l} & \cdots & b_{n-l-1,2l} \\ \vdots & & \vdots & & \vdots \\ b_{n-1,0} & \cdots & b_{n-1,l} & \cdots & b_{n-1,2l} \end{bmatrix} = \begin{bmatrix} a_{00} & \cdots & a_{0l} & 0 \\ \vdots & & \ddots & \ddots \\ a_{l0} & \cdots & a_{ll} & \cdots & a_{l,2l} \\ \vdots & & \vdots & & \vdots \\ a_{n-l-1,n-2l-1} & \cdots & b_{n-l-1,n-l-1} & \cdots & b_{n-l-1,n-1} \\ \vdots & & \vdots & & \vdots \\ a_{n-1,n-l-1} & \cdots & a_{n-1,n-1} & & 0 \end{bmatrix}$$

整个求解过程(包括列选主元)均在二维数组 B 中进行。

由于本函数采用的是列选主元高斯消去法。因此, 对于大型带型矩阵、且带宽较大的方程组, 其数值计算有可能是不稳定的。

三、函数语句

```
int aband(b, d, n, l, il, m)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示参数中半带宽 l 与带宽 il 的关系不对; 若返回的标志值为 0, 则表示系数矩阵 A 奇异; 若返回的标志值大于 0, 则表示程序工作正常。在前两种情况下, 本函数还输出信息“fail”。

四、形参说明

b——双精度实型二维数组, 体积为 $n \times il$ 。存放带型矩阵 A 中带区内的元素, 存放格式见本节的方法说明。此数组在本函数中要被破坏。

d——双精度实型二维数组, 体积为 $n \times m$ 。存放方程组右端的 m 组常数向量; 返回时存放方程组的 m 组解。

n——整型变量。方程组的阶数。

l——整型变量。系数矩阵的半带宽。

il——整型变量。系数矩阵的带宽, 应为 $il = 2l + 1$ 。

m——整型变量。方程组右端的常数向量的个数。

五、函数程序(文件名:aband.c)

六、例

求解 8 阶五对角线方程组 $AX = D$ 。其中

$$A = \begin{bmatrix} 3 & -4 & 1 & & & & & \\ -2 & -5 & 6 & 1 & & & & \\ 1 & 3 & -1 & 2 & -3 & & & \\ & 2 & 5 & -5 & 6 & -1 & & \\ & & -3 & 1 & -1 & 2 & -5 & \\ & & & 6 & 1 & -3 & 2 & -9 \\ & & & & -4 & 1 & -1 & 2 \\ & & & & & 5 & 1 & -7 \end{bmatrix}$$

$$D = \begin{bmatrix} 13 & 29 & -13 \\ -6 & 17 & -21 \\ -31 & -6 & 4 \\ 64 & 3 & 16 \\ -20 & 1 & -5 \\ -22 & -41 & 56 \\ -29 & 10 & -21 \\ 7 & -24 & 20 \end{bmatrix}$$

与带型矩阵 A 所对应的二维数组 B 为

$$B = \begin{bmatrix} 3 & -4 & 1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -5 & 6 & 1 & 0 & 0 & 0 & 0 \\ 1 & 3 & -1 & 2 & -3 & 0 & 0 & 0 \\ 2 & 5 & -5 & 6 & -1 & 0 & 0 & 0 \\ -3 & 1 & -1 & 2 & -5 & 0 & 0 & 0 \\ 6 & 1 & -3 & 2 & -9 & 0 & 0 & 0 \\ -4 & 1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 5 & 1 & -7 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

在本问题中, $n = 8$, 半带宽 $l = 2$, 带宽 $il = 5$, $m = 3$ 。

主函数程序(文件名:aband.c)如下:

```
# include "aband.c"
# include "stdio.h"
main()
{ int i;
static double b[8][5]={{3.0, -4.0, 1.0, 0.0, 0.0},
{ -2.0, -5.0, 6.0, 1.0, 0.0}, {1.0, 3.0, -1.0, 2.0, -3.0},
{2.0, 5.0, -5.0, 6.0, -1.0}, {-3.0, 1.0, -1.0, 2.0, -5.0},
{6.0, 1.0, -3.0, 2.0, -9.0}, {-4.0, 1.0, -1.0, 2.0, 0.0},
{5.0, 1.0, -7.0, 0.0, 0.0}};
static double d[8][3]={{13.0, 29.0, -13.0},
{-6.0, 17.0, -21.0}, {-31.0, -6.0, 4.0}, {64.0, 3.0, 16.0},
{-20.0, 1.0, -5.0}, {-22.0, -41.0, 56.0},
{-29.0, 10.0, -21.0}, {7.0, -24.0, 20.0}};
if (aband(b, d, 8, 2, 5, 3)>0)
for (i=0; i<=7; i++)
printf("x(%d) = %13.7e, %13.7e, %13.7e\n",
i, d[i][0], d[i][1], d[i][2]);
}
```

运行结果为:

```
x(0)= 3.000000e+00, 5.000000e+00, -4.070908e-16
x(1)= -1.000000e+00, -3.000000e+00, 3.000000e+00
x(2)= 1.908873e-16, 2.000000e+00, -1.000000e+00
x(3)= -5.000000e+00, -2.220446e-15, -1.332268e-15
x(4)= 7.000000e+00, -9.688431e-16, 2.000000e+00
x(5)= 1.000000e+00, 1.000000e+00, -3.000000e+00
x(6)= 2.000000e+00, -1.000000e+00, -2.385245e-17
x(7)= -9.288423e-16, 4.000000e+00, -5.000000e+00
```

本方程组的准确解为:

第一组解 第二组解 第三组解

$x_0 = 3.0$	$x_0 = 5.0$	$x_0 = 0.0$
$x_1 = -1.0$	$x_1 = -3.0$	$x_1 = 3.0$
$x_2 = 0.0$	$x_2 = 2.0$	$x_2 = -1.0$
$x_3 = -5.0$	$x_3 = 0.0$	$x_3 = 0.0$
$x_4 = 7.0$	$x_4 = 0.0$	$x_4 = 2.0$
$x_5 = 1.0$	$x_5 = 1.0$	$x_5 = -3.0$
$x_6 = 2.0$	$x_6 = -1.0$	$x_6 = 0.0$
$x_7 = 0.0$	$x_7 = 4.0$	$x_7 = -5.0$

1.7 求解对称方程组的分解法

一、功能

用分解法求解系数矩阵为对称、且右端具有 m 组常数向量的线性代数方程组 $AX = C$ 。其中 A 为 n 阶对称矩阵, C 为

$$C = \begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0, m-1} \\ c_{10} & c_{11} & \cdots & c_{1, m-1} \\ \vdots & \vdots & & \vdots \\ c_{n-1, 0} & c_{n-1, 1} & \cdots & c_{n-1, m-1} \end{bmatrix}$$

二、方法说明

对称矩阵 A 可以分解为一个下三角矩阵 L 、一个对角线矩阵 D 和一个上三角矩阵 L^T 的乘积, 即 $A = LDL^T$, 其中:

$$L = \begin{bmatrix} 1 & & & \\ l_{10} & 1 & 0 & \\ \vdots & \vdots & \ddots & \\ l_{n-1, 0} & l_{n-1, 1} & \cdots & 1 \end{bmatrix}, D = \begin{bmatrix} d_{00} & & & & \\ & d_{11} & & & 0 \\ & & \ddots & & \\ & 0 & & & \\ & & & & d_{n-1, m-1} \end{bmatrix}$$

矩阵 L 和 D 中的各元素由下列计算公式确定:

$$\begin{aligned} d_{00} &= a_{00} \\ d_{ii} &= a_{ii} - \sum_{k=0}^{i-1} l_{ik}^2 d_{kk} \\ l_{ij} &= (a_{ij} - \sum_{k=0}^{j-1} l_{ik} l_{jk} d_{kk}) / d_{jj}, j < i \\ l_{ij} &= 0, j > i \end{aligned}$$

对于方程组 $AX = B$ 来说, 当 L 和 D 确定后, 令 $DL^T X = Y$, 则可由回代过程求解方程组 $LY = B$ 而得到 Y , 再由方程组 $DL^T X = Y$ 解出 X 。其计算公式如下

$$\begin{aligned} y_0 &= b_0 \\ y_i &= b_i - \sum_{k=0}^{i-1} l_{ik} y_k, i > 0 \\ x_{n-1} &= y_{n-1} / d_{n-1, n-1} \\ x_i &= (y_i - \sum_{k=i+1}^{n-1} d_{ii} l_{ki} x_k) / d_{ii} \end{aligned}$$

三、函数语句

int alde(a, n, m, c)

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示程序工作失败, 输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a ——双精度实型二维数组, 体积为 $n \times n$ 。存放方程组的系数矩阵(应为对称矩阵)。

此数组在本函数中要被破坏。

n ——整型变量。方程组的阶数。

m ——整型变量。方程组右端常数向量的个数。

c ——双精度实型二维数组, 体积为 $n \times m$ 。存放方程组右端的 m 组常数向量; 返回时存放方程组的 m 组解。

五、函数程序(文件名 alde.c)

六、例

求解 5 阶对称方程组 $AX = C$ 。其中

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 & 1 \\ 7 & 10 & 8 & 7 & 2 \\ 6 & 8 & 10 & 9 & 3 \\ 5 & 7 & 9 & 10 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}, C = \begin{bmatrix} 24 & 96 \\ 34 & 136 \\ 36 & 144 \\ 35 & 140 \\ 15 & 60 \end{bmatrix}$$

主函数程序(文件名: alde 0.c)如下:

```
# include "stdio.h"
# include "alde.c"
main()
{ int i;
static double a[5][5] = { {5.0, 7.0, 6.0, 5.0, 1.0},
{7.0, 10.0, 8.0, 7.0, 2.0}, {6.0, 8.0, 10.0, 9.0, 3.0},
{5.0, 7.0, 9.0, 10.0, 4.0}, {1.0, 2.0, 3.0, 4.0, 5.0} };
static double c[5][2] = { {24.0, 96.0}, {34.0, 136.0},
{36.0, 144.0}, {35.0, 140.0}, {15.0, 60.0} };
if (alde(a, 5, 2, c) > 0)
for (i=0; i<=4; i++)
printf("%d = %13.7e %13.7e\n", i, c[i][0], c[i][1]);
}
```

运行结果为:

```
x(0) = 1.000000e+00, 4.000000e+00
x(1) = 1.000000e+00, 4.000000e+00
x(2) = 1.000000e+00, 4.000000e+00
x(3) = 1.000000e+00, 4.000000e+00
x(4) = 1.000000e+00, 4.000000e+00
```

本方程组的准确解为:

第一组解 $x_0 = x_1 = x_2 = x_3 = x_4 = 1$

第二组解 $x_0 = x_1 = x_2 = x_3 = x_4 = 4$

1.8 求解对称正定方程组的平方根法

一、功能

用乔里斯基(Cholesky)分解法(即平方根法)求解系数矩阵为对称正定、且右端具有 m 组常数向量的 n 阶线性代数方程组 $AX = D$ 。

二、方法说明

当系数矩阵 A 为对称正定时, 可以唯一地分解为 $A = U^T U$, 其中 U 为上三角矩阵。

即

$$\begin{aligned} & \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0,n-1} \\ a_{10} & a_{11} & \dots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{bmatrix} \\ &= \begin{bmatrix} u_{00} & & & u_{01} & \dots & u_{0,n-1} \\ u_{10} & u_{11} & & 0 & & \\ \vdots & \vdots & \ddots & & & \\ u_{n-1,0} & u_{n-1,1} & \dots & u_{n-1,n-1} & & \end{bmatrix} \begin{bmatrix} u_{00} & u_{01} & \dots & u_{0,n-1} \\ u_{11} & \dots & u_{1,n-1} \\ 0 & \ddots & \vdots \\ u_{n-1,n-1} & & & \end{bmatrix} \end{aligned}$$

且 $u_{ij} = u_{ji}$ ($i, j = 0, 1, \dots, n-1$)。

U 矩阵中的各元素由以下计算公式确定:

$$\begin{aligned} u_{00} &= \sqrt{a_{00}} \\ u_{ii} &= (a_{ii} - \sum_{k=0}^{i-1} u_{ki}^2)^{\frac{1}{2}}, \quad i = 1, 2, \dots, n-1 \\ u_{ij} &= (a_{ij} - \sum_{k=0}^{i-1} u_{ki} u_{kj}) / u_{ii}, \quad j > i \end{aligned}$$

于是, 方程组 $AX = B$ 的解可由下述公式计算:

$$\begin{aligned} y_i &= (b_i - \sum_{k=0}^{i-1} u_{ki} y_k) / u_{ii} \\ x_i &= (y_i - \sum_{k=i+1}^{n-1} u_{ik} x_k) / u_{ii} \end{aligned}$$

三、函数语句

int achol(a, n, m, d)

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示程序工作失败(如系数矩阵非正定), 输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放系数矩阵(应为对称正定矩阵); 返回时, 其上三角部分存放分解后的 U 矩阵, 即

$$U = \begin{bmatrix} u_{00} & u_{01} & \dots & u_{0,n-1} \\ u_{10} & u_{11} & \dots & u_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n-1,0} & u_{n-1,1} & \dots & u_{n-1,n-1} \end{bmatrix}$$

n——整型变量。方程组的阶数。

m——整型变量。方程组右端常数向量的个数。

d——双精度实型二维数组, 体积为 $n \times m$ 。存放方程组右端 m 组常数向量; 返回时, 存放方程组的 m 组解。

五、函数程序(文件名: achol.c)

六、例

求解 4 阶对称正定方程组 $AX = D$ 。其中

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \quad D = \begin{bmatrix} 23 & 92 \\ 32 & 128 \\ 33 & 132 \\ 31 & 124 \end{bmatrix}$$

主函数程序(文件名: achol 0.c)如下:

```
# include "stdio.h"
# include "achol.c"
main()
{
    int i;
    static double a[4][4] = {{5.0, 7.0, 6.0, 5.0},
                            {7.0, 10.0, 8.0, 7.0}, {6.0, 8.0, 10.0, 9.0}, {5.0, 7.0, 9.0, 10.0}};
    static double d[4][2] = {{23.0, 92.0}, {32.0, 128.0},
                            {33.0, 132.0}, {31.0, 124.0}};
    if (achol(a, 4, 2, d) > 0)
        for (i = 0; i < 4; i++)
            printf("x(%d) = %13.7e, %13.7e\n", i, d[i][0], d[i][1]);
}
```

运行结果为:

```
x(0) = 1.000000e+00, 4.000000e+00
x(1) = 1.000000e+00, 4.000000e+00
x(2) = 1.000000e+00, 4.000000e+00
x(3) = 1.000000e+00, 4.000000e+00
```

本方程组的准确解为:

第一组解 $x_0 = x_1 = x_2 = x_3 = 1$

第二组解 $x_0 = x_1 = x_2 = x_3 = 4$

1.9 求解大型稀疏方程组的全选主元 高斯-约当消去法

一、功能

用全选主元高斯-约当(Gauss-Jordan)消去法求解系数矩阵为稀疏矩阵的大型方程组 $AX = B$ 。

二、方法说明

基本方法同 1.2 节, 只是在消去过程中避免了对零元素的运算, 从而可以大大减少运算次数。本函数不考虑稀疏矩阵的压缩存储问题。

三、函数语句

```
int aggje(a, n, b)
```

本函数返回一个整型标志值。若返回的标志值为 0, 则表示系数矩阵奇异, 还输出信息“fail”; 若返回的标志值不为 0, 则表示正常返回。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放方程组的系数矩阵。此数组在本函数中要被破坏。

n——整型变量。方程组的阶数。

b——双精度实型一维数组, 长度为 n。存放方程组右端的常数向量; 返回时存放方程组的解。

五、函数程序(文件名: aggje.c)

六、例

求解 8 阶稀疏方程组 $AX = B$ 。其中

$$A = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 6 & 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & -4 \\ 3 & 0 & 0 & 0 & -2 & 0 & 1 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 5 & 0 \\ 1 & 0 & 0 & 0 & -3 & 0 & 0 & 2 \\ 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & -2 \end{bmatrix}$$

$$B = (4, 6, -8, -2, 27, -9, 2, -4)^T$$

主函数程序(文件名: aggje0.c)如下:

```
# include "stdio.h"
# include "aggje.c"
main()
{ int i;
  static double a[8][8] = { {0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 2.0, 0.0},
                            {0.0, 6.0, 0.0, 0.0, 0.0, -6.0, 0.0, 0.0},
                            {0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, -4.0},
                            {3.0, 0.0, 0.0, 0.0, -2.0, 0.0, 1.0, 0.0},
                            {0.0, 0.0, 6.0, 0.0, 0.0, 0.0, 5.0, 0.0},
                            {1.0, 0.0, 0.0, 0.0, -3.0, 0.0, 0.0, 2.0},
                            {0.0, 4.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0},
                            {0.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0, -2.0} };
  static double b[8] = { 4.0, 6.0, -8.0, -2.0, 27.0, -9.0, 2.0, -4.0 };
  if (aggje(a, 8, b) != 0)
    for (i = 0; i < 8; i++)
      printf("x(%d) = %13.7e\n", i, b[i]);
}
```

运行结果为:

```
x(0) = 1.000000e+00
x(1) = 0.000000e+00
x(2) = 2.000000e+00
x(3) = -2.000000e+00
x(4) = 4.000000e+00
x(5) = -1.000000e+00
x(6) = 3.000000e+00
x(7) = 1.000000e+00
```

1.10 求解托伯利兹方程组的列文逊方法

一、功能

用列文逊(Levinson)递推算法求解对称托伯利兹(Toeplitz)型方程组。

二、方法说明

n 阶对称托伯利兹矩阵为如下形式的矩阵:

$$T^{(n)} = \begin{bmatrix} t_0 & t_1 & t_2 & \dots & t_{n-1} \\ t_1 & t_0 & t_1 & \dots & t_{n-2} \\ t_2 & t_1 & t_0 & \dots & t_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & t_{n-3} & \dots & t_0 \end{bmatrix}$$

简称 n 阶对称 T 型矩阵。

设线性代数方程组 $AX = B$ 的系数矩阵为 n 阶对称 T 型矩阵。即 $A = T^{(n)}$

假设已知

$$T^{(k)} \begin{bmatrix} y_0^{(k)} \\ \vdots \\ y_{k-2}^{(k)} \\ y_{k-1}^{(k)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \alpha_{k-1} \end{bmatrix}$$

令

$$\beta_{k-1} = y_{k-1}^{(k)} t_k + y_{k-2}^{(k)} t_{k-1} + \cdots + y_0^{(k)} t_1 = \sum_{j=0}^{k-1} t_{j+1} y_j^{(k)} \quad (1)$$

由于 $T^{(k)}$ 与 $T^{(k+1)}$ 均为对称 T 型矩阵, 因此有

$$T^{(k+1)} \begin{bmatrix} 0 \\ y_0^{(k)} \\ \vdots \\ y_{k-2}^{(k)} \\ y_{k-1}^{(k)} \end{bmatrix} = \begin{bmatrix} \beta_{k-1} \\ 0 \\ \vdots \\ 0 \\ \alpha_{k-1} \end{bmatrix} \quad (2)$$

与

$$T^{(k+1)} \begin{bmatrix} y_{k-1}^{(k)} \\ y_{k-2}^{(k)} \\ \vdots \\ y_0^{(k)} \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_{k-1} \\ 0 \\ \vdots \\ 0 \\ \beta_{k-1} \end{bmatrix} \quad (3)$$

若将方程组(2)减去方程组(3)的 β_{k-1}/α_{k-1} 倍, 并且令

$$c_{k-1} = -\beta_{k-1}/\alpha_{k-1} \quad (4)$$

则可得到如下方程组

$$T^{(k+1)} \begin{bmatrix} y_0^{(k)} + c_{k-1} y_{k-1}^{(k)} \\ \vdots \\ y_{k-2}^{(k)} + c_{k-1} y_0^{(k)} \\ y_{k-1}^{(k)} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \alpha_{k-1} + c_{k-1} \beta_{k-1} \end{bmatrix} \quad (5)$$

若再令

$$\begin{cases} y_0^{(k+1)} = c_{k-1} y_{k-1}^{(k)} \\ y_i^{(k+1)} = y_{i-1}^{(k)} + c_{k-1} y_{k-i-1}^{(k)}, i = 1, 2, \dots, k-1 \\ y_k^{(k+1)} = y_k^{(k)} \end{cases} \quad (6)$$

及

$$\alpha_k = \alpha_{k-1} + c_{k-1} \beta_{k-1} \quad (7)$$

则方程组(5)变为

$$T^{(k+1)} \begin{bmatrix} y_0^{(k+1)} \\ y_1^{(k+1)} \\ \vdots \\ y_{k-1}^{(k+1)} \\ y_k^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_k \end{bmatrix} \quad (8)$$

显然, 式(6)与(7)是递推公式。若取 $\alpha_0 = t_0$, 则 $y_0^{(1)} = 1$ 。

下面再考虑方程组

$$T^{(k)} X^{(k)} = B^{(k)}, k = 2, 3, \dots, n \quad (9)$$

其中

$$X^{(k)} = (x_0^{(k)}, x_1^{(k)}, \dots, x_{k-1}^{(k)})^T$$

$$B^{(k)} = (b_0, b_1, \dots, b_{k-1})^T$$

现假设对于某个 k , 方程组(9)已经解出, 则

$$T^{(k+1)} \begin{bmatrix} X^{(k+1)} \\ 0 \end{bmatrix} = \begin{bmatrix} B^{(k)} \\ q_k \end{bmatrix} \quad (10)$$

其中

$$q_k = x_0^{(k)} t_k + \cdots + x_{k-1}^{(k)} t_1 = \sum_{j=0}^{k-1} x_j^{(k)} t_{k-j} \quad (11)$$

又因为

$$T^{(k+1)} X^{(k+1)} = B^{(k+1)} \quad (12)$$

于是, 方程组(12)减去(10)得

$$T^{(k+1)} \begin{bmatrix} x_0^{(k+1)} - x_0^{(k)} \\ x_1^{(k+1)} - x_1^{(k)} \\ \vdots \\ x_{k-1}^{(k+1)} - x_{k-1}^{(k)} \\ x_k^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b_k - q_k \end{bmatrix} \quad (13)$$

在上述过程中, 当 $k=1$ 时有 $x_0^{(1)} = b_0/t_0$, 如果用

$$w_k = (b_k - q_k)/\alpha_k \quad (14)$$

乘方程组(8), 可以得到:

$$T^{(k+1)} \begin{bmatrix} w_k y_0^{(k+1)} \\ w_k y_1^{(k+1)} \\ \vdots \\ w_k y_{k-1}^{(k+1)} \\ w_k y_k^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b_k - q_k \end{bmatrix} \quad (15)$$

比较方程组(13)与(15), 可以得到由 $X^{(k)}$ 计算 $X^{(k+1)}$ 的递推公式如下:

$$\begin{cases} x_i^{(k+1)} = x_i^{(k)} + w_k y_i^{(k+1)}, i = 0, 1, \dots, k-1 \\ x_k^{(k+1)} = w_k y_k^{(k+1)} \end{cases} \quad (16)$$

综上所述, 由式(11)、(3)、(4)、(6)、(7)、(14)及(16), 可以得到求解对称 T 型方程组的

递推算法如下。

取初值 $\alpha_0 = t_0$, $y_0^{(1)} = 1$, $x_0^{(1)} = b_0/t_0$ 。

对于 k 从 1 到 $n-1$, 作如下计算:

$$\begin{aligned} q_k &= \sum_{j=0}^{k-1} x_j^{(k)} t_{k-j} \\ \beta_{k-1} &= \sum_{j=0}^{k-1} y_j^{(k)} t_{j+1} \\ c_{k-1} &= -\beta_{k-1}/\alpha_{k-1} \\ \begin{cases} y_0^{(k+1)} = c_{k-1} y_{k-1}^{(k)} \\ y_i^{(k+1)} = y_{i-1}^{(k)} + c_{k-1} y_{k-i-1}^{(k)}, i = 1, 2, \dots, k-1 \\ y_k^{(k+1)} = y_{k-1}^{(k)} \end{cases} \\ \alpha_k &= \alpha_{k-1} + c_{k-1} \beta_{k-1} \\ w_k &= (b_k - q_k)/\alpha_k \\ \begin{cases} x_i^{(k+1)} = x_i^{(k)} + w_k y_i^{(k+1)}, i = 0, 1, \dots, k-1 \\ x_k^{(k+1)} = w_k y_k^{(k+1)} \end{cases} \end{aligned}$$

三、函数语句

```
int atlvs(t, n, b, x)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示程序工作失败, 输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

t —双精度实型一维数组, 长度为 n 。存放对称 T 型矩阵中的元素 t_0, t_1, \dots, t_{n-1} 。

n —整型变量。方程组的阶数。

b —双精度实型一维数组, 长度为 n 。存放方程组右端的常数向量。

x —双精度实型一维数组, 长度为 n 。返回方程组的解。

五、函数程序(文件名: atlvs.c)

六、例

求解 6 阶对称 T 型方程组 $AX = B$ 。其中

$$A = \begin{bmatrix} 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 6 & 5 & 4 & 3 & 2 \\ 4 & 5 & 6 & 5 & 4 & 3 \\ 3 & 4 & 5 & 6 & 5 & 4 \\ 2 & 3 & 4 & 5 & 6 & 5 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} = T^{(6)}$$

即 $t = (6, 5, 4, 3, 2, 1)$, 常数向量为 $B = (11, 9, 9, 9, 13, 17)^T$ 。

主函数程序(文件名: atlvs0.c)如下:

```
# include "stdio.h"
# include "atlvs.c"
main()
{ int i;
  static double x[6];
  static double t[6] = {6.0, 5.0, 4.0, 3.0, 2.0, 1.0};
  static double b[6] = {11.0, 9.0, 9.0, 9.0, 13.0, 17.0};
  if (atlvs(t, 6, b, x)>0)
    for (i=0; i<=5; i++)
      printf("%x(%d) = %13.7e\n", i, x[i]);
}
```

运行结果为:

```
x(0) = 3.000000e+00
x(1) = -1.000000e+00
x(2) = -1.173206e-15
x(3) = -2.000000e+00
x(4) = -2.109424e-15
x(5) = 4.000000e+00
```

1.11 高斯-赛德尔迭代法

一、功能

用高斯-赛德尔(Gauss-Seidel)迭代法求解系数矩阵具有主对角线占绝对优势的线性代数方程组 $AX = B$ 。其中

$$\sum_{\substack{j=0 \\ j \neq i}}^{n-1} |a_{ij}| < |a_{ii}|, \quad i = 0, 1, \dots, n-1$$

二、方法说明

高斯-赛德尔迭代格式为

$$x_i^{(k+1)} = \left(b_i - \sum_{j=0}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n-1} a_{ij} x_j^{(k)} \right) / a_{ii}, \quad i = 0, 1, \dots, n-1$$

其中初值取 $x_i^{(0)} = 0$ ($i = 0, 1, \dots, n-1$)。

结束迭代的条件为

$$\max_{0 \leq i \leq n-1} \frac{|x_i^{(k+1)} - x_i^{(k)}|}{1 + |x_i^{(k+1)}|} < \epsilon$$

其中 ϵ 为给定的精度要求。

三、函数语句

```
int agsdl(a, b, n, x, eps)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则说明系数矩阵不具有主对角线占绝对优势, 还输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放方程组的系数矩阵。
b——双精度实型一维数组, 长度为 n 。存放方程组右端的常数向量。
n——整型变量。方程组的阶数。
x——双精度实型一维数组, 长度为 n 。返回方程组的解。
eps——双精度实型变量。给定的精度要求。

五、函数程序(文件名: agsdl.c)

六、例

用高斯-赛德尔迭代法求解下列 4 阶方程组:

$$\begin{cases} 7x_0 + 2x_1 + x_2 - 2x_3 = 4 \\ 9x_0 + 15x_1 + 3x_2 - 2x_3 = 7 \\ -2x_0 - 2x_1 + 11x_2 + 5x_3 = -1 \\ x_0 + 3x_1 + 2x_2 + 13x_3 = 0 \end{cases}$$

取 $\epsilon = 0.000001$ 。

主函数程序(文件名: agsdl 0.c)如下:

```
# include "stdio.h"
# include "agsdl.c"
main()
{ int i;
  double eps;
  static double a[4][4] = { {7.0, 2.0, 1.0, -2.0}, {9.0, 15.0, 3.0, -2.0},
                           {-2.0, -2.0, 11.0, 5.0}, {1.0, 3.0, 2.0, 13.0} };
  static double x[5], b[4] = {4.0, 7.0, -1.0, 0.0};
  eps = 0.000001;
  if (agsdl(a, b, 4, x, eps) > 0)
    for (i = 0; i <= 3; i++)
      printf("x(%d) = %13.7e\n", i, x[i]);
```

运行结果为:

```
x(0) = 4.97931e-01
x(1) = 1.444939e-01
x(2) = 6.285805e-02
x(3) = -8.131763e-02
```

1.12 求解对称正定方程组的共轭梯度法

一、功能

用共轭梯度法求解 n 阶对称正定方程组 $AX = B$ 。

二、方法说明

共轭梯度法的递推计算公式如下。
取初值 $X_0 = (0, 0, \dots, 0)^T$, $R_0 = P_0 = B$ 。
对于 $i = 0, 1, \dots, n-1$ 作如下计算:

$$\alpha_i = \frac{(P_i, B)}{(P_i, AP_i)}$$

$$X_{i+1} = X_i + \alpha_i P_i$$

$$R_{i+1} = B - AX_{i+1}$$

$$\beta_i = \frac{(R_{i+1}, AP_i)}{(P_i, AP_i)}$$

$$P_{i+1} = R_{i+1} - \beta_i P_i$$

上述过程一直作到 $\|R_i\| < \epsilon$ 或 $i = n-1$ 为止。

特别要指出, 本方法只适用于对称正定方程组。

三、函数语句

```
void agrad(a, n, b, eps, x)
```

本函数要调用实矩阵相乘的函数 brmul, 参看 2.1 节。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放对称正定矩阵 A 。
n——整型变量。方程组的阶数。
b——双精度实型一维数组, 长度为 n 。存放方程组右端的常数向量。
eps——双精度实型变量。控制精度要求。
x——双精度实型一维数组, 长度为 n 。返回方程组的解向量。

五、函数程序(文件名: agrad.c)

六、例

用共轭梯度法求解 4 阶对称正定方程组 $AX = B$ 。其中

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, B = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

取 $\epsilon = 0.000001$ 。

主函数程序(文件名: agrad 0.c)如下:

```
# include "stdio.h"
# include "agrad.c"
# include "brmul.c"
main()
```

```

int i;
double eps, x[4];
static double a[4][4] = { {5.0, 7.0, 6.0, 5.0},
                        {7.0, 10.0, 8.0, 7.0},
                        {6.0, 8.0, 10.0, 9.0},
                        {5.0, 7.0, 9.0, 10.0} };
static double b[4] = {23.0, 32.0, 33.0, 31.0};
eps = 0.000001;
agrad(a, 4, b, eps, x);
printf("\n");
for (i = 0; i <= 3; i++)
    printf("%e\n", x[i]);
printf("\n");

```

运行结果为：

```

x[0] = 1.00000e+00
x[1] = 1.00000e+00
x[2] = 1.00000e+00
x[3] = 1.00000e+00

```

1.13 求解线性最小二乘问题的豪斯荷尔德变换法

一、功能

用豪斯荷尔德(Householder)变换求解线性最小二乘问题。

二、方法说明

设超定方程组 $AX = B$, 其中 A 为 $m \times n$ ($m \geq n$) 列线性无关的矩阵, X 为 n 维列向量, B 为 m 维列向量。

用豪斯荷尔德变换将 A 进行 QR 分解。即

$$A = QR$$

其中 Q 为 $m \times m$ 的正交矩阵, R 为上三角矩阵。具体分解过程见 2.11 节的方法说明。

设 $E = B - AX$, 用 Q^T 乘上式两端得

$$Q^T E = Q^T B - Q^T A X = Q^T B - R X$$

因为 Q^T 为正交矩阵, 所以有

$$\|E\|_2^2 = \|Q^T E\|_2^2 = \|Q^T B - R X\|_2^2$$

若令

$$Q^T B = \begin{bmatrix} C \\ D \end{bmatrix}, R X = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} X$$

其中 C 为 n 维列向量, D 为 $m - n$ 维列向量, R_1 为 $n \times n$ 上三角方阵, 0 为 $(m - n) \times n$ 的零矩阵。则有

$$\|E\|_2^2 = \|C - R_1 X\|_2^2 + \|D\|_2^2$$

显然, 当 X 满足 $R_1 X = C$ 时, $\|E\|_2^2$ 将取最小值。

由上所述, 求解线性最小二乘问题 $AX = B$ 的步骤如下:

(1) 对 A 进行 QR 分解。即 $A = QR$, 其中 Q 为 $m \times m$ 的正交矩阵, R 为右上三角矩阵。且令

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

其中 R_1 为 $n \times n$ 的上三角方阵。

(2) 计算

$$\begin{bmatrix} C \\ D \end{bmatrix} = Q^T B$$

其中 C 为 n 维列向量。

(3) 利用回代求解方程组 $R_1 X = C$

三、函数语句

int agmqr(a, m, n, b, q)

本函数返回一个整型标志值。若返回的标志值为 0, 则表示程序工作失败(如系数矩阵 A 列线性相关), 还输出信息“fail”; 若返回的标志值不为 0, 则表示正常返回。

本函数要调用 QR 分解的函数 bmaqr, 参看 2.11 节。

四、形参说明

a ——双精度实型二维数组, 体积为 $m \times n$ 。存放超定方程组的系数矩阵 A ; 返回时存放 QR 分解式中的 R 矩阵。

m ——整型变量。系数矩阵 A 的行数。 $m \geq n$ 。

n ——整型变量。系数矩阵 A 的列数。 $n \leq m$ 。

b ——双精度实型一维数组, 长度为 m 。存放方程组右端常数向量; 返回时前 n 个元素(即 $b(0), b(1), \dots, b(n-1)$)存放方程组的最小二乘解。

q ——双精度实型二维数组, 体积为 $m \times m$ 。返回时存放 QR 分解式中的正交矩阵 Q 。

五、函数程序(文件名: agmqr.c)

六、例

求下列超定方程组的最小二乘解, 并求系数矩阵的 QR 分解式。

$$\begin{cases} x_0 + x_1 - x_2 = 2 \\ 2x_0 + x_1 = -3 \\ x_0 - x_1 = 1 \\ -x_0 + 2x_1 + x_2 = 4 \end{cases}$$

主函数程序(文件名: agmqr 0.c)如下:

```
# include "bmaqr.c"
# include "stdio.h"
# include "agmqr.c"
```

```

main()
{
    int m, n;
    static double a[4][3] = { {1.0, 1.0, -1.0}, {2.0, 1.0, 0.0},
                            {1.0, -1.0, 0.0}, {-1.0, 2.0, 1.0} };
    static double b[4] = {2.0, -3.0, 1.0, 4.0};
    static double q[4][4];
    m = 4; n = 3;
    i = agmqr(a, m, n, b, q);
    if (i != 0)
        for (i = 0; i <= 2; i++)
            printf("x(%d) = %13.7e\n", i, b[i]);
    printf("\n");
    printf("MAT Q IS:\n");
    for (i = 0; i <= 3; i++)
        printf("%13.7e %13.7e %13.7e %13.7e\n",
               q[i][0], q[i][1], q[i][2], q[i][3]);
    printf("\n");
    printf("MAT R IS:\n");
    for (i = 0; i <= 3; i++)
        printf("%13.7e %13.7e %13.7e\n",
               a[i][0], a[i][1], a[i][2]);
}

```

运行结果为：

```

x(0) = -1.190476e+00
x(1) = 9.523810e-01
x(2) = -6.666667e-01

MAT Q IS:
-3.779645e-01 -3.779645e-01 7.559289e-01 3.779645e-01
-7.559289e-01 -3.779645e-01 -3.779645e-01 -3.779645e-01
-3.779645e-01 3.779645e-01 -3.779645e-01 7.559289e-01
3.779645e-01 -7.559289e-01 -3.779645e-01 3.779645e-01

MAT R IS:
-2.645751e+00 -1.955901e-16 7.559289e-01
0.000000e+00 -2.645751e+00 -3.779645e-01
0.000000e+00 0.000000e+00 -1.133893e+00
0.000000e+00 0.000000e+00 0.000000e+00

```

1.14 求解线性最小二乘问题的广义逆法

一、功能

利用广义逆求超定方程组 $AX = B$ 的最小二乘解。其中 A 为 $m \times n$ ($m \geq n$) 的矩阵，且列线性无关。当 $m = n$ 时，即为求线性代数方程组的解。

二、方法说明

首先对矩阵 A 进行奇异值分解。即

$$A = U \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V^T$$

(参看 2.12 节的方法说明)。

然后利用奇异值分解式计算 A 的广义逆 A^+ 。即

$$A^+ = V_1 \Sigma^{-1} U_1^T$$

(参看 2.13 节的方法说明)。

最后利用广义逆 A^+ 求超定方程组 $AX = B$ 的最小二乘解。即 $X = A^+ B$

三、函数语句

int agmiv(a, m, n, b, x, aa, eps, u, v, ka)

本函数返回一个整型标志值。若返回的标志值小于 0，则表示在奇异值分解过程中迭代超过了 60 次还未达到精度要求，此时矩阵 A 的分解式为 UAV ，求最小二乘解失败；若返回的标志值大于 0，则表示正常返回。

本函数要调用求广义逆的函数 bginv，参看 2.13 节；在函数 bginv 中又要调用奇异值分解的函数 bmuav，参看 2.12 节。

四、形参说明

a——双精度实型二维数组，体积为 $m \times n$ 。存放超定方程组的系数矩阵 A ；返回时其对角线依次给出奇异值 $\sigma_0, \sigma_1, \dots, \sigma_p$ ，其余元素均为零。

m——整型变量。系数矩阵 A 的行数。

n——整型变量。系数矩阵 A 的列数。

b——双精度实型一维数组，长度为 m 。存放超定方程组右端常数向量。

x——双精度实型一维数组，长度为 n 。返回时存放超定方程组的最小二乘解。

aa——双精度实型二维数组，体积为 $n \times m$ 。返回时存放系数矩阵 A 的广义逆 A^+ 。

eps——双精度实型变量。奇异值分解函数中的控制精度要求。

u——双精度实型二维数组，体积为 $m \times m$ 。返回时存放系数矩阵 A 的奇异值分解式中的左奇异向量 U 。

v——双精度实型二维数组，体积为 $n \times n$ 。返回时存放系数矩阵 A 的奇异值分解式中的右奇异向量 V^T 。

ka——整型变量。 $ka = \max(m, n) + 1$

五、函数程序(文件名：agmiv.c)

六、例

求下列超定方程组的最小二乘解，同时输出系数矩阵的广义逆：

$$\begin{cases} x_0 + x_1 - x_2 = 2 \\ 2x_0 + x_1 = -3 \\ x_0 - x_1 = 1 \\ -x_0 + 2x_1 + x_2 = 4 \end{cases}$$

取 $\epsilon = 0.000001$

主函数程序(文件名: agmiv 0.c)如下:

```
# include "stdio.h"
# include "agmiv.c"
# include "bginv.c"
# include "bmuav.c"
main()
{
    int i, m, n, ka;
    static double x[3], aa[3][4], u[4][4], v[3][3];
    static double a[4][3] = { {1.0, 1.0, -1.0}, {2.0, 1.0, 0.0}, {1.0, -1.0, 0.0}, {-1.0, 2.0, 1.0} };
    static double b[4] = {2.0, -3.0, 1.0, 4.0};
    double eps;
    m = 4; n = 3; ka = 5; eps = 0.000001;
    i = agmiv(a, m, n, b, x, aa, eps, u, v, ka);
    if (i > 0)
        for (i = 0; i <= 2; i++)
            printf("%e\n", x[i]);
    printf("\n");
    printf("MAT A+:\n");
    for (i = 0; i <= 2; i++)
        printf("%e %e %e %e\n", aa[i][0], aa[i][1], aa[i][2], aa[i][3]);
    printf("\n");
}
```

运行结果为:

```
x(0) = -1.190476e+00
x(1) = 9.523810e-01
x(2) = -6.666667e-01
MAT A+ :
-4.761905e-02 3.809524e-01 2.380952e-01 -4.761905e-02
2.380952e-01 9.523810e-02 -1.904762e-01 2.380952e-01
-6.666667e-01 2.333333e-01 3.333333e-01 3.333333e-01
```

1.15 病态方程组的求解

一、功能

求解病态线性代数方程组 $AX = B$ 。

二、方法说明

设线性代数方程组 $AX = B$ 是病态的。其求解的步骤如下:

- (1) 用全选主元高斯消去法求解, 得到一组近似解 $X^{(1)} = (x_0^{(1)}, x_1^{(1)}, \dots, x_{n-1}^{(1)})^T$ 。
- (2) 计算剩余向量

$$R = B - AX^{(1)}$$

- (3) 用全选主元高斯消去法求解线性代数方程组

$$AE = R$$

解出 $e = (e_0, e_1, \dots, e_{n-1})^T$ 。

- (4) 计算

$$X^{(2)} = X^{(1)} + E$$

- (5) 令 $X^{(1)} = X^{(2)}$, 转(2)重复这个过程。直到

$$\max_{0 \leq i \leq n-1} \frac{|x_i^{(2)} - x_i^{(1)}|}{1 + |x_i^{(2)}|} < \epsilon$$

为止。

三、函数语句

int abint(a, n, b, eps, x)

本函数返回一个整型标志值。若返回的标志值为 0, 则表示求解失败(如系数矩阵奇异); 若返回的标志值不为 0, 则表示正常返回。

本函数要调用全选主元高斯消去法求解线性代数方程组的函数 agaus, 参看 1.1 节。

本函数还要调用实矩阵相乘的函数 brmul, 参看 2.1 节。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放方程组的系数矩阵。

n——整型变量。方程组的阶数。

b——双精度实型一维数组, 长度为 n 。存放方程组右端的常数向量。

eps——双精度实型变量。控制精度要求。

x——双精度实型一维数组, 长度为 n 。返回方程组的解向量。

五、函数程序(文件名: abint.c)

六、例

求解下列 4 阶病态方程组

$$\begin{bmatrix} 3.4336 & -0.5238 & 0.67105 & -0.15272 \\ -0.5238 & 3.28326 & -0.73051 & -0.2689 \\ 0.67105 & -0.73051 & 4.02612 & 0.01835 \\ -0.15272 & -0.2689 & 0.01835 & 2.75702 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.5 \\ 2.5 \\ -2.0 \end{bmatrix}$$

取 $\epsilon = 0.000001$ 。

主函数程序(文件名: abint 0.c)如下:

```

#include "stdio.h"
#include "abint.c"
#include "brmul.c"
#include "agaus.c"
main()
{
    int k;
    double eps=0.000001,x[4];
    static double a[4][4]=
        {{3.4336,-0.5238,0.67105,-0.15272},
         {-0.5238,3.28326,-0.73051,-0.2689},
         {0.67105,-0.73051,4.02612,0.01835},
         {-0.15272,-0.2689,0.01835,2.75702}};
    static double b[4]={-1.0,1.5,2.5,-2.0};
    if (abint(a,b,eps,x)!=0)
        printf("\n");
    for (k=0;k<=3;k++)
        printf("x[%d] = %13.7e\n",k,x[k]);
    printf("\n");
}

```

运行结果为：

```

x[0] = -3.977180e-01
x[1] = 5.100536e-01
x[2] = 7.829837e-01
x[3] = -7.029161e-01

```

第2章 矩阵运算

2.1 实矩阵相乘

一、功能

求 $m \times n$ 阶矩阵 A 与 $n \times k$ 阶矩阵 B 的乘积矩阵 $C = AB$ 。

二、方法说明

乘积矩阵 C 中的各元素为

$$c_{ij} = \sum_{t=0}^{n-1} a_{it} b_{tj}, i = 0, 1, \dots, m-1; j = 0, 1, \dots, k-1$$

三、函数语句

```
void brmul (a, b, m, n, k, c)
```

四、形参说明

a ——双精度实型二维数组，体积为 $m \times n$ 。存放矩阵 A 的元素。

b ——双精度实型二维数组，体积为 $n \times k$ 。存放矩阵 B 的元素。

m ——整型变量。矩阵 A 的行数，也是乘积矩阵 $C = AB$ 的行数。

n ——整型变量。矩阵 A 的列数，也是矩阵 B 的行数。

k ——整型变量。矩阵 B 的列数，也是乘积矩阵 $C = AB$ 的列数。

c ——双精度实型二维数组，体积为 $m \times k$ 。返回乘积矩阵 $C = AB$ 的元素。

五、函数程序(文件名：brmul.c)

六、例

求 4×5 阶矩阵 A 与 5×3 阶矩阵 B 的乘积矩阵 $C = AB$ 。其中

$$A = \begin{bmatrix} 1 & 3 & -2 & 0 & 4 \\ -2 & -1 & 5 & -7 & 2 \\ 0 & 8 & 4 & 1 & -5 \\ 3 & -3 & 2 & -4 & 1 \end{bmatrix}, B = \begin{bmatrix} 4 & 5 & 1 \\ 2 & -2 & 6 \\ 7 & 8 & 1 \\ 0 & 3 & -5 \\ 9 & 8 & -6 \end{bmatrix}$$

主函数程序(文件名：brmul 0.c)如下：

```

#include "stdio.h"
#include "brmul.c"
main()
{
    int i, j;
}

```

```

static double a[4][5] = { {1.0, 3.0, -2.0, 0.0, 4.0},
                        {-2.0, -1.0, 5.0, -7.0, 2.0},
                        {0.0, 8.0, 4.0, 1.0, -5.0},
                        {3.0, -3.0, 2.0, -4.0, 1.0} };
static double c[4][3], b[5][3] = { {4.0, 5.0, -1.0},
                                  {2.0, -2.0, 6.0}, {7.0, 8.0, 1.0},
                                  {0.0, 3.0, -5.0}, {9.0, 8.0, -6.0} };
bcmul(a, b, 4, 5, 3, c);
for (i = 0; i < 3; i++)
    for (j = 0; j <= 2; j++)
        printf("%13.7e", c[i][j]);
    printf("\n");
}
printf("\n");

```

运行结果为：

```

3.200000e+01 1.500000e+01 -9.000000e+00
4.300000e+01 2.700000e+01 2.400000e+01
-1.000000e+00 -2.100000e+01 7.700000e+01
2.900000e+01 3.300000e+01 -5.000000e+00

```

2.2 复矩阵相乘

一、功能

求 $m \times n$ 阶复矩阵 A 与 $n \times k$ 阶复矩阵 B 的乘积矩阵 $C = AB$ 。其中

$$A = AR + jAI$$

$$B = BR + jBI$$

$$C = CR + jCI$$

二、方法说明

基本方法同 2.1 节。其中两个复数相乘采用如下算法：

设

$$e + jf = (a + jb)(c + jd)$$

令

$$p = ac, q = bd, s = (a + b)(c + d)$$

则

$$e = p - q, f = s - p - q$$

三、函数语句

void bcmul(ar, ai, br, bi, m, n, k, cr, ci)

四、形参说明

ar——双精度实型二维数组，体积为 $m \times n$ 。存放矩阵 A 的实部。

ai——双精度实型二维数组，体积为 $m \times n$ 。存放矩阵 A 的虚部。

br——双精度实型二维数组，体积为 $n \times k$ 。存放矩阵 B 的实部。

bi——双精度实型二维数组，体积为 $n \times k$ 。存放矩阵 B 的虚部。

m——整型变量。矩阵 A 的行数，也是乘积矩阵 C 的行数。

n——整型变量。矩阵 A 的列数，也是矩阵 B 的行数。

k——整型变量。矩阵 B 的列数，也是乘积矩阵 C 的列数。

cr——双精度实型二维数组，体积为 $m \times k$ 。返回乘积矩阵 C 的实部。

ci——双精度实型二维数组，体积为 $m \times k$ 。返回乘积矩阵 C 的虚部。

五、函数程序(文件名：bcmul.c)

六、例

求 3×4 阶复矩阵 A 与 4×4 阶复矩阵 B 的乘积矩阵 $C = AB$ 。其中

$$A = \begin{bmatrix} 1+j & 2-j & 3+2j & -2+j \\ 1-j & 5-j & 1+2j & 3+0j \\ 0-3j & 4-j & 2+2j & -1+2j \end{bmatrix}$$

$$B = \begin{bmatrix} 1-j & 4-j & 5+j & -2+j \\ 3+2j & 0+j & 2+0j & -1+5j \\ 6-3j & 3+2j & 1+j & 2-j \\ 2-j & -3-2j & -2+j & 1-2j \end{bmatrix}$$

主函数程序(文件名：bcmul 0.c)如下：

```

#include "stdio.h"
#include "bcmul.c"
main()
{
    int i, j;
    static double cr[3][4], ci[3][4];
    static double ar[3][4] = { {1.0, 2.0, 3.0, -2.0},
                             {1.0, 5.0, 1.0, 3.0}, {0.0, 4.0, 2.0, -1.0} };
    static double ai[3][4] = { {1.0, -1.0, 2.0, 1.0},
                             {-1.0, -1.0, 2.0, 0.0}, {-3.0, -1.0, 2.0, 2.0} };
    static double br[4][4] = { {1.0, 4.0, 5.0, -2.0},
                             {3.0, 0.0, 2.0, -1.0}, {6.0, 3.0, 1.0, 2.0},
                             {2.0, -3.0, -2.0, 1.0} };
    static double bi[4][4] = { {-1.0, -1.0, 1.0, 1.0},
                             {2.0, 1.0, 0.0, 5.0}, {-3.0, 2.0, 1.0, -1.0},
                             {-1.0, -2.0, 1.0, -2.0} };
    bcmul(ar, ai, br, bi, 3, 4, 4, cr, ci);
    for (i = 0; i <= 2; i++)
        for (j = 0; j <= 3; j++)
            printf("%13.7e", cr[i][j]);
        printf("\n");
    printf("\n");
    for (i = 0; i <= 2; i++)
        for (j = 0; j <= 3; j++)
            printf("%13.7e", ci[i][j]);
        printf("\n");
    }

```

运行结果为：

乘积矩阵 C 的实部

3.100000e+01	1.900000e+01	1.200000e+01	8.000000e+00
3.500000e+01	-6.000000e+00	9.000000e+00	6.000000e+00
2.900000e+01	7.000000e+00	1.100000e+01	1.300000e+01

乘积矩阵 C 的虚部

8.000000e+00	1.800000e+01	5.000000e+00	1.600000e+01
1.100000e+01	2.000000e+00	0.000000e+00	2.600000e+01
1.300000e+01	-2.000000e+00	-1.800000e+01	3.300000e+01

准确结果为：

$$C = \begin{bmatrix} 31 + 8j & 19 + 18j & 12 + 5j & 8 + 16j \\ 35 + 11j & -6 + 2j & 9 + 0j & 6 + 26j \\ 29 + 13j & 7 - 2j & 11 - 18j & 13 + 33j \end{bmatrix}$$

2.3 实矩阵求逆的全选主元高斯-约当法

一、功能

用全选主元高斯-约当(Gauss-Jordan)消去法求 n 阶实矩阵 A 的逆矩阵 A^{-1} 。

二、方法说明

高斯-约当法(全选主元)求逆的步骤如下。

首先, 对于 k 从 0 到 $n-1$ 作如下几步:

(1) 从第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素, 并记住此元素所在的行号和列号, 再通过行交换与列交换将它交换到主元素位置上。这一步称为全选主元。

(2) $1/a_{kk} \Rightarrow a_{kk}$

(3) $a_{kj}a_{kk} \Rightarrow a_{kj}, j = 0, 1, \dots, n-1; j \neq k$

(4) $a_{ij} - a_{ik}a_{kj} \Rightarrow a_{ij}, i, j = 0, 1, \dots, n-1; i, j \neq k$

(5) $-a_{ik}a_{kk} \Rightarrow a_{ik}, i = 0, 1, \dots, n-1; i \neq k$

最后, 根据在全选主元过程中所记录的行、列交换的信息进行恢复, 恢复的原则如下:

在全选主元过程中, 先交换的行、列后进行恢复; 原来的行(列)交换用列(行)交换来恢复。

三、函数语句

```
int brinv(a, n)
```

本函数返回一个整型标志值。若返回的标志值为 0, 则表示矩阵 A 奇异, 还输出信息“err * * not inv”; 若返回的标志值不为 0, 则表示正常返回。

四、形参说明

a ——双精度实型二维数组, 体积为 $n \times n$ 。存放原矩阵 A ; 返回时存放其逆矩阵 A^{-1} 。

n ——整型变量。矩阵的阶数。

五、函数程序(文件名: brinv.c)

六、例

求下列 4 阶矩阵的逆矩阵, 并计算 AA^{-1} 以检验其结果的正确性:

$$A = \begin{bmatrix} 0.2368 & 0.2471 & 0.2568 & 1.2671 \\ 1.1161 & 0.1254 & 0.1397 & 0.1490 \\ 0.1582 & 1.1675 & 0.1768 & 0.1871 \\ 0.1968 & 0.2071 & 1.2168 & 0.2271 \end{bmatrix}$$

主函数程序(文件名: brinv 0.c)如下:

```
# include "stdio.h"
# include "brinv.c"
# include "brmul.c"
main()
{ int i, j;
  static double a[4][4] = { {0.2368, 0.2471, 0.2568, 1.2671},
                            {1.1161, 0.1254, 0.1397, 0.1490},
                            {0.1582, 1.1675, 0.1768, 0.1871},
                            {0.1968, 0.2071, 1.2168, 0.2271} };
  static double b[4][4], c[4][4];
  for (i=0; i<=3; i++)
    for (j=0; j<=3; j++)
      b[i][j]=a[i][j];
  i=brinv(a, 4);
  if (i!=0)
    { printf("MAT A IS: \n");
      for (i=0; i<=3; i++)
        { for (j=0; j<=3; j++)
          printf("%13.7e", b[i][j]);
          printf("\n");
        }
      printf("\n");
      printf("MAT A- IS: \n");
      for (i=0; i<=3; i++)
        { for (j=0; j<=3; j++)
          printf("%13.7e", a[i][j]);
          printf("\n");
        }
      printf("\n");
      printf("MAT AA- IS: \n");
      brmul(b, a, 4, 4, c);
      for (i=0; i<=3; i++)
    }
```

```

    { for (j=0; j<=3; j++)
        printf("%13.7e", c[i][j]);
        printf("\n");
    }
}

```

运行结果为：

```

MAT A IS:
 2.368000e-01  2.471000e-01  2.568000e-01  1.267100e+00
 1.116100e+00  1.254000e-01  1.397000e-01  1.490000e-01
 1.582000e-01  1.167500e+00  1.768000e-01  1.871000e-01
 1.968000e-01  2.071000e-01  1.216800e+00  2.271000e-01

MAT A- IS:
 -8.592075e-02  9.379443e-01  -6.843720e-02  -7.960772e-02
 -1.055899e-01  -8.852432e-02  9.059826e-01  -9.919081e-02
 -1.270733e-01  -1.113511e-01  -1.169667e-01  8.784253e-01
 8.516058e-01  -1.354557e-01  -1.401826e-01  -1.438075e-01

MAT AA- IS:
 1.000000e+00  -3.375935e-17  -2.303930e-19  -1.111307e-17
 -7.589415e-18  1.000000e+00  -2.817232e-18  9.640929e-18
 -8.104411e-18  3.671549e-17  1.000000e+00  2.012211e-17
 -2.813505e-17  -1.471296e-17  -1.776397e-17  1.000000e+00

```

2.4 复矩阵求逆的全选主元高斯-约当法

一、功能

用全选主元高斯-约当(Gauss-Jordan)消去法求 n 阶复矩阵的逆矩阵。

二、方法说明

算法同实矩阵的求逆，见 2.3 节的方法说明。其中复矩阵分成实部与虚部两部分，即

$$A = AR + jAI$$

算法中的所有运算均为复数运算。有关复数相乘与复数相除的算法见 1.3 节的方法说明。

三、函数语句

```
int bcinv(ar, ai, n)
```

本函数返回一个整型标志值。若返回的标志值为 0，则表示原矩阵奇异，还输出信息“err * * not inv”；若返回的标志值不为 0，则表示正常返回。

四、形参说明

ar——双精度实型二维数组，体积为 $n \times n$ 。存放原矩阵 A 的实部；返回时存放逆矩阵

阵的实部。

ai——双精度实型二维数组，体积为 $n \times n$ 。存放原矩阵 A 的虚部；返回时存放逆矩阵的虚部。

n——整型变量。矩阵阶数。

五、函数程序(文件名：bcinv.c)

六、例

求下列 4 阶复矩阵 A 的逆矩阵 A^{-1} ，并计算 AA^{-1} 以检验结果的正确性：

$$A = AR + jAI$$

其中

$$AR = \begin{bmatrix} 0.236800 & 0.247100 & 0.256800 & 1.26710 \\ 1.11610 & 0.125400 & 0.139700 & 0.149000 \\ 0.158200 & 1.16750 & 0.176800 & 0.187100 \\ 0.196800 & 0.207100 & 1.21680 & 0.227100 \end{bmatrix}$$

$$AI = \begin{bmatrix} 0.134500 & 0.167800 & 0.182500 & 1.11610 \\ 1.26710 & 0.201700 & 0.702400 & 0.272100 \\ -0.283600 & -1.19670 & 0.355800 & -0.207800 \\ 0.357600 & -1.23450 & 2.11850 & 0.477300 \end{bmatrix}$$

主函数程序(文件名：bcinv 0.c)如下：

```

#include "stdio.h"
#include "bcinv.c"
#include "bcmul.c"

main()
{
    int i, j;
    static double br[4][4], bi[4][4], cr[4][4], ci[4][4];
    static double ar[4][4] = { 0.2368, 0.2471, 0.2568, 1.2671,
                             { 1.1161, 0.1254, 0.1397, 0.1490 },
                             { 0.1582, 1.1675, 0.1768, 0.1871 },
                             { 0.1968, 0.2071, 1.2168, 0.2271 } };
    static double ai[4][4] = { 0.1345, 0.1678, 0.1875, 1.1161,
                             { 1.2671, 0.2017, 0.7024, 0.2721 },
                             { -0.2836, -1.1967, 0.3558, -0.2078 },
                             { 0.3576, -1.2345, 2.1185, 0.4773 } };

    for (i=0; i<=3; i++)
        for (j=0; j<=3; j++)
            br[i][j] = ar[i][j]; bi[i][j] = ai[i][j];
    i = bcinv(ar, ai, 4);
    if (i != 0)
        printf("MAT AR IS: \n");
        for (i=0; i<=3; i++)
            for (j=0; j<=3; j++)
                printf("%13.7e", br[i][j]);
                printf("\n");
}

```

```

printf("\n");
printf("MAT AI IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
| printf("%13.7e", bi[i][j]);
| printf("\n");
|
printf("\n");
printf("MAT AR- IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
| printf("%13.7e", ar[i][j]);
| printf("\n");
|
printf("\n");
printf("MAT AI- IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
| printf("%13.7e", ai[i][j]);
| printf("\n");
|
bcmul (br,bi,ar,ai,4,4,4,cr,ci);
printf("\n");
printf("MAT CR = AA- IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
| printf("%13.7e", cr[i][j]);
| printf("\n");
|
printf("\n");
printf("MAT CI = AA- IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
| printf("%13.7e", ci[i][j]);
| printf("\n");
|
}

```

运行结果为：

```

MAT AR IS:
2.368000e-01 2.471000e-01 2.568000e-01 1.267100e+00
1.116100e+00 1.254000e-01 1.397000e-01 1.490000e-01
1.582000e-01 1.167500e+00 1.768000e-01 1.871000e-01
1.968000e-01 2.071000e-01 1.216800e+00 2.271000e-01

MAT AI IS:
1.345000e-01 1.678000e-01 1.875000e-01 1.116100e+00
1.267100e+00 2.017000e-01 7.024000e-01 2.721000e-01

```

```

- 2.836000e-01 - 1.196700e+00 3.558000e-01 - 2.078000e-01
3.576000e-01 - 1.234500e+00 2.118500e+00 4.773000e-01

MAT AR- IS:
- 5.669986e-03 4.851151e-01 2.166018e-02 - 1.874068e-01
- 6.996602e-02 - 4.714739e-02 5.546471e-01 - 5.583729e-02
- 1.763916e-01 - 1.421441e-01 7.371922e-02 2.620023e-01
4.848230e-01 - 3.106411e-02 - 1.258590e-01 - 2.498309e-03

MAT AI- IS:
4.506499e-02 - 4.816759e-01 - 2.382550e-01 1.211980e-01
1.162225e-01 1.487048e-01 5.124807e-01 - 1.430340e-01
1.032388e-01 1.142015e-01 4.515873e-01 - 4.689941e-01
- 4.430876e-01 4.101886e-02 - 1.227346e-01 9.101123e-02

MAT CR = AA- IS:
1.000000e+00 - 2.775558e-17 - 1.734723e-17 - 1.170938e-17
- 1.387779e-17 1.000000e+00 - 6.938894e-17 - 4.065758e-17
0.000000e+00 1.301043e-17 1.000000e+00 - 2.168404e-18
- 5.551115e-17 - 3.903128e-17 3.469447e-17 1.000000e+00

MAT CI = AA- IS:
3.122502e-17 2.081668e-17 1.110223e-16 - 2.168404e-18
- 1.387779e-17 1.474515e-17 - 1.040834e-17 - 2.873136e-17
1.019150e-17 2.913793e-17 2.133710e-16 - 1.973248e-17
2.081668e-17 - 1.474515e-17 3.677614e-16 2.699663e-17

```

2.5 对称正定矩阵的求逆

一、功能

求 n 阶对称正定矩阵 A 的逆矩阵 A^{-1} 。

二、方法说明

采用“变量循环重新编号法”，其计算公式如下：

$$\begin{aligned}
a'_{n-1,n-1} &= 1/a_{00} \\
a'_{n-1,j-1} &= -a_{0j}/a_{00}, j = 1, 2, \dots, n-1 \\
a'_{i-1,n-1} &= a_{i0}/a_{00}, i = 1, 2, \dots, n-1 \\
a'_{i-1,j-1} &= a_{ij} - a_{i0}a_{0j}/a_{00}, i, j = 1, 2, \dots, n-1
\end{aligned}$$

当 A 为对称正定矩阵时，其逆矩阵 A^{-1} 也是对称正定矩阵。

三、函数语句

int bssgi(a, n)

本函数返回一个整型标志值。若返回的标志值小于 0，则表示程序工作失败（如矩阵 A

非对称正定), 还输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放实对称正定矩阵 A; 返回时存放逆矩阵 A^{-1} 。

n——整型变量。矩阵阶数。

五、函数程序(文件名: bssgj.c)

六、例

求下列 4 阶对称正定矩阵 A 的逆矩阵 A^{-1} , 并计算 AA^{-1} 以检验结果的正确性:

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

主函数程序(文件名: bssgj 0.c)如下:

```
# include "stdio.h"
# include "bssgj.c"
# include "brmul.c"

main()
{
    int i, j;
    static double a[4][4] = { {5.0, 7.0, 6.0, 5.0},
                             {7.0, 10.0, 8.0, 7.0},
                             {6.0, 8.0, 10.0, 9.0},
                             {5.0, 7.0, 9.0, 10.0} };
    static double b[4][4], c[4][4];
    for (i=0; i<=3; i++)
        for (j=0; j<=3; j++)
            b[i][j] = a[i][j];
    i = bssgj(a, 4);
    if (i>0)
        printf ("MAT A IS: \n");
        for (i=0; i<=3; i++)
            for (j=0; j<=3; j++)
                printf ("%13.7e", b[i][j]);
            printf ("\n");
        printf ("\n");
        printf ("MAT A- IS: \n");
        for (i=0; i<=3; i++)
            for (j=0; j<=3; j++)
                printf ("%13.7e", a[i][j]);
            printf ("\n");
        printf ("\n");
        printf ("MAT AA- IS: \n");
}
```

```
for (i=0; i<=3; i++)
    for (j=0; j<=3; j++)
        printf ("%13.7e", c[i][j]);
    printf ("\n");
}
```

运行结果为:

MAT A IS:

5.000000e+00	7.000000e+00	6.000000e+00	5.000000e+00
7.000000e+00	1.000000e+01	8.000000e+00	7.000000e+00
6.000000e+00	8.000000e+00	1.000000e+01	9.000000e+00
5.000000e+00	7.000000e+00	9.000000e+00	1.000000e+01

MAT A- IS:

6.800000e+01	-4.100000e+01	-1.700000e+01	1.000000e+01
-4.100000e+01	2.500000e+01	1.000000e+01	-6.000000e+00
-1.700000e+01	1.000000e+01	5.000000e+00	-3.000000e+00
1.000000e+01	-6.000000e+00	-3.000000e+00	2.000000e+00

MAT AA- IS:

1.000000e+00	2.398082e-14	8.881784e-16	0.000000e+00
6.039613e-14	1.000000e+00	-1.509903e-14	-5.329071e-15
6.750156e-14	3.819167e-14	1.000000e+00	-3.552714e-15
7.105427e-15	1.243450e-14	-1.776357e-15	1.000000e+00

2.6 托伯利兹矩阵求逆的特兰持方法

一、功能

用特兰持(Trench)方法求托伯利兹(Toeplitz)矩阵的逆矩阵。

二、方法说明

设 n 阶托伯利兹矩阵为

$$T^{(n)} = \begin{bmatrix} t_0 & t_1 & t_2 & \dots & t_{n-1} \\ t_1 & t_0 & t_1 & \dots & t_{n-2} \\ t_2 & t_1 & t_0 & \dots & t_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & t_{n-3} & \dots & t_0 \end{bmatrix}$$

简称 n 阶 T 型矩阵。其求逆过程如下。

取初值 $\alpha_0 = t_0$, $c_0^{(0)} = \tau_1/t_0$, $r_0^{(0)} = t_1/t_0$ 。

第一步 对于 k 从 0 到 $n - 3$ 作如下工作:

$$(1) \quad c_i^{(k+1)} = c_i^{(k)} + \frac{r_{k-i}^{(k)}}{\alpha_k} \left(\sum_{j=1}^{k+1} c_{k+1-j}^{(k)} \tau_j - \tau_{k+2} \right), i = 0, 1, \dots, k$$

$$c_{k+1}^{(k+1)} = \frac{1}{\alpha_k} \left(\tau_{k+2} - \sum_{j=1}^{k+1} c_{k+1-j}^{(k)} \tau_j \right)$$

$$(2) \quad r_i^{(k+1)} = r_i^{(k)} + \frac{c_{k-i}^{(k)}}{\alpha_k} \left(\sum_{j=1}^{k+1} r_{k+1-j}^{(k)} t_j - t_{k+2} \right), i = 0, 1, \dots, k$$

$$r_{k+1}^{(k+1)} = \frac{1}{\alpha_k} \left(t_{k+2} - \sum_{j=1}^{k+1} r_{k+1-j}^{(k)} t_j \right)$$

$$(3) \quad a_{k+1} = t_0 - \sum_{j=1}^{k+2} t_j c_{j-1}^{(k+1)}$$

最后可算出 a_{n-2} 以及 $c_i^{(n-2)}$ 和 $r_i^{(n-2)}$, ($i = 0, 1, \dots, n-2$)。

第二步 计算逆矩阵 $B^{(n)}$ 中的各元素:

$$b_{00} = 1/a_{n-2}$$

$$b_{0,j+1} = -r_j^{(n-2)}/a_{n-2}, j = 0, 1, \dots, n-2$$

$$b_{i+1,0} = -c_i^{(n-2)}/a_{n-2}, i = 0, 1, \dots, n-2$$

$$b_{i+1,j+1} = b_{ij} + \frac{1}{a_{n-2}} [c_i^{(n-2)} r_j^{(n-2)} - r_{n-i-2}^{(n-2)} c_{n-j-2}^{(n-2)}], i, j = 0, 1, \dots, n-2$$

三、函数语句

int btrch(t, tt, n, b)

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示程序工作失败, 输出信息 "fail"; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

t——双精度实型一维数组, 长度为 n 。存放 n 阶 T 型矩阵中的上三角元素 t_0, t_1, \dots, t_{n-1}

tt——双精度实型一维数组, 长度为 n 。其中后 $n-1$ 个元素 $tt(1), \dots, tt(n-1)$ 依次存放 n 阶 T 型矩阵中的元素 $\tau_1, \dots, \tau_{n-1}$ 。

n——整型变量。 T 型矩阵的阶数。

b——双精度实型二维数组, 体积为 $n \times n$ 。返回 n 阶 T 型矩阵的逆矩阵。

五、函数程序(文件名: btrch.c)

六、例

求下列 6 阶 T 型矩阵 $T^{(6)}$ 的逆矩阵 B , 并计算 $A = T^{(6)}B$ 以检验结果的正确性:

$$T^{(6)} = \begin{bmatrix} 10 & 5 & 4 & 3 & 2 & 1 \\ -1 & 10 & 5 & 4 & 3 & 2 \\ -2 & -1 & 10 & 5 & 4 & 3 \\ -3 & -2 & -1 & 10 & 5 & 4 \\ -4 & -3 & -2 & -1 & 10 & 5 \\ -5 & -4 & -3 & -2 & -1 & 10 \end{bmatrix}$$

其中

$$T = (10, 5, 4, 3, 2, 1)$$

$$TT = (0, -1, -2, -3, -4, -5)$$

$$n = 6$$

主函数程序(文件名: btrch 0.c)如下:

```
# include "stdio.h"
# include "btrch.c"
main()
{ int n, i, j, k;
  static double t[6] = {10.0, 5.0, 4.0, 3.0, 2.0, 1.0};
  static double tt[6] = {0.0, -1.0, -2.0, -3.0, -4.0, -5.0};
  static double b[6][6], a[6][6];
  n=6;
  i=btrch(t, tt, n, b);
  if (i>0)
    { printf("B=inv(T):\n");
      for (i=0; i<=5; i++)
        { for (j=0; j<=5; j++)
          { printf("%12.6e", b[i][j]);
            printf("\n");
          }
        printf("\n");
      printf("A=T*B:\n");
      for (i=1; i<=6; i++)
        for (j=1; j<=6; j++)
          { a[i-1][j-1]=0.0;
            for (k=1; k<=j-1; k++)
              a[i-1][j-1]=a[i-1][j-1]+b[i-1][k-1]*t[j-k];
            a[i-1][j-1]=a[i-1][j-1]+b[i-1][j-1]*t[0];
            for (k=j+1; k<=6; k++)
              a[i-1][j-1]=a[i-1][j-1]+b[i-1][k-1]*tt[k-j];
          }
        for (i=0; i<=5; i++)
          { for (j=0; j<=5; j++)
            { printf("%12.6e", a[i][j]);
              printf("\n");
            }
          }
        }
    }
}
```

运行结果为:

```
B=inv(T):
9.46884e-02 -4.69953e-02 -1.37211e-02 -1.77270e-03 1.90745e-03 3.80190e-03
-4.27534e-03 9.49232e-02 -4.69003e-02 -1.37122e-02 -1.81915e-03 1.90745e-03
-9.88143e-04 -4.73167e-03 9.48032e-02 -4.69175e-02 -1.37122e-02 -1.77270e-03
1.77270e-03 -9.88082e-04 -4.74397e-03 9.48032e-02 -4.69003e-02 -1.37211e-02
1.30644e-02 2.09908e-03 -9.88082e-04 -4.73167e-03 9.49232e-02 -4.69953e-02
```

```

4.69986e-02 1.30644e-02 1.77270e-03 -9.88143e-04 -4.27534e-03 9.46884e-02
A = T * B
1.00000e+00 -1.74668e-04 -6.88163e-03 2.40309e-04 -8.43695e-05 -1.30104e-17
-5.00000e-03 1.00001e+00 1.39531e-04 -6.75264e-03 -4.23289e-05 -3.72966e-17
9.70874e-03 -4.99818e-03 1.00008e+00 2.07183e-04 -6.89547e-03 -1.77809e-17
4.85723e-17 9.70547e-03 -5.12863e-03 1.00002e+00 3.39164e-04 -6.93481e-03
9.71445e-17 -2.40994e-05 8.74343e-03 -5.60017e-03 1.00104e+00 3.46741e-05
2.77556e-17 -8.66964e-05 -3.48283e-03 7.13412e-03 -2.10127e-03 1.00000e+00

```

2.7 求行列式值的全选主元高斯消去法

一、功能

用全选主元高斯(Gauss)消去法计算 n 阶方阵 A 所对应的行列式值。

二、方法说明

用高斯消去法对方阵 A 进行一系列变换,使之成为上三角矩阵,其主对角线上的各元素的乘积即为行列式的值。

变换过程如下:

对于 $k = 0, 1, \dots, n-2$ 作变换

$$a_{ij} - a_{ik}a_{kj}/a_{kk} \Rightarrow a_{ij}, \quad i, j = k+1, \dots, n-1$$

为保证数值计算的稳定性,在实际变换过程中采用全选主元。

三、函数语句

double bsdet(a, n)

本函数返回一个双精度实型函数值,即行列式值。

四、形参说明

a ——双精度实型二维数组,体积为 $n \times n$ 。存放 n 阶方阵 A ,在本函数中要被破坏。

n ——整型变量。方阵 A 的阶数。

五、函数程序(文件名: bsdet.c)

六、例

求下列两个方阵 A 与 B 的行列式值 $\det(A)$ 与 $\det(B)$:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & -3 & -2 & 4 \\ 5 & -5 & 1 & 8 \\ 11 & 8 & 5 & -7 \\ 5 & -1 & -3 & -1 \end{bmatrix}$$

主函数程序(文件名: bsdet 0.c)如下:

```
# include "stdio.h"
```

```

# include "bsdet.c"
main()
{
    static double a[4][4] = { {1.0, 2.0, 3.0, 4.0},
                             {5.0, 6.0, 7.0, 8.0},
                             {9.0, 10.0, 11.0, 12.0},
                             {13.0, 14.0, 15.0, 16.0} };
    static double b[4][4] = { {3.0, -3.0, -2.0, 4.0},
                            {5.0, -5.0, 1.0, 8.0},
                            {11.0, 8.0, 5.0, -7.0},
                            {5.0, -1.0, -3.0, -1.0} };

    double det;
    det = bsdet(a, 4);
    printf("det (A) = %13.7e\n", det);
    printf("\n");
    det = bsdet(b, 4);
    printf("det (B) = %13.7e\n", det);
    printf("\n");
}

```

运行结果为:

```

det(A) = 0.000000e+00
det(B) = 5.950000e+02

```

2.8 求矩阵秩的全选主元高斯消去法

一、功能

用全选主元高斯(Gauss)消去法计算矩阵 A 的秩。

二、方法说明

设有 $m \times n$ 阶矩阵

$$A = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0,n-1} \\ a_{10} & a_{11} & \dots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1,n-1} \end{bmatrix}$$

取 $k = \min\{m, n\}$ 。对于 $r = 0, 1, \dots, k-1$, 用全选主元高斯消去法(参看 1.1 节)将 A 变为上三角矩阵,直到某次 $a_{rr} = 0$ 为止,矩阵 A 的秩为 r 。

三、函数语句

int brank(a, m, n)

本函数返回一个整型值,即矩阵 A 的秩。

四、形参说明

a ——双精度实型二维数组,体积为 $m \times n$ 。存放 $m \times n$ 阶矩阵 A ,返回时该矩阵将被

破坏。

m——整型变量。矩阵 A 的行数。

n——整型变量。矩阵 A 的列数。

五、函数程序(文件名: brank.c)

六、例

求下列 5×4 阶矩阵 A 的秩:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \end{bmatrix}$$

主函数程序(文件名: brank 0.c)如下:

```
# include "stdio.h"
# include "brank.c"
main()
{ static double a[5][4] = { {1.0, 2.0, 3.0, 4.0},
                           {5.0, 6.0, 7.0, 8.0},
                           {9.0, 10.0, 11.0, 12.0},
                           {13.0, 14.0, 15.0, 16.0},
                           {17.0, 18.0, 19.0, 20.0}};

  printf("\n");
  printf("RANK = %d\n", brank(a, 5, 4));
  printf("\n");
}
```

运行结果为:

RANK = 4

2.9 对称正定矩阵的乔里斯基分解与行列式的求值

一、功能

用乔里斯基(Cholesky)分解法求对称正定矩阵的三角分解，并求行列式值。

二、方法说明

设 n 阶矩阵 A 为对称正定，则存在一个实的非奇异的下三角阵 L，使 $A = LL^T$ ，其中

$$L = \begin{bmatrix} l_{00} & & & \\ l_{10} & l_{11} & 0 & \\ \vdots & \vdots & \ddots & \\ l_{n-1,0} & l_{n-1,1} & \dots & l_{n-1,n-1} \end{bmatrix}$$

乔里斯基分解的步骤为：

对于 $j = 0, 1, \dots, n-1$

$$(1) \quad l_{jj} = \left(a_{jj} - \sum_{k=0}^{j-1} l_{jk}^2 \right)^{\frac{1}{2}}$$

$$(2) \quad l_{ij} = \left(a_{ij} - \sum_{k=0}^{j-1} l_{ik} l_{jk} \right) / l_{jj}, \quad i = j+1, \dots, n-1$$

$$A \text{ 的行列式值为 } \det(A) = \left(\prod_{k=0}^{n-1} l_{kk} \right)^2$$

三、函数语句

int bchol(a, n, det)

本函数返回一个整型标志值。若返回的标志值小于 0，则表示程序工作失败(如矩阵 A 非对称正定)，还输出信息“fail”；若返回的标志值大于 0，则表示正常返回。

四、形参说明

a——双精度实型二维数组，体积为 $n \times n$ 。存放 n 阶对称正定矩阵 A；返回时，其下三角部分存放分解后的下三角矩阵 L，其余元素均为零。

n——整型变量。对称正定矩阵 A 的阶数。

det——双精度实型变量指针。在该指针指向的变量中返回行列式值。

五、函数程序(文件名: bchol.c)

六、例

求下列 4 阶对称正定矩阵 A 的乔里斯基分解式，并求行列式值 $\det(A)$:

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

主函数程序(文件名: bchol 0.c)如下:

```
# include "stdio.h"
# include "bchol.c"
main()
{ int i, j;
  double det;
  static double a[4][4] = { {5.0, 7.0, 6.0, 5.0},
                           {7.0, 10.0, 8.0, 7.0},
                           {6.0, 8.0, 10.0, 9.0},
                           {5.0, 7.0, 9.0, 10.0}};

  i = bchol(a, 4, &det);
  if (i > 0)
    { printf("MAT L IS: \n");
      for (i = 0; i < = 3; i++)
        { for (j = 0; j < = 3; j++)
          printf("%13.7e", a[i][j]);
    }
  }
}
```

```

    printf("\n");
    |
    printf("\n");
    printf("det(A) = %13.7e\n", det);
}

```

运行结果为：

```

MAT L IS:
2.236068e+00 0.000000e+00 0.000000e+00 0.000000e+00
3.130495e+00 4.472136e-01 0.000000e+00 0.000000e+00
2.683282e+00 -8.944272e-01 1.414214e+00 0.000000e+00
2.236068e+00 -6.332402e-16 2.121320e+00 7.071068e-01
det(A) = 1.000000e+00

```

2.10 矩阵的三角分解

一、功能

对 n 阶实矩阵进行 LU 分解。即 $A = LU$, 其中

$$L = \begin{bmatrix} 1 & & & \\ l_{10} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{k0} & l_{k1} & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ l_{n-1,0} & l_{n-1,1} & \cdots & l_{n-1,k} & \cdots & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{00} & u_{01} & \cdots & u_{0,n-1} \\ u_{11} & \cdots & u_{1,n-1} \\ \vdots & & \vdots \\ u_{n-1,n-1} \end{bmatrix}$$

二、方法说明

令

$$Q = L + U - I_n = \begin{bmatrix} u_{00} & u_{01} & \cdots & u_{0k} & \cdots & u_{0,n-1} \\ l_{10} & u_{11} & \cdots & u_{1k} & \cdots & u_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ l_{k0} & l_{k1} & \cdots & u_{kk} & \cdots & u_{k,n-1} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ l_{n-1,0} & l_{n-1,1} & \cdots & l_{n-1,k} & \cdots & u_{n-1,n-1} \end{bmatrix}$$

则 LU 分解问题变为求 Q 矩阵的问题。

由 n 阶实矩阵求矩阵 Q 的计算步骤为：

对于 $k = 0, 1, \dots, n-2$

(1) $a_{ik}/a_{kk} \Rightarrow a_{ik}, i = k+1, \dots, n-1$

(2) $a_{ij} - a_{ik}a_{kj} \Rightarrow a_{ij}, i, j = k+1, \dots, n-1$

由于本方法没有选主元，因此数值计算是不稳定的。

三、函数语句

```
int blluu(a, n, l, u)
```

本函数返回一个整型标志值。若返回的标志值为 0，则表示程序工作失败（如 $|a_{kk}| \approx 0$ ），输出信息“fail”；若返回的标志值不为 0，则表示正常返回。

四、形参说明

a ——双精度实型二维数组，体积为 $n \times n$ 。存放 n 阶实矩阵 A ；返回时存放 Q 矩阵。

n ——整型变量。矩阵 A 的阶数。

l ——双精度实型二维数组，体积为 $n \times n$ 。返回时存放下三角矩阵 L 。

u ——双精度实型二维数组，体积为 $n \times n$ 。返回时存放上三角矩阵 U 。

五、函数程序(文件名：blluu.c)

六、例

求下列 4 阶矩阵的 LU 分解：

$$A = \begin{bmatrix} 2 & 4 & 4 & 2 \\ 3 & 3 & 12 & 6 \\ 2 & 4 & -1 & 2 \\ 4 & 2 & 1 & 1 \end{bmatrix}$$

主函数程序(文件名：blluu 0.c)如下：

```
#include "stdio.h"
#include "blluu.c"
main()
| int i, j;
static double l[4][4], u[4][4];
static double a[4][4] = { {2.0, 4.0, 4.0, 2.0},
                        {3.0, 3.0, 12.0, 6.0},
                        {2.0, 4.0, -1.0, 2.0},
                        {4.0, 2.0, 1.0, 1.0} };
i = blluu(a, 4, l, u);
if (i != 0)
| printf("MAT L IS: \n");
for (i=0; i<=3; i++)
| for (j=0; j<=3; j++)
printf("%13.7e", l[i][j]);
printf("\n");
printf("\n");
```

```

printf("MAT U IS: \n");
for (i=0; i<=3; i++)
    { for (j=0; j<=3; j++)
        printf("%13.7e", u[i][j]);
        printf("\n");
    }
printf("\n");
}

```

运行结果为：

```

MAT L IS:
1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
1.500000e+00 1.000000e+00 0.000000e+00 0.000000e+00
1.000000e+00 -0.000000e+00 1.000000e+00 0.000000e+00
2.000000e+00 2.000000e+00 3.800000e+00 1.000000e+00

MAT U IS:
2.000000e+00 4.000000e+00 4.000000e+00 2.000000e+00
0.000000e+00 -3.000000e+00 6.000000e+00 3.000000e+00
0.000000e+00 0.000000e+00 -5.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 -9.000000e+00

```

2.11 一般实矩阵的 QR 分解

一、功能

用豪斯荷尔德(Householder)变换对一般 $m \times n$ 阶的实矩阵进行 QR 分解。

二、方法说明

设 $m \times n$ 的实矩阵 A 列线性无关，则可以将 A 分解为 $A = QR$ 的形式。其中 Q 为 $m \times m$ 的正交矩阵， R 为 $m \times n$ 的上三角矩阵。

豪斯荷尔德方法对 A 进行 QR 分解的步骤如下。

令 $s = \min(m-1, n)$, $Q = I_{m \times m}$ 。

对于 k 从 0 到 $s-1$ 作以下几步：

(1) 确定豪斯荷尔德变换

$$Q^{(k)} = \begin{bmatrix} I_k & | & 0 \\ 0 & | & \tilde{Q}_{m-k} \end{bmatrix}$$

其中

$$\tilde{Q}_{m-k} = \begin{bmatrix} (1-2u_k^2) & -2u_ku_{k+1} & \cdots & -2u_ku_{m-1} \\ -2u_{k+1}u_k & (1-2u_{k+1}^2) & \cdots & -2u_{k+1}u_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ -2u_{m-1}u_k & -2u_{m-1}u_{k+1} & \cdots & (1-2u_{m-1}^2) \end{bmatrix}$$

矩阵中 $u_j (j = k, k+1, \dots, m-1)$ 的计算公式如下：

$$\eta = \max_{k \leq i \leq m-1} |a_{ik}|$$

$$\alpha = -\text{sign}(a_{kk}) \sqrt{\sum_{i=k}^{m-1} (a_{ik}/\eta)^2} \cdot \eta$$

$$\rho = \sqrt{2\alpha(\alpha - a_{kk})}$$

$$u_k = \frac{1}{\rho}(a_{kk} - \alpha) \Rightarrow a_{kk}$$

$$u_i = \frac{1}{\rho}a_{ik} \Rightarrow a_{ik}, i = k+1, \dots, m-1$$

(2) 用 $Q^{(k)}$ 左乘 Q 。即 $Q^{(k)}Q \Rightarrow Q$

其计算公式如下：

$$\left. \begin{aligned} t &= \sum_{l=k}^{m-1} a_{lk} \cdot q_{lj} \\ q_{ij} - 2a_{ik}t &\Rightarrow a_{ij} \end{aligned} \right\} \begin{aligned} j &= 0, 1, \dots, m-1 \\ i &= k, k+1, \dots, m-1 \end{aligned}$$

(3) 用 $Q^{(k)}$ 左乘 A 。即 $Q^{(k)}A \Rightarrow A$

其计算公式如下：

$$\left. \begin{aligned} t &= \sum_{l=k}^{m-1} a_{lk} \cdot a_{lj} \\ a_{ij} - 2a_{ik}t &\Rightarrow a_{ij} \end{aligned} \right\} \begin{aligned} j &= k+1, \dots, n-1 \\ i &= k, \dots, m-1 \\ \alpha &\Rightarrow a_{kk} \end{aligned}$$

最后，矩阵 A 的右上三角部分即为 R ，而矩阵 Q^T 为正交矩阵。

三、函数语句

int bmaqr(a, m, n, q)

本函数返回一个整型标志值。若返回的标志值为 0，则表示程序工作失败(矩阵 A 列线性相关)，输出信息“fail”；若返回的标志值不为 0，则表示正常返回。

四、形参说明

a ——双精度实型二维数组，体积为 $m \times n$ 。存放 $m \times n$ 的实矩阵 A ；返回时，其右上三角部分存放上三角矩阵 R 。

m ——整型变量。实矩阵 A 的行数。

n ——整型变量。实矩阵 A 的列数。

q ——双精度实型二维数组，体积为 $m \times m$ 。返回时存放正交矩阵 Q 。

五、函数程序(文件名：bmaqr.c)

六、例

对下列 4×3 阶的矩阵 A 进行 QR 分解：

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 2 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 2 & 1 \end{bmatrix}$$

主函数程序(文件名: bmaqr0.c)如下:

```
# include "stdio.h"
# include "bmaqr.c"
main()
| int i, j;
static double q[4][4], a[4][3] = { {1.0, 1.0, -1.0},
{2.0, 1.0, 0.0}, {1.0, -1.0, 0.0}, {-1.0, 2.0, 1.0} };
i=bmaqr(a, 4, 3, q);
if (!i == 0)
| printf("MAT Q IS: \n");
for (i = 0; i < 3; i++)
| for (j = 0; j <= 3; j++)
printf("%13.7e", q[i][j]);
printf("\n");
|
printf("\n");
printf("MAT R IS: \n");
for (i = 0; i < 3; i++)
| for (j = 0; j <= 2; j++)
printf("%13.7e", a[i][j]);
printf("\n");
|
printf("\n");
|
```

运行结果为:

```
MAT Q IS:
-3.779645e-01 -3.779645e-01 7.559289e-01 3.779645e-01
-7.559289e-01 -3.779645e-01 -3.779645e-01 -3.779645e-01
-3.779645e-01 3.779645e-01 -3.779645e-01 -7.559289e-01
3.779645e-01 -7.559289e-01 -3.779645e-01 3.779645e-01

MAT R IS:
-2.645751e+00 -1.955901e-16 7.559289e-01
0.000000e+00 -2.645751e+00 -3.779645e-01
0.000000e+00 0.000000e+00 -1.133893e+00
0.000000e+00 0.000000e+00 0.000000e+00
```

2.12 一般实矩阵的奇异值分解

一、功能

用豪斯荷尔德(Householder)变换及变形 QR 算法对一般实矩阵进行奇异值分解。

二、方法说明

设 A 为 $m \times n$ 的实矩阵, 则存在一个 $m \times m$ 的列正交矩阵 U 和 $n \times n$ 的列正交矩阵 V , 使

$$A = U \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V^T$$

成立。其中 $\Sigma = \text{diag}(\sigma_0, \sigma_1, \dots, \sigma_p)$ ($p \leq \min(m, n) - 1$) 且 $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_p > 0$ 。

上式称为实矩阵 A 的奇异值分解式, σ_i ($i = 0, 1, \dots, p$) 称为 A 的奇异值。

利用 A 的奇异值分解式, 可以计算 A 的广义逆 A^+ (见 2.13 节)。利用 A 的广义逆可以求解线性最小二乘问题(见 1.14 节)。

奇异值分解分两大步:

第一步 用豪斯荷尔德变换将 A 约化为双对角线矩阵。即

$$B = \tilde{U}^T A \tilde{V} = \begin{bmatrix} s_0 & e_0 & & & \\ & s_1 & e_1 & 0 & \\ & & \ddots & \ddots & \\ & 0 & & s_{p-1} & e_{p-1} \\ & & & & s_p \end{bmatrix}$$

其中

$$\tilde{U} = U_0 U_1 \cdots U_{k-1}, k = \min(n, m - 1)$$

$$\tilde{V} = V_0 V_1 \cdots V_{l-1}, l = \min(m, n - 2)$$

\tilde{U} 中的每一个变换 U_j ($j = 0, 1, \dots, k - 1$) 将 A 中第 j 列主对角线以下的元素变为零; 而 \tilde{V} 中的每一个变换 V_j ($j = 0, 1, \dots, l - 1$) 将 A 中第 j 行中与主对角线紧邻的右次对角线元素右边的元素变为零。

对于每一个变换 V_j 具有如下形式:

$$I - \rho V_j V_j^T$$

其中 ρ 为一个比例因子, 以避免计算过程中的溢出现象与误差的积累。 V_j 是一个列向量, 即

$$V_j = (v_0, v_1, \dots, v_{n-1})^T$$

则

$$AV_j = A - \rho AV_j V_j^T = A - WV_j^T$$

其中

$$W = \rho AV_j = \rho \left(\sum_{i=0}^{n-1} v_i a_{0i}, \sum_{i=0}^{n-1} v_i a_{1i}, \dots, \sum_{i=0}^{n-1} v_i a_{m-1, i} \right)^T$$

第二步 用变形的 QR 算法进行迭代, 计算所有的奇异值。即: 用一系列的平面旋转变换将双对角线矩阵 B 逐步变成对角矩阵。

在每一次的迭代中, 用变换

$$B' = U_{p-1, p}^T \cdots U_{12}^T U_{01}^T BV_{01} V_{12} \cdots V_{m-2, m-1}$$

其中变换 $U_{j,j+1}^T$ 将 B 中第 j 列主对角线下的一个非零元素变为零, 同时在第 j 行的次对角

线元素的右边出现一个非零元素；而变换 $V_{j,j+1}$ 将第 $j - 1$ 行的次对角线元素右边的一个非零元素变为零，同时在第 j 列的主对角线元素的下方出现一个非零元素。由此可知，经过一次迭代 ($j = 0, 1, \dots, p - 1$) 后， B' 仍为双对角矩阵。但随着迭代的进行，最后收敛为对角矩阵，其对角线上的元素即为奇异值。

在每次迭代时，经过初始变换 V_{01} 后，将在第 0 列的主对角线下方出现一个非零元素。在变换 V_{01} 中，选择位移值 u 的计算公式如下：

$$b = [(s_{p-1} + s_p)(s_{p-1} - s_p) + e_{p-1}^2]/2$$

$$c = (s_p e_{p-1})^2$$

$$d = \text{sign}(b) \sqrt{b^2 + c}$$

$$u = s_p^2 - c/(b + d)$$

最后还需要对奇异值按非递增次序进行排序。

在上述变换过程中，若对于某个次对角线元素 e_j 满足 $|e_j| \leq \epsilon(|s_{j+1}| + |s_j|)$ ，则可认为 e_j 为零。

若对角线元素 s_j 满足 $|s_j| \leq \epsilon(|e_{j-1}| + |e_j|)$ ，则可认为 s_j 为零（即为零奇异值）。其中 ϵ 为给定的精度要求。

三、函数语句

```
int bmuav(a, m, n, u, v, eps, ka)
```

本函数返回一个整型标志值。若返回的标志值小于 0，则表示出现了迭代 60 次还未求得某个奇异值的情况，此时，矩阵的分解式为 UAV ；若返回的标志值大于 0，则表示正常返回。

四、形参说明

a ——双精度实型二维数组，体积为 $m \times n$ 。存放 $m \times n$ 的实矩阵 A ；返回时其对角线给出奇异值（以非递增次序排列），其余元素均为零。

m ——整型变量。实矩阵 A 的行数。

n ——整型变量。实矩阵 A 的列数。

u ——双精度实型二维数组，体积为 $m \times m$ 。返回时存放左奇异向量 U 。

v ——双精度实型二维数组，体积为 $n \times n$ 。返回时存放右奇异向量 V^T 。

ϵ ——双精度实型变量。给定的精度要求。

ka ——整型变量。其值为 $\max(m, n) + 1$ 。

五、函数程序(文件名：bmuav.c)

六、例

求下列两个矩阵 A 与 B 的奇异值分解式 UAV 与 UBV ：(取 $\epsilon = 0.000001$)

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 2 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 2 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 2 & 1 & 0 & 2 \\ 1 & -1 & 0 & 1 \end{bmatrix}$$

主函数程序(文件名：bmuav 0.c)如下：

```
# include "stdio.h"
# include "bmuav.c"
# include "brmul.c"
main()
{ int i, j;
  static double a[4][3] = { {1.0, 1.0, -1.0}, {2.0, 1.0, 0.0}, {1.0, -1.0, 0.0}, {-1.0, 2.0, 1.0} };
  static double b[3][4] = { {1.0, 1.0, -1.0, -1.0}, {2.0, 1.0, 0.0, 2.0}, {1.0, -1.0, 0.0, 1.0} };
  static double u[4][4], v[3][3], c[4][3], d[3][4];
  double eps;
  eps = 0.000001;
  i = bmuav(a, 4, 3, u, v, eps, 5);
  printf("\n");
  printf("EXAMPLE(1)\n");
  printf("\n");
  printf("i = %d\n", i);
  printf("\n");
  printf("MAT U IS:\n");
  for (i=0; i<=3; i++)
    for (j=0; j<=3; j++)
      printf("%13.7e", u[i][j]);
  printf("\n");
  printf("\n");
  printf("MAT V IS:\n");
  for (i=0; i<=2; i++)
    for (j=0; j<=2; j++)
      printf("%13.7e", v[i][j]);
  printf("\n");
  printf("\n");
  printf("MAT A IS:\n");
  for (i=0; i<=3; i++)
    for (j=0; j<=2; j++)
      printf("%13.7e", a[i][j]);
  printf("\n");
  printf("\n");
  printf("MAT UAV IS:\n");
  brmul(u, a, 4, 4, 3, c);
  brmul(c, v, 4, 3, 3, a);
  for (i=0; i<=3; i++)
    for (j=0; j<=2; j++)
      printf("%13.7e", a[i][j]);
  printf("\n");
  printf("\n");
  printf("MAT UBV IS:\n");
  printf("EXAMPLE(2)\n");
```

```

printf("\n");
i=bmuav(b,3,4,v,u,eps,5);
printf("i= %d\n",i);
printf("\n");
printf("MAT U IS:\n");
for (i=0; i<=2; i++)
    | for (j=0; j<=2; j++)
        printf("%13.7e",v[i][j]);
    printf("\n");
|
printf("\n");
printf("MAT V IS:\n");
for (i=0; i<=3; i++)
    | for (j=0; j<=3; j++)
        printf("%13.7e",u[i][j]);
    printf("\n");
|
printf("\n");
printf("MAT B IS:\n");
for (i=0; i<=2; i++)
    | for (j=0; j<=3; j++)
        printf("%13.7e",b[i][j]);
    printf("\n");
|
printf("\n\n");
printf("MAT UV IS:\n");
brmul(v,b,3,3,d);
brmul(d,u,3,4,b);
for (i=0; i<=2; i++)
    | for (j=0; j<=3; j++)
        printf("%13.7e",b[i][j]);
    printf("\n");
|
printf("\n");
}

运行结果为：
EXAMPLE(1)
i=1

MAT U IS:
2.763932e-01 5.070926e-01 7.236068e-01 0.000000e+00
4.472136e-01 6.761234e-01 -4.472136e-01 0.000000e+00
4.472136e-01 -1.690309e-01 -4.472136e-01 0.000000e+00
-7.236068e-01 5.070926e-01 -2.763932e-01 0.000000e+00

MAT V IS:
8.355492e-01 -4.177746e-01 -3.568221e-01
4.472136e-01 8.944272e-01 4.430095e-15

```

```

-3.191514e-01 1.595757e-01 -9.341724e-01
MAT A IS:
2.802517e+00 0.000000e+00 0.000000e+00
0.000000e+00 2.645751e+00 0.000000e+00
0.000000e+00 0.000000e+00 1.070466e+00
0.000000e+00 0.000000e+00 0.000000e+00

MAT UAV IS:
1.000000e+00 1.000000e+00 -1.000000e+00
2.000000e+00 1.000000e+00 1.658829e-16
1.000000e+00 -1.000000e+00 1.658829e-16
-1.000000e+00 2.000000e+00 1.000000e+00

EXAMPLE(2)
i=1

MAT U IS:
-8.849941e-02 -9.028798e-01 -4.206851e-01
-9.253548e-01 -8.177683e-02 3.701772e-01
-3.686278e-01 4.220434e-01 -8.282469e-01

MAT V IS:
-7.194191e-01 -2.011355e-01 2.758774e-02 -6.642436e-01
-3.018290e-01 -6.588911e-01 4.229043e-01 5.439795e-01
-4.739923e-01 7.248497e-01 3.920768e-01 3.101613e-01
-4.082483e-01 -4.129676e-17 -8.164966e-01 4.082483e-01

MAT B IS:
3.207917e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 2.134951e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 1.072966e+00 -0.000000e+00

MAT UV IS:
1.000000e+00 1.000000e+00 -1.000000e+00 -1.000000e+00
2.000000e+00 9.999996e-01 2.266166e-07 2.000000e+00
9.999999e-01 -1.000000e+00 9.027583e-08 1.000000e+00

```

2.13 求广义逆的奇异值分解法

一、功能

利用奇异值分解求一般 $m \times n$ 阶实矩阵 A 的广义逆 A^+ 。

二、方法说明

设 $m \times n$ 矩阵 A 的奇异值分解式为

$$A = U \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V^T$$

其中 $\Sigma = \text{diag}(\sigma_0, \sigma_1, \dots, \sigma_p)$ ($p \leq \min(m, n) - 1$), 且 $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_p > 0$ 。

关于奇异值分解参看 2.12 节的方法说明。

设 $U = (U_1, U_2)$, 其中 U_1 为 U 中前 $p+1$ 列列正交向量组构成的 $m \times (p+1)$ 矩阵; $V = (V_1, V_2)$, 其中 V_1 为 V 中前 $p+1$ 列列正交向量组构成的 $n \times (p+1)$ 矩阵。则 A 的广义逆为

$$A^+ = V_1 \Sigma^{-1} U_1^T$$

三、函数语句

```
int bginv(a, m, n, aa, eps, u, v, ka)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示在奇异值分解过程中迭代超过了 60 次还未满足精度要求; 若返回的标志值大于 0, 则表示正常返回。

本函数要调用奇异值分解的函数 bmuav(参看 2.12 节)。

四、形参说明

a ——双精度实型二维数组, 体积为 $m \times n$ 。存放 $m \times n$ 的矩阵 A ; 返回时其对角线依次给出奇异值, 其余元素均为零。

m ——整型变量。矩阵 A 的行数。

n ——整型变量。矩阵 A 的列数。

aa ——双精度实型二维数组, 体积为 $n \times m$ 。返回时存放矩阵 A 的广义逆 A^+ 。

eps ——双精度实型变量。奇异值分解时的控制精度要求。

u ——双精度实型二维数组, 体积为 $m \times m$ 。返回时存放奇异值分解式中的左奇异向量 U 。

v ——双精度实型二维数组, 体积为 $n \times n$ 。返回时存放奇异值分解式中的右奇异向量 V^T 。

ka ——整型变量。其值为 $\max(m, n) + 1$ 。

五、函数程序(文件名: bginv.c)

六、例

求下列 5×4 阶矩阵 A 的广义逆 A^+ , 再求 A^+ 的广义逆 $(A^+)^+$:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 \\ 1 & 2 & 13 & 0 \\ 16 & 17 & 8 & 9 \\ 2 & 4 & 3 & 4 \end{bmatrix}$$

取 $\epsilon = 0.000001$ 。

主函数程序(文件名: bginv 0.c)如下:

```
# include "bmuav.c"
# include "stdio.h"
# include "bginv.c"
```

```
main()
{
    int m, n, ka, i, j;
    static double a[5][4] = { {1.0, 2.0, 3.0, 4.0},
                             {6.0, 7.0, 8.0, 9.0}, {1.0, 2.0, 13.0, 0.0},
                             {16.0, 17.0, 8.0, 9.0}, {2.0, 4.0, 3.0, 4.0} };
    static double aa[4][5], c[5][4], u[5][5], v[4][4];
    double eps;
    m = 5; n = 4; ka = 6; eps = 0.000001;
    printf("MAT A IS: \n");
    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            printf("%13.7e", a[i][j]);
        printf("\n");
    printf("\n");
    printf("MAT A+ IS: \n");
    i = bginv(a, m, n, aa, eps, u, v, ka);
    if (i > 0)
        for (i = 0; i < 3; i++)
            for (j = 0; j < 4; j++)
                printf("%12.6e", aa[i][j]);
            printf("\n");
        printf("\n");
    printf("MAT A+ + IS: \n");
    i = bginv(aa, n, m, c, eps, v, u, ka);
    if (i > 0)
        for (i = 0; i < 4; i++)
            for (j = 0; j < 3; j++)
                printf("%13.7e", c[i][j]);
            printf("\n");
        printf("\n");
    }
}
```

运行结果为:

```
MAT A IS:
1.000000e+00 2.000000e+00 3.000000e+00 4.000000e+00
6.000000e+00 7.000000e+00 8.000000e+00 9.000000e+00
1.000000e+00 2.000000e+00 1.300000e+01 0.000000e+00
1.600000e+01 1.700000e+01 8.000000e+00 9.000000e+00
2.000000e+00 4.000000e+00 3.000000e+00 4.000000e+00
```

```
MAT A+ IS:
-1.13487e-01 2.64449e-01 -2.97724e-02 4.51837e-02 -5.83187e-01
7.18009e-02 -3.47061e-01 2.86578e-02 4.90204e-02 5.98791e-01
-3.85819e-03 3.36949e-02 7.47631e-02 -1.12316e-02 -4.66841e-02
```

5.79800e-02	1.60308e-01	-6.79684e-02	-5.34359e-02	-4.84417e-02
MAT A++ IS:				
1.000001e+00	2.000001e+00	2.999997e+00	4.000000e+00	
6.000003e+00	7.000003e+00	7.999991e+00	9.000001e+00	
1.000002e+00	2.000001e+00	1.300000e+01	6.334456e-07	
1.600001e+01	1.700000e+01	7.999985e+00	9.000002e+00	
2.000002e+00	4.000001e+00	2.999996e+00	4.000001e+00	

第3章 矩阵特征值与特征向量的计算

3.1 约化对称矩阵为对称三对角阵的豪斯荷尔德变换法

一、功能

用豪斯荷尔德(Householder)变换将 n 阶实对称矩阵约化为对称三对角阵。

二、方法说明

化 n 阶实对称矩阵 A 为对称三对角阵的豪斯荷尔德方法, 就是使 A 经过 $n-2$ 次正交变换后变成三对角阵 A_{n-2} 。即

$$A_{n-2} = P_{n-3} \cdots P_1 P_0 A P_0 P_1 \cdots P_{n-3}$$

每一次的正交变换 P_i ($i=0, 1, \dots, n-3$) 具有如下形式

$$P_i = I - U_i U_i^T / H_i$$

其中 P_i 为对称正交矩阵, 且

$$H_i = \frac{1}{2} U_i^T U_i$$

$$U_i = (a_{i0}^{(i)}, a_{i1}^{(i)}, \dots, a_{i,l-2}^{(i)}, a_{i,l-1}^{(i)} \pm \sigma_i^{1/2}, 0, \dots, 0)^T$$

$$\sigma_i = (a_{i0}^{(i)})^2 + (a_{i1}^{(i)})^2 + \dots + (a_{i,l-1}^{(i)})^2$$

式中 $l = n - i - 1$ 。

对 A 的每一次变换为

$$A_{i+1} = P_i A_i P_i = (I - U_i U_i^T / H_i) A_i (I - U_i U_i^T / H_i)$$

若令

$$\begin{cases} s_i \doteq A_i U_i / H_i \\ k_i = U_i^T s_i / 2H_i \\ q_i = s_i - k_i U_i \end{cases}$$

则有

$$A_{i+1} = A_i - U_i q_i^T - q_i U_i^T$$

其中 s_i 的形式为

$$s_i = (s_{i0}, s_{i1}, \dots, s_{il}, 0, \dots, 0)^T$$

q_i 的形式为

$$q_i = (s_{i0} - k_i u_{i0}, s_{i1} - k_i u_{i1}, \dots, s_{il-1} - k_i u_{il-1}, s_{il}, 0, \dots, 0)^T$$

三、函数语句

void cstrq(a, n, q, b, c)

本函数与 3.2 节的函数 csstq 联用, 可以计算实对称矩阵 A 的全部特征值与相应的特征向量。

四、形参说明

a——双精度实型二维数组, 体积为 $n \times n$ 。存放 n 阶实对称矩阵 A。

n——整型变量。实对称矩阵 A 的阶数。

q——双精度实型二维数组, 体积为 $n \times n$ 。返回豪斯荷尔德变换的乘积矩阵 Q。在与 3.2 节中的函数 csstq 联用时, 若将 Q 矩阵作为函数 csstq 中的一个参数, 可计算实对称矩阵 A 的特征值及相应的特征向量。

b——双精度实型一维数组, 长度为 n。返回对称三对角阵中的主对角线元素。

c——双精度实型一维数组, 长度为 n。其中前 $n - 1$ 个元素返回对称三对角阵中的次对角线元素。

由上可知, 本函数返回的对称三对角阵为

$$T = \begin{bmatrix} b_0 & c_0 & & & \\ c_0 & b_1 & c_1 & & 0 \\ & c_1 & b_2 & c_2 & \\ & & \ddots & \ddots & \ddots \\ 0 & & c_{n-3} & b_{n-2} & c_{n-2} \\ & & & c_{n-2} & b_{n-1} \end{bmatrix}$$

五、函数程序(文件名: cstrq.c)

六、例

用豪斯荷尔德变换将下列 5 阶实对称矩阵约化为对称三对角阵, 并给出豪斯荷尔德变换的乘积矩阵 Q:

$$A = \begin{bmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & -1 \\ 4 & -3 & -5 & -1 & 15 \end{bmatrix}$$

主函数程序(文件名: cstrq 0.c)如下:

```
# include "stdio.h"
# include "cstrq.c"
main()
{ int i, j;
  static double b[5], c[5], q[5][5];
  static double a[5][5] = { {10.0, 1.0, 2.0, 3.0, 4.0},
```

```
{1.0, 9.0, -1.0, 2.0, -3.0}, {2.0, -1.0, 7.0, 3.0, -5.0},
{3.0, 2.0, 3.0, 12.0, -1.0}, {4.0, -3.0, -5.0, -1.0, 15.0} };
cstrq(a, 5, q, b, c);
printf("MAT A IS: \n");
for (i = 0; i <= 4; i++)
  for (j = 0; j <= 4; j++)
    printf("%13.7e", a[i][j]);
printf("\n");
printf("MAT Q IS: \n");
for (i = 0; i <= 4; i++)
  for (j = 0; j <= 4; j++)
    printf("%13.7e", q[i][j]);
printf("\n");
printf("MAT B IS: \n");
for (i = 0; i <= 4; i++)
  printf("%13.7e", b[i]);
printf("\n\n");
printf("MAT C IS: \n");
for (i = 0; i <= 4; i++)
  printf("%13.7e", c[i]);
printf("\n\n");
```

运行结果为:

MAT A IS:				
1.000000e+01	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00
1.000000e+00	9.000000e+00	-1.000000e+00	2.000000e+00	-3.000000e+00
2.000000e+00	-1.000000e+00	7.000000e+00	3.000000e+00	-5.000000e+00
3.000000e+00	2.000000e+00	3.000000e+00	1.200000e+01	-1.000000e+00
4.000000e+00	-3.000000e+00	-5.000000e+00	-1.000000e+00	1.500000e+01

MAT Q IS:				
-3.845416e-02	-8.269598e-01	3.054904e-02	5.601120e-01	0.000000e+00
-8.685653e-01	-2.401904e-01	1.069216e-01	-4.200840e-01	0.000000e+00
4.492444e-01	-5.037103e-01	-2.329364e-01	-7.001400e-01	0.000000e+00
2.056576e-01	-6.871667e-02	9.661134e-01	-1.400280e-01	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00

MAT B IS:				
9.295202e+00	1.162671e+01	1.096044e+01	6.117647e+00	1.500000e+01

MAT C IS:				
-7.494847e-01	-4.496268e+00	-2.157041e+00	7.141428e+00	0.000000e+00

即返回的三对角阵为:

$$T = \begin{bmatrix} 9.295202 & -0.7494847 & 0.000000 & 0.000000 & 0.000000 \\ -0.7494847 & 11.62671 & -4.496268 & 0.000000 & 0.000000 \\ 0.000000 & -4.496268 & 10.96044 & -2.157041 & 0.000000 \\ 0.000000 & 0.000000 & -2.157041 & 6.117647 & 7.141428 \\ 0.000000 & 0.000000 & 0.000000 & 7.141428 & 15.000000 \end{bmatrix}$$

3.2 实对称三对角阵的全部特征值与特征向量的计算

一、功能

用变形 QR 方法计算实对称三对角阵的全部特征值与相应的特征向量。

二、方法说明

有关 QR 方法的说明参看 3.4 节。

三、函数语句

int csstq(n, b, c, q, eps, l)

本函数返回一个整型标志值。若返回的标志值小于 0，则表示迭代了 l 次还未求得一个特征值，输出信息“fail”；若返回的标志值大于 0，则表示正常返回。

本函数如果与 3.1 节的函数 cstrq 联用，则可以计算一般实对称矩阵的全部特征值与相应的特征向量。

四、形参说明

n ——整型变量。实对称三对角阵的阶数。

b ——双精度实型一维数组，长度为 n 。存放 n 阶对称三对角阵的主对角线上的各元素；返回时存放全部特征值。

c ——双精度实型一维数组，长度为 n 。前 $n-1$ 个元素存放 n 阶对称三对角阵的次对角线上的元素。

由形参 b 与 c 的说明可知， n 阶实对称三对角阵为

$$T = \begin{bmatrix} b_0 & c_0 & & & \\ c_0 & b_1 & c_1 & & 0 \\ & c_1 & b_2 & c_2 & \\ & & \ddots & \ddots & \\ 0 & & c_{n-3} & b_{n-2} & c_{n-2} \\ & & & c_{n-2} & b_{n-1} \end{bmatrix}$$

q ——双精度实型二维数组，体称为 $n \times n$ 。若存放单位矩阵，则返回 n 阶实对称三对角阵 T 的特征向量组；若存放由 3.1 节中函数 cstrq 所返回的一般实对称矩阵 A 的豪斯荷

尔德变换的乘积矩阵 Q ，则返回实对称矩阵 A 的特征向量组。其中 q 中的第 j 列为与数组 b 中第 j 个特征值对应的特征向量。

eps ——双精度实型变量。本函数在迭代过程中的控制精度要求。

l ——整型变量。为求得一个特征值所允许的最大迭代次数。

五、函数程序(文件名:csstq.c)

六、例

与 3.1 节中的函数 cstrq 联用，计算下列 5 阶实对称矩阵的全部特征值与相应的特征向量：

$$A = \begin{bmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & -1 \\ 4 & -3 & -5 & -1 & 15 \end{bmatrix}$$

取 $\epsilon = 0.000001$ ，最大迭代次数为 60。

主函数程序(文件名: csstq 0.c)如下：

```
# include "stdio.h"
# include "csstq.c"
# include "cstrq.c"
main()
{
    int i, j, k, l = 60;
    static double q[5][5], b[5], c[5];
    static double a[5][5] = { {10.0, 1.0, 2.0, 3.0, 4.0},
                            {1.0, 9.0, -1.0, 2.0, -3.0}, {2.0, -1.0, 7.0, 3.0, -5.0},
                            {3.0, 2.0, 3.0, 12.0, -1.0}, {4.0, -3.0, -5.0, -1.0, 15.0} };
    double eps = 0.000001;
    cstrq(a, 5, q, b, c);
    k = csstq(5, b, c, q, eps, 1);
    printf("MAT A IS: \n");
    for (i = 0; i <= 4; i++)
        for (j = 0; j <= 4; j++)
            printf("%13.7e", a[i][j]);
    printf("\n");
    printf("c\n");
    if (k > 0)
        printf("MAT B IS: \n");
        for (i = 0; i <= 4; i++)
            printf("%13.7e", b[i]);
        printf("\n\n");
        printf("MAT Q IS: \n");
        for (i = 0; i <= 4; i++)
            for (j = 0; j <= 4; j++)
                printf("%13.7e", q[i][j]);
        printf("\n");
    }
}
```

```

printf("\n");
}

```

运行结果为：

```

MAT A IS:
1.000000e+01 1.000000e+00 2.000000e+00 3.000000e+00 4.000000e+00
1.000000e+00 9.000000e+00 -1.000000e+00 2.000000e+00 -3.000000e+00
2.000000e+00 -1.000000e+00 7.000000e+00 3.000000e+00 -5.000000e+00
3.000000e+00 2.000000e+00 3.000000e+00 1.200000e+01 -1.000000e+00
4.000000e+00 -3.000000e+00 -5.000000e+00 -1.000000e+00 1.500000e+01

MAT B IS:
6.994838e+00 9.365555e+00 1.655266e+00 1.580892e+01 1.917542e+01

MAT Q IS:
6.540830e-01 5.215112e-02 3.872969e-01 -6.237025e-01 1.745051e-01
1.996813e-01 -8.599639e-01 -3.662210e-01 -1.591011e-01 -2.473025e-01
2.565105e-01 5.055751e-01 -7.043773e-01 -2.272975e-01 -3.616417e-01
-6.604027e-01 2.011666e-04 1.189262e-01 -6.926844e-01 -2.644109e-01
-1.742799e-01 -4.621920e-02 -4.534231e-01 -2.328223e-01 8.412441e-01

```

3.3 约化一般实矩阵为赫申伯格矩阵 的初等相似变换法

一、功能

用初等相似变换将一般实矩阵约化为上 H 阵，即赫申伯格(Hessen berg)矩阵。

二、方法说明

为了要将 n 阶实方阵 A 化为上 H 阵，只需依次将 A 中的每一列化为上 H 阵。为此，只要对于 $k = 1, 2, \dots, n-2$ 作如下变换：

(1) 从第 $k-1$ 列的第 $k-1$ 个以下的元素中选取绝对值最大的元素，设为 $a_{l,k-1}$ 。

(2) 交换第 l 行与第 k 行；交换第 l 列与第 k 列。

(3) 对于 $i = k+1, \dots, n-1$ 作变换

$$a_{i,k-1}/a_{k,k-1} \Rightarrow m, \quad 0 \Rightarrow a_{i,k-1}$$

$$a_{ij} - ma_{ji} \Rightarrow a_{ij}, \quad j = k, \dots, n-1$$

$$a_{jk} + ma_{ji} \Rightarrow a_{jk}, \quad j = 0, 1, \dots, n-1$$

三、函数语句

```
void chhbg(a, n)
```

四、形参说明

a ——双精度实型二维数组，体积为 $n \times n$ 。存放一般 n 阶实矩阵 A ；返回时存放 A 的

上 H 阵。

n ——整型变量。实矩阵 A 的阶数。

五、函数程序(文件名:chhbg.c)

六、例

用初等相似变换将下列 5 阶矩阵约化为上 H 阵：

$$A = \begin{bmatrix} 1 & 6 & -3 & -1 & 7 \\ 8 & -15 & 18 & 5 & 4 \\ -2 & 11 & 9 & 15 & 20 \\ -13 & 2 & 21 & 30 & -6 \\ 17 & 22 & -5 & 3 & 6 \end{bmatrix}$$

主函数程序(文件名: chhbg 0.c)如下：

```

#include "stdio.h"
#include "chhbg.c"
main()
{
    int i, j;
    static double a[5][5] = {{1.0, 6.0, -3.0, -1.0, 7.0},
                            {8.0, -15.0, 18.0, 5.0, 4.0}, {-2.0, 11.0, 9.0, 15.0, 20.0},
                            {-13.0, 2.0, 21.0, 30.0, -6.0}, {17.0, 22.0, -5.0, 3.0, 6.0}};
    chhbg(a, 5);
    printf("MAT A IS: \n");
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            printf("%13.7e", a[i][j]);
    printf("\n");
    printf("\n");
}

```

运行结果为：

```

MAT A IS:
1.000000e+00 1.094118e+01 6.185671e+00 8.267828e+00 -3.000000e+00
1.700000e+01 1.464706e+01 2.487305e+01 2.577971e+01 -5.000000e+00
0.700000e+00 -1.926990e+01 3.500961e+01 5.839105e+00 1.717647e+01
0.000000e+00 0.000000e+00 -6.137333e+01 -4.555436e+01 6.186748e+00
0.000000e+00 0.000000e+00 0.000000e+00 -2.285263e+01 2.589769e+01

```

3.4 求赫申伯格矩阵全部特征值的 QR 方法

一、功能

用带原点位移的双重步 QR 方法计算实上 H 阵的全部特征值。

二、方法说明

设上 H 阵 A 为不可约的，通过一系列双重步 QR 变换，使上 H 阵变为对角块全部

是一阶块或二阶块，进而从中解出全部特征值。

双重步 QR 变换的步骤可以归纳如下。

(1) 确定一个初等正交对称矩阵 Q_0 ，对 A 作相似变换

$$A_1 = Q_0 A Q_0$$

其中 Q_0 为对称正交矩阵，具有如下形式

$$Q_0 = \begin{bmatrix} \tilde{Q}_0 & 0 \\ 0 & I_{n-3} \end{bmatrix}$$

且 \tilde{Q}_0 为 3×3 的矩阵。若令

$$\begin{cases} \alpha = a_{n-2,n-2} + a_{n-1,n-1} \\ \beta = a_{n-2,n-2} \cdot a_{n-1,n-1} - a_{n-2,n-1} \cdot a_{n-1,n-2} \\ p_0 = a_{00}(a_{00} - \alpha) + a_{01}a_{10} + \beta \\ q_0 = a_{10}(a_{00} + a_{11} - \alpha) \\ r_0 = a_{10}a_{21} \end{cases}$$

则 \tilde{Q}_0 中的各元素为

$$\tilde{Q}_0 = \begin{bmatrix} -\frac{p_0}{s_0} & -\frac{q_0}{s_0} & -\frac{r_0}{s_0} \\ -\frac{q_0}{s_0} & \frac{p_0}{s_0} + \frac{r_0^2}{s_0(p_0 + s_0)} & -\frac{q_0r_0}{s_0(p_0 + s_0)} \\ -\frac{r_0}{s_0} & -\frac{q_0r_0}{s_0(p_0 + s_0)} & \frac{p_0}{s_0} + \frac{q_0^2}{s_0(p_0 + s_0)} \end{bmatrix}$$

其中 $s_0 = \text{sign}(p_0)\sqrt{p_0^2 + q_0^2 + r_0^2}$ 。

由此可得

$$A_1 = Q_0 A Q_0 = \begin{bmatrix} * & * & * & * & \cdots & * \\ p_1 & * & * & * & \cdots & * \\ q_1 & * & * & * & \cdots & * \\ r_1 & * & * & * & \cdots & * \\ \vdots & \ddots & \ddots & \vdots & & \\ 0 & * & * & * & & \end{bmatrix}$$

其中有下划线的为次对角线以下新增加的三个非零元素。

(2) 利用同样的方法，依次确定正交对称矩阵 Q_1, Q_2, \dots, Q_{n-2} 对 A_1, A_2, \dots, A_{n-2} 作相似变换

$$A_{i+1} = Q_i A_i Q_i, \quad i = 1, 2, \dots, n-2$$

最后可得到上 H 矩阵

$$A_{n-1} = Q_{n-2} A_{n-2} Q_{n-2}$$

在这个过程中，一般有

$$A_i = \begin{bmatrix} * & * & * & \cdots & * & * & * & * & \cdots & * \\ * & * & * & \cdots & * & * & * & * & \cdots & * \\ * & * & \cdots & * & * & * & * & * & \cdots & * \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & * & * & * & * & * & * & \cdots & * \\ p_i & * & * & * & * & * & * & * & \cdots & * \\ q_i & * & * & * & * & * & * & * & \cdots & * \\ r_i & * & * & * & * & * & * & * & \cdots & * \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & \end{bmatrix}$$

为了消去 A_i 中次对角线以下有下划线的三个非零元素， Q_i 可以取如下形式的正交对称矩阵：

$$Q_i = \begin{bmatrix} I_i & 0 & 0 \\ 0 & \tilde{Q}_i^{(0)} & 0 \\ 0 & 0 & I_{n-i-3} \end{bmatrix}$$

其中 $\tilde{Q}_i^{(0)}$ 为 3×3 的矩阵，且

$$\tilde{Q}_i^{(0)} = \begin{bmatrix} -\frac{p_i}{s_i} & -\frac{q_i}{s_i} & -\frac{r_i}{s_i} \\ -\frac{q_i}{s_i} & \frac{p_i}{s_i} + \frac{r_i^2}{s_i(p_i + s_i)} & -\frac{q_ir_i}{s_i(p_i + s_i)} \\ -\frac{r_i}{s_i} & -\frac{q_ir_i}{s_i(p_i + s_i)} & \frac{p_i}{s_i} + \frac{q_i^2}{s_i(p_i + s_i)} \end{bmatrix}$$

其中 $s_i = \text{sign}(p_i)\sqrt{p_i^2 + q_i^2 + r_i^2}$ 。

上式中的 $p_i, q_i, r_i (i=1, 2, \dots, n-2)$ 是从相应的 A_i 的第 $i-1$ 列中取得。即

$$p_i = a_{i,i-1}^{(i)}, \quad q_i = a_{i+1,i-1}^{(i)}, \quad r_i = a_{i+2,i-1}^{(i)}$$

而

$$p_{n-2} = a_{n-2,n-3}^{(n-2)}, \quad q_{n-2} = a_{n-1,n-3}^{(n-2)}, \quad r_{n-2} = 0$$

反复进行以上两步，直到将上 H 矩阵 A 变为对角块全部是一阶块或二阶块为止，此时，就可以直接从各一阶块或二阶块中解出全部特征值。

QR 方法是一种迭代的方法。在每次迭代计算过程中，如果次对角线元素的模小到一定程度，就可以把它们当作零看待，将矩阵分割为各不可约上 H 矩阵，以便逐步降低主子阵的阶数，从而减少计算工作量。

在本函数中，判断次对角线元素的模是否可认为是零的准则为

$$|a_{k,k-1}| \leq (\epsilon + |a_{k-1,k-1}| + |a_{kk}|)$$

当上 H 矩阵 A 类似正交阵时，本函数程序工作可能失败或误差较大，其原因是由于舍入误差的影响，但这种情况在一般工程应用中是很少见的。

三、函数语句

```
int chhqr(a, n, u, v, eps, jt)
```

本函数返回一个整型标志值。若返回的标志值小于0，则表示迭代超过jt次而还未满足精度要求，输出信息“fail”；若返回的标志值大于0，则表示正常返回。

本函数与3.3节的函数chhbqg联用，可以计算一般实矩阵的全部特征值。

四、形参说明

a——双精度实型二维数组，体积为 $n \times n$ 。存放上H矩阵A。

n——整型变量。上H矩阵的阶数。

u——双精度实型一维数组，长度为n。返回n个特征值的实部。

v——双精度实型一维数组，长度为n。返回n个特征值的虚部。

eps——双精度实型变量。控制精度要求。

jt——整型变量。控制最大迭代次数。

五、函数程序(文件名:chhqr.c)

六、例

与3.3节中的函数chhbqg联用，计算下列5阶矩阵的全部特征值：

$$A = \begin{bmatrix} 1 & 6 & -3 & -1 & 7 \\ 8 & -15 & 18 & 5 & 4 \\ -2 & 11 & 9 & 15 & 20 \\ -13 & 2 & 21 & 30 & -6 \\ 17 & 22 & -5 & 3 & 6 \end{bmatrix}$$

取 $\epsilon = 0.000001$ ，最大迭代次数为60。

主函数程序(文件名:chhqr0.c)如下：

```
#include "stdio.h"
#include "chhqr.c"
#include "chhbqg.c"

main()
{
    int i, j, jt = 60;
    double eps = 0.000001;
    static double u[5], v[5];
    static double a[5][5] = {{1.0, 6.0, -3.0, -1.0, 7.0},
                           {8.0, -15.0, 18.0, 5.0, 4.0}, {-2.0, 11.0, 9.0, 15.0, 20.0},
                           {-13.0, 2.0, 21.0, 30.0, -6.0}, {17.0, 22.0, -5.0, 3.0, 6.0}};
    chhbqg(a, 5);
    printf("MAT H IS: \n");
    for (i = 0; i <= 4; i++)
        for (j = 0; j <= 4; j++)
            printf("%13.7e", a[i][j]);
    printf("\n");
}
```

```
printf("\n");
i = chhqr(a, 5, u, v, eps, jt);
if (i > 0)
    for (i = 0, i <= 4, i++)
        printf("%13.7e + %13.7e \n", u[i], v[i]);
printf("\n");
```

运行结果为：

MAT H IS:

1.000000e+00	1.094118e+01	6.185671e+00	8.267828e+00	-3.000000e+00
1.700000e+01	1.464706e+01	2.487305e+01	2.577971e+01	-5.000000e+00
0.000000e+00	-1.926990e+01	3.500961e+01	5.839105e+00	1.717647e+01
0.000000e+00	0.000000e+00	-6.137333e+01	-4.555436e+01	6.186748e+00
0.000000e+00	0.000000e+00	0.000000e+00	-2.285263e+01	2.589769e+01
-6.617160e-01	+ J	0.000000e+00		
4.296101e+01	+ J	0.000000e+00		
-1.533964e+01	+ J	-6.755658e+00		
-1.533964e+01	+ J	6.755658e+00		
1.937998e+01	+ J	0.000000e+00		

3.5 求实对称矩阵特征值与特征向量的雅可比法

一、功能

用雅可比(Jacobi)方法求实对称矩阵的全部特征值与相应的特征向量。

二、方法说明

雅可比方法的基本思想如下。

设n阶矩阵A为对称矩阵。在n阶对称矩阵A的非对角线元素中选取一个绝对值最大的元素，设为 a_{pq} 。利用平面旋转变换矩阵 $R_0(p, q, \theta)$ 对A进行正交相似变换：

$$A_1 = R_0(p, q, \theta)^T A R_0(p, q, \theta)$$

其中 $R_0(p, q, \theta)$ 的元素为

$$\begin{aligned} r_{pp} &= \cos \theta, & r_{qq} &= \cos \theta, & r_{pq} &= -\sin \theta, & r_{qp} &= \sin \theta \\ r_{ij} &= 0, & i, j &\neq p, q \end{aligned}$$

如果按下式确定角度 θ

$$\tan 2\theta = \frac{2a_{pq}}{a_{pp} - a_{qq}}$$

则对称矩阵A经上述变换后，其非对角线元素的平方和将减少 $2a_{pq}^2$ ，对角线元素的平方和增加 $2a_{pq}^2$ ，而矩阵中所有元素的平方和保持不变。由此可知，对称矩阵A每经过一次变换，其非对角线元素的平方和“向零接近了一步”。因此，只要反复进行上述变换，就可以逐步将矩阵A变为对角矩阵。对角矩阵中对角线上的元素 $\lambda_0, \lambda_1, \dots, \lambda_{n-1}$ 即为特征值，而每一步中的平面旋转矩阵的乘积的第i列($i = 0, 1, \dots, n-1$)即为与 λ_i 对应的特征向量。

综上所述,用雅可比方法求 n 阶对称矩阵 A 的特征值及相应特征向量的步骤如下:

- (1) 令 $S = I_n$ (I_n 为单位矩阵)。
- (2) 在 A 中选取非对角线元素中绝对值最大者,设为 a_{pq} 。
- (3) 若 $|a_{pq}| < \epsilon$, 则迭代过程结束。此时对角线元素 a_{ii} ($i = 0, 1, \dots, n-1$)即为特征值 λ_i , 矩阵 S 的第 i 列为与 λ_i 对应的特征向量。否则,继续下一步。
- (4) 计算平面旋转矩阵的元素及其变换后的矩阵 A_1 的元素。其计算公式如下:

$$x = -a_{pq}$$

$$y = \frac{1}{2}(a_{qq} - a_{pp})$$

$$w = \text{sign}(y) \frac{x}{\sqrt{x^2 + y^2}}$$

$$\sin 2\theta = w$$

$$\sin \theta = \frac{w}{\sqrt{2(1 + \sqrt{1 - w^2})}}$$

$$\cos \theta = \sqrt{1 - \sin^2 \theta}$$

$$a_{pp}^{(1)} = a_{pp}\cos^2 \theta + a_{qq}\sin^2 \theta + a_{pq}\sin 2\theta$$

$$a_{qq}^{(1)} = a_{pp}\sin^2 \theta - a_{qq}\cos^2 \theta - a_{pq}\sin 2\theta$$

$$a_{pq}^{(1)} = a_{qp}^{(1)} = 0$$

$$\left. \begin{array}{l} a_{pj}^{(1)} = a_{pj}\cos \theta + a_{qj}\sin \theta \\ a_{qj}^{(1)} = -a_{pj}\sin \theta + a_{qj}\cos \theta \end{array} \right\} j \neq p, q$$

$$\left. \begin{array}{l} a_{ip}^{(1)} = a_{ip}\cos \theta + a_{iq}\sin \theta \\ a_{iq}^{(1)} = -a_{ip}\sin \theta + a_{iq}\cos \theta \end{array} \right\} i \neq p, q$$

$$a_{ij}^{(1)} = a_{ij}, i, j \neq p, q$$

(5) $S = S \cdot R(p, q, \theta)$, 转(2)。

三、函数语句

```
int cjcbi(a, n, v, eps, jt)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示迭代了 jt 次还达不到精度要求; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a ——双精度实型二维数组, 体积为 $n \times n$ 。存放 n 阶实对称矩阵 A ; 返回时, 对角线上存放 n 个特征值。

n ——整型变量。实对称矩阵 A 的阶数。

v ——双精度实型二维数组, 体积为 $n \times n$ 。返回特征向量。其中第 i 列为与 λ_i (即返回的 a_{ii} , $i = 0, 1, \dots, n-1$) 对应的特征向量。

eps ——双精度实型变量。控制精度要求。

jt ——整型变量。控制最大迭代次数。

五、函数程序(文件名:cjcbi.c)

六、例

用雅可比方法计算下列 3 阶实对称矩阵的全部特征值与相应的特征向量:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

取 $\epsilon = 0.000001$, 最大迭代次数为 100。

主函数程序(文件名:cjcbi 0.c)如下:

```
# include "stdio.h"
# include "cjcbi.c"
main()
| int i, j;
double eps;
static double v[3][3];
static double a[3][3] = {{2.0, -1.0, 0.0}, {-1.0, 2.0, -1.0},
| 0.0, -1.0, 2.0}};
eps = 0.000001;
i = cjcbi(a, 3, v, eps, 100);
if (i > 0)
| for (i = 0; i < = 2; i++)
printf("%13.7e ", a[i][i]);
printf("\n\n");
for (i = 0; i < = 2; i++)
| for (j = 0, j < = 2; j++)
printf("%13.7e ", v[i][j]);
printf("\n");
}
printf("\n");
```

运行结果为:

3.414214e+00	5.857864e-01	2.000000e+00
5.000000e-01	5.000000e-01	-7.071068e-01
-7.071068e-01	7.071068e-01	8.702502e-09
5.000000e-01	5.000000e-01	7.071068e-01

本问题的准确结果为:

特征值 $\lambda_0 = 2 + \sqrt{2}$, $\lambda_1 = 2 - \sqrt{2}$, $\lambda_2 = 2.0$

特征向量

$$V_0 = \begin{bmatrix} 0.5 \\ -\sqrt{2}/2 \\ 0.5 \end{bmatrix}, V_1 = \begin{bmatrix} 0.5 \\ \sqrt{2}/2 \\ 0.5 \end{bmatrix}, V_2 = \begin{bmatrix} -\sqrt{2}/2 \\ 0 \\ \sqrt{2}/2 \end{bmatrix}$$

3.6 求实对称矩阵特征值与特征向量 的雅可比过关法

一、功能

用雅可比(Jacobi)过关法求实对称矩阵的全部特征值与对应的特征向量。

二、方法说明

基本方法同 3.5 节。在选取非对角线上的绝对值最大的元素时改用如下方法：

首先计算实对称矩阵 A 的非对角线元素的平方和的平方根

$$v_0 = \left(2 \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} a_{ij}^2 \right)^{\frac{1}{2}}$$

然后设置关口 $v_1 = v_0/n$, 在非对角线元素中按行扫描选取第一个绝对值大于或等于 v_1 的元素 a_{pq} 进行平面旋转变换, 直到所有非对角线元素的绝对值均小于 v_1 为止。再设关口 $v_2 = v_1/n$, 重复这个过程。以此类推, 这个过程一直作到对于某个 $v_k < \epsilon$ 为止。

三、函数语句

```
void cjcbj(a, n, v, eps)
```

四、形参说明

a ——双精度实型二维数组, 体积为 $n \times n$ 。存放 n 阶实对称矩阵 A ; 返回时, 对角线上存放 n 个特征值。

n ——整型变量。实对称矩阵 A 的阶数。

v ——双精度实型二维数组, 体积为 $n \times n$ 。返回特征向量。其中第 i 列为与 λ_i (即返回的 a_{ii} , $i = 0, 1, \dots, n-1$) 对应的特征向量。

ϵ ——双精度实型变量。控制精度要求。

五、函数程序(文件名:cjcbj.c)

六、例

用雅可比过关法求下列 5 阶实对称矩阵的全部特征值与相应的特征向量:

$$A = \begin{bmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & -1 \\ 4 & -3 & -5 & -1 & 15 \end{bmatrix}$$

取 $\epsilon = 0.000001$ 。

主函数程序(文件名:cjcbj_0.c)如下:

```
# include "stdio.h"
```

```
# include "cjcbj.c"
main()
{
    int i, j;
    double eps;
    static double v[5][5];
    static double a[5][5] = { {10.0, 1.0, 2.0, 3.0, 4.0},
                            {1.0, 9.0, -1.0, 2.0, -3.0},
                            {2.0, -1.0, 7.0, 3.0, -5.0},
                            {3.0, 2.0, 3.0, 12.0, -1.0},
                            {4.0, -3.0, -5.0, -1.0, 15.0} };
    eps = 0.000001;
    cjcbj(a, 5, v, eps);
    for (i = 0; i <= 4; i++)
        printf("%13.7e\n", a[i][i]);
    printf("\n\n");
    for (i = 0; i <= 4; i++)
        for (j = 0, j <= 4; j++)
            printf("%12.6e", v[i][j]);
    printf("\n");
    printf("\n");
}
```

运行结果为:

6.994838e+00				
9.365555e+00				
1.655266e+00				
1.580892e+01				
1.917542e+01				
6.54083e-01	-5.21511e-02	-3.87297e-01	6.23702e-01	1.74505e-01
1.99681e-01	8.59964e-01	3.66221e-01	1.59101e-01	-2.47303e-01
2.56510e-01	-5.05575e-01	7.04377e-01	2.27297e-01	-3.61642e-01
-6.60403e-01	-2.01167e-04	-1.18926e-01	6.92684e-01	-2.64411e-01
-1.74280e-01	4.62192e-02	4.53423e-01	2.32822e-01	8.41244e-01

第4章 非线性方程与方程组的求解

4.1 求非线性方程实根的对分法

一、功能

用对分法搜索方程 $f(x)=0$ 在区间 $[a, b]$ 内的实根。

二、方法说明

从端点 $x_0 = a$ 开始, 以 h 为步长, 逐步往后进行搜索。

对于每一个子区间 $[x_i, x_{i+1}]$ (其中 $x_{i+1} = x_i + h$):

若 $f(x_i) = 0$, 则 x_i 为一个实根, 且从 $x_i + \frac{h}{2}$ 开始再往后搜索; 若 $f(x_{i+1}) = 0$, 则 x_{i+1}

为一个实根, 且从 $x_{i+1} + \frac{h}{2}$ 开始再往后搜索; 若 $f(x_i)f(x_{i+1}) > 0$, 则说明当前子区间内无实根, 从 x_{i+1} 开始再往后搜索; 若 $f(x_i)f(x_{i+1}) < 0$, 则说明在当前子区间内有实根。此时, 反复将子区间减半, 直到发现一个实根或子区间长度小于 ϵ 为止。在后一种情况下, 子区间的中点即取为方程的一个实根。然后再从 x_{i+1} 开始往后搜索。其中 ϵ 为预先给定的精度要求。以上过程一直进行到区间右端点 b 为止。

在使用本方法时, 要注意步长 h 的选择。若步长 h 选得过大, 可能会导致某些实根的丢失; 若步长 h 选得过小, 则会增加计算工作量。

三、函数语句

```
int ddhrt(a, b, h, eps, x, m)
```

本函数返回一个整型值, 表示本函数在区间 $[a, b]$ 内实际搜索到的实根个数。若该值等于 m 时, 则有可能在求根区间 $[a, b]$ 内的实根未搜索完。

本函数需要调用计算方程左端函数 $f(x)$ 值的函数 ddhrtf, 由用户自编, 其形式为:

```
double ddhrtf(x)
double x;
double z;
z = f(x)的表达式;
return(z);
```

四、形参说明

a —双精度实型变量。求根区间的左端点。

b —双精度实型变量。求根区间的右端点。

h —双精度实型变量。搜索求根时采用的步长。

eps —双精度实型变量。控制精度要求。

x —双精度实型一维数组, 长度为 m 。返回在区间 $[a, b]$ 内搜索到的实根, 其实根个数由函数值返回。

m —整型变量。在 $[a, b]$ 内实根个数的预估值。

五、函数程序(文件名: ddhrt.c)

六、例

求方程

$$f(x) = x^6 - 5x^5 + 3x^4 + x^3 - 7x^2 + 7x - 20 = 0$$

在区间 $[-2, 5]$ 内的所有实根。取步长 $h = 0.2$, $\epsilon = 0.000001$ 。

由于本方程为 6 次代数方程, 最多有 6 个实根, 因此取 $m = 6$ 。

主函数程序与计算 $f(x)$ 的函数程序(文件名: ddhrt0.c)如下:

```
#include "math.h"
#include "stdio.h"
#include "ddhrt.c"

main()
{ int i, n;
  static int m = 6;
  static double x[6];
  n = ddhrt(-2.0, 5.0, 0.2, 0.000001, x, m);
  printf("M = %d\n", n);
  for (i = 0; i <= n - 1; i++)
    printf("x(%d) = %13.7e\n", i, x[i]);
  printf("\n");
}

double ddhrtf(x)
double x;
{ double z;
  z = (((((x - 5.0) * x + 3.0) * x + 1.0) * x - 7.0) * x + 7.0 * x - 20.0;
  return(z);
}
```

运行结果为:

```
M = 2
x(0) = -1.402463e+00
x(1) = 4.333755e+00
```

4.2 求非线性方程一个实根的牛顿法

一、功能

用牛顿(Newton)迭代法求方程 $f(x)=0$ 的一个实根。

二、方法说明

设方程 $f(x)=0$, 满足以下条件:

- (1) $f(x)$ 在闭区间 $[a, b]$ 上, 其 $f'(x)$ 与 $f''(x)$ 均存在, 且各自保持固定符号;
 (2) $f(a)f(b) < 0$;
 (3) $f(x_0)f''(x) > 0$, 且 $x, x_0 \in [a, b]$

则方程 $f(x) = 0$ 在区间 $[a, b]$ 上有且只有一个实根, 取初值 x_0 , 由牛顿迭代格式

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

计算得到的序列 $x_0, x_1, \dots, x_n, \dots$ 收敛于方程 $f(x) = 0$ 的根。

结束迭代过程的条件为 ($|f(x_{n+1})| < \epsilon$) 与 ($|x_{n+1} - x_n| < \epsilon$) 同时成立, 其中 ϵ 为预先给定的精度要求。

三、函数语句

`int dnewt(x, eps, js)`

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示在牛顿迭代公式中出现了 $f'(x) = 0$ 的情况, 输出信息“err”; 若返回的标志值等于最大迭代次数 js , 则表示迭代了 js 次还未满足精度要求, 返回的实根只作为参考; 若返回的标志值大于等于 0 且小于最大迭代次数 js , 则表示正常返回。

本函数需要调用计算 $f(x)$ 及 $f'(x)$ 值的函数 `dnewtf`, 由用户自编, 其形式为:

```
void dnewtf(x, y)
double x, y[2];
{ y[0] = f(x)的表达式;
  y[1] = f'(x)的表达式;
  return;
}
```

四、形参说明

x ——双精度实型变量指针。在此指针指向的单元中存放迭代初值; 返回时存放迭代终值。

ϵ ——双精度实型变量。控制精度要求。

js ——整型变量。最大迭代次数。

五、函数程序(文件名: `dnewt.c`)

六、例

用牛顿迭代法求方程 $f(x) = x^3 - x^2 - 1 = 0$ 在 $x_0 = 1.5$ 附近的一个实根。取 $\epsilon = 0.000001$, 最大迭代次数为 60。其中 $f'(x) = 3x^2 - 2x$ 。

主函数程序及计算 $f(x)$ 、 $f'(x)$ 的函数程序(文件名: `dnewt.c`)如下:

```
#include "math.h"
#include "stdio.h"
#include "dnewt.c"
main()
{ int js, k;
```

```
double x, eps;
eps = 0.000001; js = 60; x = 1.5;
k = dnewt(&x, eps, js);
if (k >= 0)
  printf("k = %d x = %13.7e\n", k, x);
  printf("\n");
}
```

```
void dnewtf(x, y)
double x, y[2]
{ y[0] = x * x * (x - 1.0) - 1.0;
  y[1] = 3.0 * x * x - 2.0 * x;
  return;
}
```

运行结果为:

$k = 4 x = 1.465571e + 00$

4.3 求非线性方程一个实根的埃特金迭代法

一、功能

用埃特金(Aitken)迭代法求非线性方程 $x = \varphi(x)$ 的一个实根。

二、方法说明

设非线性方程为

$$x = \varphi(x)$$

取初值 x_0 。埃特金迭代一次的过程如下:

预报 $u = \varphi(x_k)$

再预报 $v = \varphi(u)$

校正 $x_{k+1} = v - \frac{(v - u)^2}{v - 2u + x_k}$

结束迭代过程的条件为

$$|v - u| < \epsilon$$

此时, v 即为非线性方程的一个实根。其中 ϵ 为预先给定的精度要求。

三、函数语句

`int datkn(x, eps, js)`

本函数返回一个整型标志值。若返回的标志值为 0, 则表示实际迭代次数已等于最大迭代次数, 但还不能满足精度要求, 此时返回的实根值只作为参考; 若返回的标志值大于 0, 则表示正常返回。

本函数需要调用计算 $\varphi(x)$ 值的函数 `datknf`, 由用户自编, 其形式为:

```
double datknf(x)
double x;
```

```

    | double y;
    | y = φ(x) 的表达式;
    | return(y);
}

```

四、形参说明

x——双精度实型变量指针。在该指针指向的单元中存放迭代初值;返回时存放迭代终值。

eps——双精度实型变量。控制精度要求。

js——整型变量。最大迭代次数。

五、函数程序(文件名: datkn.c)

六、例

用埃特金迭代法求方程 $x = 6 - x^2$ 在 $x_0 = 0.0$ 附近的一个实根。取 $\epsilon = 0.0000001$, 最大迭代次数为 20。

主函数程序及计算 $\varphi(x)$ 值的函数程序(文件名: datkn0.c)如下:

```

#include "math.h"
#include "stdio.h"
#include "datkn.c"
main()
{
    int js, k;
    double x, eps;
    eps = 0.0000001; js = 20; x = 0.0;
    k = datkn(&x, eps, js);
    printf("\n");
    printf("k = %d x = %13.7e\n", k, x);
    printf("\n");
}
double datknf(x);
double x;
{ double y;
    y = 6.0 - x * x;
    return(y);
}

```

运行结果为:

k = 12 x = 2.000000e+00

4.4 求非线性方程一个实根的连分式解法

一、功能

利用连分式求非线性方程 $f(x) = 0$ 的一个实根。

二、方法说明

设非线性方程为 $f(x) = 0$, 而 $f(x)$ 的反函数为 $x = F(y)$, 并用连分式表示为

$$x = a_0 + \frac{y - y_0}{a_1} + \frac{y - y_1}{a_2} + \cdots + \frac{y - y_{i-1}}{a_i} + \cdots$$

其中 $y_i = f(x_i)$ 。因此, 满足方程 $f(x) = 0$ 的根为

$$a = a_0 - \frac{y_0}{a_1} - \frac{y_1}{a_2} - \cdots - \frac{y_{i-1}}{a_i} - \cdots$$

由上所述, 可以得到求非线性方程 $f(x) = 0$ 的一个实根的过程如下。

选取初值 x_0 及 x_1 , 并计算出 $y_0 = f(x_0)$, $y_1 = f(x_1)$ 。根据 (x_0, y_0) 及 (x_1, y_1) 可以计算出

$$\begin{aligned} a_0 &= x_0 \\ a_1 &= (y_1 - y_0)/(x_1 - x_0) \end{aligned}$$

由此可以计算出 $x_2 = a_0 - \frac{y_0}{a_1}$

一般来说, 若已知点列 $(x_0, y_0), (x_1, y_1), \dots, (x_{i-1}, y_{i-1})$, 并已求得 a_0, a_1, \dots, a_{i-1} 。则可以计算出

$$x_i = a_0 - \frac{y_0}{a_1} - \frac{y_1}{a_2} - \cdots - \frac{y_{i-2}}{a_{i-1}}$$

及

$$y_i = f(x_i)$$

然后通过下列递推公式计算出 a_i :

$$a_0 = x_0$$

$$a_{0i} = x_i$$

$$a_{j+1,i} = \frac{y_i - y_j}{a_{ji} - a_j}, j = 0, 1, \dots, i-1$$

$$a_i = a_{ii}$$

以上过程一直进行到 $|y_i| < \epsilon$ 为止。其中 ϵ 为预先给定的精度要求。

在实际计算过程中, 上述的连分式最多作到七节为止。如果此时还不满足精度要求, 则将最后得到的实根近似值作为新的初值再重新计算。

本方法对函数 $f(x)$ 的要求比较低, 在有实根的情况下, 一般都能通过本方法求出一个实根。

三、函数语句

int dpqr(x, eps)

本函数返回一个整型值, 即实际迭代次数。若该值等于 10, 则表示迭代了 10 次, 可能没有满足精度要求, 返回的实根只作为参考; 若该值小于 10, 则表示正常返回。

本函数需要调用计算 $f(x)$ 值的函数 dpqrtf, 由用户自编, 其形式为:

```
double dpqrtf(x)
double x;
double y;
y = f(x)的表达式;
return(y);
}
```

四、形参说明

x——双精度实型变量指针。该指针指向的单元中存放迭代初值; 返回迭代终值。

eps——双精度实型变量。控制精度要求。

五、函数程序(文件名: dpqrt.c)

六、例

用连分式法求方程 $f(x) = x^3 - x^2 - 1 = 0$ 在 $x_0 = 1.0$ 附近的一个实根。取 $\epsilon = 0.000001$ 。

主函数程序及计算 $f(x)$ 值的函数程序(文件名: dpqrt0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dpqrt.c"
main()
{
    double x, eps;
    int k;
    x = 1.0; eps = 0.000001;
    k = dpqrt(&x, eps);
    printf("\n");
    printf("k = %d x = %13.7e\n", k, x);
    printf("\n");
}

double dpqrtf(x)
double x;
double y;
y = x * x * (x - 1.0) - 1.0;
return(y);
}
```

运行结果为:

```
k = 1 x = 1.465571e + 00
```

4.5 求实系数代数方程全部根的 QR 方法

一、功能

用 QR 方法求实系数 n 次多项式方程

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

的全部根。

二、方法说明

令 $b_i = a_i / a_n, i = n-1, \dots, 1, 0$

则一般实系数 n 次多项式方程 $P_n(x) = 0$ 化为 n 次首一多项式方程

$$Q_n(x) = x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0 = 0$$

由线性代数可知, $Q_n(x) = 0$ 可以看成是如下实矩阵

$$Q = \begin{bmatrix} -b_{n-1} & -b_{n-2} & -b_{n-3} & \cdots & -b_1 & -b_0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

的特征多项式方程

$$Q_n(\lambda) = \lambda^n + b_{n-1} \lambda^{n-1} + \cdots + b_1 \lambda + b_0 = 0$$

因此, 求方程 $Q_n(x) = 0$ 的全部根就变为求上述 Q 矩阵的全部特征值。矩阵 Q 为一个上 H 阵, 可以直接用 QR 方法求出全部特征值。

有关 QR 方法求上 H 阵全部特征值的问题请参看 3.4 节。

三、函数语句

```
int dqrqt(a, n, xr, xi, eps, jt)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示在 QR 方法中迭代已超过最大迭代次数但还未满足精度要求, 输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

本函数要调用 QR 方法求上 H 阵全部特征值的函数 chhqr, 参看 3.4 节。

四、形参说明

a——双精度实型一维数组, 长度为 $n+1$ 。存放 n 次多项式方程

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

的 $n+1$ 个系数。

n——整型变量。多项式方程的次数。

xr——双精度实型一维数组, 长度为 n 。返回 n 个根的实部。

xi——双精度实型一维数组, 长度为 n 。返回 n 个根的虚部。

eps——双精度实型变量。QR 方法中的控制精度要求。

jt——整型变量。控制 QR 方法中最大迭代次数。

五、函数程序(文件名: dqrrt.c)

六、例

用 QR 方法求 6 次多项式方程

$$P_6(x) = 1.5x^6 - 7.5x^5 + 4.5x^4 + 1.5x^3 - 10.5x^2 + 10.5x - 30 = 0$$

的全部根。取 $\epsilon = 0.000001$, 最大迭代次数为 60。

主函数程序(文件名: dqrrt 0.c)如下:

```
# include "stdio.h"
# include "chhqr.c"
# include "dqrrt.c"

main()
{ int i, jt, n;
  static double xr[6], xi[6];
  static double a[7] = { -30.0, 10.5, -10.5, 1.5, 4.5, -7.5, 1.5 };
  double eps;
  eps = 0.000001; jt = 60; n = 6;
  i = dqrrt(a, n, xr, xi, eps, jt);
  printf("\n");
  if (i > 0)
    for (i = 0; i < = 5; i++)
      printf("x(%d) = %13.7e\n", i, xr[i], xi[i]);
  printf("\n");
}
```

运行结果为:

```
x(0) = 4.333759e+00 0.000000e+00 j
x(1) = -1.402461e+00 0.000000e+00 j
x(2) = 1.183973e+00 -9.360990e-01 j
x(3) = 1.183973e+00 9.360990e-01 j
x(4) = -1.496217e-01 -1.192507e+00 j
x(5) = -1.496217e-01 1.192507e+00 j
```

4.6 求实系数代数方程全部根的牛顿-下山法

一、功能

用牛顿-下山法求实系数代数方程

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 = 0$$

的全部根。

二、方法说明•

牛顿-下山法的迭代格式为

$$z_{i+1} = z_i - t f(z_i) / f'(z_i)$$

选取适当的 t 可以保证有

$$|f(z_{i+1})|^2 < |f(z_i)|^2$$

迭代过程一直作到 $|f(z_i)|^2 < \epsilon$ 为止。

迭代格式在鞍点或接近重根点时, 可能因 $f'(z) \approx 0$ 而 $|f(z)|^2 \neq 0$ 而失败。在本函数中, 采用撤网络的方法。即选取适当的 d 与 c , 用

$$x_{i+1} = x_i + d \cos(c)$$

$$y_{i+1} = y_i + d \sin(c)$$

计算, 使 $|f(z_{i+1})|^2 < |f(z_i)|^2$ 而冲过鞍点或使

$$|f(z)|^2 < \epsilon$$

而求得一个根。

每当求得一个根 z^* 后, 在 $f(z)$ 中劈去因子 $(z - z^*)$, 再求另一个根。

以上过程直到求出全部根为止。

在实际计算时, 每求一个根都要作变换 $z = \sqrt[n]{|a_0|} z'$, 使得当 $a_n = 1$ 时, $|a_0| = 1$, 保证寻根在单位圆内进行。

三、函数语句

```
int dsrrt(a, n, xr, xi)
```

本函数返回一个整型标志值。若返回的标志值小于 0, 则表示方程中的所有系数 $a_i = 0$ ($i = 1, 2, \dots, n$), 输出信息“fail”; 若返回的标志值大于 0, 则表示正常返回。

四、形参说明

a ——双精度实型一维数组, 长度为 $n + 1$ 。存放 n 次多项式方程的实系数 a_0, a_1, \dots, a_n 。

n ——整型变量。多项式方程的次数。

xr ——双精度实型一维数组, 长度为 n 。返回 n 个根的实部。

xi ——双精度实型一维数组, 长度为 n 。返回 n 个根的虚部。

五、函数程序(文件名: dsrrt.c)

六、例

用牛顿-下山法求实系数代数方程

$$f(z) = z^6 - 5z^5 + 3z^4 + z^3 - 7z^2 + 7z - 20 = 0$$
 的全部根。

主函数程序(文件名: dsrrt 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dsrrt.c"

main()
{ int i;
  static double xr[6], xi[6];
  static double a[7] = { -20.0, 7.0, -7.0, 1.0, 3.0, -5.0, 1.0 };
  ...
```

```

i = dsrrt(a, 6, xr, xi);
printf("\n");
if (i > 0)
{ for (i = 0; i <= 5; i++)
    printf("x(%d) = %13.7e j %13.7e\n", i, xr[i], xi[i]);
    printf("\n");
}

```

运行结果为：

```

x(0) = 4.333755e+00 j 0.000000e+00
x(1) = -1.496217e-01 j 1.192507e+00
x(2) = -1.496217e-01 j -1.192507e+00
x(3) = -1.402463e+00 j 0.000000e+00
x(4) = 1.183975e+00 j 9.360988e-01
x(5) = 1.183975e+00 j -9.360988e-01

```

4.7 求复系数代数方程全部根的牛顿-下山法

一、功能

用牛顿-下山法求复系数代数方程

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 = 0$$

的全部根。其中

$$a_k = ar(k) + jai(k), k = 0, 1, \dots, n$$

二、方法说明

同 4.6 节。

三、函数语句

int dsrt(ar, ai, n, xr, xi)

本函数返回一个整型标志值。若返回的标志值小于 0，则表示方程中所有系数 $a_i = 0$ ($i = 1, 2, \dots, n$)，输出信息“fail”；若返回的标志值大于 0，则表示正常返回。

四、形参说明

ar——双精度实型一维数组，长度为 $n + 1$ 。存放方程系数的实部。此数组在本函数中将被破坏。

ai——双精度实型一维数组，长度为 $n + 1$ 。存放方程系数的虚部。此数组在本函数中将被破坏。

n——整型变量。方程的最高次数。

xr——双精度实型一维数组，长度为 n 。返回 n 个根的实部。

xi——双精度实型一维数组，长度为 n 。返回 n 个根的虚部。

五、函数程序(文件名: dsrt.c)

六、例

用牛顿-下山法求复系数代数方程

$$f(z) = z^5 + (3 + 3j)z^4 - 0.01jz^3 + (4.9 - 19j)z^2 + 21.33z + (0.1 - 100j) = 0$$

的全部根。

主函数程序(文件名: dsrt0.c)如下：

```

#include "math.h"
#include "stdio.h"
#include "dsrt.c"
main()
{ int i;
    static double xr[5], xi[5];
    static double ar[6] = {0.1, 21.33, 4.9, 0.0, 3.0, 1.0};
    static double ai[6] = {-100.0, 0.0, -19.0, -0.01, 2.0, 0.0};
    i = dsrt(ar, ai, 5, xr, xi);
    printf("\n");
    if (i > 0)
        for (i = 0; i <= 4; i++)
            printf("x(%d) = %13.7e j %13.7e\n", i, xr[i], xi[i]);
    printf("\n");
}

```

运行结果为：

```

x(0) = -2.166684e+00 j -2.767200e+00
x(1) = 2.093864e+00 j 1.102211e+00
x(2) = -3.430948e+00 j 3.040846e-01
x(3) = 5.434543e-01 j -2.187455e+00
x(4) = -3.968538e-02 j 1.548360e+00

```

4.8 求非线性方程组一组实根的梯度法

一、功能

用梯度法(即最速下降法)求实函数方程组

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, n-1$$

的一组实根。

二、方法说明

设非线性方程组为

$$f_i = f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, n-1$$

并定义目标函数为

$$F = F(x_0, x_1, \dots, x_{n-1}) = \sum_{i=0}^{n-1} f_i^2$$

则梯度法的计算过程如下。

- (1) 选取一组初值 x_0, x_1, \dots, x_{n-1} 。
- (2) 计算目标函数值

$$F = F(x_0, x_1, \dots, x_{n-1}) = \sum_{i=0}^{n-1} f_i^2$$

(3) 若 $F < \epsilon$, 则 $X = (x_0, x_1, \dots, x_{n-1})^T$ 即为方程组的一组实根, 过程结束; 否则继续。

- (4) 计算目标函数在 $(x_0, x_1, \dots, x_{n-1})$ 点的偏导数

$$\frac{\partial F}{\partial x_i} = 2 \sum_{j=0}^{n-1} f_j \cdot \frac{\partial f_j}{\partial x_i}, i = 0, 1, \dots, n-1$$

然后再计算

$$D = \sum_{j=0}^{n-1} \left(\frac{\partial F}{\partial x_j} \right)^2$$

- (5) 计算

$$x_i - \lambda \frac{\partial F}{\partial x_i} \Rightarrow x_i, i = 0, 1, \dots, n-1$$

其中 $\lambda = F/D$ 。

重复(2)~(5)各步骤, 直到满足精度要求为止。

在上述过程中, 如果 $D=0$, 则说明遇到了目标函数的局部极值点, 此时可改变初值再试一试。

三、函数语句

int dsnse(n, eps, x, js)

本函数返回实际迭代次数(整型)。当返回值小于 0 时, 说明在迭代过程中遇到了目标函数的局部极值点, 即

$$D = \sum_{j=0}^{n-1} \left(\frac{\partial F}{\partial x_j} \right)^2 = 0$$

此时可改变初值重新再试; 当返回值等于 js 时, 说明迭代了 js 次还未满足精度要求, 此时可改变下列条件之一再进行尝试:

- 改变初值;
- 放松精度要求;
- 增大 js 。

本函数需调用计算目标函数值及偏导数值的函数 $dsnsef$, 由用户自编, 其形式为:

```
double dsnsef(x, y, n)
int n;
double x[], y[];
{ double z;
```

$z = \sum_{i=0}^{n-1} f_i^2$ 的表达式;

$y[0] = 2 \sum_{j=0}^{n-1} f_j \cdot \frac{\partial f_j}{\partial x_0}$ 的表达式;

\vdots

$y[n-1] = 2 \sum_{j=0}^{n-1} f_j \cdot \frac{\partial f_j}{\partial x_{n-1}}$ 的表达式;

return (z);

}

四、形参说明

n ——整型变量。方程的个数, 也是未知数的个数。

eps ——双精度实型变量。控制精度要求。

x ——双精度实型一维数组, 长度为 n 。存放一组初值 x_0, x_1, \dots, x_{n-1} ; 返回时存放方程组的一组实根。

js ——整型变量。允许最大迭代次数。

五、函数程序(文件名: dsnse.c)

六、例

用梯度法求下列非线性方程组的一组实根:

$$\begin{cases} f_0 = x_0 - 5x_1^2 + 7x_2^2 + 12 = 0 \\ f_1 = 3x_0x_1 + x_0x_2 - 11x_0 = 0 \\ f_2 = 2x_1x_2 + 40x_0 = 0 \end{cases}$$

取初值为 $(1.5, 6.5, -5.0)$, 精度要求 ϵ 为 0.000001 , 最大迭代次数为 500。

主函数程序及计算目标函数值、偏导数值的函数程序(文件名: dsnse0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dsnse.c"
main()
{ int i, js;
  double eps;
  static double x[3] = {1.5, 6.5, -5.0};
  js = 500; eps = 0.000001;
  i = dsnse(3, eps, x, js);
  printf("\n");
  if ((i>0) && (i<js))
    for (i=0; i<=2; i++)
      printf("x(%d) = %13.7e\n", i, x[i]);
  printf("\n");
}

double dsnsef(x, y, n)
```

```

int n;
double x[], y[];
{ double z, f1, f2, f3, df1, df2, df3;
n = n;
f1 = x[0] - 5.0 * x[1] * x[1] + 7.0 * x[2] * x[2] + 12.0;
f2 = 3.0 * x[0] * x[1] + x[0] * x[2] - 11.0 * x[0];
f3 = 2.0 * x[1] * x[2] + 40.0 * x[0];
z = f1 * f1 + f2 * f2 + f3 * f3;
df1 = 1.0; df2 = 3.0 * x[1] * x[2] - 11.0; df3 = 40.0;
y[0] = 2.0 * (f1 * df1 + f2 * df2 + f3 * df3);
df1 = 10.0 * x[1]; df2 = 3.0 * x[0]; df3 = 2.0 * x[2];
y[1] = 2.0 * (f1 * df1 + f2 * df2 + f3 * df3);
df1 = 14.0 * x[2]; df2 = x[0]; df3 = 2.0 * x[1];
y[2] = 2.0 * (f1 * df1 + f2 * df2 + f3 * df3);
return(z);
}

```

运行结果为：

```

x(0) = 1.000188e+00
x(1) = 5.000432e+00
x(2) = -4.000388e+00

```

4.9 求非线性方程组一组实根的拟牛顿法

一、功能

用拟牛顿法求非线性方程组

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, n-1$$

的一组实数解。

二、方法说明

设非线性方程组

$$f_i(X) = 0, i = 0, 1, \dots, n-1$$

其中 $X = (x_0, x_1, \dots, x_{n-1})^T$ 。

若假设 X 的第 k 次迭代近似值为

$$X^{(k)} = (x_0^{(k)}, x_1^{(k)}, \dots, x_{n-1}^{(k)})^T$$

则计算第 $k+1$ 次迭代值的牛顿迭代格式为

$$X^{(k+1)} = X^{(k)} - F(X^{(k)})^{-1} f(X^{(k)}) \quad (1)$$

其中

$$\begin{aligned} f(X^{(k)}) &= (f_0^{(k)}, f_1^{(k)}, \dots, f_{n-1}^{(k)})^T \\ f_i^{(k)} &= f_i(X^{(k)}) \end{aligned}$$

$F(X)$ 为雅可比 (Jacobi) 矩阵, 即

$$F(X) = \begin{bmatrix} \frac{\partial f_0(X)}{\partial x_0} & \frac{\partial f_0(X)}{\partial x_1} & \dots & \frac{\partial f_0(X)}{\partial x_{n-1}} \\ \frac{\partial f_1(X)}{\partial x_0} & \frac{\partial f_1(X)}{\partial x_1} & \dots & \frac{\partial f_1(X)}{\partial x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{n-1}(X)}{\partial x_0} & \frac{\partial f_{n-1}(X)}{\partial x_1} & \dots & \frac{\partial f_{n-1}(X)}{\partial x_{n-1}} \end{bmatrix}$$

令

$$\delta^{(k)} = F(X^{(k)})^{-1} f(X^{(k)})$$

其中

$$\delta^{(k)} = (\delta_0^{(k)}, \delta_1^{(k)}, \dots, \delta_{n-1}^{(k)})^T$$

则有

$$F(X^{(k)}) \delta^{(k)} = f(X^{(k)})$$

$$X^{(k+1)} = X^{(k)} - \delta^{(k)} \quad (3)$$

若在雅可比矩阵中, 用差商代替偏导数,

即

$$\frac{\partial f_i(X^{(k)})}{\partial x_j} \approx \frac{f_i(X_j^{(k)}) - f_i(X^{(k)})}{h}$$

其中 h 足够小, 且

$$f_i(X_j^{(k)}) = f_i(x_0^{(k)}, \dots, x_{j-1}^{(k)}, x_j^{(k)} + h, x_{j+1}^{(k)}, \dots, x_{n-1}^{(k)})$$

则方程组(2)变为

$$\sum_{j=0}^{n-1} f_i(X_j^{(k)}) z_j^{(k)} = f_i(X^{(k)}), i = 0, 1, \dots, n-1$$

其中

$$z_j^{(k)} = \frac{\delta_j^{(k)}}{h + \sum_{s=0}^{n-1} \delta_s^{(k)}}, j = 0, 1, \dots, n-1$$

综上所述, 拟牛顿法求解非线性方程组的计算过程如下。

取初值 $X = (x_0, x_1, \dots, x_{n-1})^T, h > 0, 0 < t < 1$ 。

(1) 计算 $f_i(X) \Rightarrow B(i), i = 0, 1, \dots, n-1$ 。

(2) 若满足

$$\max_{0 \leq i \leq n-1} |B(i)| < \epsilon$$

则方程组的一组实数解为

$$X = (x_0, x_1, \dots, x_{n-1})^T$$

计算过程结束; 否则继续。

(3) 计算

$$f_i(X_j) \Rightarrow A(i, j), i, j = 0, 1, \dots, n-1$$

其中 $X = (x_0, x_1, \dots, x_{n-1}, x_j + h, x_{j+1}, \dots, x_{n-1})^T$ 。

(4) 解线性代数方程组 $AZ = B$, 其中 $Z = (z_0, z_1, \dots, z_{n-1})^T$ 。且计算 $\beta = 1 - \sum_{i=0}^{n-1} z_i$

(5) 计算 $x_i - hz_i / \beta \Rightarrow x_i, i = 0, 1, \dots, n-1$

(6) $t * h \Rightarrow h$, 转(1)。

以上过程一直作到 X 满足精度要求为止。

在计算过程中,若发现线性代数方程组 $AZ = B$ 奇异或 $\beta = 0$ (即 $\sum_{j=0}^{n-1} z_j = 1$)的情况,可以采取以下一些措施试一试:

- 放宽精度要求 ϵ ;
- 适当改变 t 与 h 的初值;
- 改变 X 的初值;
- 改变方程组中各方程的顺序。

三、函数语句

```
int dnetn(n, eps, t, h, x, k)
```

本函数返回实际迭代次数。若返回值等于 0, 则说明迭代了 k 次还未满足精度要求, 程序工作失败, 并输出信息“fail”; 若返回值等于 -1, 说明线性代数方程组 $AZ = B$ 奇异, 并输出信息“fail”; 若返回值等于 -2, 说明 $\beta = 0$, 即 $\sum_{j=0}^{n-1} z_j = 1$, 并输出信息“fail”。

在本函数中要调用全选主元高斯消去法求解线性代数方程组的函数 agaus, 参看 1.1 节。

在本函数中还要调用计算方程组左端函数值 $f_i(X)$ 的函数 dnetnf, 由用户自编, 其形式为

```
void dnetnf(x, y, n)
int n;
double x[], y[];
| y[0] = f_0(x_0, x_1, ..., x_{n-1}) 的表达式;
:
y[n-1] = f_{n-1}(x_0, x_1, ..., x_{n-1}) 的表达式;
return;
```

四、形参说明

n —整型变量。方程组中方程个数, 也是未知数个数。

eps —双精度实型变量。控制精度要求。

t —双精度实型变量。控制 h 大小的变量, $0 < t < 1$ 。

h —双精度实型变量。增量初值, 在本函数中要被破坏。

x —双精度实型一维数组, 长度为 n 。存放初值 $(x_0^{(0)}, x_1^{(0)}, \dots, x_{n-1}^{(0)})$, 返回时存放方程组的一组实数解 $(x_0, x_1, \dots, x_{n-1})$ 。

k —整型变量。最多允许的迭代次数。

五、函数程序(文件名: dnetn.c)

六、例

用拟牛顿法求非线性方程组

$$\begin{cases} f_0 = x_0^2 + x_1^2 + x_2^2 - 1 = 0 \\ f_1 = 2x_0^2 + x_1^2 - 4x_2 = 0 \\ f_2 = 3x_0^2 - 4x_1 + x_2^2 = 0 \end{cases}$$

的一组实数解。取初值为 $(1.0, 1.0, 1.0)^T$, $t = 0.1$, $h = 0.1$, $eps = 0.0000001$, 最大迭代次数 $k = 100$ 。

主函数程序及计算左端函数值 $f_i(X)$ 的函数程序(文件名: dnetn 0.c)如下:

```
# include "stdio.h"
# include "dnetn.c"
# include "agaus.c"
main()
| int i, k;
double eps, t, h;
static double x[3] = {1.0, 1.0, 1.0};
t = 0.1; h = 0.1; eps = 0.0000001; k = 100;
i = dnetn(3, eps, t, h, x, k);
printf("\n");
printf("i = %d\n", i);
printf("\n");
for (i = 0; i <= 2; i++)
    printf("x(%d) = %13.7e\n", i, x[i]);
printf("\n");
}

void dnetnf(x, y, n)
int n;
double x[], y[];
| y[0] = x[0] * x[0] + x[1] * x[1] + x[2] * x[2] - 1.0;
y[1] = 2.0 * x[0] * x[0] + x[1] * x[1] - 4.0 * x[2];
y[2] = 3.0 * x[0] * x[0] - 4.0 * x[1] + x[2] * x[2];
n = n;
return;
```

运行结果为:

```
i = 5
x(0) = 7.851969e-01
x(1) = 4.966114e-01
x(2) = 3.699228e-01
```

4.10 求非线性方程组最小二乘解的广义逆法

一、功能

利用广义逆求解无约束条件下的优化问题

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, m-1, m \geq n$$

当 $m = n$ 时, 即为求解非线性方程组。

二、方法说明

设非线性方程组为

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, m-1; m \geq n$$

其雅可比(Jacobi)矩阵为

$$A = \begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} & \dots & \frac{\partial f_0}{\partial x_{n-1}} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_{n-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_{m-1}}{\partial x_0} & \frac{\partial f_{m-1}}{\partial x_1} & \dots & \frac{\partial f_{m-1}}{\partial x_{n-1}} \end{bmatrix}$$

计算非线性方程组最小二乘解的迭代公式为

$$X^{(k+1)} = X^{(k)} - \alpha_k Z^{(k)}$$

其中 $Z^{(k)}$ 为线性代数方程组 $A^{(k)}Z^{(k)} = F^{(k)}$ 的线性最小二乘解, 即

$$Z^{(k)} = (A^{(k)})^{-1}F^{(k)}$$

式中 $A^{(k)}$ 为 k 次迭代值 $X^{(k)}$ 的雅可比矩阵; $F^{(k)}$ 为 k 次迭代值的左端函数值, 即

$$F^{(k)} = (f_0^{(k)}, f_1^{(k)}, \dots, f_{m-1}^{(k)})^T$$

$$f_i^{(k)} = f_i(x_0^{(k)}, x_1^{(k)}, \dots, x_{n-1}^{(k)}), i = 0, 1, \dots, m-1$$

α_k 为使 α 的一元函数 $\sum_{i=0}^{m-1} (f_i^{(k+1)})^2$ 达到极小值的点。在本函数中利用有理极值法计算 α_k 。

三、函数语句

```
int dngin(m, n, eps1, eps2, x, ka)
```

本函数返回一个整型标志。若返回标志值小于 0, 则说明在奇异值分解中迭代超过了 60 次还未达到精度要求; 若返回标志值等于 0, 则说明本函数迭代了 60 次还未达到精度要求; 若返回标志值大于 0, 则正常返回。

本函数要调用以下函数:

(1) 利用广义逆求解线性最小二乘问题 agmiv, 参看 1.14 节。

在函数 agmiv 中又要调用求广义逆的函数 bginv, 参看 2.13 节。

在函数 bginv 中又要调用奇异值分解的函数 bmuav, 参看 2.12 节。

(2) 计算非线性方程组中各方程左端函数值 $f_i(x_0, x_1, \dots, x_{n-1}) (i = 0, 1, \dots, m-1)$ 的函数, 由用户自编, 其形式为:

```
void dnginf(m, n, x, d)
int m, n;
double x[], d[];
d[0] = f_0(x_0, x_1, ..., x_{n-1}) 的表达式;
:
d[m-1] = f_{n-1}(x_0, x_1, ..., x_{n-1}) 的表达式;
```

```
return;
```

(3) 计算雅可比矩阵函数, 由用户自编, 其形式为:

```
void dngins(m, n, x, p)
int m, n;
double x[n], p[m][n]; (其中 m, n 的值见方法说明)
int 用到的整型变量表列;
double 用到的实型变量表列;
p[i][j] = \frac{\partial f_i}{\partial x_j} (i = 0, 1, \dots, m-1; j = 0, 1, \dots, n-1) 的表达式;
return;
}
```

四、形参说明

m——整型变量。非线性方程组中方程个数。

n——整型变量。非线性方程组中未知数个数。

eps1——双精度实型变量。控制最小二乘解的精度要求。

eps2——双精度实型变量。用于奇异值分解中的控制精度要求。

x——双精度实型一维数组, 长度为 n。存放非线性方程组解的初始近似值 $X^{(0)}$, 要求各分量 $x_0^{(0)}, x_1^{(0)}, \dots, x_{n-1}^{(0)}$ 不全为零; 返回时存放最小二乘解, 当 $m = n$ 时, 即为非线性方程组的解。

ka——整型变量。ka = max {m, n} + 1。

五、函数程序(文件名: dngin.c)

六、例

(1) 求解非线性方程组

$$\begin{cases} x_0^2 + 10x_0x_1 + 4x_1^2 + 0.7401006 = 0 \\ x_0^2 - 3x_0x_1 + 2x_1^2 - 1.0201228 = 0 \end{cases}$$

其中 $m = 2, n = 2$ 。取初值为 $(0.5, -1.0)$, $eps1 = 0.000001$, $eps2 = 0.000001$ 。

主函数程序及计算非线性方程组左端各函数值的函数程序、计算雅可比矩阵的函数程序(文件名: dngin0.c)如下:

```
#include "stdio.h"
#include "agmiv.c"
#include "bginv.c"
#include "bmuav.c"
#include "dngin.c"
main()
{
    int m, n, i, ka;
    double eps1, eps2;
    static double x[2] = {0.5, -1.0};
```

```

m = 2; n = 2; ka = 3; eps1 = 0.000001; eps2 = 0.000001;
i = dnginf(m, n, eps1, eps2, x, ka);
printf("\n");
printf("i = %d\n", i);
printf("\n");
for (i = 0; i <= 1; i++)
    printf("x(%d) = %13.7e\n", i, x[i]);
printf("\n");
}

void dnginf(m, n, x, d)
int m, n;
double x[], d[];
{ m = m; n = n;
d[0] = x[0] * x[0] + 10.0 * x[0] * x[1] + 4.0 * x[1] * x[1] + 0.7401006;
d[1] = x[0] * x[0] - 3.0 * x[0] * x[1] + 2.0 * x[1] * x[1] - 1.0201228;
return;
}

void dngins(m, n, x, p)
int m, n;
double x[2], p[2][2];
{ m = m; n = n;
p[0][0] = 2.0 * x[0] + 10.0 * x[1];
p[0][1] = 10.0 * x[0] + 8.0 * x[1];
p[1][0] = 2.0 * x[0] - 3.0 * x[1];
p[1][1] = -3.0 * x[0] + 4.0 * x[1];
return;
}

```

运行结果为：

```

i = 1
x(0) = 3.765472e-01
x(1) = -4.379528e-01

```

(2) 求非线性方程组

$$\begin{cases} x_0^2 + 7x_0x_1 + 3x_1^2 + 0.5 = 0 \\ x_0^2 - 2x_0x_1 + x_1^2 - 1 = 0 \\ x_0 + x_1 + 1 = 0 \end{cases}$$

的最小二乘解。其中 $m = 3, n = 2$ 。取初值 $(1.0, -1.0)$, $eps1 = eps2 = 0.000001$ 。

主函数程序及计算非线性方程组左端函数值的函数程序、计算雅可比矩阵的函数程序
(文件名: dngin1.c)如下：

```

#include "stdio.h"
#include "agmiv.c"
#include "bginv.c"
#include "bmuav.c"
#include "dngin.c"
main()
{ int m, n, i, ka;

```

```

    double eps1, eps2;
    static double x[2] = {1.0, -1.0};
    m = 3; n = 2; ka = 4; eps1 = 0.000001; eps2 = 0.000001;
    i = dnginf(m, n, eps1, eps2, x, ka);
    printf("\n");
    printf("i = %d\n", i);
    printf("\n");
    for (i = 0; i <= 1; i++)
        printf("x(%d) = %13.7e\n", i, x[i]);
    printf("\n");
}

void dnginf(m, n, x, d)
int m, n;
double x[], d[];
{ m = m; n = n;
d[0] = x[0] * x[0] + 7.0 * x[0] * x[1] + 3.0 * x[1] * x[1] + 0.5;
d[1] = x[0] * x[0] - 2.0 * x[0] * x[1] + x[1] * x[1] - 1.0;
d[2] = x[0] + x[1] + 1.0;
return;
}

void dngins(m, n, x, p)
int m, n;
double x[2], p[3][2];
{ m = m; n = n;
p[0][0] = 2.0 * x[0] + 7.0 * x[1];
p[0][1] = 7.0 * x[0] + 6.0 * x[1];
p[1][0] = 2.0 * x[0] - 2.0 * x[1];
p[1][1] = -2.0 * x[0] + 2.0 * x[1];
p[2][0] = 1.0;
p[2][1] = 1.0;
return;
}

```

运行结果为：

```

i = 1
x(0) = 3.789467e-01
x(1) = -6.925762e-01

```

4.11 求非线性方程一个实根的蒙特卡洛法

一、功能

用蒙特卡洛(Monte Carlo)法求非线性方程 $f(x) = 0$ 的一个实根。

二、方法说明

设实函数方程为

$$f(x) = 0$$

蒙特卡洛法求一个实根的过程如下：

选取一个初值 x , 并计算 $F_0 = f(x)$ 。再选取一个 $b > 0$ 。

在区间 $[-b, b]$ 上反复产生均匀分布的随机数 r , 对于每一个 r 计算 $F_1 = f(x + r)$, 直到发现一个 r 使 $|F_1| < |F_0|$ 为止, 此时 $x + r \Rightarrow x$, $F_1 \Rightarrow F_0$ 。如果连续产生了 m 个随机数 r 还不满足 $|F_1| < |F_0|$, 则将 b 减半再进行。

重复上述过程, 直到 $|F_0| < \epsilon$ 为止, 此时的 x 即为非线性方程 $f(x) = 0$ 的一个实根。

在使用本方法时, 如遇到迭代不收敛, 则可以适当调整 b 与 m 的值。

三、函数语句

```
void dmtcl(x, b, m, eps)
```

本函数需要调用产生 0 到 1 之间均匀分布的一个随机数的函数 mrnd1, 参看 13.1 节。

本函数还要调用计算方程左端函数值 $f(x)$ 的函数 dmtclf, 由用户自编, 其形式为:

```
double dmtclf(x)
double x;
double y;
y = f(x)的表达式;
return(y);
|
```

四、形参说明

x —双精度实型变量指针。该指针指向的单元存放初值; 返回方程根的终值。

b —双精度实型变量。均匀分布随机数的端点初值。

m —整型变量。控制调节 b 的参数。

ϵ —双精度实型变量。控制精度要求。

五、函数程序(文件名: dmtcl.c)

六、例

用蒙特卡洛法求实函数方程

$$f(x) = e^{-x^3} - \frac{\sin x}{\cos x} + 800 = 0$$

在区间 $(0, \frac{\pi}{2})$ 内的一个实根。

取初值 $x = 0.5$, $b = 1.0$, $m = 10$, $\epsilon = 0.00001$ 。

主函数程序及计算 $f(x)$ 的函数程序(文件名: dmtcl0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dmtcl.c"
# include "mrnd1.c"
main ()
| int n;
double x, b, eps;
```

```
x = 0.5; b = 1.0; m = 10; eps = 0.00001;
dmtcl(&x, b, m, eps);
printf("\n");
printf("x = %13.7e\n", x);
printf("\n");
```

```
| double dmtclf(x)
double x;
| double y;
y = exp(-x * x * x) - sin(x)/cos(x) + 800.0;
return(y);
|
```

运行结果为:

```
x = 1.569546e+00
```

4.12 求实函数或复函数方程一个复根的蒙特卡洛法

一、功能

用蒙特卡洛(Monte Carlo)法求实函数或复函数方程 $f(z) = 0$ 的一个复根。

二、方法说明

设非线性方程为

$$f(z) = 0$$

左端函数 $f(z)$ 的模函数记为 $\|f(z)\|$ 。

蒙特卡洛法求一个复根的过程如下:

选取一个初值 $z = x + jy$, 其中 $j = \sqrt{-1}$ 。并计算 $F_0 = \|f(z)\|$ 。再选取一个 $b > 0$ 。

在区间 $[-b, b]$ 上反复产生均匀分布的随机数 r_x 与 r_y , 对于每一对 (r_x, r_y) 计算

$$F_1 = \|f(x + r_x + j(y + r_y))\|$$

直到发现一对 (r_x, r_y) 使 $F_1 < F_0$ 为止, 此时

$$x + r_x + j(y + r_y) \Rightarrow z, F_1 \Rightarrow F_0$$

如果连续产生了 m 对随机数 (r_x, r_y) 还不满足 $F_1 < F_0$, 则将 b 减半再进行。

重复上述过程, 直到 $F_0 < \epsilon$ 为止, 此时的 z 即为 $f(z) = 0$ 的一个复根。

在使用本方法时, 如遇到迭代不收敛, 则可以适当调整 b 与 m 的值。

本方法可用于求只包含两个未知量的非线性方程组

$$\begin{cases} f_1(x, y) = 0 \\ f_2(x, y) = 0 \end{cases}$$

的一组实根。此时将 x 当作实部, y 当作虚部。

三、函数语句

```
void demtc(x, y, b, m, eps)
```

本函数需要调用产生 0 到 1 之间均匀分布的一个随机数的函数 mrnd1, 参看 13.1 节。

本函数还要调用计算模函数 $\| f(z) \|$ 值的函数 dcmtcf, 由用户自编, 其形式为:

```
double dcmtcf(x, y)
double x, y;
{ double z;
  z = \| f(x + jy) \| 的表达式;
  return(z);
}
```

四、形参说明

x, y——均为双精度实型变量指针。该指针指向的单元分别存放初值的实部与虚部; 分别返回非线性方程复根终值的实部与虚部。

b——双精度实型变量。均匀分布随机数的端点初值。

m——整型变量。控制调节 b 的参数。

eps——双精度实型变量。控制精度要求。

五、函数程序(文件名: dcmtc.c)

六、例

(1) 求实函数方程

$$f(z) = z^2 - 6z + 13 = 0$$

的一个复根。取初值 $z = 0.5 + 0.5j$, $b = 1.0$, $m = 10$, $\epsilon = 0.00001$ 。其中

$$f(z) = (x^2 - y^2 - 6x + 13) + j(2xy - 6y)$$

主函数程序及计算 $\| f(z) \|$ 值的函数程序(文件名: dcmtc 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dcmtc.c"
# include "mrndl.c"
main()
{ int m;
  double x, y, b, eps;
  x = 0.5; y = 0.5; b = 1.0; m = 10; eps = 0.00001;
  dcmtc(&x, &y, b, m, eps);
  printf("\n");
  printf("z = %e + j %e\n", x, y);
  printf("\n");
}

double dcmtcf(x, y)
double x, y;
{ double u, v, z;
  u = x * x - y * y + x - y - 2.0;
  v = 2.0 * x * y + x + y + 2.0;
  z = sqrt(u * u + v * v);
  return(z);
}
```

运行结果为:

$$z = 3.00000e+00 + j 2.00000e+00$$

(2) 求复函数方程

$$f(z) = z^2 + (1 + j)z - 2 + 2j = 0$$

的一个复根。取初值 $z = 0.5 + 0.5j$, $b = 1.0$, $m = 10$, $\epsilon = 0.00001$ 。其中

$$f(z) = (x^2 - y^2 + x - y - 2) + j(2xy + x + y + 2)$$

主函数程序及计算 $\| f(z) \|$ 值的函数程序(文件名: dcmtc 1.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "dcmtc.c"
# include "mrndl.c"
main()
{ int m;
  double x, y, b, eps;
  x = 0.5; y = 0.5; b = 1.0; m = 10; eps = 0.00001;
  dcmtc(&x, &y, b, m, eps);
  printf("\n");
  printf("z = %e + j %e\n", x, y);
  printf("\n");
}

double dcmtcf(x, y)
double x, y;
{ double u, v, z;
  u = x * x - y * y + x - y - 2.0;
  v = 2.0 * x * y + x + y + 2.0;
  z = sqrt(u * u + v * v);
  return(z);
}
```

运行结果为:

$$z = -2.00000e+00 + j 3.14415e-06$$

4.13 求非线性方程组一组实根的蒙特卡洛法

一、功能

用蒙特卡洛(Monte Carlo)法求非线性方程组

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, n - 1$$

的一组实根。

二、方法说明

设非线性方程组为

$$f_i(x_0, x_1, \dots, x_{n-1}) = 0, i = 0, 1, \dots, n - 1$$

定义模函数为

$$\| F \| = \sqrt{\sum_{i=0}^{n-1} f_i^2}$$

蒙特卡洛法求一组实根的过程如下：

选取初值 $X = (x_0, x_1, \dots, x_{n-1})^T$, 并计算模函数值 $F_0 = \| F \|$ 。再选取一个 $b > 0$ 。
在区间 $[-b, b]$ 上反复产生均匀分布的随机数 $(r_0, r_1, \dots, r_{n-1})$, 对于每一组 $(r_0, r_1, \dots, r_{n-1})$ 计算 $(x_0 + r_0, x_1 + r_1, \dots, x_{n-1} + r_{n-1})^T$ 的模函数值 F_1 , 直到发现一组使 $F_1 < F_0$ 为止, 此时令

$$x_i + r_i \Rightarrow x_i, i = 0, 1, \dots, n - 1$$

$$F_1 \Rightarrow F_0$$

如果连续产生了 m 组随机数还不满足 $F_1 < F_0$, 则将 b 减半再进行。

重复上述过程, 直到 $F_0 < \epsilon$ 为止, 此时的 $X = (x_0, x_1, \dots, x_{n-1})^T$ 即为非线性方程组的一组实根。

在使用本方法时, 如遇到迭代不收敛, 则可以适当调整 b 与 m 的值。

三、函数语句

void dnmvc(x, n, b, m, eps)

本函数需要调用产生 0 到 1 之间均匀分布的一个随机数的函数 mrnd1, 参看 13.1 节。

本函数还要调用计算模函数值 $\| F \|$ 的函数 dnmtcf, 由用户自编, 其形式为:

```
double dnmtcf(x, n)
double x[];
int n;
double f;
f =  $\sqrt{\sum_{i=0}^{n-1} f_i^2}$  的表达式;
return(f);
```

四、形参说明

x——双精度实型一组数组, 长度为 n 。存放一组实根初值; 返回一组实根的终值。

n——整型变量。方程个数, 也是未知量的个数。

b——双精度实型变量。随机数的端点初值。

m——整型变量。控制调节 b 的参数。

eps——双精度实型变量。控制精度要求。

五、函数程序(文件名: dnmvc.c)

六、例

求非线性方程组

$$\begin{cases} 3x_1 + x_2 + 2x_3^2 - 3 = 0 \\ -3x_1 + 5x_2^2 + 2x_1x_3 - 1 = 0 \\ 25x_1x_2 + 20x_3 + 12 = 0 \end{cases}$$

的一组实根。取初值 $X = (0, 0, 0)^T$, $b = 2.0$, $m = 10$, $\epsilon = 0.00001$ 。

主函数程序及计算模函数值的函数程序(文件名: dnmvc.c)如下:

```
#include "math.h"
#include "stdio.h"
#include "dnmtc.c"
#include "mrnd1.c"
main()
{
    int i, n, m;
    double b, eps, x[3] = {0.0, 0.0, 0.0};
    b = 2.0; m = 10; n = 3; eps = 0.00001;
    dnmtc(x, n, b, m, eps);
    printf("\n");
    for (i = 0; i < 2; i++)
        printf("x(%d) = %13.7e\n", i, x[i]);
    printf("\n");
}

double dnmtcf(x, n)
int n;
double x[];
double f, f1, f2, f3;
n = n;
f1 = 3.0 * x[0] + x[1] + 2.0 * x[2] * x[2] - 3.0;
f2 = -3.0 * x[0] + 5.0 * x[1] * x[1] + 2.0 * x[0] * x[2] - 1.0;
f3 = 25.0 * x[0] * x[1] + 20.0 * x[2] + 12.0;
f = sqrt(f1 * f1 + f2 * f2 + f3 * f3);
return(f);
```

运行结果为:

```
x(0) = 1.096032e+00
x(1) = -7.997541e-01
x(2) = 4.956811e-01
```

第5章 插 值

5.1 一元全区间不等距插值

一、功能

给定 n 个不等距结点 x_i ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 用拉格朗日 (Lagrange) 插值公式, 计算指定插值点 t 处的函数近似值 $z = f(t)$ 。

二、方法说明

为了避免龙格 (Runge) 现象对计算结果的影响, 在 n 个结点中自动选择八个结点进行插值, 且使指定插值点 t 位于它们的中间。即选取满足 $x_k < x_{k+1} < x_{k+2} < x_{k+3} < t < x_{k+4} < x_{k+5} < x_{k+6} < x_{k+7}$ 的八个结点, 利用七次拉格朗日插值多项式计算插值点 t 处的函数近似值 $z = f(t)$, 即

$$z = \sum_{i=k}^{k+7} y_i \prod_{\substack{j=k \\ j \neq i}}^{k+7} [(t - x_j) / (x_i - x_j)]$$

当插值点 t 位于包含 n 个结点的区间外时, 将仅取区间某端的四个结点进行插值; 而当插值点 t 靠近 n 个结点的两端时, 选取的结点数将少于八个。

三、函数语句

double enlgr(x, y, n, t)

本函数返回一个双精度实型函数值, 即在插值点 t 处的函数近似值 $z = f(t)$ 。

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放给定 n 个结点的值 x_i , 要求 $x_0 < x_1 < \dots < x_{n-1}$ 。

y ——双精度实型一维数组, 长度为 n 。存放 n 个给定结点上的函数值 $y_i = f(x_i)$ ($i = 0, 1, \dots, n - 1$)。

n ——整型变量。给定结点的个数。

t ——双精度实型变量。存放指定插值点的值。

五、函数程序(文件名:enlgr.c)

六、例

已知一元不等距列表函数如下:

i	1	2	3	4	5
x_i	0.10	0.15	0.25	0.40	0.50
y_i	0.904837	0.860708	0.778801	0.670320	0.606531
i	6	7	8	9	10
x_i	0.57	0.70	0.85	0.93	1.00
y_i	0.565525	0.496585	0.427415	0.394554	0.367879

利用拉格朗日插值公式计算插值点 $t = 0.63$ 处的函数近似值。

主函数程序(文件名:enlgr.c)如下:

```
# include "enlgr.c"
# include "stdio.h"
main()
{ double t, z;
  static double x[10] = {0.10, 0.15, 0.25, 0.40, 0.50,
                        0.57, 0.70, 0.85, 0.93, 1.00};
  static double y[10] = {0.904837, 0.860708, 0.778801, 0.670320, 0.606531,
                        0.565525, 0.496585, 0.427415, 0.394554, 0.367879};
  t = 0.63; z = enlgr(x, y, 10, t);
  printf("\n");
  printf("t = %6.3f, z = %e\n", t, z);
  printf("\n");
}
```

运行结果为:

$t = 0.630, z = 5.32591e-01$

5.2 一元全区间等距插值

一、功能

给定 n 个等距结点 $x_i = x_0 + ih$ ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 用拉格朗日 (Lagrange) 插值公式计算指定插值点 t 处的函数近似值 $z = f(t)$ 。

二、方法说明

同 5.1 节。

三、函数语句

double eelgr(x0, h, n, y, t)

本函数返回一个双精度实型函数值, 即 $z = f(t)$ 。

四、形参说明

$x0$ ——双精度实型变量。等距结点中的第一个结点值。

h ——双精度实型变量。等距结点的步长。

n ——整型变量。等距结点的个数。

y ——双精度实型一维数组，长度为 n 。存放 n 个等距结点上的函数值。

t ——双精度实型变量。指定插值点的值。

五、函数程序(文件名:eelgr.c)

六、例

设函数 $y = e^{-x}$ 的列表如下：

i	1	2	3	4	5
x_i	0.1	0.2	0.3	0.4	0.5
y_i	0.904837	0.818731	0.740818	0.670320	0.606531
i	6	7	8	9	10
x_i	0.6	0.7	0.8	0.9	1.0
y_i	0.548812	0.496585	0.449329	0.406570	0.367879

计算在插值点 $t = 0.25, 0.63, 0.95$ 处的函数近似值。

在利用本函数时，其中 $x0 = 0.1, h = 0.1, n = 10$ 。

主函数程序(文件名:eelgr 0.c)如下：

```
#include "stdio.h"
#include "eelgr.c"

main()
{ double x0, h, t, z;
  static double y[10] = {0.904837, 0.818731, 0.740818, 0.670320, 0.606531,
                        0.548812, 0.496585, 0.449329, 0.406570, 0.367879};

  x0 = 0.1; h = 0.1;
  printf("\n");
  t = 0.25; z = eelgr(x0, h, 10, y, t);
  printf("t = %6.3f,      z = %e\n", t, z, );
  t = 0.63; z = eelgr(x0, h, 10, y, t);
  printf("t = %6.3f,      z = %e\n", t, z, );
  t = 0.95; z = eelgr(x0, h, 10, y, t);
  printf("t = %6.3f,      z = %e\n", t, z, );
  printf("\n");
}
```

运行结果为：

```
t = 0.250,      z = 7.78801e-01
t = 0.630,      z = 5.32592e-01
t = 0.950,      z = 3.86741e-01
```

5.3 一元三点不等距插值

一、功能

给定 n 个不等距结点 $x_i (i = 0, 1, 2, \dots, n-1)$ 上的函数值 $y_i = f(x_i)$ ，用抛物插值公式计算指定插值点 t 处的函数近似值 $z = f(t)$ 。

二、方法说明

设 n 个不等距结点为 $x_0 < x_1 < \dots < x_{n-1}$ ，其相应的函数值为 $y_i (i = 0, 1, \dots, n-1)$ 。

为了计算指定插值点 t 处的函数近似值，选取最靠近插值点 t 的三个结点，即如果 $x_k < t < x_{k+1}$ ，则当 $|x_k - t| > |t - x_{k+1}|$ 时，取三个结点为 x_k, x_{k+1}, x_{k+2} ，而当 $|x_k - t| < |t - x_{k+1}|$ 时，取三个结点为 x_{k-1}, x_k, x_{k+1} ，然后用抛物插值公式计算 t 点的函数近似值，即

$$z = \sum_{i=m}^{m+2} y_i \prod_{\substack{j=m \\ j \neq i}}^{m+2} [(t - x_j) / (x_i - x_j)]$$

其中：

当 $|x_k - t| > |t - x_{k+1}|$ 时， $m = k$ ；

当 $|x_k - t| < |t - x_{k+1}|$ 时， $m = k - 1$ 。

如果插值点 t 位于包含 n 个结点的区间外，则取区间某端的两个结点作线性插值。

三、函数语句

double enlg3(x, y, n, t)

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x ——双精度实型一维数组，长度为 n 。存放给定 n 个不等距结点的值。

y ——双精度实型一维数组，长度为 n 。存放给定 n 个不等距结点上的函数值。

n ——整型变量。给定不等距结点的个数。

t ——双精度实型变量。指定插值点的值。

五、函数程序(文件名:enlg 3.c)

六、例

给定列表函数如下：

x	1.615	1.634	1.702	1.828	1.921
$y = f(x)$	2.41450	2.46459	2.65271	3.03035	3.34066

用抛物插值法求 $f(x)$ 在 $x = 1.682, 1.813$ 处的函数近似值。

主函数程序(文件名:enlg 30.c)如下：

```

#include "stdio.h"
#include "enlg3.c"
main()
{ double t, z;
static double x[5] = {1.615, 1.634, 1.702, 1.828, 1.921};
static double y[5] = {2.41450, 2.46459, 2.65271, 3.03035, 3.34066};
printf("\n");
t = 1.682; z = enlg3(x, y, 5, t);
printf("x = %6.3f, f(x) = %e\n", t, z);
t = 1.813; z = enlg3(x, y, 5, t);
printf("x = %6.3f, f(x) = %e\n", t, z);
printf("\n");
}

```

运行结果为：

```

x = 1.682,      f(x) = 2.59624e+00
x = 1.813,      f(x) = 2.98281e+00

```

5.4 一元三点等距插值

一、功能

给定 n 个等距结点 $x_i = x_0 + ih$ ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 用抛物插值公式计算指定插值点 t 处的函数近似值。

二、方法说明

同 5.3 节。

三、函数语句

```
double eelg3(x0, h, n, y, t)
```

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x_0 ——双精度实型变量。给定 n 个等距结点中的第一个结点值。

h ——双精度实型变量。给定 n 个等距结点的步长。

n ——整型变量。给定等距结点的个数。

y ——双精度实型一维数组, 长度为 n 。存放给定 n 个等距结点上的函数值。

t ——双精度实型变量。存放指定插值点的值。

五、函数程序(文件名: eelg3.c)

六、例

设函数 $f(x) = e^{-x}$ 在 10 个等距结点上的函数值如下表:

x	0.1	0.2	0.3	0.4	0.5
e^{-x}	0.904837	0.818731	0.740818	0.670320	0.606531
x	0.6	0.7	0.8	0.9	1.0
e^{-x}	0.548812	0.496585	0.449329	0.406570	0.367879

计算在插值点 $t = 0.23, 0.63, 0.95$ 处的函数近似值。

在利用本函数时, $x_0 = 0.1, h = 0.1, n = 10$ 。

主函数程序(文件名: eelg3.c)如下:

```

#include "stdio.h"
#include "eelg3.c"
main()
{ double t, z, x0, h;
static double y[10] = {0.904837, 0.818731, 0.740818, 0.670320, 0.606531,
                      0.548812, 0.496585, 0.449329, 0.406570, 0.367879};
x0 = 0.1; h = 0.1;
printf("\n");
t = 0.23; z = eelg3(x0, h, 10, y, t);
printf("t = %6.3f, z = %e\n", t, z);
t = 0.63; z = eelg3(x0, h, 10, y, t);
printf("t = %6.3f, z = %e\n", t, z);
t = 0.95; z = eelg3(x0, h, 10, y, t);
printf("t = %6.3f, z = %e\n", t, z);
printf("\n");

```

运行结果为:

```

t = 0.230, z = 7.94497e-01
t = 0.630, z = 5.32567e-01
t = 0.950, z = 3.86716e-01

```

5.5 连分式不等距插值

一、功能

给定 n 个不等距结点 x_i ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 利用连分式插值法计算指定插值点 t 处的函数近似值。

二、方法说明

设给定的 n 个不等距结点为 $x_0 < x_1 < \dots < x_{n-1}$, 相应的函数值为 y_i ($i = 0, 1, \dots, n - 1$), 则可以构造一个 n 节连分式

$$\varphi(x) = b_0 + \frac{x - x_0}{b_1} + \frac{x - x_1}{b_2} + \dots + \frac{x - x_{n-2}}{b_{n-1}}$$

其中常数项 b_0 及部分分母 b_1, b_2, \dots, b_{n-1} 由下列递推公式计算:

$$\varphi_0(x_j) = y_j, j = 0, 1, \dots, n - 1$$

$$\varphi_{i+1}(x_j) = \frac{x_j - x_i}{\varphi_i(x_j) - \varphi_i(x_i)}, i = 0, 1, \dots, n - 2, j = i + 1, \dots, n - 1$$

$$b_i = \varphi_i(x_i), i = 0, 1, \dots, n - 1$$

在实际进行插值计算时,一般在指定插值点 t 的前后各取四个结点就够了。此时,计算八节连分式的值 $\varphi(t)$ 作为插值点 t 处的函数近似值。

三、函数语句

double enpqz(x, y, n, t)

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x —双精度实型一维数组,长度为 n 。存放给定的 n 个不等距结点值。

y —双精度实型一维数组,长度为 n 。存放给定 n 个不等距结点上的函数值。

n —整型变量。给定不等距结点的个数。

t —双精度实型变量。指定插值点的值。

五、函数程序(文件名:enpqz.c)

六、例

设函数 $f(x) = \frac{1}{1 + 25x^2}$ 的 10 个不等距结点处的函数值如下表:

x	-1.00	-0.80	-0.65	-0.40	-0.30
$f(x)$	0.0384615	0.0588236	0.0864865	0.200000	0.307692
x	0.00	0.20	0.45	0.80	1.00
$f(x)$	1.00000	0.500000	0.164948	0.0588236	0.0384615

利用连分式插值计算 $t = -0.85$ 及 $t = 0.25$ 处的函数近似值。

主函数程序(文件名:enpqz0.c)如下:

```
# include "stdio.h"
# include "enpqz.c"
main()
{ double t, z;
  static double x[10] = { -1.0, -0.8, -0.65, -0.4, -0.3,
    0.0, 0.2, 0.45, 0.8, 1.0 };
  static double y[10] = { 0.0384615, 0.0588236, 0.0864865, 0.2, 0.307692,
    1.0, 0.5, 0.164948, 0.0588236, 0.0384615 };
  printf("\n");
  t = -0.85; z = enpqz(x, y, 10, t);
}
```

```
printf("t = %6.3f,      z = %e\n", t, z);
t = 0.25; z = enpqz(x, y, 10, t);
printf("t = %6.3f,      z = %e\n", t, z);
printf("\n");
```

运行结果为:

```
t = -0.850,      z = 5.24591e-02
t = 0.250,      z = 3.90244e-01
```

5.6 连分式等距插值

一、功能

给定 n 个等距结点 $x_i = x_0 + ih$ ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 利用连分式插值法计算指定插值点 t 处的函数近似值。

二、方法说明

同 5.5 节。

三、函数语句

double eepqz(x0, h, n, y, t)

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

$x0$ —双精度实型变量。 n 个等距结点中的第一个结点值。

h —双精度实型变量。 n 个等距结点的步长。

n —整型变量。给定等距结点的个数。

y —双精度实型一维数组,长度为 n 。存放 n 个等距结点上的函数值。

t —双精度实型变量。指定插值点的值。

五、函数程序(文件名:eepqz.c)

六、例

设函数 $f(x) = \frac{1}{1 + 25x^2}$ 的 11 个等距结点上的函数值如下表:

x	-1.00	-0.80	-0.60	-0.40	-0.20	0.00
$f(x)$	0.0384615	0.0588236	0.100000	0.200000	0.500000	1.00000
x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

利用连分式插值法计算在插值点 $t = -0.75$ 及 $t = -0.05$ 处的函数近似值。

在利用本函数时, $x_0 = -1.00$, $h = 0.20$, $n = 11$ 。

主函数程序(文件名: eepqs 0.c)如下:

```
# include "stdio.h"
# include "eepqs.c"
main()
{
    double x0, h, t, z;
    static double y[11] = {0.0384615, 0.0588236, 0.1, 0.2, 0.5, 1.0,
                          0.5, 0.2, 0.1, 0.0588236, 0.0384615};
    printf("\n");
    x0 = -1.0; h = 0.2;
    t = -0.75; z = eepqs(x0, h, 11, y, t);
    printf("t = %6.3f,      z = %e\n", t, z);
    t = -0.05; z = eepqs(x0, h, 11, y, t);
    printf("t = %6.3f,      z = %e\n", t, z);
    printf("\n");
}
```

运行结果为:

```
t = -0.750,      z = 6.63901e-02
t = -0.050,      z = 9.41176e-01
```

5.7 埃尔米特不等距插值

一、功能

给定 n 个不等距结点 x_i ($i = 0, 1, \dots, n-1$) 上的函数值 $f(x_i)$ 及一阶导数值 $f'(x_i)$, 用埃尔米特(Hermite)插值公式计算指定插值点 t 处的函数近似值 $f(t)$ 。

二、方法说明

设函数 $f(x)$ 在 n 个不等距结点 $x_0 < x_1 < \dots < x_{n-1}$ 上的函数值及一阶导数值分别为:

$$y_i = f(x_i), \quad y'_i = f'(x_i), \quad i = 0, 1, \dots, n-1$$

则 $f(x)$ 可用埃尔米特插值多项式

$$P_{2n-1}(x) = \sum_{i=0}^{n-1} [y_i + (x - x_i)(y'_i - 2y_i l'_i(x_i))] l_i^2(x)$$

近似代替。其中:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n-1} [(x - x_j)/(x_i - x_j)]$$

$$l'_i(x_i) = \sum_{j=0}^{n-1} \frac{1}{x_i - x_j}$$

在实际进行插值计算时, 为了减少计算工作量, 用户可以适当地将远离插值点的结点抛弃。通常, 只需取插值点 t 的前后各四个结点就够了。

三、函数语句

```
double enhmt(x, y, dy, n, t)
```

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x —双精度实型一维数组, 长度为 n 。存放给定 n 个不等距结点的值。

y —双精度实型一维数组, 长度为 n 。存放给定 n 个不等距结点上的函数值。

dy —双精度实型一维数组, 长度为 n 。存放给定 n 个不等距结点上的一阶导数值。

n —整型变量。给定的不等距结点个数。

t —双精度实型变量。指定插值点的值。

五、函数程序(文件名: enhmt.c)

六、例

设函数 $f(x) = e^{-x}$ 在 10 个不等距结点上的函数值与一阶导数值如下表:

x	0.10	0.15	0.30	0.45	0.55
$f(x)$	0.904837	0.860708	0.740818	0.637628	0.576950
$f'(x)$	-0.904837	-0.860708	-0.740818	-0.637628	-0.576950
x	0.60	0.70	0.85	0.90	1.00
$f(x)$	0.548812	0.496585	0.427415	0.406570	0.367879
$f'(x)$	-0.548812	-0.496585	-0.427415	-0.406570	-0.367879

利用埃尔米特插值公式计算在插值点 $t = 0.356$ 处的函数近似值 $f(t)$ 。

主函数程序(文件名: enhmt 0.c)如下:

```
# include "stdio.h"
# include "enhmt.c"
main()
{
    int i;
    double t, z;
    static double x[10] = {0.1, 0.15, 0.3, 0.45, 0.55, 0.6, 0.7, 0.85, 0.9, 1.0};
    static double y[10] = {0.904837, 0.860708, 0.740818, 0.637628, 0.576950,
                          0.548812, 0.496585, 0.427415, 0.406570, 0.367879};
    static double dy[10];
    printf("\n");
    for (i = 0; i < = 9; i++)
        dy[i] = -y[i];
    t = 0.356; z = enhmt(x, y, dy, 10, t);
    printf("t = %6.3f,      z = %e\n", t, z);
    printf("\n");
}
```

运行结果为：

t = 0.356, z = 7.00480e - 01

5.8 埃尔米特等距插值

一、功能

给定 n 个等距结点 x_i ($i = 0, 1, \dots, n-1$) 上的函数值 $f(x_i)$ 及一阶导数值 $f'(x_i)$, 利用埃尔米特(Hermite)插值公式计算指定插值点 t 处的函数近似值 $f(t)$ 。

二、方法说明

同 5.7 节。

三、函数语句

double eehmt(x0, h, n, y, dy, t)

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x_0 ——双精度实型变量。给定 n 个等距结点中的第一个结点值。

h ——双精度实型变量。给定 n 个等距结点的步长。

n ——整型变量。给定等距结点的个数。

y ——双精度实型一维数组, 长度为 n 。存放给定 n 个等距结点上的函数值。

dy ——双精度实型一维数组, 长度为 n 。存放给定 n 个等距结点上的一阶导数值。

t ——双精度实型变量。存放指定插值点的值。

五、函数程序(文件名:eehmt.c)

六、例

设函数 $f(x) = e^{-x}$ 在 10 个等距结点上的函数值与一阶导数值如下表:

x	0.10	0.20	0.30	0.40	0.50
$f(x)$	0.904837	0.818731	0.740818	0.670320	0.606531
$f'(x)$	-0.904837	-0.818731	-0.740818	-0.670320	-0.606531
x	0.60	0.70	0.80	0.90	1.00
$f(x)$	0.548812	0.496585	0.449329	0.406570	0.367879
$f'(x)$	-0.548812	-0.496585	-0.449329	-0.406570	-0.367879

利用埃尔米特插值法计算在插值点 $t = 0.752$ 处的函数近似值。

主函数程序(文件名:eehmt 0.c)如下:

```
# include "stdio.h"
# include "eehmt.c"
main()
{ int i;
  double x0, h, t, z;
  static double y[10] = {0.904837, 0.818731, 0.740818, 0.670320, 0.606531,
                        0.548812, 0.496585, 0.449329, 0.406570, 0.367879};
  static double dy[10];
  printf("\n");
  for (i = 0; i < = 9; i++) dy[i] = -y[i];
  x0 = 0.1; h = 0.1;
  t = 0.752; z = eehmt(x0, h, 10, y, dy, t);
  printf("t = %6.3f, z = %e\n", t, z);
  printf("\n");
```

运行结果为:

t = 0.752, z = 4.71423e - 01

5.9 埃特金不等距逐步插值

一、功能

给定 n 个不等距结点 x_i ($i = 0, 1, \dots, n-1$) 上的函数值 $y_i = f(x_i)$ 及精度要求, 利用埃特金(Aitken)逐步插值法计算指定插值点 t 处的函数近似值 $f(t)$ 。

二、方法说明

设给定的 n 个不等距结点为 $x_0 < x_1 < \dots < x_{n-1}$, 其函数值为 y_i ($i = 0, 1, \dots, n-1$)。

首先, 从给定的 n 个结点中选取最靠近插值点 t 的 m 个结点 $x'_0, x'_1, \dots, x'_{m-1}$, 相应的函数值为 $y'_0, y'_1, \dots, y'_{m-1}$ 。其中 $m \leq n$ 。在本函数程序中, m 的最大值取为 10。

然后用此 m 个结点作埃特金逐步线性插值。

埃特金逐步线性插值的步骤如下:

(1) $y'_0 \Rightarrow p_0, 1 \Rightarrow i$;

(2) $y'_i \Rightarrow z$,

$$p_j + \frac{t - x'_j}{x'_j - x'_i} [p_j - z] \Rightarrow z, j = 0, 1, \dots, i-1$$

(3) $z \Rightarrow p_i$;

(4) 若 $i = m-1$ 或 $|p_i - p_{i-1}| < \epsilon$, 则结束, p_i 即为 $f(t)$ 的近似值; 否则, $i+1 \Rightarrow i$, 转(2)。

三、函数语句

double enatk(x, y, n, t, eps)

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

x——双精度实型一维数组，长度为 n 。存放 n 个不等距结点的值。
y——双精度实型一维数组，长度为 n 。存放 n 个不等距结点上的函数值。
n——整型变量。给定不等距结点的个数。
t——双精度实型变量。指定插值点的值。
eps——双精度实型变量。插值的精度要求。

五、函数程序(文件名:enatk.c)

六、例

设函数 $f(x)$ 在 11 个结点上的函数值如下表：

x	-1.00	-0.80	-0.65	-0.40	-0.30	0.00
$f(x)$	0.0384615	0.0588236	0.0864865	0.200000	0.307692	1.00000
x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

利用埃特金逐步线性插值计算在插值点 $t = -0.75$ 及 $t = 0.05$ 处的函数近似值。取 $\text{eps} = 10^{-6}$ 。

主函数程序(文件名:enatk 0.c)如下：

```
# include "stdio.h"
# include "enatk.c"
main()
{ double t, z, eps;
  static double x[11] = {-1.0, -0.8, -0.65, -0.4, -0.3,
                        0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
  static double y[11] = {0.0384615, 0.0588236, 0.0864865, 0.2, 0.307692,
                        1.0, 0.5, 0.2, 0.1, 0.0588236, 0.0384615};
  eps = 1.0e-6;
  printf("\n");
  t = -0.75; z = enatk(x, y, 11, t, eps);
  printf("t = %6.3f,      z = %e\n", t, z, );
  t = 0.05; z = enatk(x, y, 11, t, eps);
  printf("t = %6.3f,      z = %e\n", t, z, );
  printf("\n");
```

运行结果为：

```
t = -0.750,      z = -3.08891e-03
t = 0.050,      z = 9.59859e-01
```

5.10 埃特金等距逐步插值

一、功能

给定 n 个等距结点 $x_i = x_0 + ih$ ($i = 0, 1, \dots, n-1$) 上的函数值 $y_i = f(x_i)$ 及精度要求，利用埃特金(Aitken)逐步插值法计算指定插值点 t 处的函数近似值 $f(t)$ 。

二、方法说明

同 5.9 节。

三、函数语句

```
double eeatk(x0, h, n, y, t, eps)
```

本函数返回一个双精度实型函数值 $f(t)$ 。

四、形参说明

$x0$ ——双精度实型变量。给定 n 个等距结点中的第一个结点值。

h ——双精度实型变量。给定 n 个等距结点的步长。

n ——整型变量。给定等距结点的个数。

y ——双精度实型一维数组，长度为 n 。存放 n 个等距结点上的函数值。

t ——双精度实型变量。指定插值点的值。

eps ——双精度实型变量。插值精度要求。

五、函数程序(文件名:eeatk.c)

六、例

设函数 $f(x)$ 在 10 个等距结点上的函数值如下表：

x	0.10	0.20	0.30	0.40	0.50
$f(x)$	0.904837	0.818731	0.740818	0.670320	0.606531
x	0.60	0.70	0.80	0.90	1.00
$f(x)$	0.548812	0.496585	0.449329	0.406570	0.367879

利用埃特金逐步线性插值法计算插值点 $t = 0.15$ 及 $t = 0.55$ 处的函数近似值。取 $\epsilon = 10^{-6}$ 。

主函数程序(文件名:eeatk 0.c)如下：

```
# include "stdio.h"
# include "eeatk.c"
main()
{ double t, z, x0, h, eps;
  static double y[10] = {0.904837, 0.818731, 0.740818, 0.670320, 0.606531,
```

```

0.548812, 0.496585, 0.449329, 0.406570, 0.367879];
eps = 1.0e-6; x0 = 0.1; h = 0.1;
printf("\n");
t = 0.15; z = eeatk(x0, h, 10, y, t, eps);
printf("t = %6.3f,      z = %e\n", t, z);
t = 0.55; z = eeatk(x0, h, 10, y, t, eps);
printf("t = %6.3f,      z = %e\n", t, z);
printf("\n");
}

```

运行结果为：

```

t = 0.150,      z = 8.60708e-01
t = 0.550,      z = 5.76950e-01

```

5.11 光滑不等距插值

一、功能

给定 n 个不等距结点 x_i ($i = 0, 1, \dots, n - 1$) 上的函数值 $y_i = f(x_i)$, 利用阿克玛(Aki-ma)方法, 计算指定子区间上的三次插值多项式与指定插值点上的函数值。

二、方法说明

设给定的 n 个不等距结点为 $x_0 < x_1 < \dots < x_{n-1}$, 相应的函数值为 y_i ($i = 0, 1, \dots, n - 1$)。

若在子区间 $[x_k, x_{k+1}]$ ($k = 0, 1, \dots, n - 2$) 上的两个端点处有以下四个条件:

$$\begin{cases} y_k = f(x_k) \\ y_{k+1} = f(x_{k+1}) \\ y'_k = g_k \\ y'_{k+1} = g_{k+1} \end{cases}$$

则在此区间上可以唯一确定一个三次多项式

$$s(x) = s_0 + s_1(x - x_k) + s_2(x - x_k)^2 + s_3(x - x_k)^3$$

并且就用此三次多项式计算该子区间中的插值点 t 处的函数近似值。

根据阿克玛几何条件, g_k 与 g_{k+1} 由下式计算:

$$g_k = \frac{|u_{k+1} - u_k| |u_{k-1} - u_{k-2}| |u_k|}{|u_{k+1} - u_k| + |u_{k-1} - u_{k-2}|}$$

$$g_{k+1} = \frac{|u_{k+2} - u_{k+1}| |u_k| |u_{k-1} - u_{k-2}| |u_{k+1}|}{|u_{k+2} - u_{k+1}| + |u_k - u_{k-1}|}$$

其中

$$u_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

并且在端点处有

$$u_{-1} = 2u_0 - u_1, \quad u_{-2} = 2u_{-1} - u_0$$

$$u_{n-1} = 2u_{n-2} - u_{n-3}, \quad u_n = 2u_{n-1} - u_{n-2}$$

当 $u_{k+1} = u_k$ 与 $u_{k-1} = u_{k-2}$ 时

$$g_k = \frac{u_{k-1} + u_k}{2}$$

当 $u_{k+2} = u_{k+1}$ 与 $u_k = u_{k-1}$ 时

$$g_{k+1} = \frac{u_k + u_{k+1}}{2}$$

最后可以得到区间 $[x_k, x_{k+1}]$ ($k = 0, 1, \dots, n - 2$) 上的三次多项式的系数为:

$$s_0 = y_k$$

$$s_1 = g_k$$

$$s_2 = (3u_k - 2g_k - g_{k+1})/(x_{k+1} - x_k)$$

$$s_3 = (g_{k+1} + g_k - 2u_k)/(x_{k+1} - x_k)^2$$

插值点 t ($t \in [x_k, x_{k+1}]$) 处的函数近似值为

$$s(t) = s_0 + s_1(t - x_k) + s_2(t - x_k)^2 + s_3(t - x_k)^3$$

三、函数语句

```
void enspl(x, y, n, k, t, s)
```

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放给定的 n 个不等距结点值。

y ——双精度实型一维数组, 长度为 n 。存放给定的 n 个不等距结点上的函数值。

n ——整型变量。给定不等距结点的个数。

k ——整型变量。若 $k \geq 0$, 表示只计算第 k ($k = 0, 1, \dots, n - 2$) 个子区间 $[x_k, x_{k+1}]$ 上的三次多项式的系数 s_0, s_1, s_2, s_3 (当 $k \geq n - 1$ 时, 按 $k = n - 2$ 处理); 若 $k < 0$, 表示需要计算指定插值点 t 处的函数近似值 $f(t)$, 并计算插值点 t 所在子区间的三次多项式的系数 s_0, s_1, s_2, s_3 。

t ——双精度实型变量。指定插值点值。当 $k \geq 0$ 时, 此形参对本函数无用, 可任意。

s ——双精度实型一维数组, 长度为 5。其中 s_0, s_1, s_2, s_3 返回三次多项式的系数, s_4 返回指定插值点 t 处的函数近似值(当 $k < 0$ 时)或任意(当 $k \geq 0$ 时)。

五、函数程序(文件名: enspl.c)

六、例

设函数 $f(x)$ 在 11 个给定不等距结点上的函数值如下表:

x	-1.00	-0.95	-0.75	-0.55	-0.30	0.00
$f(x)$	0.0384615	0.0424403	0.0663900	0.116788	0.307692	1.00000
x	0.20	0.45	0.60	0.80	1.00	
$f(x)$	0.500000	0.164948	0.100000	0.0588236	0.0384615	

利用光滑插值计算指定插值点 $t = -0.85$ 与 $t = 0.15$ 处的函数近似值及插值点所在子

区间上的三次多项式

$$s(x) = s_0 + s_1(x - x_k) + s_2(x - x_k)^2 + s_3(x - x_k)^3$$

的系数 s_0, s_1, s_2, s_3 。其中 $t \in [x_k, x_{k+1}]$ 。

主函数程序(文件名:enspl 0.c)如下:

```
# include "enspl.c"
# include "stdio.h"
main()
{ int k, n;
  double t;
  static double x[11] = { -1.0, -0.95, -0.75, -0.55, -0.3, 0.0,
                         0.2, 0.45, 0.6, 0.8, 1.0 };
  static double y[11] = { 0.0384615, 0.0424403, 0.06639, 0.116788,
                        0.307692, 1.0, 0.5, 0.164948, 0.1, 0.0588236, 0.0384615 };
  static double s[5];
  k = -1; n = 11;
  printf("\n");
  t = -0.85; enspl(x, y, n, k, t, s);
  printf("x = %6.3f, f(x) = %e\n", t, s[4]);
  printf("s0 = %e, s1 = %e, s2 = %e, s3 = %e\n", s[0], s[1], s[2], s[3]);
  printf("\n");
  t = 0.15; enspl(x, y, n, k, t, s);
  printf("x = %6.3f, f(x) = %e\n", t, s[4]);
  printf("s0 = %e, s1 = %e, s2 = %e, s3 = %e\n", s[0], s[1], s[2], s[3]);
  printf("\n");
}
```

运行结果为:

```
x = -0.850, f(x) = 5.34042e-02
s0 = 4.24403e-02, s1 = 8.89362e-02, s2 = 2.59985e-01, s3 = -5.29619e-01
x = 0.150, f(x) = 6.16892e-01
s0 = 1.00000e+00, s1 = -4.37798e-01, s2 = -2.55004e+01, s3 = 7.59470e+01
```

5.12 光滑等距插值

一、功能

给定 n 个等距结点 $x_i = x_0 + ih$ ($i = 0, 1, \dots, n-1$) 上的函数值 $y_i = f(x_i)$, 利用阿克玛(Akima)方法, 计算指定子区间上的三次插值多项式与指定插值点上的函数值。

二、方法说明

同 5.11 节。

三、函数语句

```
void eespl(x0, h, n, y, k, t, s)
```

四、形参说明

$x0$ ——双精度实型变量。给定 n 个等距结点中的第一个结点值。

h ——双精度实型变量。给定 n 个等距结点的步长。

n ——整型变量。给定等距结点的个数。

y ——双精度实型一维数组, 长度为 n 。存放给定 n 个等距结点上的函数值。

k ——整型变量。若 $k \geq 0$, 表示只计算第 k ($k = 0, 1, \dots, n-2$) 个子区间 $[x_k, x_{k+1}]$ 上的三次多项式

$$s(x) = s_0 + s_1(x - x_k) + s_2(x - x_k)^2 + s_3(x - x_k)^3$$

的系数 s_0, s_1, s_2, s_3 (当 $k \geq n-1$ 时, 按 $k = n-2$ 处理); 若 $k < 0$, 表示需要计算指定插值点 t 处的函数近似值 $f(t)$, 并计算插值点 t 所在子区间的三次多项式的系数 s_0, s_1, s_2, s_3 。

t ——双精度实型变量。指定插值点值。当 $k \geq 0$ 时, 此形参对本函数无用, 可任意。

s ——双精度实型一维数组, 长度为 5。其中 s_0, s_1, s_2, s_3 返回三次多项式的系数, s_4 返回指定插值点 t 处的函数近似值(当 $k < 0$ 时)或任意(当 $k \geq 0$ 时)。

五、函数程序(文件名: eespl.c)

六、例

设函数 $f(x)$ 在 11 个等距结点上的函数值如下表:

x	-1.00	-0.80	-0.60	-0.40	-0.20	0.00
$f(x)$	0.0384615	0.0588236	0.100000	0.200000	0.500000	1.000000
x	0.20	0.40	0.60	0.80	1.00	
$f(x)$	0.500000	0.200000	0.100000	0.0588236	0.0384615	

利用光滑插值计算插值点 $t = -0.65, 0.25$ 处的函数近似值及插值点所在区间的三次多项式

$$s(x) = s_0 + s_1(x - x_k) + s_2(x - x_k)^2 + s_3(x - x_k)^3$$

的系数 s_0, s_1, s_2, s_3 。其中 $t \in [x_k, x_{k+1}]$, $x0 = -1.0$, $h = 0.2$ 。

主函数程序(文件名:eespl 0.c)如下:

```
# include "eespl.c"
# include "stdio.h"
main()
{ int k, n;
  double t, x0, h;
  static double y[11] = { 0.0384615, 0.0588236, 0.1, 0.2, 0.5,
                        1.0, 0.5, 0.2, 0.1, 0.0588236, 0.0384615 };
  static double s[5];
  k = -1; n = 11; x0 = -1.0; h = 0.2;
  printf("\n");
  t = -0.65; eespl(x0, h, n, y, k, t, s);
```

```

printf("x = %6.3f,      f(x) = %e\n", t, s[4]);
printf("s0 = %e, s1 = %e, s2 = %e, s3 = %e\n", s[0], s[1], s[2], s[3]);
printf("\n");
t = 0.25; espl(x0, h, n, y, k, t, s);
printf("x = %6.3f,      f(x) = %e\n", t, s[4]);
printf("s0 = %e, s1 = %e, s2 = %e, s3 = %e\n", s[0], s[1], s[2], s[3]);
printf("\n");
}

运行结果为:
x = -0.650,      f(x) = 8.82055e-02
s0 = 5.88236e-02, s1 = 1.29011e-01, s2 = 6.30092e-01, s3 = -1.22868e+00
x = 0.250,      f(x) = 4.13068e-01
s0 = 5.00000e-01, s1 = -1.66667e+00, s2 = -2.19697e+00, s3 = 1.51515e+01

```

5.13 第一种边界条件的三次样条 函数插值、微商与积分

一、功能

给定 n 个结点 $x_i (i=0, 1, \dots, n-1)$ 上的函数值 $y_i = f(x_i)$ 及两端点上的一阶导数值 $y'_0 = f'(x_0), y'_{n-1} = f'(x_{n-1})$, 利用三次样条函数计算各结点上的数值导数及插值区间 $[x_0, x_{n-1}]$ 上的积分近似值 $s = \int_{x_0}^{x_{n-1}} f(x) dx$, 并对函数 $f(x)$ 进行成组插值及成组微商。

二、方法说明

设给定的 n 个结点为 $x_0 < x_1 < \dots < x_{n-1}$, 其函数值为 $y_i = f(x_i) (i=0, 1, \dots, n-1)$, 第一种边界条件为

$$\begin{cases} y'_0 = f'(x_0) \\ y'_{n-1} = f'(x_{n-1}) \end{cases}$$

计算其余 $n-2$ 个结点上的一阶导数值 $y'_j (j=1, 2, \dots, n-2)$ 的公式如下:

$$\begin{aligned} a_0 &= 0 \\ b_0 &= y'_0 \\ h_j &= x_{j+1} - x_j, j = 0, 1, \dots, n-2 \\ a_j &= h_{j-1}/(h_{j-1} + h_j), j = 1, 2, \dots, n-2 \\ \beta_j &= 3[(1-a_j)(y_j - y_{j-1})/h_{j-1} + a_j(y_{j+1} - y_j)/h_j], j = 1, 2, \dots, n-2 \\ a_j &= -a_j/[2 + (1-a_j)a_{j-1}], j = 1, 2, \dots, n-2 \\ b_j &= [\beta_j - (1-a_j)b_{j-1}]/[2 + (1-a_j)a_{j-1}], j = 1, 2, \dots, n-2 \\ y'_j &= a_j y'_{j+1} + b_j, j = n-2, n-3, \dots, 1 \end{aligned}$$

计算 n 个结点上的二阶导数值 $y''_j (j=0, 1, \dots, n-1)$ 的公式如下:

$$y''_j = 6(y_{j+1} - y_j)/h_j^2 - 2(2y'_j + y'_{j+1})/h_j, j = 0, 1, \dots, n-2$$

$$y''_{n-1} = 6(y_{n-2} - y_{n-1})/h_{n-2}^2 + 2(2y'_{n-1} + y'_{n-2})/h_{n-2}$$

利用各结点上的数值导数, 并根据辛卜生(Simpson)公式, 可以得到在插值区间 $[x_0, x_{n-1}]$ 上的求积公式为

$$\begin{aligned} s &= \int_{x_0}^{x_{n-1}} f(x) dx \\ &= \frac{1}{2} \sum_{i=0}^{n-2} (x_{i+1} - x_i)(y_i + y_{i+1}) - \frac{1}{24} \sum_{i=0}^{n-2} (x_{i+1} - x_i)^3 (y''_i + y''_{i+1}) \end{aligned}$$

利用各结点上的函数值 y_i 、一阶导数值 $y'_i (i=0, 1, \dots, n-1)$ 计算插值点 t 处的函数值、一阶导数值及二阶导数值的公式如下(其中 $t \in [x_i, x_{i+1}]$):

$$\begin{aligned} y(t) &= \left[\frac{3}{h_i^2} (x_{i+1} - t)^2 - \frac{2}{h_i^3} (x_{i+1} - t)^3 \right] y_i + \left[\frac{3}{h_i^2} (t - x_i)^2 - \frac{2}{h_i^3} (t - x_i)^3 \right] y_{i+1} \\ &\quad + h_i \left[\frac{1}{h_i^2} (x_{i+1} - t)^2 - \frac{1}{h_i^3} (x_{i+1} - t)^3 \right] y'_i - h_i \left[\frac{1}{h_i^2} (t - x_i)^2 - \frac{1}{h_i^3} (t - x_i)^3 \right] y'_{i+1} \\ y'(t) &= \frac{6}{h_i} \left[\frac{1}{h_i^2} (x_{i+1} - t)^2 - \frac{1}{h_i} (x_{i+1} - t) \right] y_i - \frac{6}{h_i} \left[\frac{1}{h_i^2} (t - x_i)^2 - \frac{1}{h_i} (t - x_i) \right] y_{i+1} \\ &\quad + \left[\frac{3}{h_i^2} (x_{i+1} - t)^2 - \frac{2}{h_i} (x_{i+1} - t) \right] y'_i + \left[\frac{3}{h_i^2} (t - x_i)^2 - \frac{2}{h_i} (t - x_i) \right] y'_{i+1} \\ y''(t) &= \frac{1}{h_i^2} \left[6 - \frac{12}{h_i} (x_{i+1} - t) \right] y_i + \frac{1}{h_i^2} \left[6 - \frac{12}{h_i} (t - x_i) \right] y_{i+1} \\ &\quad + \frac{1}{h_i} \left[2 - \frac{6}{h_i} (x_{i+1} - t) \right] y'_i - \frac{1}{h_i} \left[2 - \frac{6}{h_i} (t - x_i) \right] y'_{i+1} \end{aligned}$$

三、函数语句

double espl1(x, y, n, dy, ddy, t, m, z, dz, ddz)

本函数返回一个双精度实型积分值

$$\int_{x_0}^{x_{n-1}} f(x) dx$$

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放给定 n 个结点的值。

y ——双精度实型一维数组, 长度为 n 。存放给定 n 个结点上的函数值。

n ——整型变量。给定结点的个数。

dy ——双精度实型一维数组, 长度为 n 。调用时, $dy(0)$ 存放给定的左端点处的一阶导数值 y'_0 , $dy(n-1)$ 存放给定的右端点处的一阶导数值 y'_{n-1} 。返回时存放 n 个给定结点处的一阶导数值 $y'_i (i=0, 1, \dots, n-1)$ 。

ddy ——双精度实型一维数组, 长度为 n 。返回时存放 n 个给定结点处的二阶导数值 $y''_i (i=0, 1, \dots, n-1)$ 。

t ——双精度实型一维数组, 长度为 m 。存放 m 个指定插值点的值。要求 $x_0 < t(j) < x_{n-1} (j=0, 1, \dots, m-1)$ 。

m ——整型变量。指定插值点的个数。

z ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的函数值。

dz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的一阶导数值。

ddz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的二阶导数值。

五、函数程序(文件名: espl1.c)

六、例

已知某直升飞机旋转机翼外形曲线上的部分坐标值如下表:

x	0.52	8.0	17.95	28.65	50.65	104.6
y	5.28794	13.8400	20.2000	24.9000	31.1000	36.5000
x	156.6	260.7	364.4	468.0	507.0	520.0
y	36.6000	31.0000	20.9000	7.80000	1.50000	0.200000

且两端点上的一阶导数值为: $y'_0 = 1.86548$, $y'_{n-1} = -0.046115$

计算各结点处的一阶及二阶导数值, 在区间 $[0.52, 520.0]$ 内的积分值。并计算在八个插值点 $4.0, 14.0, 30.0, 60.0, 130.0, 230.0, 450.0, 515.0$ 处的函数值、一阶导数值及二阶导数值。

在本例中, $n = 12$, $m = 8$ 。

主函数程序(文件名: espl10.c)如下:

```
# include "espl1.c"
# include "stdio.h"
main()
{ int n, m, i;
  double s;
  static double dy[12], ddy[12], z[8], dz[8], ddz[8];
  static double x[12] = {0.52, 8.0, 17.95, 28.65, 50.65, 104.6,
                        156.6, 260.7, 364.4, 468.0, 507.0, 520.0};
  static double y[12] = {5.28794, 13.84, 20.2, 24.9, 31.1, 36.5,
                        36.6, 31.0, 20.9, 7.8, 1.5, 0.2};
  static double t[8] = {4.0, 14.0, 30.0, 60.0, 130.0, 230.0,
                       450.0, 515.0};
  dy[0] = 1.86548; dy[11] = -0.046115;
  n = 12; m = 8;
  printf("\n");
  s = espl1(x, y, n, dy, ddy, t, m, z, dz, ddz);
  printf(
    "x(i)           y(i)           dy(i)           ddy(i)\n");
  for (i = 0; i <= 11; i++)
  printf("%14.6e%14.6e%14.6e\n", x[i], y[i], dy[i], ddy[i]);
  printf("\n");
```

```
printf("s = %e\n", s);
printf("\n");
printf(
  "t(i)           z(i)           dz(i)           ddz(i)\n");
for (i = 0; i <= 7; i++)
printf("%14.6e%14.6e%14.6e\n", t[i], z[i], dz[i], ddz[i]);
printf("\n");
```

运行结果为:

x(i)	y(i)	dy(i)	ddy(i)
5.20000e-01	5.28794e+00	1.86548e+00	-2.79319e-01
8.00000e+00	1.38400e+01	7.43662e-01	-2.06327e-02
1.79500e+01	2.02000e+01	5.32912e-01	-2.17292e-02
2.86500e+01	2.49000e+01	3.68185e-01	-9.06092e-03
5.06500e+01	3.11000e+01	2.08755e-01	-5.43268e-03
1.04600e+02	3.65000e+01	2.93142e-02	-1.21944e-03
1.56600e+02	3.66000e+01	-2.11538e-02	-7.21641e-04
2.60700e+02	3.10000e+01	-8.15142e-02	-4.38020e-04
3.64400e+02	2.09000e+01	-1.06449e-01	-4.28883e-05
4.68000e+02	7.80000e+00	-1.64223e-01	-1.07244e-03
5.07000e+02	1.50000e+00	-1.35256e-01	2.55796e-03
5.20000e+02	2.00000e-01	-4.61150e-02	1.11560e-02

s = 1.29044e+04

t(i)	z(i)	dz(i)	ddz(i)
4.00000e+00	1.03314e+01	1.10286e+00	-1.58967e-01
1.40000e+01	1.79266e+01	6.17882e-01	-2.12939e-02
3.00000e+01	2.53889e+01	3.56103e-01	-8.83827e-03
6.00000e+01	3.28250e+01	1.61373e-01	-4.70249e-03
1.30000e+02	3.68774e+01	1.42856e-03	-9.76284e-04
2.30000e+02	3.32829e+01	-6.67831e-02	-5.21662e-04
4.50000e+02	1.05919e+01	-1.46529e-01	-8.93563e-04
5.15000e+02	5.56246e-01	-9.36277e-02	7.84907e-03

5.14 第二种边界条件的三次样条函数插值、微商与积分

一、功能

给定 n 个结点 x_i ($i = 0, 1, \dots, n-1$) 上的函数值 $y_i = f(x_i)$ 以及两端点上的二阶导数值 $y''_0 = f''(x_0)$, $y''_{n-1} = f''(x_{n-1})$, 利用三次样条函数计算各结点上的数值导数 y'_i , y''_i ($i = 0, 1, \dots, n-1$) 及插值区间 $[x_0, x_{n-1}]$ 上的积分近似值 $s = \int_{x_0}^{x_{n-1}} f(x) dx$, 并对函数 $f(x)$ 进行成组插值及成组微商。

二、方法说明

设给定的 n 个结点为 $x_0 < x_1 < \dots < x_{n-1}$, 其函数值为 y_i ($i = 0, 1, \dots, n-1$), 第二种边

界条件为

$$\begin{cases} y''_0 = f''(x_0) \\ y''_{n-1} = f''(x_{n-1}) \end{cases}$$

计算 n 个结点上一阶导数值 $y'_j (j=0, 1, \dots, n-1)$ 的公式如下：

$$a_0 = -0.5$$

$$b_0 = 3 \cdot \frac{y_1 - y_0}{2(x_1 - x_0)} - y''_0(x_1 - x_0)/4$$

$$h_j = x_{j+1} - x_j, j = 0, 1, \dots, n-2$$

$$\alpha_j = h_{j-1}/(h_{j-1} + h_j), j = 1, 2, \dots, n-2$$

$$\beta_j = 3[(1-\alpha_j)(y_j - y_{j-1})/h_{j-1} + \alpha_j(y_{j+1} - y_j)/h_j], j = 1, 2, \dots, n-2$$

$$a_j = -a_j/[2 + (1-\alpha_j)a_{j-1}], j = 1, 2, \dots, n-2$$

$$b_j = [\beta_j - (1-\alpha_j)b_{j-1}]/[2 + (1-\alpha_j)a_{j-1}], j = 1, 2, \dots, n-2$$

$$y'_{n-1} = [3(y_{n-1} - y_{n-2})/h_{n-2} + y''_{n-1}h_{n-2}/2 - b_{n-2}]/(2 + a_{n-2})$$

$$y'_j = a_jy'_{j+1} + b_j, j = n-2, n-3, \dots, 0$$

计算 n 个结点上的二阶导数值 $y''_j (j=0, 1, \dots, n-1)$, 插值区间 $[x_0, x_{n-1}]$ 上的积分值, 计算插值点 t 处的函数值、一阶导数值及二阶导数值的公式同 5.13 节。

三、函数语句

```
double espl2(x, y, n, dy, ddy, t, m, z, dz, ddz)
```

本函数返回一个双精度实型的积分值

$$\int_{x_0}^{x_{n-1}} f(x) dx$$

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放 n 个给定结点的值。

y ——双精度实型一维数组, 长度为 n 。存放 n 个给定结点上的函数值。

n ——整型变量。给定结点的个数。

dy ——双精度实型一维数组, 长度为 n 。返回时存放 n 个给定结点上的一阶导数值。

ddy ——双精度实型一维数组, 长度为 n 。调用时, $ddy(0)$ 存放给定的左端点处的二阶导数值 y''_0 , $ddy(n-1)$ 存放给定的右端点处的二阶导数值 y''_{n-1} 。返回时存放 n 个给定结点处的二阶导数值 $y''_i (i=0, 1, \dots, n-1)$ 。

t ——双精度实型一维数组, 长度为 m 。存放 m 个指定插值点的值。要求 $x_0 < t(j) < x_{n-1} (j=0, 1, \dots, m-1)$ 。

m ——整型变量。指定插值点的个数。

z ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的函数值。

dz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的一阶导数值。

ddz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的二阶导数值。

数值。

五、函数程序(文件名: espl2.c)

六、例

同 5.13 节的例, 其中边界条件改为:

$$y''_0 = -0.279319, \quad y''_{n-1} = 0.0111560$$

主函数程序(文件名: espl20.c)如下:

```
# include "espl2.c"
# include "stdio.h"
main()
{ int n, m, i;
  double s;
  static double dy[12], ddy[12], z[8], dz[8], ddz[8];
  static double x[12] = {0.52, 8.0, 17.95, 28.65, 50.65, 104.6,
                        156.6, 260.7, 364.4, 468.0, 507.0, 520.0};
  static double y[12] = {5.28794, 13.84, 20.2, 24.9, 31.1, 36.5,
                        36.6, 31.0, 20.9, 7.8, 1.5, 0.2};
  static double t[8] = {4.0, 14.0, 30.0, 60.0, 130.0, 230.0,
                      450.0, 515.0};
  ddy[0] = -0.279319; ddy[11] = 0.011156;
  n = 12; m = 8;
  printf("\n");
  s = espl2(x, y, n, dy, ddy, t, m, z, dz, ddz);
  printf(
    "x(i)           y(i)           dy(i)           ddy(i)\n");
  for (i = 0; i < 11; i++)
  printf("%14.6e%14.6e%14.6e%14.6e\n", x[i], y[i], dy[i], ddy[i]);
  printf("\n");
  printf("s = %e\n", s);
  printf("\n");
  printf(
    "t(i)           z(i)           dz(i)           ddz(i)\n");
  for (i = 0; i < 7; i++)
  printf("%14.6e%14.6e%14.6e%14.6e\n", t[i], z[i], dz[i], ddz[i]);
  printf("\n");
}
```

运行结果为:

x(i)	y(i)	dy(i)	ddy(i)
5.20000e-01	5.28794e+00	1.86548e+00	-2.79319e-01
8.00000e+00	1.38400e+01	7.43662e-01	-2.06326e-02
1.79500e+01	2.02000e+01	5.32912e-01	-2.17292e-02
2.86500e+01	2.49000e+01	3.68185e-01	-9.06091e-03
5.06500e+01	3.11000e+01	2.08755e-01	-5.43268e-03
1.04600e+02	3.65000e+01	2.93142e-02	-1.21944e-03
1.56600e+02	3.66000e+01	-2.11538e-02	-7.21641e-04
2.60700e+02	3.10000e+01	-8.15142e-02	-4.38020e-04

3.64400e+02	2.09000e+01	-1.06449e-01	-4.28882e-05
4.68000e+02	7.80000e+00	-1.64223e-01	-1.07244e-03
5.07000e+02	1.50000e+00	-1.35256e-01	2.55796e-03
5.20000e+02	2.00000e-01	-4.61151e-02	1.11560e-02
s = 1.29044e+04			
t(i)	z(i)	dz(i)	ddz(i)
4.00000e+00	1.03314e+01	1.10286e+00	-1.58968e-01
1.40000e+01	1.79266e+01	6.17882e-01	-2.12939e-02
3.00000e+01	2.53889e+01	3.56103e-01	-8.83827e-03
6.00000e+01	3.28250e+01	1.61373e-01	-4.70249e-03
1.30000e+02	3.68774e+01	1.42856e-03	-9.76284e-04
2.30000e+02	3.32829e+01	-6.67831e-02	-5.21662e-04
4.50000e+02	1.05919e+01	-1.46529e-01	-8.93563e-04
5.15000e+02	5.56247e-01	-9.36277e-02	7.84906e-03

5.15 第三种边界条件的三次样条函数插值、微商与积分

一、功能

给定 n 个结点 x_i ($i = 0, 1, \dots, n-1$) 上的函数值 $y_i = f(x_i)$ 以及第三种边界条件, 利用三次样条函数, 计算各给定结点上的一阶导数值 y'_i 、二阶导数值 y''_i ($i = 0, 1, \dots, n-1$) 及插值区间 $[x_0, x_{n-1}]$ 上的积分近似值 $s = \int_{x_0}^{x_{n-1}} f(x) dx$, 并对函数 $f(x)$ 进行成组插值及成组微商。

二、方法说明

设给定的 n 个结点为 $x_0 < x_1 < \dots < x_{n-1}$, 其函数值为 y_i ($i = 0, 1, \dots, n-1$)。第三种边界条件为:

$$y_0 = y_{n-1}, \quad y'_0 = y'_{n-1}, \quad y''_0 = y''_{n-1}$$

计算 n 个给定结点上一阶导数值 y'_j ($j = 0, 1, \dots, n-1$) 的公式如下:

$$a_0 = 0, \quad b_0 = 1, \quad c_1 = 0$$

$$h_0 = h_{n-1}, \quad y_0 - y_{n-1} = y_{n-1} - y_{n-2}$$

$$h_j = x_{j+1} - x_j, \quad j = 0, 1, \dots, n-2$$

$$a_j = h_{j-1}/(h_{j-1} + h_j), \quad j = 1, 2, \dots, n-2$$

$$\beta_j = 3[(1 - a_j)(y_j - y_{j-1})/h_{j-1} + a_j(y_{j+1} - y_j)/h_j], \quad j = 1, 2, \dots, n-2$$

$$a_j = -a_{j-1}/[2 + (1 - a_{j-1})a_{j-1}], \quad j = 1, 2, \dots, n-2$$

$$b_j = -(1 - a_{j-1})b_{j-1}/[2 + (1 - a_{j-1})a_{j-1}], \quad j = 1, 2, \dots, n-2$$

$$c_j = [\beta_{j-1} - (1 - a_{j-1})c_{j-1}]/[2 + (1 - a_{j-1})a_{j-1}], \quad j = 1, 2, \dots, n-2$$

$$u_{n-1} = 1, \quad v_{n-1} = 0$$

$$u_j = a_j u_{j+1} + b_j, \quad j = n-2, n-3, \dots, 1$$

$$v_j = a_j v_{j+1} + c_j, \quad j = n-2, n-3, \dots, 1$$

$$y_{n-2}' = [\beta_{n-2} - a_{n-2}v_1 - (1 - a_{n-2})v_{n-2}]/[2 + a_{n-2}u_1 + (1 - a_{n-2})u_{n-2}]$$

$$y'_j = u_{j+1} y'_{n-2} + v_{j+1}, \quad j = 0, 1, \dots, n-3$$

$$y'_{n-1} = y'_0$$

计算 n 个给定结点上的二阶导数值 y''_j ($j = 0, 1, \dots, n-1$), 插值区间 $[x_0, x_{n-1}]$ 上的积分值, 指定插值点 t 处的函数值、一阶导数值及二阶导数值的公式同 5.13 节。

三、函数语句

double espl3(x, y, n, dy, ddy, t, m, z, dz, ddz)

本函数返回一个双精度实型积分值

$$\int_{x_0}^{x_{n-1}} f(x) dx$$

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放 n 个给定结点的值。

y ——双精度实型一维数组, 长度为 n 。存放 n 个给定结点上的函数值。要求 $y_0 = y_{n-1} = 0$

n ——整型变量。给定结点的个数。

dy ——双精度实型一维数组, 长度为 n 。返回时存放 n 个给定结点上的一阶导数值。

ddy ——双精度实型一维数组, 长度为 n 。返回时存放 n 个给定结点上的二阶导数值。

t ——双精度实型一维数组, 长度为 m 。存放 m 个指定插值点的值。

m ——整型变量。指定插值点的个数。

z ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的函数值。

dz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的一阶导数值。

ddz ——双精度实型一维数组, 长度为 m 。返回时存放 m 个指定插值点处的二阶导数值。

五、函数程序(文件名: espl3.c)

六、例

给定间隔为 10° 的 $\sin x$ 函数表, 利用三次样条插值计算间隔为 5° 的 $\sin x$ 函数表, 并计算其一阶、二阶导数值以及在一个周期内的积分值。

在本例中, $n = 37, m = 36$ 。

主函数程序(文件名: espl30.c)如下:

```
#include "math.h"
#include "stdio.h"
#include "espl3.c"
main()
{ int n, m, i, j;
```

```

double u, s;
static double x[37], y[37], dy[37], ddy[37];
static double t[36], z[36], dz[36], ddz[36];
for (i = 0; i <= 36; i++)
    { x[i] = i * 6.2831852/36.0;
      y[i] = sin(x[i]);
    }
for (i = 0; i <= 35; i++)
    t[i] = (0.5 + i) * 6.2831852/36.0;
n = 37; m = 36;
printf("\n");
s = esp13(x, y, n, dy, ddy, t, m, z, dz, ddz);
printf("x(i) y(i) = sin(x) dy(i) = cos(x) ddy(i) = -sin(x)\n");
printf("%6.1f%10.6f%11.6f%12.6f\n", x[0], y[0], dy[0], ddy[0]);
for (i = 0; i <= 35; i++)
    { u = t[i] * 36.0/0.62831852;
      printf("%6.1f%10.6f%11.6f%12.6f\n", u, z[i], dz[i], ddz[i]);
      u = x[i + 1] * 36.0/0.62831852;
      j = i + 1;
      printf("%6.1f%10.6f%11.6f%12.6f\n", u, y[j], dy[j], ddy[j]);
    }
printf("\n");
printf("s = %e\n", s);
printf("\n");
}

```

运行结果为：

x(i)	y(i) = sin(x)	dy(i) = cos(x)	ddy(i) = -sin(x)
0.0	0.000000	0.999995	0.000000
5.0	0.087156	0.996197	-0.087045
10.0	0.173648	0.984803	-0.174089
15.0	0.258818	0.965928	-0.258489
20.0	0.342020	0.939688	-0.342889
25.0	0.422617	0.906310	-0.422080
30.0	0.500000	0.866021	-0.501271
35.0	0.573575	0.819154	-0.572846
40.0	0.642788	0.766040	-0.644421
45.0	0.707105	0.707108	-0.706206
50.0	0.766044	0.642784	-0.767991
55.0	0.819150	0.573578	-0.818108
60.0	0.866025	0.499997	-0.868226
65.0	0.906306	0.422619	-0.905153
70.0	0.939693	0.342018	-0.942080
75.0	0.965923	0.258820	-0.964695
80.0	0.984808	0.173647	-0.987310
85.0	0.996192	0.087156	-0.994926
90.0	1.000000	0.000000	-1.002541
95.0	0.996192	-0.087156	-0.994926
100.0	0.984808	-0.173647	-0.987310
105.0	0.965923	-0.258820	-0.964695

110.0	0.939693	-0.342018	-0.942080
115.0	0.906306	-0.422619	-0.905153
120.0	0.866025	-0.499997	-0.868226
125.0	0.819150	-0.573578	-0.818109
130.0	0.766044	-0.642784	-0.767991
135.0	0.707105	-0.707108	-0.706206
140.0	0.642788	-0.766040	-0.644421
145.0	0.573575	-0.819154	-0.572846
150.0	0.500000	-0.866021	-0.501271
155.0	0.422617	-0.906310	-0.422080
160.0	0.342020	-0.939688	-0.342889
165.0	0.258818	-0.965928	-0.258489
170.0	0.173648	-0.984803	-0.174089
175.0	0.087156	-0.996197	-0.087045
180.0	0.000000	-0.999995	-0.000000
185.0	-0.087155	-0.996197	0.087045
190.0	-0.173648	-0.984803	0.174089
195.0	-0.258818	-0.965928	0.258489
200.0	-0.342020	-0.939688	0.342889
205.0	-0.422617	-0.906310	0.422080
210.0	-0.500000	-0.866021	0.501270
215.0	-0.573575	-0.819154	0.572846
220.0	-0.642788	-0.766041	0.644421
225.0	-0.707105	-0.707108	0.706206
230.0	-0.766044	-0.642784	0.767991
235.0	-0.819150	-0.573578	0.818108
240.0	-0.866025	-0.499997	0.868226
245.0	-0.906306	-0.422619	0.905153
250.0	-0.939693	-0.342018	0.942080
255.0	-0.965923	-0.258820	0.964695
260.0	-0.984808	-0.173647	0.987310
265.0	-0.996192	-0.087156	0.994926
270.0	-1.000000	-0.000000	1.002541
275.0	-0.996192	0.087156	0.994926
280.0	-0.984808	0.173647	0.987310
285.0	-0.965923	0.258820	0.964695
290.0	-0.939693	0.342018	0.942080
295.0	-0.906306	0.422619	0.905153
300.0	-0.866025	0.499997	0.868226
305.0	-0.819150	0.573578	0.818109
310.0	-0.766045	0.642784	0.767991
315.0	-0.707105	0.707108	0.706206
320.0	-0.642788	0.766040	0.644421
325.0	-0.573575	0.819154	0.572846
330.0	-0.500000	0.866021	0.501271
335.0	-0.422617	0.906310	0.422080
340.0	-0.342020	0.939688	0.342889
345.0	-0.258819	0.965928	0.258489
350.0	-0.173648	0.984803	0.174090
355.0	-0.087156	0.996197	0.087045
360.0	-0.000000	0.999995	0.000000

s = 5.07754e - 15

5.16 二元三点插值

一、功能

给定矩形域上 $n \times m$ 个结点 (x_i, y_j) ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$) 上的函数值 $z_{ij} = z(x_i, y_j)$, 利用二元三点插值公式计算指定插值点 (u, v) 处的函数近似值 $w = z(u, v)$ 。

二、方法说明

设给定矩形域上的 $n \times m$ 个结点在两个方向上的坐标分别为

$$x_0 < x_1 < \dots < x_{n-1}$$

$$y_0 < y_1 < \dots < y_{m-1}$$

相应的函数值为

$$z_{ij} = z(x_i, y_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$$

选取最靠近插值点 (u, v) 的 9 个结点, 其两个方向上的坐标分别为

$$x_p < x_{p+1} < x_{p+2}$$

及

$$y_q < y_{q+1} < y_{q+2}$$

然后用二元三点插值公式

$$z(x, y) = \sum_{i=p}^{p+2} \sum_{j=q}^{q+2} \left(\prod_{k=p}^{p+2} \frac{x - x_k}{x_i - x_k} \right) \left(\prod_{l=q}^{q+2} \frac{y - y_l}{y_j - y_l} \right) z_{ij}$$

计算插值点 (u, v) 处的函数近似值。

三、函数语句

double eslq3(x, y, z, n, m, u, v)

本函数返回一个双精度实型函数值 $f(u, v)$ 。

四、形参说明

x——双精度实型一维数组, 长度为 n 。存放给定 $n \times m$ 个结点 X 方向的 n 个坐标。

y——双精度实型一维数组, 长度为 m 。存放给定 $n \times m$ 个结点 Y 方向的 m 个坐标。

z——双精度实型二维数组, 体积为 $n \times m$ 。存放给定 $n \times m$ 个结点上的函数值。

$$z_{ij} = z(x_i, y_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$$

n——整型变量。给定结点在 X 方向上的坐标个数。

m——整型变量。给定结点在 Y 方向上的坐标个数。

u——双精度实型变量。指定插值点的 X 坐标。

v——双精度实型变量。指定插值点的 Y 坐标。

五、函数程序(文件名: eslq 3.c)

六、例

设二元函数

$$z(x, y) = e^{-(x-y)}$$

取以下 6×5 个结点:

$$x_i = 0.2i, i = 0, 1, \dots, 5$$

$$y_i = 0.25j, j = 0, 1, \dots, 4$$

其函数值为

$$z(i, j) = e^{-(x_i-y_j)}, i = 0, 1, \dots, 5; j = 0, 1, \dots, 4$$

利用二元三点插值法计算插值点 $(0.9, 0.8)$ 及 $(0.3, 0.9)$ 处的函数近似值。

主函数程序(文件名: eslq 30.c)如下:

```
# include "stdio.h"
# include "math.h"
# include "eslq3.c"
main()
{
    int i, j;
    double u, v, w;
    static double x[6], y[5], z[6][5];
    for (i = 0; i <= 5; i++) x[i] = 0.2 * i;
    for (j = 0; j <= 4; j++) y[j] = 0.25 * j;
    for (j = 0; j <= 5; j++)
        for (j = 0; j <= 4; j++)
            z[i][j] = exp(-(x[i] - y[j]));
    printf("\n");
    u = 0.9; v = 0.8;
    w = eslq3(x, y, z, 6, 5, u, v);
    printf("x = %7.3f,      y = %7.3f,      z(x, y) = %e\n", u, v, w);
    u = 0.3; v = 0.9;
    w = eslq3(x, y, z, 6, 5, u, v);
    printf("x = %7.3f,      y = %7.3f,      z(x, y) = %e\n", u, v, w);
    printf("\n");
}
```

运行结果为:

$$x = 0.900, \quad y = 0.800 \quad z(x, y) = 9.04786e - 01$$

$$x = 0.300, \quad y = 0.900 \quad z(x, y) = 1.82277e + 00$$

5.17 二元全区间插值

一、功能

给定矩形域上 $n \times m$ 个结点 (x_i, y_j) ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$) 上的函数值

$z_{ij} = z(x_i, y_j)$, 利用二元插值公式计算指定插值点 (u, v) 处的函数近似值 $w = z(u, v)$ 。

二、方法说明

设给定矩形域上的 $n \times m$ 个结点在两个方向上的坐标分别为

$$x_0 < x_1 < \cdots < x_{n-1}$$

$$y_0 < y_1 < \cdots < y_{m-1}$$

相应的函数值为

$$z_{ij} = z(x_i, y_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$$

计算插值点 (u, v) 处的函数值 $w = z(u, v)$ 。

以插值点 (u, v) 为中心, 在 X 方向上, 前后各取四个坐标:

$$x_p < x_{p+1} < x_{p+2} < x_{p+3} < u < x_{p+4} < x_{p+5} < x_{p+6} < x_{p+7}$$

在 Y 方向上, 前后也各取四个坐标:

$$y_q < y_{q+1} < y_{q+2} < y_{q+3} < v < y_{q+4} < y_{q+5} < y_{q+6} < y_{q+7}$$

然后用二元插值公式

$$z(x, y) = \sum_{i=p}^{p+7} \sum_{j=q}^{q+7} \left(\prod_{\substack{k=p \\ k \neq i}}^{p+7} \frac{x - x_k}{x_i - x_k} \right) \left(\prod_{\substack{l=q \\ l \neq j}}^{q+7} \frac{y - y_l}{y_j - y_l} \right) z_{ij}$$

计算插值点 (u, v) 处的函数近似值。

三、函数语句

double eslgq(x, y, z, n, m, u, v)

本函数返回一个双精度实型函数值 $f(u, v)$ 。

四、形参说明

x —双精度实型一维数组, 长度为 n 。存放给定 $n \times m$ 个结点 X 方向的 n 个坐标。

y —双精度实型一维数组, 长度为 m 。存放给定 $n \times m$ 个结点 Y 方向的 m 个坐标。

z —双精度实型二维数组, 体积为 $n \times m$ 。存放给定 $n \times m$ 个结点上的函数值

$$z_{ij} = z(x_i, y_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$$

n —整型变量。给定结点中 X 方向的坐标个数。

m —整型变量。给定结点中 Y 方向的坐标个数。

u —双精度实型变量。指定插值点的 X 方向坐标。

v —双精度实型变量。指定插值点的 Y 方向坐标。

五、函数程序(文件名:eslgq.c)

六、例

设二元函数

$$z(x, y) = e^{-(x-y)}$$

取以下 11×11 个结点:

$$x_i = 0.1i, i = 0, 1, \dots, 10$$

$$y_j = 0.1j, j = 0, 1, \dots, 10$$

其函数值为

$$z(i, j) = e^{-(x_i - y_j)}, i = 0, 1, \dots, 10, j = 0, 1, \dots, 10$$

利用二元插值法计算插值点 $(0.35, 0.65)$ 及 $(0.45, 0.55)$ 处的函数近似值。

主函数程序(文件名: eslgq.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "eslgq.c"
main()
{ int i, j;
  double u, v, w;
  static double x[11], y[11], z[11][11];
  for (i = 0; i <= 10; i++)
    | x[i] = 0.1 * i; y[i] = x[i];
  for (i = 0; i <= 10; i++)
    for (j = 0; j <= 10; j++)
      z[i][j] = exp(-(x[i] - y[j]));
  printf("\n");
  u = 0.35; v = 0.65;
  w = eslgq(x, y, z, 11, 11, u, v);
  printf("x = %7.3f, y = %7.3f, z(x, y) = %e\n", u, v, w);
  u = 0.45; v = 0.55;
  w = eslgq(x, y, z, 11, 11, u, v);
  printf("x = %7.3f, y = %7.3f, z(x, y) = %e\n", u, v, w);
  printf("\n");
}
```

运行结果为:

$x = 0.350$	$y = 0.650$	$z(x, y) = 1.34986e+00$
$x = 0.450$	$y = 0.550$	$z(x, y) = 1.10517e+00$

第6章 数值积分

6.1 变步长梯形求积法

一、功能

用变步长梯形求积法计算定积分 $T = \int_a^b f(x) dx$

二、方法说明

首先用梯形公式计算

$$T_n = \frac{h}{2} [f(a) + f(b)]$$

其中 $n = 1, h = b - a$

然后用下列递推公式计算：

$$T_{2n} = \frac{1}{2} T_n + \frac{h}{2} \sum_{i=0}^{n-1} f(x_i + 0.5h)$$

$2n \Rightarrow n, 0.5h \Rightarrow h$

直至 $|T_{2n} - T_n| < \epsilon$ 为止。

三、函数语句

double fffts(a, b, eps)

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 ffftsf，由用户自编，其形式为：

```
double ffftsf(x)
double x;
double y;
y = f(x) 的表达式;
return(y);
```

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: fffts.c)

六、例

用变步长梯形求积法计算定积分

$$T = \int_0^1 e^{-x^2} dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: fffts 0.c)如下：

```
# include "stdio.h"
# include "fffts.c"
main()
{
    double a, b, eps, t;
    a = 0.0; b = 1.0; eps = 0.000001;
    t = fffts(a, b, eps);
    printf("\n");
    printf("t = %e\n", t);
    printf("\n");
}
```



```
# include "math.h"
double ffftsf(x)
double x;
double y;
y = exp(-x * x);
return(y);
}
```

运行结果为：

t = 7.46824e-01

6.2 变步长辛卜生求积法

一、功能

用变步长辛卜生(Simpson)求积法计算定积分 $S = \int_a^b f(x) dx$ 。

二、方法说明

变步长辛卜生求积法的步骤如下：

- (1) 用梯形公式计算 $T_n = h[f(a) + f(b)]/2$ ，其中 $n = 1, h = b - a$ 。且令 $S_n = T_n$ 。
- (2) 用变步长梯形法则计算

$$T_{2n} = \frac{1}{2} T_n + \frac{h}{2} \sum_{k=0}^{n-1} f(x_k + h/2)$$

- (3) 用辛卜生求积公式计算

$$S_{2n} = (4T_{2n} - T_n)/3$$

- (4) 若 $|S_{2n} - S_n| \geq \epsilon$ ，则令 $2n \Rightarrow n, h/2 \Rightarrow h$ ，转(2)继续进行计算；否则结束， S_{2n} 即为所求积分的近似值。其中 ϵ 为事先给定的求积精度。

三、函数语句

double fsimp(a, b, eps)

• 140 •

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数值 $f(x)$ 的函数 `fsimpf`, 由用户自编, 其形式为:

```
double fsimpf(x)
double x;
| double y;
y=被积函数 f(x)的表达式;
return (y);
|
```

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分的精度要求。

五、函数程序(文件名: `fsimp.c`)

六、例

用变步长辛卜生求积法计算定积分

$$S = \int_0^1 \frac{\ln(1+x)}{1+x^2} dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: `fsimp0.c`)如下:

```
# include "stdio.h"
# include "fsimp.c"
main()
| double a,b,eps,t;
a=0.0;b=1.0;eps=0.000001;
t=fsimp(a,b,eps);
printf("\n");
printf("t=%e\n",t);
printf("\n");
```

```
# include "math.h"
double fsimpf(x)
double x;
| double y;
y=log(1.0+x)/(1.0+x*x);
return(y);
|
```

运行结果为:

t=2.72198e-01

6.3 自适应梯形求积法

一、功能

用自适应梯形求积法计算被积函数 $f(x)$ 在积分区间内有强峰的定积分 $T = \int_a^b f(x) \times dx$ 。

二、方法说明

自适应梯形求积法的过程如下:

首先将积分区间 $[a, b]$ 分割成两个相等的子区间(称为 1 级子区间) $\Delta_0^{(1)}$ 和 $\Delta_1^{(1)}$ 。并在每一个子区间上分别用梯形公式计算积分近似值, 设其结果分别为 $t_0^{(1)}$ 和 $t_1^{(1)}$ 。

然后将子区间 $\Delta_0^{(1)}$ 再分割为两个相等的子区间(称为 2 级子区间) $\Delta_2^{(2)}$ 和 $\Delta_3^{(2)}$ 。并在每一个子区间上也分别用梯形公式计算积分近似值, 设分别为 $t_0^{(2)}$ 和 $t_1^{(2)}$ 。如果下列不等式

$$| t_0^{(1)} - (t_0^{(2)} + t_1^{(2)}) | < \epsilon / 1.4$$

成立, 则保留 $t_0^{(2)}$ 和 $t_1^{(2)}$ 。再将 $\Delta_1^{(1)}$ 也分割为两个相等的 2 级子区间 $\Delta_2^{(2)}$ 和 $\Delta_3^{(2)}$, 其相应的积分近似值为 $t_2^{(2)}$ 和 $t_3^{(2)}$ 。如果下列不等式

$$| t_1^{(1)} - (t_2^{(2)} + t_3^{(2)}) | < \epsilon / 1.4$$

成立, 则保留 $t_2^{(2)}$ 和 $t_3^{(2)}$ 。最后可得到满足精度要求的积分近似值为

$$t = t_0^{(2)} + t_1^{(2)} + t_2^{(2)} + t_3^{(2)}$$

如果在上述不等式中有一个不成立, 则将对应的 2 级子区间再分割成两个相等的 3 级子区间。在考虑 3 级子区间时, 其精度要求变为 $\epsilon / 1.4^2$ 。

同样, 对 3 级子区间中不满足精度要求的子区间又可以分割成两个相等的 4 级子区间, 其精度要求变为 $\epsilon / 1.4^3$ 。依此类推, 这个过程一直进行到在所考虑的所有子区间上都满足精度要求为止。

本算法为递归算法。

三、函数语句

double ffpts(a, b, eps, d)

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 `ffptsf`, 由用户自行编写, 其形式为:

```
double ffptsf(x)
double x;
| double y;
y=被积函数 f(x)的表达式;
return (y);
|
```

四、形参说明

a——双精度实型变量。积分下限。
 b——双精度实型变量。积分上限。要求 $b > a$ 。
 eps——双精度实型变量。积分精度要求。
 d——双精度实型变量。对积分区间进行分割的最小步长,当子区间的宽度小于 d 时,即使没有满足精度要求,也不再往下进行分割。

五、函数程序(文件名: ffpts.c)

六、例

用自适应梯形求积法计算定积分

$$T = \int_{-1}^1 \frac{1}{1 + 25x^2} dx$$

取 $\epsilon = 0.000001$, 最小步长为 0.0001。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: ffpts0.c)如下:

```
# include "stdio.h"
# include "ffpts.c"
main()
{
    double a, b, eps, t, d;
    a = -1.0; b = 1.0; eps = 0.000001; d = 0.0001;
    t = ffpts(a, b, eps, d);
    printf("%n", t);
    printf("t = %e\n", t);
    printf("\n");
}

double ffptsf(x)
double x;
{
    double y;
    y = 1.0 / (1.0 + 25.0 * x * x);
    return(y);
}
```

运行结果为:

t = 5.49363e-01

6.4 龙贝格求积法

一、功能

用龙贝格(Romberg)求积法计算定积分 $T = \int_a^b f(x) dx$ 。

二、方法说明

设 $T_m(h)$ 为步长为 h 时利用 $2m-2$ 阶牛顿-柯特斯(Newton-Cotes)公式计算得到的

结果; $T_m\left(\frac{h}{2}\right)$ 为将步长 h 减半后用 $2m-2$ 阶牛顿-柯特斯公式计算得到的结果。将它们进行线性组合,便得到步长为 h 的 $2m$ 阶牛顿-柯特斯公式,即

$$T_{m+1}(h) = \frac{4^m T_m(h/2) - T_m(h)}{4^m - 1}$$

其中 $T_1(h)$ 为步长为 h 时的梯形公式计算得到的结果, $T_1(h/2)$ 为步长为 $h/2$ 时的梯形公式计算得到的结果。

在实际进行计算时,龙贝格求积法按下表所示的计算格式进行,直到 $|T_{m+1}(h) - T_m(h)| < \epsilon$ 为止。

梯形法则	二阶公式	四阶公式	六阶公式	八阶公式	...
$T_1(h)$	$T_2(h)$	$T_3(h)$	$T_4(h)$	$T_5(h)$...
$T_1\left(\frac{h}{2}\right)$	$T_2\left(\frac{h}{2}\right)$	$T_3\left(\frac{h}{2}\right)$	$T_4\left(\frac{h}{2}\right)$	$T_5\left(\frac{h}{2}\right)$...
$T_1\left(\frac{h}{2^2}\right)$	$T_2\left(\frac{h}{2^2}\right)$	$T_3\left(\frac{h}{2^2}\right)$	$T_4\left(\frac{h}{2^2}\right)$	$T_5\left(\frac{h}{2^2}\right)$...
$T_1\left(\frac{h}{2^3}\right)$	$T_2\left(\frac{h}{2^3}\right)$	$T_3\left(\frac{h}{2^3}\right)$	$T_4\left(\frac{h}{2^3}\right)$	$T_5\left(\frac{h}{2^3}\right)$...
$T_1\left(\frac{h}{2^4}\right)$	$T_2\left(\frac{h}{2^4}\right)$	$T_3\left(\frac{h}{2^4}\right)$	$T_4\left(\frac{h}{2^4}\right)$	$T_5\left(\frac{h}{2^4}\right)$...
\vdots	\vdots	\vdots	\vdots	\vdots	...

在本函数中,最多可以算到 $m=10$,如果此时还达不到精度要求,就取 T_{10} 作为最后结果。

三、函数语句

double fromb(a, b, eps)

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 frombf,由用户自编,其形式为:

```
double frombf(x)
double x;
{
    double y;
    y = 被积函数 f(x) 的表达式;
    return(y);
}
```

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: fromb.c)

六、例

用龙贝格求积法计算定积分

$$T = \int_0^1 \frac{x}{4+x^2} dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: fromb 0.c)如下:

```
# include "stdio.h"
# include "fromb.c"
main()
{
    double a, b, eps, t;
    a=0.0; b=1.0; eps=0.000001;
    t=fromb(a,b,eps);
    printf("\n");
    printf("t= %e\n",t);
    printf("\n");
}

double frombf(x)
double x;
{
    double y;
    y=x/(4.0+x*x);
    return(y);
}
```

运行结果为:

$t = 1.11572e-01$

6.5 计算一维积分的连分式法

一、功能

用连分式法计算定积分 $s = \int_a^b f(x) dx$ 。

二、方法说明

利用变步长梯形法则计算出步长分别为

$$h_i = (b - a)/2^i, i = 0, 1, \dots$$

的一系列积分近似值 $s_i (i = 0, 1, \dots)$ 。显然, 积分近似值 s 是步长 h 的函数 $s(h)$ 。

将函数 $s(h)$ 用如下连分式表示

$$s(h) = b_0 + \frac{h - h_0}{b_1} + \frac{h - h_1}{b_2} + \dots + \frac{h - h_{i-1}}{b_i} + \dots$$

其中 $b_0, b_1, \dots, b_i, \dots$ 可以由一系列的积分近似值点 $(h_i, s_i) (i = 0, 1, \dots)$ 来确定, 其计算公式如下:

$$\begin{aligned} b_0 &= s_0 \\ b_1 &= \frac{h_1 - h_0}{s_1 - b_0} \\ b_{21} &= \frac{h_2 - h_0}{s_2 - b_0}, \quad b_2 = \frac{h_2 - h_1}{b_{21} - b_1} \\ b_{31} &= \frac{h_3 - h_0}{s_3 - b_0}, \quad b_{32} = \frac{h_3 - h_1}{b_{31} - b_1}, \quad b_3 = \frac{h_3 - h_2}{b_{32} - b_2}, \\ &\vdots \end{aligned}$$

一般来说, 如果已知 b_0, b_1, \dots, b_{i-1} , 则在计算出一个新的积分近似值点 (h_i, s_i) 后, 可以用以下递推公式计算 b_i :

$$\left\{ \begin{array}{l} b_{i1} = s_i \\ b_{ij} = \frac{h_i - h_{j-1}}{b_{i,j-1} - b_{j-1}}, j = 1, 2, \dots, i \\ b_i = b_{ii} \end{array} \right.$$

当 $h = 0$ 时, $s(0)$ 即为积分的精确值。即

$$s = s(0) = b_0 - \frac{h_0}{b_1} - \frac{h_1}{b_2} - \dots - \frac{h_{i-1}}{b_i} - \dots$$

在实际进行计算时, 一般取到七节就能满足精度要求。在本函数中, 最多可以取到十节, 如果此时还不满足精度要求, 就直接取当前的积分近似值。

三、函数语句

double ffpqg(a, b, eps)

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 ffpqgf, 由用户自编, 其形式为:

```
double ffpqgf(x)
double x;
{
    double y;
    y=被积函数 f(x) 的表达式;
    return (y);
}
```

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: ffpqg.c)

六、例

用连分式法计算定积分

$$s = \int_0^{4.3} e^{-x^2} dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: ffpqg 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "ffpqg.c"
main()
{
    double a, b, eps, s;
    a = 0.0; b = 4.3; eps = 0.000001;
    s = ffpqg(a, b, eps);
    printf("\n");
    printf("s = %e\n", s);
    printf("\n");
}

double ffpqgf(x)
double x;
{
    double y;
    y = exp(-x * x);
    return(y);
}
```

运行结果为:

$s = 8.86227e - 01$

6.6 高振荡函数求积法

一、功能

用分部积分法计算高振荡函数的积分

$$\int_a^b f(x) \sin mx dx \text{ 与 } \int_a^b f(x) \cos mx dx$$

中 m 比较大。

二、方法说明

当 m 较大时, 积分

$$s_1(m) = \int_a^b f(x) \cos mx dx$$

与

$$s_2(m) = \int_a^b f(x) \sin mx dx$$

称为高振荡积分。

令

$$s(m) = \int_a^b f(x) e^{imx} dx$$

其中 $e^{imx} = \cos mx + j \sin mx$ 。则有

$$s(m) = s_1(m) + j s_2(m)$$

反复利用分部积分法可以得到

$$s(m) = \int_a^b f(x) e^{imx} dx$$

$$\approx - \sum_{k=0}^{n-1} \left(\frac{j}{m}\right)^{k+1} [(f^{(k)}(b) \cos mb - f^{(k)}(a) \cos ma) + j(f^{(k)}(b) \sin mb - f^{(k)}(a) \sin ma)]$$

分离出实部和虚部后就得到

$$s_1(m) = \int_a^b f(x) \cos mx dx$$

$$\approx \sum_{k=0}^{n-1} \frac{1}{m^{k+1}} [f^{(k)}(b) \sin\left(\frac{k\pi}{2} + mb\right) - f^{(k)}(a) \sin\left(\frac{k\pi}{2} + ma\right)]$$

$$s_2(m) = \int_a^b f(x) \sin mx dx$$

$$\approx \sum_{k=0}^{n-1} \frac{-1}{m^{k+1}} [f^{(k)}(b) \cos\left(\frac{k\pi}{2} + mb\right) - f^{(k)}(a) \cos\left(\frac{k\pi}{2} + ma\right)]$$

其中

$$\sin\left(\frac{k\pi}{2} + x\right) = \begin{cases} \sin x, \text{mod}(k, 4) = 0 \\ \cos x, \text{mod}(k, 4) = 1 \\ -\sin x, \text{mod}(k, 4) = 2 \\ -\cos x, \text{mod}(k, 4) = 3 \end{cases}$$

$$\cos\left(\frac{k\pi}{2} + x\right) = \begin{cases} \cos x, \text{mod}(k, 4) = 0 \\ -\sin x, \text{mod}(k, 4) = 1 \\ -\cos x, \text{mod}(k, 4) = 2 \\ \sin x, \text{mod}(k, 4) = 3 \end{cases}$$

n 的大小可根据 $f(x)$ 及 m 的大小而定。一般取 $n = 3$ 左右。

三、函数语句

void fpart(a, b, m, n, fa, fb, s)

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

m——整型变量。被积函数中振荡函数的角频率。

n——整型变量。给定积分区间两端点上 $f(x)$ 的导数最高阶数 + 1。

fa——双精度实型一维数组, 长度为 n 。存放 $f(x)$ 在积分区间端点 $x = a$ 处的各阶导

数值, 即

$$fa(i) = f^{(i)}(a), i = 0, 1, \dots, n - 1$$

fb——双精度实型一维数组, 长度为 n 。存放 $f(x)$ 在积分区间端点 $x = b$ 处的各阶导数值, 即

$$fb(i) = f^{(i)}(b), i = 0, 1, \dots, n - 1$$

s——双精度实型一维数组, 长度为 2。其中 $s(0)$ 返回积分

$$\int_a^b f(x) \cos mx dx$$

的值; $s(1)$ 返回积分

$$\int_a^b f(x) \sin mx dx$$

的值。

五、函数程序(文件名: fpart.c)

六、例

用分部积分法计算下列高振荡积分

$$s_1 = \int_0^{2\pi} x \cos x \cos 30x dx$$

与

$$s_2 = \int_0^{2\pi} x \cos x \sin 30x dx$$

其中 $a = 0.0$, $b = 6.2831852$, $m = 30$ 。

$$\begin{aligned} \text{取 } n = 4 & \quad f(x) = x \cos x, & f'(x) = \cos x - x \sin x \\ & \quad f''(x) = -2 \sin x - x \cos x, & f'''(x) = -3 \cos x + x \sin x \end{aligned}$$

则有

$$\begin{aligned} fa(0) = f^{(0)}(0) &= 0.0, & fa(1) = f^{(1)}(0) &= 1.0 \\ fa(2) = f^{(2)}(0) &= 0.0, & fa(3) = f^{(3)}(0) &= -3.0 \\ fb(0) = f^{(0)}(2\pi) &= 6.2831852, & fb(1) = f^{(1)}(2\pi) &= 1.0 \\ fb(2) = f^{(2)}(0) &= -6.2831852, & fb(3) = f^{(3)}(2\pi) &= -3.0 \end{aligned}$$

主函数程序(文件名: fpart 0.c)如下:

```
#include "fpart.c"
#include "stdio.h"
main()
{
    int n, m;
    double a, b;
    static double s[2], fa[4] = {0.0, 1.0, 0.0, -3.0};
    static double fb[4] = {6.2831852, 1.0, -6.2831852, -3.0};
    a = 0.0; b = 6.2831852;
    m = 30; n = 4;
    fpart(a, b, m, n, fa, fb, s);
    printf("\n");
    printf("s(0) = %e, s(1) = %e\n", s[0], s[1]);
    printf("\n");
}
```

}

运行结果为:

$s(0) = -6.74177e-07, s(1) = -2.09672e-01$

6.7 勒让德-高斯求积法

一、功能

用勒让德-高斯(Legendre-Gauss)求积法计算定积分 $G = \int_a^b f(x) dx$ 。

二、方法说明

对积分变量 x 作变换 $x = \frac{b-a}{2}t + \frac{b+a}{2}$

将原积分化为在区间 $[-1, 1]$ 上的积分, 即

$$\begin{aligned} G &= \int_a^b f(x) dx = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right) dt \\ &= \frac{b-a}{2} \int_{-1}^1 \varphi(t) dt \end{aligned}$$

根据插值求积公式有

$$\int_{-1}^1 \varphi(t) dt = \sum_{k=0}^{n-1} \lambda_k \varphi(t_k)$$

其中 $t_k (k = 0, 1, \dots, n-1)$ 为在区间 $[-1, 1]$ 上的 n 个求积结点, 且

$$\lambda_k = \int_{-1}^1 A_k(t) dt$$

$$A_k(t) = \prod_{\substack{j=0 \\ j \neq k}}^{n-1} [(t - t_j)/(t_k - t_j)]$$

如果 n 个结点 $t_k (k = 0, 1, \dots, n-1)$ 取勒让德(Legendre)多项式

$$\frac{1}{2^n n!} \frac{d^n}{dt^n} [(t^2 - 1)^n]$$

在区间 $[-1, 1]$ 上的 n 个零点, 则上述插值求积公式称为勒让德-高斯求积公式, 其代数精确度为 $2n-1$ 。

下表列出了 n 从 2 到 10 之间的求积结点 t_k 和求积系数 λ_k 。

n	求积结点 t_k	求积系数 λ_k
2	± 0.5773502692	1.0
3	0.0 ± 0.7745966692	0.8888888899 0.5555555556
4	± 0.3399810436 ± 0.8611363116	0.6521451549 0.3478548451

续表

<i>n</i>	求积结点 t_k	求积系数 λ_k
5	0.0	0.5688888889
	± 0.5384693101	0.4786286705
	± 0.9061798459	0.2369268851
6	± 0.2386191861	0.4679139346
	± 0.6612093865	0.3607615731
	± 0.9324695142	0.1713244924
7	0.0	0.4179591837
	± 0.4058451514	0.3818300505
	± 0.7415311856	0.2797053915
	± 0.9491079123	0.1294849662
8	± 0.1834346425	0.3626837834
	± 0.5255324099	0.3137066459
	± 0.7966664774	0.2223810345
	± 0.9602898565	0.1012285363
9	0.0	0.3302393550
	± 0.3242534234	0.3123470770
	± 0.6133714327	0.2606106964
	0.8360311073	0.1806481607
	± 0.9681602395	0.0812743884
10	± 0.1488743390	0.2955242247
	± 0.4333953941	0.2692667193
	± 0.6794095683	0.2190863625
	± 0.8650633667	0.1494513492
	± 0.9739065285	0.0666713443

在本函数中, 取 $n = 5$, 并采用变步长求积法。

三、函数语句

```
double flrgs(a, b, eps)
```

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 flrgsf, 由用户自编, 其形式为:

```
double flrgsf(x)
double x;
double y;
y = 被积函数 f(x) 的表达式;
return(y);
```

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分的精度要求。

五、函数程序(文件名: flrgs.c)

六、例

用勒让德-高斯求积法计算定积分

$$G = \int_{2.5}^{8.4} (x^2 + \sin x) dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: flrgs0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "flrgs.c"
main()
{
    double a, b, eps, g;
    a = 2.5; b = 8.4; eps = 0.000001;
    g = flrgs(a, b, eps);
    printf("\n");
    printf("g = %e\n", g);
    printf("\n");
}

double flrgsf(x)
double x;
double y;
y = x * x + sin(x);
return(y);
}
```

运行结果为:

```
g = 1.92078e+02
```

6.8 拉盖尔-高斯求积法

一、功能

用拉盖尔-高斯(Laguerre-Gauss)求积公式计算半无限区间 $[0, \infty)$ 上的积分

$$G = \int_0^\infty f(x) dx$$

二、方法说明

设半无限区间 $[0, \infty)$ 上的积分为

• 152 •

$$G = \int_0^{\infty} f(x) dx$$

n 点拉盖尔-高斯求积公式为

$$G = \sum_{i=0}^{n-1} \lambda_i f(x_i)$$

其中 $x_i (i=0, 1, \dots, n-1)$ 为 n 阶拉盖尔多项式

$$L_n(x) = e^x \frac{d^n}{dx^n} (x^n e^{-x}), 0 \leq x < +\infty$$

的 n 个零点; λ_i 为求积系数。

在本函数中, 取 $n=5$ 。5 阶拉盖尔-高斯求积公式的结点为

$$\begin{aligned} x_0 &= 0.26355990, & x_1 &= 1.41340290, & x_2 &= 3.59642600 \\ x_3 &= 7.08580990, & x_4 &= 12.64080000 \end{aligned}$$

求积系数为

$$\begin{aligned} \lambda_0 &= 0.6790941054, & \lambda_1 &= 1.638487956, & \lambda_2 &= 2.769426772 \\ \lambda_3 &= 4.315944000, & \lambda_4 &= 7.104896230 \end{aligned}$$

本方法特别适用于计算如下形式的积分:

$$\int_0^{\infty} e^{-x} g(x) dx$$

三、函数语句

double flags()

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 flagsf, 由用户自编, 其形式为

```
double flagsf(x)
double x;
| double y;
y = 被积函数 f(x) 的表达式;
return(y);
```

四、形参说明

无。

五、函数程序(文件名: flags.c)

六、例

计算半无限区间的积分

$$G = \int_0^{\infty} x e^{-x} dx$$

主程序及计算被积函数 $f(x)$ 值的函数程序(文件名: flags0.c)如下:

```
# include "stdio.h"
```

```
# include "flags.c"
main()
| printf("\n");
printf("g= %e\n", flags());
printf("\n");
|
```

```
double flagsf(x)
double x;
| double y;
y = x * exp(-x);
return(y);
|
```

运行结果为:

```
g= 9.99995e-01
```

6.9 埃尔米特-高斯求积法

一、功能

用埃尔米特-高斯(Hermite-Gauss)求积公式计算无限区间 $(-\infty, \infty)$ 上的积分

$$G = \int_{-\infty}^{\infty} f(x) dx$$

二、方法说明

设无限区间 $(-\infty, \infty)$ 上的积分为

$$G = \int_{-\infty}^{\infty} f(x) dx$$

n 点埃尔米特-高斯求积公式为

$$G = \sum_{i=0}^{n-1} \lambda_i f(x_i)$$

其中 $x_i (i=0, 1, \dots, n-1)$ 为 n 阶埃尔米特多项式

$$H_n(x) = (-1)^n x^n \frac{d^n}{dx^n} (e^{-x^2}), -\infty < x < +\infty$$

的 n 个零点; λ_i 为求积系数。

在本函数中, 取 $n=5$ 。5 阶埃尔米特-高斯求积公式的结点为

$$\begin{aligned} x_0 &= -2.02018200, & x_1 &= -0.95857190, & x_2 &= 0.0 \\ x_3 &= 0.95857190, & x_4 &= 2.02018200 \end{aligned}$$

求积系数为

$$\begin{aligned} \lambda_0 &= 1.181469599, & \lambda_1 &= 0.9865791417, & \lambda_2 &= 0.9453089237 \\ \lambda_3 &= 0.9865791417 & \lambda_4 &= 1.181469599 \end{aligned}$$

本方法特别适用于计算如下形式的积分:

$$\int_{-\infty}^{\infty} e^{-x^2} g(x) dx$$

三、函数语句

```
double fhmgs()
```

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 $fhmgsf$, 由用户自编, 其形式为

```
double fhmgsf(x)
double x;
{ double y;
  y = 被积函数 f(x) 的表达式;
  return(y);
}
```

四、形参说明

无。

五、函数程序(文件名: fhmgs.c)

六、例

计算无限区间积分

$$G = \int_{-\infty}^{\infty} x^2 e^{-x^2} dx$$

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: fhmgs 0.c)如下:

```
# include "stdio.h"
# include "fhmgs.c"
main()
{ printf("\n");
  printf("g= %e\n", fhmgs());
  printf("\n");
}

double fhmgsf(x)
double x;
{ double y;
  y = x*x * exp(-x*x);
  return(y);
}
```

运行结果为:

```
g= 8.86223e - 01
```

6.10 切比雪夫求积法

一、功能

用切比雪夫(Chebyshev)求积公式计算定积分 $s = \int_a^b f(x) dx$ 。

二、方法说明

对积分变量 x 作变换

$$x = \frac{b-a}{2}t + \frac{b+a}{2}$$

将原积分化为在区间 $[-1, 1]$ 上的积分

$$s = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right) dt = \frac{b-a}{2} \int_{-1}^1 \varphi(t) dt$$

切比雪夫求积公式为

$$\int_{-1}^1 \varphi(t) dt = \frac{2}{n} \sum_{k=0}^{n-1} \varphi(t_k)$$

当 $n=5$ 时, 有

$$t_0 = -0.8324975, \quad t_1 = -0.3745414$$

$$t_2 = 0.0, \quad t_3 = 0.3745414$$

$$t_4 = 0.8324975$$

三、函数语句

```
double fcbsv(a, b, eps)
```

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 $fcbsvf$, 由用户自编, 其形式为:

```
double fcbsvf(x)
double x;
{ double y;
  y = 被积函数 f(x) 的表达式;
  return(y);
}
```

四、形参说明

a ——双精度实型变量。积分下限。

b ——双精度实型变量。积分上限。要求 $b > a$ 。

eps ——双精度实型变量。积分精度要求。

五、函数程序(文件名: fcbsv.c)

六、例

用切比雪夫求积法计算定积分

• 156 •

$$s = \int_{2.5}^{8.4} (x^2 + \sin x) dx$$

取 $\epsilon = 0.000001$ 。

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: fcbsv 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "fcbsv.c"
main()
{
    double a, b, eps, s;
    a = 2.5; b = 8.4; eps = 0.000001;
    s = fcbsv(a, b, eps);
    printf("\n");
    printf("s = %e\n", s);
    printf("\n");
}

double fcbsvf(x)
double x;
{
    double y;
    y = x * x + sin(x);
    return(y);
}
```

运行结果为:

$s = 1.92078e+02$

6.11 计算一维积分的蒙特卡洛法

一、功能

用蒙特卡洛(Monte Carlo)法计算定积分

二、方法说明

设定积分为

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

取 0 到 1 之间均匀分布的随机数序列 γ_i ($i = 0, 1, \dots, m - 1$), 并令

$$x_i = a + (b - a)\gamma_i, \quad i = 0, 1, \dots, m - 1$$

只要 m 足够大, 则有

$$s \approx \frac{b - a}{m} \sum_{i=0}^{m-1} f(x_i)$$

在本函数中, 取 $m = 10000$ 。

三、函数语句

double fmtcl(a, b)

本函数返回一个双精度实型积分值。

本函数需要调用计算被积函数 $f(x)$ 值的函数 fmtclf, 由用户自编, 其形式为:

```
double fmtclf(x)
double x;
{
    double y;
    y = 被积函数 f(x) 的表达式;
    return(y);
}
```

本函数还要调用产生 0 到 1 之间均匀分布随机数的函数 mrnd1, 参看 13.1 节。

四、形参说明

a——双精度实型变量。积分下限。

b——双精度实型变量。积分上限。

五、函数程序(文件名: fmtcl.c)

六、例

用蒙特卡洛法计算定积分

$$s = \int_{2.5}^{8.4} (x^2 + \sin x) dx$$

主函数程序及计算被积函数 $f(x)$ 值的函数程序(文件名: fmtcl 0.c)如下:

```
# include "stdio.h"
# include "fmtcl.c"
# include "mrnd1.c"
main()
{
    printf("\n");
    printf("s = %e\n", fmtcl(2.5, 8.4));
    printf("\n");
}
```

```
double fmtclf(x)
double x;
{
    double y;
    y = x * x + sin(x);
    return(y);
}
```

运行结果为:

$s = 1.91553e+02$

6.12 变步长辛卜生二重积分法

一、功能

用变步长辛卜生(Simpson)方法计算二重积分

$$s = \int_a^b dx \int_{y_0(x)}^{y_1(x)} f(x, y) dy$$

二、方法说明

首先将二重积分化为两个单积分，即

$$g(x) = \int_{y_0(x)}^{y_1(x)} f(x, y) dy \text{ 与 } s = \int_a^b g(x) dx$$

然后对每个单积分采用变步长辛卜生法则。其计算步骤如下。

1. 固定一个 x , 设为 \bar{x} 。

(1) 用梯形公式计算

$$t_1 = [y_1(\bar{x}) - y_0(\bar{x})][f(\bar{x}, y_0(\bar{x})) + f(\bar{x}, y_1(\bar{x}))]/2$$

(2) 将区间分半, 每一个子区间长度为

$$h_k = [y_1(\bar{x}) - y_0(\bar{x})]/2^k, k = 1, 2, \dots$$

用辛卜生公式计算

$$t_{k+1} = \frac{1}{2} t_k + h_k \sum_{i=1}^n f(\bar{x}, y_0(\bar{x}) + (2i-1)h_k)$$

$$g_k = (4t_{k+1} - t_k)/3$$

其中 $n = 2^{k-1}$ 。

重复(2), 直至 $|g_k - g_{k-1}| < \epsilon(1 + |g_k|)$ 为止, 此时即有 $g(\bar{x}) = g_k$ 。

2. 利用 1. 中所计算得到的一系列 $g(\bar{x})$ 值计算二重积分的近似值 s 。

(1) 用梯形公式计算 $u_1 = (b-a)(g(b) + g(a))/2$

(2) 将区间二等分, 其子区间长度为 $h'_k = (b-a)/2^k, k = 1, 2, \dots$

用辛卜生公式计算

$$u_{k+1} = \frac{1}{2} u_k + h'_k \sum_{i=1}^n g(a + h'_k(2i-1))$$

$$s_k = (4u_{k+1} - u_k)/3$$

其中

$$n = 2^{k-1}$$

重复(2), 直至 $|s_k - s_{k-1}| < \epsilon(1 + |s_k|)$ 为止, 此时即有 $s \approx s_k$ 。

三、函数语句

double fsim2(a, b, eps)

本函数返回一个双精度实型积分值。

本函数需要调用计算上、下限 $y_1(x), y_0(x)$ (要求 $y_1(x) > y_0(x)$) 的函数 fsim2s, 由用户自编, 其形式为:

```
void fsim2s(x, y)
double x, y[2];
| y[0] = 下限 y_0(x) 的表达式;
| y[1] = 上限 y_1(x) 的表达式;
| return;
```

本函数还要调用计算被积函数 $f(x, y)$ 值的函数 fsim2f, 由用户自编, 其形式为:

```
double fsim2f(x, y)
double x, y;
| double z;
| z = 被积函数 f(x, y) 的表达式;
| return(z);
|
```

四、形参说明

a——双精度实型变量。对单变量 x 的积分下限。

b——双精度实型变量。对单变量 x 的积分上限。要求 $b > a$

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: fsim 2.c)

六、例

用变步长辛卜生求积法计算二重积分

$$s = \int_0^1 dx \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} e^{x^2+y^2} dy$$

取 $\epsilon = 0.0001$ 。

主函数程序及计算上、下限值 $y_1(x), y_0(x)$ 的函数程序与计算被积函数 $f(x, y)$ 值的函数程序(文件名: fsim 2 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "fsim2.c"
main()
| double a, b, eps, s;
| a = 0.0; b = 1.0; eps = 0.0001;
| s = fsim2(a, b, eps);
| printf("\n");
| printf("s = %e\n", s);
| printf("\n");
```

```
void fsim2s(x, y)
double x, y[2];
| y[0] = -sqrt(1.0 - x * x);
| y[1] = -y[0];
| return;
```

```
double fsim2f(x, y)
double x, y;
| double z;
```

```

z = exp(x * x + y * y);
return(z);
}

```

运行结果为：

s = 2.69893e+00

6.13 计算多重积分的高斯方法

一、功能

用高斯(Gauss)方法计算 n 重积分

$$s = \int_{c_0}^{d_0} dx_0 \int_{c_1(x_0)}^{d_1(x_0)} dx_1 \int_{c_2(x_0, x_1)}^{d_2(x_0, x_1)} dx_2 \cdots \int_{c_{n-1}(x_0, x_1, \dots, x_{n-2})}^{d_{n-1}(x_0, x_1, \dots, x_{n-2})} f(x_0, x_1, \dots, x_{n-1}) dx_{n-1}$$

二、方法说明

在计算 n 重积分时，分别将 $0, 1, \dots, n-1$ 层区间分为各自相等的 $js_0, js_1, \dots, js_{n-1}$ 个子区间。首先求出各层积分区间上的第一个子区间中第一个高斯型点 $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_{n-1}$ ；然后固定 $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_{n-2}$ ；按高斯方法计算最内层(即第 $n-1$ 层)的积分，再从内到外计算各层积分值。最后就得到所要求的 n 重积分的近似值。

计算单重积分的高斯方法见 2.5 节的方法说明。

在本函数中，每个子区间上取五个高斯型结点。

三、函数语句

```
double fgaus(n, js)
```

本函数返回一个双精度实型积分值。

本函数需要调用计算各层积分上、下限(要求所有的上限 > 下限)的函数 fgauss，由用户自编，其形式为：

```

void fgauss(j, n, x, y)
int j, n;
double x[], y[2];
switch(j)
{
    case 0: | y[0] = c_0;
              y[1] = d_0;
              break;
    case 1: | y[0] = c_1(x_0) 的表达式;
              y[1] = d_1(x_0) 的表达式;
              break;
    ...
    case n-1: | y[0] = c_{n-1}(x_0, x_1, \dots, x_{n-2}) 的表达式;
                y[1] = d_{n-1}(x_0, x_1, \dots, x_{n-2}) 的表达式;
                break;
}

```

```

        |
        default: |
        |
        return;
}

```

本函数还要调用计算被积函数 $f(x_0, x_1, \dots, x_{n-1})$ 值的函数 fgauf，由用户自编，其形式为：

```

double fgauf(n, x)
int n;
double x[];
| double z;
z = 被积函数 f(x_0, x_1, \dots, x_{n-1}) 的表达式;
return (z);
|

```

四、形参说明

n ——整型变量。积分重数。

js ——整型一维数组，长度为 n 。其 $js(i)(i=0, 1, \dots, n-1)$ 表示第 i 层积分区间所划分的子区间个数。

五、函数程序(文件名：fgaus.c)

六、例

用高斯求积法计算三重积分

$$s = \int_0^1 dx \int_0^{\sqrt{1-x^2}} dy \int_{\sqrt{x^2+y^2}}^{\sqrt{2-x^2-y^2}} z^2 dz$$

其中 $n = 3$ 。若将变量 x, y, z 分别用 x_0, x_1, x_2 表示，则有

$$c_0 = 0.0, \quad d_0 = 1.0$$

$$c_1(x_0) = 0.0, \quad d_1(x_0) = \sqrt{1 - x_0^2}$$

$$c_2(x_0, x_1) = \sqrt{x_0^2 + x_1^2}, \quad d_2(x_0, x_1) = \sqrt{2 - x_0^2 - x_1^2}$$

及

$$f(x_0, x_1, x_2) = x_2^2$$

设将每一层的积分区均分为 4 个子区间，即 $js_0 = js_1 = js_2 = 4$ 。

主函数程序、计算各层积分上下限的函数程序及计算被积函数值的函数程序(文件名：fgaus 0, c)如下：

```

# include "math.h"
# include "stdio.h"
# include "fgaus.c"
main()
{
    static int js[3] = {4, 4, 4};
    double s;
    s = fgaus(3, js);
    printf("\n");
}

```

```

printf("s = %e\n", s);
printf("\n");
}

void fgauss(j, n, x, y)
int j, n;
double x[], y[];
{
    double q;
    n = n;
    switch (j)
    {
        case 0: if y[0] == 0.0; y[1] = 1.0; break;
        case 1: if y[0] == 0.0; y[1] = sqrt(1.0 - x[0] * x[0]); break;
        case 2: {q = x[0] * x[0] + x[1] * x[1]; y[0] = sqrt(q);
                  y[1] = sqrt(2.0 - q); break;
        }
        default: {}
    }
    return;
}

double fgausf(n, x)
int n;
double x[];
{
    double z;
    n = n;
    z = x[2] * x[2];
    return(z);
}

```

运行结果为：

s = 3.82944e-01

6.14 计算二重积分的连分式法

一、功能

用连分式法计算二重积分

$$s = \int_a^b dx \int_{y_0(x)}^{y_1(x)} f(x, y) dy$$

二、方法说明

首先将二重积分化为两个单积分

$$s(x) = \int_{y_0(x)}^{y_1(x)} f(x, y) dy \text{ 与 } s = \int_a^b s(x) dx$$

然后利用连分式法计算每个单积分。关于用连分式法计算单积分的详细叙述见 6.5 节的方法说明。

计算二重积分的步骤如下。

1. 固定一个 x , 设为 \bar{x} , 用连分式法计算单积分

$$s(\bar{x}) = \int_{y_0(\bar{x})}^{y_1(\bar{x})} f(\bar{x}, y) dy$$

2. 利用 1. 中所计算得到的一系列 $s(\bar{x})$ 值, 再利用连分式法计算 $s = \int_a^b s(x) dx$

三、函数语句

double fpqg2(a, b, eps)

本函数返回一个双精度实型积分值。

在本函数中需要调用计算上下限 $y_1(x)$ 与 $y_0(x)$ (要求 $y_1(x) > y_0(x)$) 的函数 fpqg2s 及计算被积函数 $f(x, y)$ 值的函数 fpqg2f。这两个函数由用户自编, 它们的形式为:

```

void fpqg2s(x, y)
double x, y[2];
{
    y[0] = 下限 y_0(x) 的表达式;
    y[1] = 上限 y_1(x) 的表达式;
    return;
}

```

double fpqg2f(x, y)

```

double x, y;
{
    double z;
    z = 被积函数 f(x, y) 的表达式;
    return (z);
}

```

四、形参说明

a——双精度实型变量。积分变量 x 的积分下限。

b——双精度实型变量。积分变量 x 的积分上限。要求 $b > a$ 。

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: fpqg2.c)

六、例

用连分式法计算二重积分

$$s = \int_0^1 dx \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} e^{x^2+y^2} dy$$

其中 $a = 0.0$, $b = 1.0$, $y_0(x) = -\sqrt{1-x^2}$, $y_1(x) = \sqrt{1-x^2}$, $f(x, y) = e^{x^2+y^2}$, 取 $eps = 0.00005$ 。

主函数程序、计算上下限 $y_1(x)$ 与 $y_0(x)$ 的函数程序以及计算被积函数 $f(x, y)$ 值的函数程序(文件名: fpqg20.c)如下:

```

#include "math.h"
#include "stdio.h"
#include "fpqg2.c"
main()

```

```

    | double a,b,eps,s;
    a=0.0; b=1.0; eps=0.00005;
    s=fpqg2(a,b,eps);
    printf("\n");
    printf("s=%e\n",s);
    printf("\n");
}

void fpqg2s(x,y)
double x,y[2];
{ y[1]=sqrt(1.0-x*x);
  y[0]=-y[1];
  return;
}

double fpqg2f(x,y)
double x,y;
{ double z;
  z=exp(x*x+y*y);
  return(z);
}

```

运行结果为：

s = 2.69911e+00

6.15 计算多重积分的蒙特卡洛法

一、功能

用蒙特卡洛(Monte Carlo)法计算多重积分

$$s = \int_{a_0}^{b_0} \int_{a_1}^{b_1} \cdots \int_{a_{n-1}}^{b_{n-1}} f(x_0, x_1, \dots, x_{n-1}) dx_0 dx_1 \cdots dx_{n-1}$$

二、方法说明

取 0 到 1 之间均匀分布的随机数点列

$$(t_0^{(i)}, t_1^{(i)}, \dots, t_{n-1}^{(i)}), i = 0, 1, \dots, m - 1$$

并令

$$x_j^{(i)} = a_j + (b_j - a_j)t_j^{(i)}, j = 0, 1, \dots, n - 1$$

只要 m 足够大，则有

$$s = \frac{1}{m} \left[\sum_{i=0}^{m-1} f(x_0^{(i)}, x_1^{(i)}, \dots, x_{n-1}^{(i)}) \right] \prod_{j=0}^{n-1} (b_j - a_j)$$

在本函数中，取 $m = 10000$ 。

三、函数语句

double ftmlf(n, a, b)

本函数返回一个双精度实型积分值。

本函数需要调用产生 0 到 1 之间均匀分布随机数的函数 mrnd1，参看 13.1 节。

本函数还要调用计算被积函数 $f(x_0, x_1, \dots, x_{n-1})$ 值的函数 ftmlf，由用户自编，其形式为：

```

double ftmlf(n,x)
int n;
double x[ ];
| double y;
| y = 被积函数 f(x_0, x_1, \dots, x_{n-1}) 的表达式;
| return(y);
|

```

四、形参说明

n——整型变量。积分的重数。

a——双精度实型一维数组，长度为 n。存放积分的下限值 a_0, a_1, \dots, a_{n-1} 。

b——双精度实型一维数组，长度为 n。存放积分的上限值 b_0, b_1, \dots, b_{n-1} 。

五、函数程序(文件名：ftmlf.c)

六、例

用蒙特卡洛法计算三重积分

$$x = \int_1^2 \int_1^2 \int_1^2 (x_0^2 + x_1^2 + x_2^2) dx_0 dx_1 dx_2$$

主函数程序及计算被积函数值的函数程序(文件名：ftmlf 0.c)如下：

```

#include "stdio.h"
#include "ftmlf.c"
#include "mrnd1.c"
main()
{ static double a[3]={1.0,1.0,1.0};
  static double b[3]={2.0,2.0,2.0};
  printf("\n");
  printf("s=%e\n",ftmlf(3,a,b));
  printf("\n");
}

double ftmlf(n,x)
int n;
double x[ ];
| int i;
| double f;
| f=0.0;
| for (i=0; i<=n-1; i++)
|   f=f+x[i]*x[i];
| return(f);
|

```

运行结果为：

s = 6.97043e+00

第7章 常微分方程(组)的求解

7.1 全区间积分的定步长欧拉方法

一、功能

用改进的欧拉(Euler)公式对一阶微分方程组进行定步长全区间积分。

二、方法说明

设一阶微分方程组

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), y(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

已知 $t=t_{j-1}$ 点上的函数值 $y_{i,j-1}$ ($i=0, 1, \dots, n-1$)，求 $t_j=t_{j-1}+h$ 点处的函数值 y_{ij} ($i=0, 1, \dots, n-1$)。

改进的欧拉公式为：

$$\begin{aligned} p_i &= y_{i,j-1} + h f_i(t_{j-1}, y_{0,j-1}, \dots, y_{n-1,j-1}), i = 0, 1, \dots, n-1 \\ q_i &= y_{i,j-1} + h f_i(t_j, p_0, p_1, \dots, p_{n-1}), i = 0, 1, \dots, n-1 \\ y_{ij} &= \frac{1}{2}(p_i + q_i), i = 0, 1, \dots, n-1 \end{aligned}$$

三、函数语句

void gelrl(t, y, n, h, k, z)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i=0, 1, \dots, n-1$) 值的函数 gelrlf，由用户自编，其形式如下：

```
void gelrlf(t, y, n, d)
int n;
double t, y[], d[];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
  :
d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
```

四、形参说明

t ——双精度实型变量。对微分方程进行积分的起始点 t_0 。

y ——双精度实型一维数组，长度为 n 。存放 n 个未知函数 y_i 在起始点 t_0 处的函数值

(即初值) $y_i(t_0)$ ($i=0, 1, \dots, n-1$)。返回时，这些初值存放在二维数组 z 的第零列中，即 $z(i, 0) = y_i(t_0)$ ($i=0, 1, \dots, n-1$)。

n ——整型变量。微分方程组中方程的个数，也是未知函数的个数。

h ——双精度实型变量。积分步长。

k ——整型变量。积分步数(包括起始点这一步)。

z ——双精度实型二维数组，体积为 $n \times k$ 。返回 k 个积分点(包括起始点)上的未知函数值，即

$$z(i, j) = y_{ij}, i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0)$ ($i=0, 1, \dots, n-1$) 为初值 $y_i(t_0)$ 。

五、函数程序(文件名：gelrl1.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = -1.0 \\ y'_1 = -y_0, & y_1(0) = 0.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用改进欧拉公式计算当步长 $h=0.01$ 时，11 个积分点(包括起始点) $t_i=i * h, i=0, 1, \dots, 10$ 上的未知函数的近似值 y_{0i}, y_{1i}, y_{2i} ($i=0, 1, \dots, 10$)。其中起始点 $t=0.0, n=3, k=11, h=0.01$ 。

主函数程序及计算微分方程组中各方程右端函数值的函数程序(文件名：gelrl10.c)如下：

```
# include "stdio.h"
# include "gelrl1.c"
main()
| int i, j;
  double y[3], z[3][11], t, h, x;
  y[0] = -1.0; y[1] = 0.0; y[2] = 1.0;
  t = 0.0; h = 0.01;
  gelrl1(t, y, 3, h, 11, z);
  printf("\n");
  for (i = 0; i < 10; i++)
  | x = i * h;
    printf("t = %5.2f\n", x);
    for (j = 0; j < 2; j++)
      printf("y(%d) = %e\n", j, z[j][i]);
    printf("\n");
  |
  printf("\n");
```

```
void gelrlf(t, y, n, d)
int n;
double t, y[], d[];
```

```

{ t=t; n=n;
d[0]=y[1];d[1]=-y[0];d[2]=-y[2];
return;
}

```

运行结果为：

```

t=0.00
y(0) = -1.00000e+00   y(1) = 0.00000e+00   y(2) = 1.00000e+00
t=0.01
y(0) = -9.9950e-01   y(1) = 1.00000e-02   y(2) = 9.90050e-01
t=0.02
y(0) = -9.99800e-01   y(1) = 1.99990e-02   y(2) = 9.80199e-01
t=0.03
y(0) = -9.99550e-01   y(1) = 2.99960e-02   y(2) = 9.70446e-01
t=0.04
y(0) = -9.99200e-01   y(1) = 3.99900e-02   y(2) = 9.60790e-01
t=0.05
y(0) = -9.98750e-01   y(1) = 4.99800e-02   y(2) = 9.51230e-01
t=0.06
y(0) = -9.98200e-01   y(1) = 5.99650e-02   y(2) = 9.41765e-01
t=0.07
y(0) = -9.97551e-01   y(1) = 6.99440e-02   y(2) = 9.32395e-01
t=0.08
y(0) = -9.96802e-01   y(1) = 7.99160e-02   y(2) = 9.23118e-01
t=0.09
y(0) = -9.95953e-01   y(1) = 8.98800e-02   y(2) = 9.13933e-01
t=0.10
y(0) = -9.95004e-01   y(1) = 9.98351e-02   y(2) = 9.04839e-01

```

本问题的解析解为

$$\begin{cases} y_0 = -\cos t \\ y_1 = \sin t \\ y_2 = e^{-t} \end{cases}$$

7.2 积分一步的变步长欧拉方法

一、功能

用变步长欧拉(Euler)方法对一阶微分方程组积分一步。

二、方法说明

基本方法同 7.1 节的方法说明。

根据改进的欧拉公式以 h 为步长, 由 $y_{i,j-1}$ ($i = 0, 1, \dots, n-1$) 计算 $y_{ij}^{(h)}$; 再以 $h/2$ 为步长, 由 $y_{i,j-1}$ 跨两步计算 $y_{ij}^{(h/2)}$ 。此时, 若

$$\max_{0 \leq i \leq n-1} |y_{ij}^{(h/2)} - y_{ij}^{(h)}| < \epsilon$$

则停止计算, 取 $y_{ij}^{(h/2)}$ 作为 y_{ij} ($i = 0, 1, \dots, n-1$); 否则, 将步长折半再进行计算。

上述过程一直作到满足条件

$$\max_{0 \leq i \leq n-1} |y_{ij}^{(h/2^m)} - y_{ij}^{(h/2^{m-1})}| < \epsilon$$

为止。最后可取 $y_{ij} = y_{ij}^{(h/2^m)}$, $i = 0, 1, \dots, n-1$, 其中 ϵ 为预先给定的精度要求。

三、函数语句

```
void gelr2(t, h, y, n, eps)
```

本函数要调用计算微分方程组中各方程的右端函数值 $f_i(t, y_0, y_1, \dots, y_{n-1})$ 的函数 gelr2f, 由用户自编, 其形式如下:

```
void gelr2f(t, y, n, d)
int n;
double t, y[], d[];
{ d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
  ;
  d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
}
```

四、形参说明

t ——双精度实型变量。积分一步的起始点。

h ——双精度实型变量。积分步长。

y ——双精度实型一维数组, 长度为 n 。存放起始点 t 处的 n 个未知函数值 $y_i(t)$ ($i = 0, 1, \dots, n-1$); 返回时存放 $t+h$ 点处的 n 个未知函数值 $y_i(t+h)$ ($i = 0, 1, \dots, n-1$)。

n ——整型变量。微分方程组中方程的个数, 也是未知函数的个数。

eps ——双精度实型变量。积分的精度要求。

五、函数程序(文件名: gelr2.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = -1.0 \\ y'_1 = -y_0, & y_1(0) = 0.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用积分一步的变步长欧拉方法计算当步长 $h = 0.01$ 时各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

上的未知函数的近似值 y_{0i}, y_{1i}, y_{2i} ($i = 0, 1, \dots, 10$)。取 $\epsilon = 0.00001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gelr20.c)如下:

```
# include "stdio.h"
# include "gelr2.c"
main()
{ int i, j;
```

```

double t, h, eps, y[3];
y[0] = -1.0; y[1] = 0.0; y[2] = 1.0;
t = 0.0; h = 0.01; eps = 0.00001;
printf("n");
printf("t= %5.2f \n", t);
for (i=0;i<=2;i++)
    printf("y(%d) = %e, i, y[i]);
printf("\n");
for (j=1;j<=10;j++)
    gelr2(t, h, y, 3, eps);
    t = t + h;
    printf("t= %5.2f \n", t);
    for (i=0;i<=2;i++)
        printf("y(%d) = %e, i, y[i]);
    printf("\n");
}

void gelr2f(t, y, n, d)
int n;
double t, y[], d[];
{
    t = t; n = n;
    d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
    return;
}

```

运行结果为：

```

t=0.00
y(0)= -1.00000e+00   y(1)= 0.00000e+00   y(2)= 1.00000e+00
t=0.01
y(0)= -9.99950e-01   y(1)= 9.99988e-03   y(2)= 9.90050e-01
t=0.02
y(0)= -9.99800e-01   y(1)= 1.99988e-02   y(2)= 9.80199e-01
t=0.03
y(0)= -9.99550e-01   y(1)= 2.99956e-02   y(2)= 9.70446e-01
t=0.04
y(0)= -9.99200e-01   y(1)= 3.99895e-02   y(2)= 9.60790e-01
t=0.05
y(0)= -9.98750e-01   y(1)= 4.99794e-02   y(2)= 9.51230e-01
t=0.06
y(0)= -9.98201e-01   y(1)= 5.99643e-02   y(2)= 9.41765e-01
t=0.07
y(0)= -9.97551e-01   y(1)= 6.99431e-02   y(2)= 9.32394e-01
t=0.08
y(0)= -9.96802e-01   y(1)= 7.99150e-02   y(2)= 9.23117e-01
t=0.09
y(0)= -9.95953e-01   y(1)= 8.98789e-02   y(2)= 9.13932e-01
t=0.10
y(0)= -9.95004e-01   y(1)= 9.98338e-02   y(2)= 9.04838e-01

```

本问题的解析解为

$$\begin{cases} y_0 = -\cos t \\ y_1 = \sin t \\ y_2 = e^{-t} \end{cases}$$

7.3 全区间积分的定步长维梯方法

一、功能

用定步长维梯(witty)方法对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

用维梯方法由 t_j 积分一步到 $t_{j+1} = t_j + h$ 的计算公式如下：

$$d_{i,0} = f_i(t_0, y_{00}, y_{10}, \dots, y_{n-1,0}), \quad i = 0, 1, \dots, n-1$$

$$y_{i,j+\frac{1}{2}} = y_{ij} + \frac{1}{2}hd_{ij}, \quad i = 0, 1, \dots, n-1$$

$$q_i = f_i\left(t_j + \frac{h}{2}, y_{0,j+\frac{1}{2}}, y_{1,j+\frac{1}{2}}, \dots, y_{n-1,j+\frac{1}{2}}\right), \quad i = 0, 1, \dots, n-1$$

$$y_{i,j+1} = y_{ij} + hq_i, \quad i = 0, 1, \dots, n-1$$

$$d_{i,j+1} = 2q_i - d_{ij}, \quad i = 0, 1, \dots, n-1$$

三、函数语句

void gwity(t, y, n, h, k, z)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 gwityf, 由用户自编, 其形式如下。

```

void gwityf(t, y, n, d)
int n;
double t, y[], d[];
{
    d[0] = f_0(t, y_0, y_1, \dots, y_{n-1}) 的表达式;
    \vdots
    d[n-1] = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}) 的表达式;
    return;
}

```

四、形参说明

t——双精度实型变量。积分起始点 t_0 。

y ——双精度实型一维数组, 长度为 n 。起始点 t 处的 n 个未知函数值 $y_i(t_0)$ ($i = 0, 1, \dots, n - 1$) (即初值)。返回时其值在二维数组 z 的第零列中, 即 $z(i, 0) = y_i(t_0)$, $i = 0, 1, \dots, n - 1$

n ——整型变量。微分方程组中方程的个数, 也是未知函数的个数。

h ——双精度实型变量。积分步长。

k ——整型变量。积分步数(包括起始点这一步)。

z ——双精度实型二维数组, 体积为 $n \times k$ 。返回 k 个积分点(包括起始点)上的未知函数值, 即

$$z(i, j) = y_{ij}, i = 0, 1, \dots, n - 1; j = 0, 1, \dots, k - 1$$

其中 $z(i, 0) = y_i(t_0)$ ($i = 0, 1, \dots, n - 1$)。

五、函数程序(文件名: gwity.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = -1.0 \\ y'_1 = -y_0, & y_1(0) = 0.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用维梯方法计算当步长 $h = 0.1$ 时, 11 个积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数的近似值 y_{0i}, y_{1i}, y_{2i} ($i = 0, 1, \dots, 10$)。

其中起始点 $t = 0.0, n = 3, k = 11, h = 0.1$ 。

主函数程序及计算微分方程右端函数值的函数程序(文件名: gwity0.c)如下:

```
#include "stdio.h"
#include "gwity.c"

main()
{
    int i, j;
    double t, h, x, y[3], z[3][11];
    y[0] = -1.0; y[1] = 0.0; y[2] = 1.0;
    t = 0.0; h = 0.1;
    gwity(t, y, 3, h, 11, z);
    for (i = 0; i <= 10; i++)
    {
        x = i * h;
        printf("t = %5.2f\n", x);
        for (j = 0; j <= 2; j++)
            printf("y(%d) = %e, j, z[j][i]");
        printf("\n");
    }
}

void gwityf(t, y, n, d)
int n;
double t, y[], d[];
```

```
{
    t = t; n = n;
    d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
    return;
}
```

运行结果为:

t = 0.00	y(0) = -1.00000e+00	y(1) = 0.00000e+00	y(2) = 1.00000e+00
t = 0.10	y(0) = -9.95000e-01	y(1) = 1.00000e-01	y(2) = 9.05000e-01
t = 0.20	y(0) = -9.80000e-01	y(1) = 1.99000e-01	y(2) = 8.19000e-01
t = 0.30	y(0) = -9.55200e-01	y(1) = 2.96000e-01	y(2) = 7.41200e-01
t = 0.40	y(0) = -9.20800e-01	y(1) = 3.90040e-01	y(2) = 6.70760e-01
t = 0.50	y(0) = -8.77192e-01	y(1) = 4.80160e-01	y(2) = 6.07048e-01
t = 0.60	y(0) = -8.24768e-01	y(1) = 5.65478e-01	y(2) = 5.49350e-01
t = 0.70	y(0) = -7.64096e-01	y(1) = 6.45114e-01	y(2) = 4.97178e-01
t = 0.80	y(0) = -6.95745e-01	y(1) = 7.18298e-01	y(2) = 4.49915e-01
t = 0.90	y(0) = -6.20437e-01	y(1) = 7.84263e-01	y(2) = 4.07195e-01
t = 1.00	y(0) = -5.38893e-01	y(1) = 8.42385e-01	y(2) = 3.68476e-01

本问题的解析解为:

$$\begin{cases} y_0 = -\cos t \\ y_1 = \sin t \\ y_2 = e^{-t} \end{cases}$$

7.4 全区间积分的定步长龙格-库塔法

一、功能

用定步长四阶龙格-库塔(Runge-Kutta)法对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

由 t_j 积分一步到 $t_{j+1} = t_j + h$ 的四阶龙格-库塔方法的计算公式为:

$$\begin{aligned}k_{0i} &= f_i(t_j, y_{0j}, y_{1j}, \dots, y_{n-1,j}), \quad i=0, 1, \dots, n-1 \\k_{1i} &= f_i\left(t_j + \frac{h}{2}, y_{0j} + \frac{h}{2}k_{00}, \dots, y_{n-1,j} + \frac{h}{2}k_{0,n-1}\right), \quad i=0, 1, \dots, n-1 \\k_{2i} &= f_i\left(t_j + \frac{h}{2}, y_{0j} + \frac{h}{2}k_{10}, \dots, y_{n-1,j} + \frac{h}{2}k_{1,n-1}\right), \quad i=0, 1, \dots, n-1 \\k_{3i} &= f_i(t_j + h, y_{0j} + hk_{20}, \dots, y_{n-1,j} + hk_{2,n-1}), \quad i=0, 1, \dots, n-1 \\y_{i,j+1} &= y_{ij} + \frac{h}{6}(k_{0i} + 2k_{1i} + 2k_{2i} + k_{3i}), \quad i=0, 1, \dots, n-1\end{aligned}$$

三、函数语句

void grkt1f(t, y, n, h, k, z)

本函数要调用计算微分方程组中各方程的右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i=0, 1, \dots, n-1$) 值的函数 grkt1f, 由用户自编, 其形式如下:

```
void grkt1f(t, y, n, d)
int n;
double t, y[ ], d[ ];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
| :
d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
```

四、形参说明

t ——双精度实型变量。积分的起始点 t_0 。

y ——双精度实型一维数组, 长度为 n 。存放 n 个未知函数在起始点 t 处的函数值(即初值) $y_i(t)$ ($i=0, 1, \dots, n-1$)。返回时, 其初值在二维数组 z 的第零列中, 即

$$z(i, 0) = y_i(t), \quad i=0, 1, \dots, n-1$$

n ——整型变量。微分方程组中方程的个数, 也是未知函数的个数。

h ——双精度实型变量。积分的步长。

k ——整型变量。积分的步数(包括起始点这一步)。

z ——双精度实型二维数组, 体积为 $n \times k$ 。返回 k 个积分点上的 n 个未知函数值, 即

$$z(i, j) = y_i(t_j), \quad i=0, 1, \dots, n-1; \quad j=0, 1, \dots, k-1$$

其中 $z(i, 0)$ ($i=0, 1, \dots, n-1$) 为给定的初值。

五、函数程序(文件名: grkt1.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = -1.0 \\ y'_1 = -y_0, & y_1(0) = 0.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用四阶龙格-库塔法计算当 $h=0.01$ 时, 11 个积分点

$$t_i = i * h, \quad i=0, 1, \dots, 10$$

上的未知函数值 y_{0i}, y_{1i}, y_{2i} ($i=0, 1, \dots, 10$)。

其中起始点 $t=0.0, h=0.01, n=3, k=11$ 。

主函数程序及计算微分方程组中各方程右端函数值的函数程序(文件名: grkt10.c)如下:

```
#include "stdio.h"
#include "grkt1.c"
main()
| int i, j;
double t, h, y[3], z[3][11];
y[0] = -1.0; y[1] = 0.0; y[2] = 1.0;
t = 0.0; h = 0.01;
grkt1(t, y, 3, h, 11, z);
printf("\n");
for (i=0; i<=10; i++)
| t = i * h;
printf("t = %5.2f \n", t);
for (j=0; j<=2; j++)
| printf("y(%d) = %e", j, z[j][i]);
printf("\n");
|
```

```
void grkt1f(t, y, n, d)
int n;
double t, y[ ], d[ ];
| t=t; n=n;
d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
return;
```

运行结果为:

$t=0.00$	$y(0) = -1.00000e+00$	$y(1) = 0.00000e+00$	$y(2) = 1.00000e+00$
$t=0.01$	$y(0) = -9.99950e-01$	$y(1) = 9.99983e-03$	$y(2) = 9.90050e-01$
$t=0.02$	$y(0) = -9.99800e-01$	$y(1) = 1.99987e-02$	$y(2) = 9.80199e-01$
$t=0.03$	$y(0) = -9.99550e-01$	$y(1) = 2.99955e-02$	$y(2) = 9.70446e-01$
$t=0.04$	$y(0) = -9.99200e-01$	$y(1) = 3.99893e-02$	$y(2) = 9.60789e-01$
$t=0.05$	$y(0) = -9.98750e-01$	$y(1) = 4.99792e-02$	$y(2) = 9.51229e-01$
$t=0.06$	$y(0) = -9.98201e-01$	$y(1) = 5.99640e-02$	$y(2) = 9.41765e-01$
$t=0.07$	$y(0) = -9.97551e-01$	$y(1) = 6.99428e-02$	$y(2) = 9.32394e-01$

```

t=0.08          y(1)=7.99147e-02    y(2)=9.23116e-01
y(0)=-9.96802e-01
t=0.09          y(1)=8.98785e-02    y(2)=9.13931e-01
y(0)=-9.95953e-01
t=0.10          y(1)=9.98334e-02    y(2)=9.04837e-01
y(0)=-9.95004e-01

```

本问题的解析解为：

$$\begin{cases} y_0 = -\cos t \\ y_1 = \sin t \\ y_2 = e^{-t} \end{cases}$$

7.5 积分一步的变步长龙格-库塔法

一、功能

用变步长四阶龙格-库塔(Runge-Kutta)法对一阶微分方程组积分一步。

二、方法说明

基本方法同 7.4 节的方法说明。

根据四阶龙格-库塔公式, 以 h 为步长由 $y_{i,j-1}$ 计算 $y_{ij}^{(k)}$ ($i = 0, 1, \dots, n-1$); 再以 $h/2$ 为步长由 $y_{i,j-1}$ 跨两步计算 $y_{ij}^{(h/2)}$ ($i = 0, 1, \dots, n-1$)。若满足条件

$$\max_{0 \leq i \leq n-1} |y_{ij}^{(h/2)} - y_{ij}^{(h)}| < \epsilon$$

则停止计算, 取 $y_i(t_j) = y_{ij}^{(h/2)}$ ($i = 0, 1, \dots, n-1$); 否则将步长折半再进行计算。

以上过程一直作到满足条件

$$\max_{0 \leq i \leq n-1} |y_{ij}^{(h/2^m)} - y_{ij}^{(h/2^{m-1})}| < \epsilon$$

为止, 最后可得

$$y_i(t_j) = y_{ij}^{(h/2^m)}, \quad i = 0, 1, \dots, n-1$$

其中 ϵ 为预先给定的精度要求。

三、函数语句

void grkt2(t, h, y, n, eps)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ 值的函数 grkt2f, 由用户自编, 其形式如下。

```

void grkt2f(t, y, n, d)
int n;
double t, y[ ], d[ ];
{ d[0]=f0(t, y0, y1, ..., yn-1) 的表达式;
  ;
  d[n-1]=fn-1(t, y0, y1, ..., yn-1) 的表达式;
  return;
}

```

四、形参说明

t——双精度实型变量。积分起始点。

h——双精度实型变量。积分步长。

y——双精度实型一维数组, 长度为 n。存放起始点 t 处的 n 个未知函数值 $y_i(t)$ ($i = 0, 1, \dots, n-1$); 返回时存放积分一步后 $t+h$ 点的 n 个未知函数值 $y_i(t+h)$ ($i = 0, 1, \dots, n-1$)。

n——整型变量。一阶微分方程组的方程个数, 也是未知函数的个数。

eps——双精度实型变量。积分精度要求。

五、函数程序(文件名: grkt2.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = 0.0 \\ y'_1 = -y_0, & y_1(0) = 1.0 \end{cases}$$

用变步长四阶龙格-库塔法计算当 $h=0.1$ 时各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

上的函数值 y_{0i}, y_{1i} ($i = 0, 1, \dots, 10$)。取 $eps = 0.00001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: grkt20.c)如下:

```

#include "stdio.h"
#include "grkt2.c"
main()
{ int i;
  double t, h, eps, y[2];
  y[0]=0.0; y[1]=1.0;
  t=0.0; h=0.1; eps=0.00001;
  printf("\n");
  printf("t = %7.3f y(0) = %e y(1) = %e\n", t, y[0], y[1]);
  for (i=1; i<=10; i++)
  { grkt2(t, h, y, 2, eps);
    t=t+h;
    printf("t = %7.3f y(0) = %e y(1) = %e\n", t, y[0], y[1]);
  }
}

void grkt2f(t, y, n, d)
int n;
double t, y[ ], d[ ];
{ t=t; n=n;
  d[0]=y[1]; d[1]=-y[0];
  return;
}

```

运行结果为:

t = 0.000	y(0) = 0.00000e+00	y(1) = 1.00000e+00
t = 0.100	y(0) = 9.98334e-02	y(1) = 9.95004e-01
t = 0.200	y(0) = 1.98669e-01	y(1) = 9.80067e-01
t = 0.300	y(0) = 2.95520e-01	y(1) = 9.55336e-01
t = 0.400	y(0) = 3.89418e-01	y(1) = 9.21061e-01
t = 0.500	y(0) = 4.79426e-01	y(1) = 8.77583e-01
t = 0.600	y(0) = 5.64642e-01	y(1) = 8.25336e-01
t = 0.700	y(0) = 6.44218e-01	y(1) = 7.64842e-01
t = 0.800	y(0) = 7.17356e-01	y(1) = 6.96707e-01
t = 0.900	y(0) = 7.83327e-01	y(1) = 6.21610e-01
t = 1.000	y(0) = 8.41471e-01	y(1) = 5.40302e-01

本问题的解析解为：

$y_0 = \sin t$

$y_1 = \cos t$

7.6 积分一步的变步长基尔方法

一、功能

用变步长基尔(Gill)公式对一阶微分方程组积分一步。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

由 t 点积分到 $t+h$ 点的基尔公式如下：

$$\begin{cases} k_{0i} = h f_i(t, y_0^{(0)}, y_1^{(0)}, \dots, y_{n-1}^{(0)}) \\ k_{1i} = h f_i\left(t + \frac{h}{2}, y_0^{(1)}, y_1^{(1)}, \dots, y_{n-1}^{(1)}\right) \\ k_{2i} = h f_i\left(h + \frac{h}{2}, y_0^{(2)}, y_1^{(2)}, \dots, y_{n-1}^{(2)}\right) \\ k_{3i} = h f_i(t+h, y_0^{(3)}, y_1^{(3)}, \dots, y_{n-1}^{(3)}) \end{cases} \quad i = 0, 1, \dots, n-1$$

其中

$$\begin{cases} y_i^{(1)} = y_i^{(0)} + \frac{1}{2}(k_{0i} - 2q_i^{(0)}) \\ y_i^{(2)} = y_i^{(1)} + \left(1 - \sqrt{\frac{1}{2}}\right)(k_{1i} - q_i^{(1)}) \\ y_i^{(3)} = y_i^{(2)} + \left(1 + \sqrt{\frac{1}{2}}\right)(k_{2i} - q_i^{(2)}) \\ y_i^{(4)} = y_i^{(3)} + \frac{1}{6}(k_{3i} - 2q_i^{(3)}) \end{cases} \quad i = 0, 1, \dots, n-1$$

式中

$$\begin{cases} q_i^{(1)} = q_i^{(0)} + 3\left[\frac{1}{2}(k_{0i} - 2q_i^{(0)})\right] - \frac{1}{2}k_{0i} \\ q_i^{(2)} = q_i^{(1)} + 3\left[\left(1 - \sqrt{\frac{1}{2}}\right)(k_{1i} - q_i^{(1)})\right] - \left(1 - \sqrt{\frac{1}{2}}\right)k_{1i} \\ q_i^{(3)} = q_i^{(2)} + 3\left[\left(1 + \sqrt{\frac{1}{2}}\right)(k_{2i} - q_i^{(2)})\right] - \left(1 + \sqrt{\frac{1}{2}}\right)k_{2i} \\ q_i^{(4)} = q_i^{(3)} + 3\left[\frac{1}{6}(k_{3i} - 2q_i^{(3)})\right] - \frac{1}{2}k_{3i} \end{cases}$$

$i = 0, 1, \dots, n-1$

其中： $y_i^{(0)}$ 为 t 点的未知函数值 $y_i(t)$ ；

$y_i^{(4)}$ 为积分一步后， $t+h$ 点的未知函数值 $y_i(t+h)$ ；

$q_i^{(0)}$ 在起始时赋值为 0，以后每积分一步，将 $q_i^{(4)}$ 作为下一步的 $q_i^{(0)}$ 。

这种方法具有抵消每一步中所积累的舍入误差的作用，可以提高精度。

本函数在积分一步的过程中，还具有自动选择步长的作用，即根据给定的精度要求，自动将步长等分，跨多步积分到 $t+h$ 。有关自动选择步长的方法参看 7.5 节的方法说明。

三、函数语句

void ggill1(t, h, y, n, eps, q)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 ggillf，由用户自编，其形式如下。

```
void ggillf(t, y, n, d)
int n;
double t, y[], d[];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
| :
| d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
```

四、形参说明

t——双精度实型变量。积分的起始点。

h——双精度实型变量。积分步长。

y——双精度实型一维数组，长度为 n 。存放起始点 t 处的 n 个未知函数值 $y_i(t)$ ($i = 0, 1, \dots, n-1$)；返回时存放 $t+h$ 点处的 n 个未知函数值 $y_i(t+h)$ ($i = 0, 1, \dots, n-1$)。

n——整型变量。一阶微分方程组中方程个数，也是未知函数的个数。

eps——双精度实型变量。积分一步的精度要求。

q——双精度实型一维数组，长度为 n 。在主函数第一次调用本函数时，应赋值以 0，即 $q(i) = 0$ ($i = 0, 1, \dots, n-1$)，以后每调用一次本函数（即每积分一步），将由本函数的返回值以便循环使用。

五、函数程序(文件名: ggill1.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = 0.0 \\ y'_1 = -y_0, & y_1(0) = 1.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用变步长基尔方法计算当 $h = 0.1$ 时, 各积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数值 y_{0i}, y_{1i}, y_{2i} ($i = 0, 1, \dots, 10$)。

取 $\text{eps} = 0.000001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: ggill10.c)如下:

```
# include "stdio.h"
# include "ggill1.c"
main()
| int i, j;
double t, h, eps;
static double q[3] = {0.0, 0.0, 0.0};
static double y[3] = {0.0, 1.0, 1.0};
t=0.0; h=0.1; eps=0.000001;
printf("\n");
printf("t= %7.3f\n", t);
for (i=0; i<=2; i++)
    printf("y(%d) = %e\n", i, y[i]);
printf("\n");
for (j=1; j<=10; j++)
    | ggill1(t, h, y, 3, eps, q);
    t=j*h;
    printf("t= %7.3f\n", t);
    for (i=0; i<=2; i++)
        printf("y(%d) = %e\n", i, y[i]);
    printf("\n");
}

void ggill1f(t, y, n, d)
int n;
double t, y[], d[];
| t=t; n=n;
d[0]=y[1]; d[1]=-y[0]; d[2]=-y[2];
return;
```

运行结果为:

t = 0.000

$y(0) = 0.00000e+00$	$y(1) = 1.00000e+00$	$y(2) = 1.00000e+00$
$t = 0.100$		
$y(0) = 9.98334e-02$	$y(1) = 9.95004e-01$	$y(2) = 9.04837e-01$
$t = 0.200$		
$y(0) = 1.98669e-01$	$y(1) = 9.80067e-01$	$y(2) = 8.18731e-01$
$t = 0.300$		
$y(0) = 2.95520e-01$	$y(1) = 9.55336e-01$	$y(2) = 7.40818e-01$
$t = 0.400$		
$y(0) = 3.89418e-01$	$y(1) = 9.21061e-01$	$y(2) = 6.70320e-01$
$t = 0.500$		
$y(0) = 4.79426e-01$	$y(1) = 8.77583e-01$	$y(2) = 6.06531e-01$
$t = 0.600$		
$y(0) = 5.64642e-01$	$y(1) = 8.25336e-01$	$y(2) = 5.48812e-01$
$t = 0.700$		
$y(0) = 6.44218e-01$	$y(1) = 7.64842e-01$	$y(2) = 4.96585e-01$
$t = 0.800$		
$y(0) = 7.17356e-01$	$y(1) = 6.96707e-01$	$y(2) = 4.49329e-01$
$t = 0.900$		
$y(0) = 7.83327e-01$	$y(1) = 6.21610e-01$	$y(2) = 4.06570e-01$
$t = 1.000$		
$y(0) = 8.41471e-01$	$y(1) = 5.40302e-01$	$y(2) = 3.67879e-01$

本问题的解析解为:

$$\begin{cases} y_0 = \sin t \\ y_1 = \cos t \\ y_2 = e^{-t} \end{cases}$$

7.7 全区间积分的变步长基尔方法

一、功能

用变步长基尔(Gill)公式对一阶微分方程组进行全区间积分。

二、方法说明

同 7.6 节。

三、函数语句

void ggill2(t, h, y, n, eps, k, z)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ 值的函数 ggill2f, 由用户自编, 其形式如下:

```
void ggill2f(t, y, n, d)
int n;
double t, y[], d[];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
|   ;
| d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
```

```

    return;
}

```

四、形参说明

t ——双精度实型变量。积分的起始点。

h ——双精度实型变量。积分步长。

y ——双精度实型一维数组，长度为 n 。存放起始点 t 处的 n 个未知函数值 $y_i(t)$ ($i = 0, 1, \dots, n-1$)；返回时其值在二维数组 z 的第零列，即

$$z(i, 0) = y_i(t), i = 0, 1, \dots, n-1$$

n ——整型变量。一阶微分方程组中方程个数，也是未知函数的个数。

eps ——双精度实型变量。积分精度要求。

k ——整型变量。积分的步数(包括起始点这一步)。

z ——双精度实型二维数组，体积为 $n \times k$ 。返回 k 个积分点上的 n 个未知函数值，即

$$z(i, j) = y_i(t_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0)$ ($i = 0, 1, \dots, n-1$) 为初值。

五、函数程序(文件名: ggil2.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = 0.0 \\ y'_1 = -y_0, & y_1(0) = 1.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用变步长基尔方法计算当 $h = 0.1$ 时，各积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数值 y_{0i}, y_{1i}, y_{2i} ($i = 0, 1, \dots, 10$)。

取 $\text{eps} = 0.000001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: ggil20.c)如下：

```

#include "stdio.h"
#include "ggil2.c"
main()
{
    int i, j;
    double a, h, eps, t, z[3][11];
    static double y[3] = {0.0, 1.0, 1.0};
    a = 0.0; h = 0.1; eps = 0.000001;
    ggil2(a, h, y, 3, eps, 11, z);
    printf("\n");
    for (i = 0; i <= 10; i++)
    {
        t = i * h;
        printf("t = %7.3f\n", t);
        for (j = 0; j <= 2; j++)
            printf("y(%d) = %e\n", j, z[j][i]);
    }
}

```

```

    printf("\n");
}

void ggil2f(t, y, n, d)
int n;
double t, y[], d[];
{
    t = t; n = n;
    d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
    return;
}

```

void ggil2f(t, y, n, d)

```

int n;
double t, y[], d[];
{
    t = t; n = n;
    d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
    return;
}

```

运行结果为：

$t = 0.000$	$y(0) = 0.00000e+00$	$y(1) = 1.00000e+00$	$y(2) = 1.00000e+00$
$t = 0.100$	$y(0) = 9.98334e-02$	$y(1) = 9.95004e-01$	$y(2) = 9.04837e-01$
$t = 0.200$	$y(0) = 1.98669e-01$	$y(1) = 9.80067e-01$	$y(2) = 8.18731e-01$
$t = 0.300$	$y(0) = 2.95520e-01$	$y(1) = 9.55336e-01$	$y(2) = 7.40818e-01$
$t = 0.400$	$y(0) = 3.89418e-01$	$y(1) = 9.21061e-01$	$y(2) = 6.70320e-01$
$t = 0.500$	$y(0) = 4.79426e-01$	$y(1) = 8.77583e-01$	$y(2) = 6.06531e-01$
$t = 0.600$	$y(0) = 5.64642e-01$	$y(1) = 8.25336e-01$	$y(2) = 5.48812e-01$
$t = 0.700$	$y(0) = 6.44218e-01$	$y(1) = 7.64842e-01$	$y(2) = 4.96585e-01$
$t = 0.800$	$y(0) = 7.17356e-01$	$y(1) = 6.96707e-01$	$y(2) = 4.49329e-01$
$t = 0.900$	$y(0) = 7.83327e-01$	$y(1) = 6.21610e-01$	$y(2) = 4.06570e-01$
$t = 1.000$	$y(0) = 8.41471e-01$	$y(1) = 5.40302e-01$	$y(2) = 3.67879e-01$

本问题的解析解为：

$$\begin{cases} y_0 = \sin t \\ y_1 = \cos t \\ y_2 = e^{-t} \end{cases}$$

7.8 全区间积分的变步长默森方法

一、功能

用变步长默森(Merson)方法对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), y_{n-1}(t_0) = y_{n-10} \end{cases}$$

由 t_j 积分一步到 $t_{j+1} = t_j + h$ 的默森方法的计算公式为:

$$\begin{aligned} y_i^{(1)} &= y_i^{(0)} + \frac{h}{3} f_i(t_j, y_0^{(0)}, y_1^{(0)}, \dots, y_{n-1}^{(0)}) \\ y_i^{(2)} &= y_i^{(1)} + \frac{h}{6} \left[f_i\left(t_j + \frac{h}{3}, y_0^{(1)}, y_1^{(1)}, \dots, y_{n-1}^{(1)}\right) - f_i(t_j, y_0^{(0)}, \dots, y_{n-1}^{(0)}) \right] \\ y_i^{(3)} &= y_i^{(2)} + \frac{3}{8} h \left[f_i\left(t_j + \frac{h}{3}, y_0^{(2)}, y_1^{(2)}, \dots, y_{n-1}^{(2)}\right) \right. \\ &\quad \left. - \frac{4}{9} \left(f_i\left(t_j + \frac{h}{3}, y_0^{(1)}, y_1^{(1)}, \dots, y_{n-1}^{(1)}\right) + \frac{1}{4} f_i(t_j, y_0^{(0)}, \dots, y_{n-1}^{(0)}) \right) \right] \\ y_i^{(4)} &= y_i^{(3)} + 2h \left[f_i\left(t_j + \frac{h}{2}, y_0^{(3)}, y_1^{(3)}, \dots, y_{n-1}^{(3)}\right) \right. \\ &\quad \left. - \frac{15}{16} \left(f_i\left(t_j + \frac{h}{3}, y_0^{(2)}, y_1^{(2)}, \dots, y_{n-1}^{(2)}\right) - \frac{1}{5} f_i(t_j, y_0^{(0)}, y_1^{(0)}, \dots, y_{n-1}^{(0)}) \right) \right] \\ y_i^{(5)} &= y_i^{(4)} + \frac{h}{6} \left[f_i(t_j + h, y_0^{(4)}, y_1^{(4)}, \dots, y_{n-1}^{(4)}) - 8 \left(f_i\left(t_j + \frac{h}{2}, y_0^{(3)}, y_1^{(3)}, \dots, y_{n-1}^{(3)}\right) \right. \right. \\ &\quad \left. \left. - \frac{9}{8} \left(f_i\left(t_j + \frac{h}{3}, y_0^{(2)}, y_1^{(2)}, \dots, y_{n-1}^{(2)}\right) - \frac{2}{9} f_i(t_j, y_0^{(0)}, y_1^{(0)}, \dots, y_{n-1}^{(0)}) \right) \right) \right] \\ i &= 0, 1, \dots, n-1 \end{aligned}$$

其中

$$\begin{aligned} y_i^{(0)} &= y_i(t_j), i = 0, 1, \dots, n-1 \\ y_i^{(5)} &= y_i(t_j + h), i = 0, 1, \dots, n-1 \end{aligned}$$

关于自动选择步长参看 7.5 节的方法说明。

三、函数语句

void gmrsn(t, h, n, y, eps, k, z)
本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1}) (i = 0, 1, \dots, n-1)$ 值的函数 gmrsnf, 由用户自编, 其形式如下:

```
void gmrsnf(t, y, n, d)
int n;
double t, y[], d[];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
|   :
| d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
```

四、形参说明

t——双精度实型变量。积分的起始点。

h——双精度实型变量。积分一步的步长。

n——整型变量。一阶微分方程组的方程个数, 也是未知函数的个数。

y——双精度实型一维数组, 长度为 n。存放在初始点 t 处的 n 个未知函数的初值
 $y_i(t) (i = 0, 1, \dots, n-1)$ 。返回时其初值在二维数组 z 的第零列中, 即

$$z(i, 0) = y_i(t), i = 0, 1, \dots, n-1$$

eps——双精度实型变量。控制积分一步的精度要求。

k——整型变量。积分的步数(包括起始点这一步)。

z——双精度实型二维数组, 体积为 $n \times k$ 。返回微分方程组中 n 个未知函数在 k 个积分点(包括起始点)处的函数值, 即

$$z(i, j) = y_{ij}, i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0) (i = 0, 1, \dots, n-1)$ 为初值。

五、函数程序(文件名: gmrsn.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = 60[0.06 + t(t-0.6)]y_1, & y_0(0) = 0.0 \\ y'_1 = -60[0.06 + t(t-0.6)]y_0, & y_1(0) = 1.0 \end{cases}$$

用默森方法计算当 $h = 0.1$ 时, 各积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数值 $y_0(t_i)$ 及 $y_1(t_i) (i = 0, 1, \dots, 10)$ 的近似值。取 $eps = 0.00001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gmrsn0.c)如下:

```
# include "stdio.h"
# include "gmrsn.c"
main()
| int i;
double t, h, eps, x, y[2], z[2][11];
t=0.0; h=0.1; eps=0.00001;
y[0]=0.0; y[1]=1.0;
gmrsn(t, h, 2, y, eps, 11, z);
printf("\n");
for (i=0; i<=10; i++)
| x=i*h;
printf("t = %7.3f y(0) = %e y(1) = %e\n",
x, z[0][i], z[1][i]);
| printf("\n");
```

```
void gmrsnf(t, y, n, d)
int n;
double t, y[], d[];
| double q;
```

```

n=n;
q=60.0*(0.06+t*(t-0.6));
d[0]=q*y[1]; d[1]=-q*y[0];
return;
}

```

运行结果为：

$t=0.000$	$y(0)=0.00000e+00$	$y(1)=1.00000e+00$
$t=0.100$	$y(0)=1.98669e-01$	$y(1)=9.80064e-01$
$t=0.200$	$y(0)=1.59318e-01$	$y(1)=9.87226e-01$
$t=0.300$	$y(0)=-1.35170e-07$	$y(1)=1.00000e+00$
$t=0.400$	$y(0)=-1.59319e-01$	$y(1)=9.87226e-01$
$t=0.500$	$y(0)=-1.98670e-01$	$y(1)=9.80064e-01$
$t=0.600$	$y(0)=-4.12307e-07$	$y(1)=1.00000e+00$
$t=0.700$	$y(0)=5.31188e-01$	$y(1)=8.47256e-01$
$t=0.800$	$y(0)=9.99577e-01$	$y(1)=-2.92037e-02$
$t=0.900$	$y(0)=-9.82556e-02$	$y(1)=-9.95165e-01$
$t=1.000$	$y(0)=-6.31263e-01$	$y(1)=7.75575e-01$

本问题的解析解为：

$$\begin{cases} y_0 = \sin[20t(t-0.3)(t-0.6)] \\ y_1 = \cos[20t(t-0.3)(t-0.6)] \end{cases}$$

7.9 积分一步的连分式法

一、功能

用连分式法对一阶微分方程组积分一步。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

由 t_j 积分一步到 $t_{j+1} = t_j + h$ 的有理分式法如下。

用定步长四阶龙格-库塔法由 t_j 跨一步到 t_{j+1} , 其中每一小步的步长为

$$h_k = h/2^k, k = 0, 1, \dots$$

由此可以计算得到

$$y_{i,j+1}^{(k)} \text{ 记作 } S_{ki}, i = 0, 1, \dots, n-1$$

显然, 在 t_{j+1} 处的各未知函数的近似值是步长 h^* 的函数, 设为 $s_i(h^*)$, 即

$$y_{i,j+1} = y_i(t_{j+1}) \approx s_i(h^*)$$

并且, 当 h^* 越小时, 计算得到的 $s_i(h^*)$ 越接近于准确值 $y_i(t_{j+1})$ 。

若将 $s_i(h^*)$ 用连分式表示为,

$$s_i(h^*) = b_{0i} + \frac{h^* - h_0}{b_{1i}} + \frac{h^* - h_1}{b_{2i}} + \dots + \frac{h^* - h_{l-1}}{b_{li}} + \dots$$

$$i = 0, 1, \dots, n-1$$

则有

$$y_i(t_{j+1}) = s_i(0)$$

$$= b_{0i} - \frac{h_0}{b_{1i}} - \frac{h_1}{b_{2i}} - \dots - \frac{h_{l-1}}{b_{li}} \quad i = 0, 1, \dots, n-1$$

其中, b_{li} ($l = 0, 1, \dots$) 可由以下递推公式计算:

$$b_{0i} = s_{0i}$$

$$\begin{cases} d_{0,i} = s_{0i} \\ d_{ki} = \frac{h_k - h_{k-1}}{d_{k-1,i} - b_{k-1,i}}, k = 1, 2, \dots, l \end{cases}$$

$$b_{li} = d_{li}$$

其中 $i = 0, 1, \dots, n-1$

通常, 上述连分式作到七节就能满足精度要求。下面是用连分式法由 t_j 积分一步到 $t_{j+1} = t_j + h$ 的计算步骤。

(1) 用定步长四阶龙格-库塔法由 t_j 跨一步到 t_{j+1} 计算 s_{0i} ($i = 0, 1, \dots, n-1$)。其中

$$h_0 = t_{j+1} - t_j = h$$

$$b_{0i} = s_{0i}, y_i^{(0)} = s_{0i}, \quad i = 0, 1, \dots, n-1$$

(2) 对于 $l = 1, 2, \dots$, 令

$$h_l = h_{l-1}/2$$

以 h_l 为步长, 用定步长四阶龙格-库塔公式由 t_j 跨 $n = 2^l$ 步到 t_{j+1} 计算 s_{li} ($i = 0, 1, \dots, n-1$)。然后用下列递推公式计算 b_{li} ($i = 0, 1, \dots, n-1$):

$$d_i = s_{li}$$

$$d_i = \frac{h_l - h_k}{d_i - b_{ki}}, \quad k = 0, 1, \dots, l-1$$

$$b_{li} = d_i$$

最后计算

$$y_i^{(l)} = b_{0i} - \frac{h_0}{b_{1i}} - \frac{h_1}{b_{2i}} - \dots - \frac{h_{l-1}}{b_{li}}, \quad i = 0, 1, \dots, n-1$$

反复作步骤(2), 直到 $\max_{0 \leq i \leq n-1} |y_i^{(l)} - y_i^{(l-1)}| < \epsilon$ 为止。

三、函数语句

void gpb81(t, h, n, y, eps)

本函数要调用计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 `gpbslf`, 由用户自编, 其形式如下:

```
void gpbslf(t, y, n, d)
int n;
double t, y[ ], d[ ];
| d[0] = f0(t, y0, y1, ..., yn-1) 的表达式;
|   :
d[n-1] = fn-1(t, y0, y1, ..., yn-1) 的表达式;
return;
{
```

四、形参说明

t ——双精度实型变量。积分的起始点。

h ——双精度实型变量。积分一步的步长。

n ——整型变量。一阶微分方程组中方程个数, 也是未知函数的个数。

y ——双精度实型一维数组, 长度为 n 。存放 n 个未知函数在起始点 t 处的函数值 $y_i(t)$ ($i = 0, 1, \dots, n-1$); 返回时存放 n 个未知函数在 $t+h$ 点的函数值 $y_i(t+h)$ ($i = 0, 1, \dots, n-1$)。

eps ——双精度实型变量。积分一步的精度要求。

五、函数程序(文件名: `gpbs 1.c`)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = -y_1, & y_0(0) = 1.0 \\ y'_1 = y_0, & y_1(0) = 0.0 \end{cases}$$

用连分式法计算当 $h = 0.1$ 时, 各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

上的未知函数值 $y_i(t_j)$ ($i = 0, 1; j = 0, 1, \dots, 10$)。取 $\text{eps} = 0.000001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: `gpbs 10.c`)如下:

```
# include "stdio.h"
# include "gpbs1.c"
main()
| int i;
| double t, h, eps, y[2];
| t = 0.0; h = 0.1; eps = 0.000001;
| y[0] = 1.0; y[1] = 0.0;
| printf("\n");
| printf("t = %7.3f y(0) = %e y(1) = %e\n", t, y[0], y[1]);
for (i = 0; i <= 9; i++)
|   gpbs1(t, h, 2, y, eps);
|   t = t + h;
|   printf("t = %7.3f y(0) = %e y(1) = %e\n", t, y[0], y[1]);
```

```
| printf("\n");
{
```

```
void gpbslf(t, y, n, d)
int n;
double t, y[ ], d[ ];
| t = t; n = n;
| d[0] = -y[1]; d[1] = y[0];
return;
{
```

运行结果为:

$t = 0.000$	$y(0) = 1.00000e+00$	$y(1) = 0.00000e+00$
$t = 0.100$	$y(0) = 9.95004e-01$	$y(1) = 9.9835e-02$
$t = 0.200$	$y(0) = 9.80067e-01$	$y(1) = 1.98669e-01$
$t = 0.300$	$y(0) = 9.55336e-01$	$y(1) = 2.95520e-01$
$t = 0.400$	$y(0) = 9.21061e-01$	$y(1) = 3.89419e-01$
$t = 0.500$	$y(0) = 8.77582e-01$	$y(1) = 4.79426e-01$
$t = 0.600$	$y(0) = 8.25335e-01$	$y(1) = 5.64643e-01$
$t = 0.700$	$y(0) = 7.64842e-01$	$y(1) = 6.44218e-01$
$t = 0.800$	$y(0) = 6.96706e-01$	$y(1) = 7.17357e-01$
$t = 0.900$	$y(0) = 6.21609e-01$	$y(1) = 7.83327e-01$
$t = 1.000$	$y(0) = 5.40302e-01$	$y(1) = 8.41471e-01$

本问题的解析解为:

$$\begin{cases} y_0 = \cos t \\ y_1 = \sin t \end{cases}$$

7.10 全区间积分的连分式法

一、功能

用连分式法对一阶微分方程组进行全区间积分。

二、方法说明

同 7.9 节。

三、函数语句

`void gpbs2(t, h, n, y, eps, k, z)`

本函数要调用计算微分方程组右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ 值的函数 `gpbs2f`, 由用户自编, 其形式如下:

```
void gpbs2f(t, y, n, d)
int n;
double t, y[ ], d[ ];
```

```

    | d[0] = f0(t, y0, y1, ..., yn-1) 的表达式;
    |
    |   :
    | d[n-1] = fn-1(t, y0, y1, ..., yn-1) 的表达式;
    | return;
}

```

四、形参说明

t——双精度实型变量。积分的起始点。

h——双精度实型变量。积分一步的步长。

n——整型变量。一阶微分方程组中方程个数，也是未知函数的个数。

y——双精度实型一维数组，长度为 n。存放在初始点 t 处的 n 个未知函数值(即初值)

$y_i(t) (i=0, 1, \dots, n-1)$ 。返回时其值在二维数组 z 的第零列。即

$$z(i, 0) = y_i(t), i = 0, 1, \dots, n-1$$

eps——双精度实型变量。积分一步的精度要求。

k——整型变量。积分的步数(包括初始点这一步)。

z——双精度实型二维数组，体积为 $n \times k$ 。返回 n 个未知函数在 k 个积分点上的函数值，即

$$z(i, j) = y_i(t_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0) = y_i(t) (i = 0, 1, \dots, n-1)$ 。

五、函数程序(文件名: gpbs 2.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = -y_1, & y_0(0) = 1.0 \\ y'_1 = y_0, & y_1(0) = 0.0 \end{cases}$$

用全区间积分的连分式法计算当 $h = 0.1$ 时，各积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数值 $y_0(t_i)$ 及 $y_1(t_j) (j = 0, 1, \dots, 10)$ 。取 $eps = 0.000001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gpbs 20.c)如下：

```

#include "stdio.h"
#include "gpbs2.c"
main()
{
    int i;
    double t, h, eps, z[2][11], y[2];
    t=0.0; h=0.1; eps=0.000001;
    y[0]=1.0; y[1]=0.0;
    printf("\n");
    gpbs2(t, h, 2, y, eps, 11, z);
    for (i=0; i<=10; i++)
        | t = i * h;
    printf("t = %7.3f y(0)= %e y(1)= %e\n",

```

```

        | t, z[0][i], z[1][i]);
    |
    | printf("\n");
    |
    void gpbs2f(t, y, n, d)
    int n;
    double t, y[], d[];
    | t=t; n=n;
    | d[0] = -y[1]; d[1] = y[0];
    | return;
}

```

运行结果为：

t = 0.000	y(0) = 1.00000e+00	y(1) = 0.00000e+00
t = 0.100	y(0) = 9.95004e-01	y(1) = 9.98335e-02
t = 0.200	y(0) = 9.80067e-01	y(1) = 1.98669e-01
t = 0.300	y(0) = 9.55336e-01	y(1) = 2.95520e-01
t = 0.400	y(0) = 9.21061e-01	y(1) = 3.89419e-01
t = 0.500	y(0) = 8.77582e-01	y(1) = 4.79426e-01
t = 0.600	y(0) = 8.25335e-01	y(1) = 5.64643e-01
t = 0.700	y(0) = 7.64842e-01	y(1) = 6.44218e-01
t = 0.800	y(0) = 6.96706e-01	y(1) = 7.17357e-01
t = 0.900	y(0) = 6.21609e-01	y(1) = 7.83327e-01
t = 1.000	y(0) = 5.40302e-01	y(1) = 8.41471e-01

本问题的解析解为：

$$\begin{cases} y_0 = \cos t \\ y_1 = \sin t \end{cases}$$

7.11 全区间积分的双边法

一、功能

用双边法对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

用双边法积分一步(步长为 h)的计算公式为：

$$\begin{aligned} p_i^{(j+2)} &= -4y_i^{(j+1)} + 5y_i^{(j)} + 2h[2f_i^{(j+1)} + f_i^{(j)}] \\ q_i^{(j+2)} &= 4y_i^{(j+1)} - 3y_i^{(j)} + \frac{2}{3}h[f_i^{(j+2)} - 2f_i^{(j+1)} - 2f_i^{(j)}] \end{aligned}$$

$$y_i^{(j+2)} = \frac{1}{2} [p_i^{(j+2)} + q_i^{(j+2)}] \quad i = 0, 1, \dots, n-1$$

其中

$$y_i^{(j)} = y_i(t_j)$$

$$f_i^{(j)} = f_i(t_j, y_0^{(j)}, y_1^{(j)}, \dots, y_{n-1}^{(j)})$$

$$f_i^{(j+1)} = f_i(t_j + h, y_0^{(j+1)}, y_1^{(j+1)}, \dots, y_{n-1}^{(j+1)})$$

$$f_i^{(j+2)} = f_i(t_j + 2h, p_0^{(j+2)}, p_1^{(j+2)}, \dots, p_{n-1}^{(j+2)}) \quad i = 0, 1, \dots, n-1$$

本方法为多步法，在进行全区间积分时，要求采用某种单步法起步算出 $y_i(t_1)$ ($i = 0, 1, \dots, n-1$)。在本函数中，采用积分一步的变步长龙格-库塔(Runge-Kutta)法起步算出 $y_i(t_1)$ ($i = 0, 1, \dots, n-1$)。

三、函数语句

`void ggjfq(t, h, n, y, eps, k, z)`

本函数要调用计算微分方程组右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 `ggjfqf`，由用户自编，其形式如下：

```
void ggjfqf(t, y, n, d)
int n;
double t, y[ ], d[ ];
| d[0]=f_0(t, y_0, y_1, ..., y_{n-1})的表达式;
:
d[n-1]=f_{n-1}(t, y_0, y_1, ..., y_{n-1})的表达式;
return;
}
```

四、形参说明

t ——双精度实型变量。积分的起始点。

h ——双精度实型变量。积分一步的步长。

n ——整型变量。一阶微分方程组中方程个数，也是未知函数的个数。

y ——双精度实型一维数组，长度为 n 。存放在起始点 t 处的 n 个未知函数值(即初值) $y_i(t)$ ($i = 0, 1, \dots, n-1$)。返回时其值在二维数组 z 的第零列中，即

$$z(i, 0) = y_i(t), \quad i = 0, 1, \dots, n-1$$

eps ——双精度实型变量。在调用变步长龙格-库塔法函数时的控制精度要求。

k ——整型变量。积分的步数(包括起始点这一步)。

z ——双精度实型二维数组，体积为 $n \times k$ 。返回 k 个积分点(包括起始点)处 n 个未知函数值，即

$$z(i, j) = y_i(t_j), \quad i = 0, 1, \dots, n-1; \quad j = 0, 1, \dots, k-1$$

其中 $z(i, 0) = y_i(t)$ 为初值。

五、函数程序(文件名: ggjfq.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = -y_1, & y_0(0) = 1.0 \\ y'_1 = y_0, & y_1(0) = 0.0 \end{cases}$$

用双边法计算当 $h = 0.1$ 时，各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

处的未知函数值 $y_0(t_i)$ 及 $y_1(t_i)$ ($i = 0, 1, \dots, 10$)。取 $eps = 0.00001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: ggjfq 0.c)如下：

```
#include "stdio.h"
#include "ggjfq.c"
main()
| int i;
double y[2], z[2][11];
double t, h, eps;
t=0.0; h=0.1; eps=0.00001;
y[0]=1.0; y[1]=0.0;
ggjfq(t, h, 2, y, eps, 11, z);
printf("\n");
for (i=0; i<=10; i++)
| t=i*h;
printf("t = %7.3f y(0) = %e, y(1) = %e\n",
t, z[0][i], z[1][i]);
|
printf("\n");
|
void ggjfqf(t, y, n, d)
int n;
double t, y[ ], d[ ];
| t=t;n=n;
d[0]=-y[1];d[1]=y[0];
return;
|
```

运行结果为：

$t = 0.000$	$y(0) = 1.00000e+00$	$y(1) = 0.00000e+00$
$t = 0.100$	$y(0) = 9.95004e-01$	$y(1) = 9.98334e-02$
$t = 0.200$	$y(0) = 9.80067e-01$	$y(1) = 1.98669e-01$
$t = 0.300$	$y(0) = 9.55337e-01$	$y(1) = 2.95520e-01$
$t = 0.400$	$y(0) = 9.21061e-01$	$y(1) = 3.89417e-01$
$t = 0.500$	$y(0) = 8.77583e-01$	$y(1) = 4.79425e-01$
$t = 0.600$	$y(0) = 8.25336e-01$	$y(1) = 5.64641e-01$
$t = 0.700$	$y(0) = 7.64843e-01$	$y(1) = 6.44217e-01$
$t = 0.800$	$y(0) = 6.96708e-01$	$y(1) = 7.17355e-01$
$t = 0.900$	$y(0) = 6.21611e-01$	$y(1) = 7.83326e-01$
$t = 1.000$	$y(0) = 5.40304e-01$	$y(1) = 8.41470e-01$

本问题的解析解为：

$$\begin{cases} y_0 = \cos t \\ y_1 = \sin t \end{cases}$$

7.12 全区间积分的阿当姆斯预报-校正法

一、功能

用阿当姆斯(Adams)预报-校正公式对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为：

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

积分步长为 h , 则阿当姆斯预报-校正公式为：

$$\text{预报公式 } \bar{y}_{i,j+1} = y_{ij} + [55f_{ij} - 59f_{i,j-1} + 37f_{i,j-2} - 9f_{i,j-3}]h/24$$

$$\text{校正公式 } \bar{y}_{i,j+1} = y_{ij} + [9f_{i,j+1} + 19f_{ij} - 5f_{i,j-1} + f_{i,j-2}]h/24$$

$$i = 0, 1, \dots, n-1$$

其中

$$f_{ik} = f_i(t_k, y_{0k}, y_{1k}, \dots, y_{n-1,k})$$

$$k = j-3, j-2, j-1, j; i = 0, 1, \dots, n-1$$

$$f_{i,j+1} = f_i(t_{j+1}, \bar{y}_{0,j+1}, \bar{y}_{1,j+1}, \dots, \bar{y}_{n-1,j+1}), i = 0, 1, \dots, n-1$$

阿当姆斯方法是线性多步法, 在计算新点上的未知函数值时要用到前面四个点上的未知函数值。在本函数中, 采用变步长四阶龙格-库塔公式计算开始四个点上的未知函数值 $y_{i0}, y_{i1}, y_{i2}, y_{i3}$ ($i = 0, 1, \dots, n-1$), 其中 y_{i0} 为给定的初值。

三、函数语句

void gadms(t, h, n, y, eps, k, z)

本函数要调用计算微分方程组右端各函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 gadmsf, 由用户自编, 其形式如下:

```
void gadmsf(t, y, n, d)
int n;
double t, h, eps, ;
| d[0] = f_0(t, y_0, y_1, \dots, y_{n-1}) 的表达式;
| :
| d[n-1] = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}) 的表达式;
| return;
|
```

四、形参说明

t —双精度实型变量。积分的起始点。

h —双精度实型变量。积分一步的步长。

n —整型变量。一阶微分方程组中方程个数, 也是未知函数的个数。

y —双精度实型一维数组, 长度为 n 。存放在起始点 t 处的 n 个未知函数值(即初值) $y_i(t)$ ($i = 0, 1, \dots, n-1$)。返回时其值在二维数组 z 的第零列, 即

$$z(i, 0) = y_i(t), i = 0, 1, \dots, n-1$$

eps —双精度实型变量。在调用变步长龙格-库塔法时的控制精度要求。

k —整型变量。积分的步数(包括起始点这一步)。

z —双精度实型二维数组, 体积为 $n \times k$ 。返回在 k 个积分点(包括起始点)处的 n 个未知函数值 $y_i(t_j)$ ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$), 即

$$z(i, j) = y_i(t_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0) = y_i(t)$ 为初值。

五、函数程序(文件名: gadms.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = 0.0 \\ y'_1 = -y_0, & y_1(0) = 1.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用阿当姆斯预报-校正公式计算当 $h = 0.05$ 时, 各积分点

$$t_i = i * h, i = 0, 1, \dots, 10$$

上的未知函数值 y_{0i}, y_{1i}, y_{2i} ($i = 0, 1, \dots, 10$)。取 $\text{eps} = 0.0001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gadms0.c)如下:

```
#include "stdio.h"
#include "gadms.c"
main()
| int i, j;
| double t, h, eps, ;
| double y[3], z[3][11];
| y[0] = 0.0; y[1] = 1.0; y[2] = 1.0;
| t = 0.0; h = 0.05; eps = 0.0001;
| gadms(t, h, 3, y, eps, 11, z);
| printf("\n");
| for (i = 0; i < 10; i++)
| | t = i * h;
| | printf("t = %7.3f\n", t);
| | for (j = 0; j <= 2; j++)
| | | printf("y (%d) = %e\n", j, z[j][i]);
| | | printf("\n");
| |
| printf("\n");
```

```
void gadmsf(t, y, n, d)
```

```

int n;
double t, y[], d[];
{ t = t; n = n;
d[0] = y[1]; d[1] = -y[0]; d[2] = -y[2];
return;
}

```

运行结果为：

```

t=0.000
y(0)=0.00000e+00 y(1)=1.00000e+00 y(2)=1.00000e+00
t=0.050
y(0)=4.99792e-02 y(1)=9.98750e-01 y(2)=9.51229e-01
t=0.100
y(0)=9.98334e-02 y(1)=9.95004e-01 y(2)=9.04837e-01
t=0.150
y(0)=1.49438e-01 y(1)=9.88771e-01 y(2)=8.60708e-01
t=0.200
y(0)=1.98669e-01 y(1)=9.80067e-01 y(2)=8.18731e-01
t=0.250
y(0)=2.47404e-01 y(1)=9.68912e-01 y(2)=7.78801e-01
t=0.300
y(0)=2.95520e-01 y(1)=9.55336e-01 y(2)=7.40818e-01
t=0.350
y(0)=3.42898e-01 y(1)=9.39373e-01 y(2)=7.04688e-01
t=0.400
y(0)=3.89418e-01 y(1)=9.21061e-01 y(2)=6.70320e-01
t=0.450
y(0)=4.34966e-01 y(1)=9.00447e-01 y(2)=6.37628e-01
t=0.500
y(0)=4.79426e-01 y(1)=8.77583e-01 y(2)=6.06531e-01

```

本问题的解析解为：

$$\begin{cases} y_0 = \sin t \\ y_1 = \cos t \\ y_2 = e^{-t} \end{cases}$$

7.13 全区间积分的哈明方法

一、功能

用哈明(Hamming)方法对一阶微分方程组进行全区间积分。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

积分步长为 h 。则哈明方法的计算公式为：

$$\text{预报 } \bar{P}_{i,j+1} = \bar{y}_{i,j-3} + 4h(2f_{ij} - f_{i,j-1} + 2f_{i,j-2})/3$$

$$\text{修正 } P_{i,j+1} = \bar{P}_{i,j+1} + \frac{112}{121}(C_{ij} - P_{ij})$$

$$\text{校正 } C_{i,j+1} = \frac{1}{8}[9y_{ij} - y_{i,j-2} + 3h(f_{i,j+1} + 2f_{ij} - f_{i,j-1})] \\ i = 0, 1, \dots, n-1$$

其中

$$f_{ik} = f_i(t_k, y_{0k}, y_{1k}, \dots, y_{n-1,k}), k = j-2, j-1, j; i = 0, 1, \dots, n-1$$

$$f_{i,j+1} = f_i(t_{j+1}, P_{0,j+1}, P_{1,j+1}, \dots, P_{n-1,j+1}), i = 0, 1, \dots, n-1$$

$$\text{终值 } y_{i,j+1} = C_{i,j+1} - \frac{9}{121}(C_{i,j+1} - P_{i,j+1}), i = 0, 1, \dots, n-1$$

哈明方法是线性多步法，在计算新点上的未知函数值时要用到前面四个点上的未知函数值。在本函数中，采用积分一步的变步长龙格-库塔法计算开始四个点上的未知函数值 $y_{i0}, y_{i1}, y_{i2}, y_{i3}$ ($i = 0, 1, \dots, n-1$)，其中 y_{i0} 为初值。

三、函数语句

void ghamg(t, h, n, y, eps, k, z)

本函数要调用计算微分方程组右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 ghamgf，由用户自编，其形式如下：

```

void ghamgf(t, y, n, d)
int n;
double t, y[], d[];
{ d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
  :
  d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
}

```

四、形参说明

t——双精度实型变量。积分起始点。

h——双精度实型变量。积分一步的步长。

n——整型变量。一阶微分方程组中方程个数，也是未知函数的个数。

y——双精度实型一维数组，长度为 n。存放起始点 t 处的 n 个未知函数的初值 $y_i(t)$ ($i = 0, 1, \dots, n-1$)。返回时其值在二维数组 z 的第零列，即

$$z(i, 0) = y_i(t), i = 0, 1, \dots, n-1$$

eps——双精度实型变量。在调用积分一步的变步长龙格-库塔法时的精度要求。

k——整型变量。积分的步数(包括起始点这一步)。

z——双精度实型二维数组，体积为 $n \times k$ 。返回 k 个积分点(包括起始点)上 n 个未知函数值 $y_i(t_j)$ ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$)，即

$$z(i, j) = y_i(t_j), i = 0, 1, \dots, n-1; j = 0, 1, \dots, k-1$$

其中 $z(i, 0)$ 为初值。

五、函数程序(文件名: ghamg.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = y_1, & y_0(0) = 1.0 \\ y'_1 = -y_0, & y_1(0) = 1.0 \\ y'_2 = -y_2, & y_2(0) = 1.0 \end{cases}$$

用哈明方法计算当 $h=0.05$ 时, 各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

上的未知函数值 y_{0i}, y_{1i}, y_{2i} ($i=0, 1, \dots, 10$)。取 $eps=0.0001$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: ghamg0.c)如下:

```
# include "stdio.h"
# include "ghamg.c"
main()
{
    int i, j;
    double t, h, eps, y[3], z[3][11];
    y[0] = 1.0; y[1] = 1.0; y[2] = 1.0;
    t = 0.0; h = 0.05; eps = 0.0001;
    ghamg(t, h, 3, y, eps, 11, z);
    printf("\n");
    for (i = 0; i <= 10; i++)
    {
        t = i * h;
        printf("t = %7.3f\n", t);
        for (j = 0; j <= 2; j++)
            printf("y(%d) = %e\n", j, z[j][i]);
        printf("\n");
    }
    printf("\n");
}

void ghamgf(t, y, n, d)
int n;
double t, y[], d[];
{
    t = t; n = n;
    d[0] = y[1]; d[1] = -y[0]; d[2] = y[2];
    return;
}
```

运行结果为:

```
t=0.000
y(0)=1.00000e+00  y(1)=1.00000e+00  y(2)=1.00000e+00
t=0.050
y(0)=1.04873e+00  y(1)=9.48771e-01  y(2)=1.05127e+00
t=0.100
```

$y(0) = 1.09484e+00$	$y(1) = 8.95171e-01$	$y(2) = 1.10517e+00$
$t = 0.150$		
$y(0) = 1.13821e+00$	$y(1) = 8.39333e-01$	$y(2) = 1.16183e+00$
$t = 0.200$		
$y(0) = 1.17874e+00$	$y(1) = 7.81397e-01$	$y(2) = 1.22140e+00$
$t = 0.250$		
$y(0) = 1.21632e+00$	$y(1) = 7.21508e-01$	$y(2) = 1.28403e+00$
$t = 0.300$		
$y(0) = 1.25086e+00$	$y(1) = 6.59816e-01$	$y(2) = 1.34986e+00$
$t = 0.350$		
$y(0) = 1.28227e+00$	$y(1) = 5.96475e-01$	$y(2) = 1.41907e+00$
$t = 0.400$		
$y(0) = 1.31048e+00$	$y(1) = 5.31643e-01$	$y(2) = 1.49182e+00$
$t = 0.450$		
$y(0) = 1.33541e+00$	$y(1) = 4.65482e-01$	$y(2) = 1.56831e+00$
$t = 0.500$		
$y(0) = 1.35701e+00$	$y(1) = 3.98157e-01$	$y(2) = 1.64872e+00$

本问题的解析解为:

$$\begin{cases} y_0 = \sin t + \cos t \\ y_1 = \cos t - \sin t \\ y_2 = e^t \end{cases}$$

7.14 积分一步的特雷纳方法

一、功能

用特雷纳(Treanor)方法对一阶刚性(stiff)微分方程组积分一步。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

如果矩阵

$$\begin{bmatrix} \frac{\partial f_0}{\partial y_0} & \frac{\partial f_0}{\partial y_1} & \cdots & \frac{\partial f_0}{\partial y_{n-1}} \\ \frac{\partial f_1}{\partial y_0} & \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_{n-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_{n-1}}{\partial y_0} & \frac{\partial f_{n-1}}{\partial y_1} & \cdots & \frac{\partial f_{n-1}}{\partial y_{n-1}} \end{bmatrix}$$

的特征值 λ_i 具有如下特性:

$\operatorname{Re}\lambda_i < 0$ 且 $\max_{0 \leq i \leq n-1} |\operatorname{Re}\lambda_i| \gg \min_{0 \leq i \leq n-1} |\operatorname{Re}\lambda_i|$

则称此微分方程组为刚性的。

求解刚性方程的特雷纳方法如下。

设已知 t_j 点处的未知函数值 y_{ij} ($i = 0, 1, \dots, n-1$), 则计算 $t_{j+1} = t_j + h$ 点处未知函数值 $y_{i,j+1}$ 的公式为

$$y_{i,j+1} = y_{ij} + \Delta y_i, \quad i = 0, 1, \dots, n-1$$

其中

$$\Delta y_i = \begin{cases} \frac{h}{6} [q_i^{(1)} + 2(q_i^{(2)} + q_i^{(3)}) + q_i^{(4)}], & p_i \leq 0 \\ h \{q_i^{(1)}r_i^{(2)} + [-3(q_i^{(1)} + p_i w_i^{(1)}) + 2(q_i^{(2)} + p_i w_i^{(2)}) \\ + 2(q_i^{(3)} + p_i w_i^{(3)}) - (q_i^{(4)} + p_i w_i^{(4)})]r_i^{(3)} \\ + 4[(q_i^{(1)} + p_i w_i^{(1)}) - (q_i^{(2)} + p_i w_i^{(2)}) - (q_i^{(3)} + p_i w_i^{(3)}) \\ + (q_i^{(4)} + p_i w_i^{(4)})]r_i^{(4)}\}, & p_i > 0 \end{cases}$$

式中

$$p_i = \frac{q_i^{(3)} - q_i^{(2)}}{w_i^{(3)} - w_i^{(2)}}, \quad i = 0, 1, \dots, n-1$$

$$r_i^{(1)} = e^{-p_i h}, \quad r_i^{(2)} = \frac{r_i^{(1)} - 1}{-p_i h}, \quad r_i^{(3)} = \frac{r_i^{(2)} - 1}{-p_i h},$$

$$r_i^{(4)} = \frac{r_i^{(3)} - \frac{1}{2}}{-p_i h}, \quad i = 0, 1, \dots, n-1$$

$$w_i^{(1)} = y_{ij}, \quad q_i^{(1)} = f_i(t_j, w_0^{(1)}, w_1^{(1)}, \dots, w_{n-1}^{(1)})$$

$$w_i^{(2)} = w_i^{(1)} + \frac{h}{2} q_i^{(1)}, \quad q_i^{(2)} = f_i\left(t_j + \frac{h}{2}, w_0^{(2)}, w_1^{(2)}, \dots, w_{n-1}^{(2)}\right)$$

$$w_i^{(3)} = w_i^{(2)} + \frac{h}{2} q_i^{(2)}, \quad q_i^{(3)} = f_i\left(t_j + \frac{h}{2}, w_0^{(3)}, w_1^{(3)}, \dots, w_{n-1}^{(3)}\right)$$

$$w_i^{(4)} = w_i^{(3)} + h q_i, \quad q_i^{(4)} = f_i(t_j + h, w_0^{(4)}, w_1^{(4)}, \dots, w_{n-1}^{(4)}), \quad i = 0, 1, \dots, n-1$$

q_i 由公式

$$q_i = \begin{cases} q_i^{(3)}, & p_i \leq 0 \\ 2(q_i^{(3)} - q_i^{(1)})r_i^{(3)} + (q_i^{(1)} - q_i^{(2)})r_i^{(2)} + q_i^{(2)}, & p_i > 0 \end{cases} \quad i = 0, 1, \dots, n-1$$

确定。

由上述计算公式可知, 当 $p_i \leq 0$ 时, 本方法便退化为四阶龙格-库塔方法。

本方法适合于求解刚性问题, 但对一般的一阶微分方程组同样适用。

三、函数语句

void gtnrlf(t, h, n, y)

本函数要调用计算微分方程组右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 gtnrlf, 由用户自编, 其形式如下:

```
void gtnrlf(t, y, n, d)
int n;
double t, y[ ], d[ ];
{ d[0] = f0(t, y0, y1, ..., yn-1) 的表达式;
  :
  d[n-1] = fn-1(t, y0, y1, ..., yn-1) 的表达式;
  return;
}
```

四、形参说明

t——双精度实型变量。积分的起始点。

h——双精度实型变量。积分一步的步长。

n——整型变量。一阶微分方程组中方程个数, 也是未知函数的个数。

y——双精度实型一维数组, 长度为 n。存放起始点 t 处的 n 个未知函数值 $y_i(t)$; 返回时存放积分点 $t + h$ 处的 n 个未知函数值 $y_i(t + h)$ ($i = 0, 1, \dots, n-1$)。

五、函数程序(文件名: gtnr 1.c)

六、例

设一阶刚性微分方程组及初值为

$$\begin{cases} y'_0 = -21y_0 + 19y_1 - 20y_2, & y_0(0) = 1.0 \\ y'_1 = 19y_0 - 21y_1 + 20y_2, & y_1(0) = 0.0 \\ y'_2 = 40y_0 - 40y_1 - 40y_2, & y_2(0) = -1.0 \end{cases}$$

用积分一步的特雷纳方法计算当 $h = 0.001$ 时, 各积分点

$$t_i = i * h, \quad i = 0, 1, \dots, 10$$

处的未知函数值 $y_0(t_i), y_1(t_i), y_2(t_i)$ ($i = 0, 1, \dots, 10$)。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gtnr 10.c)如下:

```
# include "stdio.h"
# include "gtnrlf.c"
main()
{ int i, j;
  double t, h, y[3];
  y[0] = 1.0; y[1] = 0.0; y[2] = -1.0;
  t = 0.0; h = 0.001;
  printf("\n");
  printf("t = %6.3f\n", t);
  for (i = 0; i <= 2; i++)
    printf("y(%d) = %e\n", i, y[i]);
  printf("\n");
  for (i = 0; i <= 9; i++)
    { gtnrlf(t, h, 3, y);
      t = t + h;
      printf("t = %6.3f\n", t);
      for (j = 0; j <= 2; j++)
        }
```

```

printf("y(%d) = %e\n", j, y[j]);
printf("\n");
}

void gtnr1f(t, y, n, d)
int n;
double t, y[], d[];
{ t = t; n = n;
  d[0] = -21.0 * y[0] + 19.0 * y[1] - 20.0 * y[2];
  d[1] = 19.0 * y[0] - 21.0 * y[1] + 20.0 * y[2];
  d[2] = 40.0 * y[0] - 40.0 * y[1] - 40.0 * y[2];
  return;
}

```

运行结果为：

t=0.000	y(0)=1.00000e+00	y(1)=0.00000e+00	y(2)=-1.00000e+00
t=0.001	y(0)=9.98243e-01	y(1)=-2.40944e-04	y(2)=-9.21600e-01
t=0.002	y(0)=9.95011e-01	y(1)=9.96761e-04	y(2)=-8.45389e-01
t=0.003	y(0)=9.90406e-01	y(1)=3.61195e-03	y(2)=-7.72922e-01
t=0.004	y(0)=9.84554e-01	y(1)=7.47766e-03	y(2)=-7.03802e-01
t=0.005	y(0)=9.77570e-01	y(1)=1.24796e-02	y(2)=-6.37904e-01
t=0.006	y(0)=9.69562e-01	y(1)=1.85105e-02	y(2)=-5.75151e-01
t=0.007	y(0)=9.60629e-01	y(1)=2.54688e-02	y(2)=-5.15481e-01
t=0.008	y(0)=9.50869e-01	y(1)=3.32583e-02	y(2)=-4.58830e-01
t=0.009	y(0)=9.40374e-01	y(1)=4.17877e-02	y(2)=-4.05136e-01
t=0.010	y(0)=9.29229e-01	y(1)=5.09707e-02	y(2)=-3.54333e-01

本问题的解析解为：

$$\begin{cases} y_0(t) = \frac{1}{2}e^{-2t} + \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_1(t) = \frac{1}{2}e^{-2t} - \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_2(t) = -e^{-40t}(\cos 40t - \sin 40t) \end{cases}$$

7.15 全区间积分的特雷纳方法

一、功能

用特雷纳(Treanor)方法对一阶刚性(stiff)微分方程组进行全区间积分。

二、方法说明

同 7.14 节。

三、函数语句

void gtnr2f(t, h, n, y, k, z)

本函数要调用计算一阶微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ ($i = 0, 1, \dots, n-1$) 值的函数 gtnr2f, 由用户自编, 其形式如下:

```

void gtnr2f(t, y, n, d)
int n;
double t, y[], d[];
{ d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
  ;
  d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
  return;
}

```

四、形参说明

t——双精度实型变量。积分起始点。

h——双精度实型变量。积分一步的步长。

n——整型变量。一阶微分方程组中方程个数, 也是未知函数的个数。

y——双精度实型一维数组, 长度为 n。存放在起始点 t 处的 n 个未知函数值 $y_i(t)$ (即初值)。返回时其值在二维数组 z 的第零列中。即

$$z(i, 0) = y_i(t), \quad i = 0, 1, \dots, n-1$$

k——整型变量。积分的步数(包括起始点这一步)。

z——双精度实型二维数组, 体积为 $n \times k$ 。返回 k 个积分点处的 n 个未知函数值, 即 $z(i, j) = y_i(t_j)$, $i = 0, 1, \dots, n-1$; $j = 0, 1, \dots, k-1$

其中 $z(i, 0) = y_i(t)$ ($i = 0, 1, \dots, n-1$) 为初值。

五、函数程序(文件名: gtnr2.c)

六、例

设一阶刚性微分方程组及初值为

$$\begin{cases} y'_0 = -21y_0 + 19y_1 - 20y_2, & y_0(0) = 1.0 \\ y'_1 = 19y_0 - 21y_1 + 20y_2, & y_1(0) = 0.0 \\ y'_2 = 40y_0 - 40y_1 - 40y_2, & y_2(0) = -1.0 \end{cases}$$

用全区间积分的特雷纳方法计算当 $h = 0.001$ 时, 各积分点
 $t_i = i * h, i = 0, 1, \dots, 101$

处的未知函数值 $y_0(t_i), y_1(t_i), y_2(t_i) (i = 0, 1, \dots, 101)$ 。

主函数程序及计算微分方程组右端函数值的函数程序(文件名: gtnr20.c)如下:

```
#include "stdio.h"
#include "gtnr2.c"
main()
{
    int i, j;
    double t, h, y[3], z[3][101];
    y[0] = 1.0; y[1] = 0.0; y[2] = -1.0;
    t = 0.0; h = 0.001;
    gtnr2(t, h, 3, y, 101, z);
    printf("\n");
    for (i = 0; i <= 100; i = i + 10)
    {
        t = i * h;
        printf("t = %6.3f\n", t);
        for (j = 0; j <= 2; j++)
            printf("y(%d) = %e, ", j, z[j][i]);
        printf("\n");
    }
    printf("\n");
}

void gtnr2f(t, y, n, d)
int n;
double t, y[], d[];
{
    t = t; n = n;
    d[0] = -21.0 * y[0] + 19.0 * y[1] - 20.0 * y[2];
    d[1] = 19.0 * y[0] - 21.0 * y[1] + 20.0 * y[2];
    d[2] = 40.0 * y[0] - 40.0 * y[1] - 40.0 * y[2];
    return;
}
```

运行结果为:

```
t=0.000
y(0)=1.00000e+00  y(1)=0.00000e+00  y(2)=-1.00000e+00
t=0.010
y(0)=9.29229e-01  y(1)=5.09707e-02  y(2)=-3.54333e-01
t=0.020
y(0)=7.97831e-01  y(1)=1.62959e-01  y(2)=1.08604e-02
t=0.030
y(0)=6.65430e-01  y(1)=2.76335e-01  y(2)=1.72648e-01
t=0.040
y(0)=5.59130e-01  y(1)=3.63987e-01  y(2)=2.08260e-01
t=0.050
y(0)=4.85243e-01  y(1)=4.19592e-01  y(2)=1.79452e-01
t=0.060
y(0)=4.40267e-01  y(1)=4.46652e-01  y(2)=1.27956e-01
t=0.070
y(0)=4.15985e-01  y(1)=4.53371e-01  y(2)=7.73260e-02
t=0.080
```

$y(0) = 4.04428e-01$	$y(1) = 4.47714e-01$	$y(2) = 3.79085e-02$
$t = 0.090$		
$y(0) = 3.99326e-01$	$y(1) = 4.35943e-01$	$y(2) = 1.20397e-02$
$t = 0.100$		
$y(0) = 3.96487e-01$	$y(1) = 4.22242e-01$	$y(2) = -2.13904e-03$

本问题的解析解为:

$$\begin{cases} y_0(t) = \frac{1}{2}e^{-2t} + \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_1(t) = \frac{1}{2}e^{-2t} - \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_2(t) = -e^{-40t}(\cos 40t - \sin 40t) \end{cases}$$

7.16 积分刚性方程组的吉尔方法

一、功能

用吉尔(Gear)方法积分一阶常微分方程组的初值问题。

二、方法说明

设一阶微分方程组为

$$\begin{cases} y'_0 = f_0(t, y_0, y_1, \dots, y_{n-1}), & y_0(t_0) = y_{00} \\ y'_1 = f_1(t, y_0, y_1, \dots, y_{n-1}), & y_1(t_0) = y_{10} \\ \vdots \\ y'_{n-1} = f_{n-1}(t, y_0, y_1, \dots, y_{n-1}), & y_{n-1}(t_0) = y_{n-1,0} \end{cases}$$

吉尔方法的计算公式如下:

预报 $Z_{i,(0)} = PZ_{i-1}$

校正 $Z_{i,(j+1)} = Z_{i,(j)} - L \left[\left[l_1 I - l_0 \frac{\partial F}{\partial Y} \right]^{-1} G(Z_{i,(j)}) \right]^T$

终值 $Z_i = Z_{i,(M)}$

其中

$$Z_i = (Y_i, hY'_i, \dots, h^p Y_i^{(p)} / p!)^T$$

为 $(p+1) \times n$ 的矩阵;

$$G(Z_{i,(j)}) = hF(t_i, Y_{i,(j)}) - hY_{i,(j)}$$

为 n 维列向量; $\frac{\partial F}{\partial Y}$ 为 $n \times n$ 的雅可比(Jacobi)矩阵, 即

$$\frac{\partial F}{\partial Y} = \begin{bmatrix} \frac{\partial f_0}{\partial y_0} & \frac{\partial f_0}{\partial y_1} & \cdots & \frac{\partial f_0}{\partial y_{n-1}} \\ \frac{\partial f_1}{\partial y_0} & \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{n-1}}{\partial y_0} & \frac{\partial f_{n-1}}{\partial y_1} & \cdots & \frac{\partial f_{n-1}}{\partial y_{n-1}} \end{bmatrix}$$

P 为 $(p+1)$ 阶的巴斯卡尔(PASCAL)三角矩阵, 即

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 2 & 3 & \cdots & k'-1 & k \\ 1 & 3 & \cdots & \vdots & \vdots & \\ 1 & \cdots & \vdots & \vdots & & \\ \ddots & k-1 & k \\ 1 & k \\ 1 & & & & & & \end{bmatrix}$$

L 为 $(p+1)$ 维列向量, 即

$$L = (l_0, l_1, \dots, l_p)^T$$

在本函数中, $M=3$ (即迭代校正 3 次)。向量 L 在本函数中自带。

上述计算过程是自开始、自动变步长且自动变阶的。

在本函数中, 为了满足精度要求, 首先考虑减小步长, 只有当达到最小步长还没有满足精度要求时, 才考虑提高方法的阶数。并且在用某个阶数的方法连续进行几步的计算中均满足精度要求时, 考虑降低方法的阶数。

本函数能比较有效地积分刚性问题, 也能积分非刚性问题。

三、函数语句

int ggear(a, b, hmin, hmax, h, eps, n, y, k, t, z)

本函数返回一个整型标志, 其值的意义如下:

返回值 = +1 表示全区间积分成功;

返回值 = -1 表示步长已小于最小步长 h_{\min} , 但精度未达到, 积分停止(已算出的前若干点输出有效);

返回值 = -2 表示阶数已大于 6, 积分停止(已算出的前若干点输出有效);

返回值 = -3 表示对于 $h \geq h_{\min}$ 校正迭代不收敛, 积分停止(已算出的前若干点输出有效);

返回值 = -4 表示对所处理的问题, 要求的精度太高, 积分停止(已算出的前若干点输出有效)。

本函数要调用以下三个函数:

(1) 全选主元高斯-约当法求矩阵逆的函数 brinv, 参看 2.3 节。

(2) 计算微分方程组中各方程右端函数 $f_i(t, y_0, y_1, \dots, y_{n-1})$ 值的函数 ggwarf, 由用户自编, 其形式如下:

```
void ggwarf(t, y, n, d)
int n;
double t, y[], d[];
| d[0] = f_0(t, y_0, y_1, ..., y_{n-1}) 的表达式;
|   :
| d[n-1] = f_{n-1}(t, y_0, y_1, ..., y_{n-1}) 的表达式;
return;
```

(3) 计算雅可比矩阵的函数 ggears, 由用户自编, 其形式如下:

```
void ggears(t, y, n, p)
int n;
double t, y[], p[n][n]; (其中 n 的值见方法说明)
| int 用到的整型变量表列;
| double 用到的实型变量表列;
p[i][j] =  $\frac{\partial f_i}{\partial y_j}$  (i, j = 0, 1, ..., n-1) 的表达式;
return;
```

四、形参说明

a——双精度实型变量。积分区间的起始点。

b——双精度实型变量。积分区间的终点。

hmin——双精度实型变量。积分过程中所允许的最小步长。

hmax——双精度实型变量。积分过程中所允许的最大步长。

h——双精度实型变量。积分的拟定步长, 在积分过程中将自动放大或缩小。 $h_{\min} < h < h_{\max}$

eps——双精度实型变量。误差检验常数。

n——整型变量。一阶微分方程组中方程个数, 也是未知函数的个数。

y——双精度实型一维数组, 长度为 n 。存放积分起始点 a 处的 n 个未知函数初值 $y_i(a)$, 返回时其值在二维数组 z 的第零列; 其余为本函数的工作单元。

k——整型变量。拟定输出的积分点数。

t——双精度实型一维数组, 长度为 k 。返回 k 个输出点处的自变量值(包括起始点)。

z——双精度实型二维数组, 体积为 $n \times k$ 。返回 k 个输出点处的 n 个未知函数值, 即 $z(i, j) = y_i(t_j)$, $i = 0, 1, \dots, n-1$; $j = 0, 1, \dots, k-1$

其中 $z(i, 0)$ ($i = 0, 1, \dots, n-1$) 为初值 $y_i(a)$ 。

五、函数程序(文件名: ggear.c)

六、例

设一阶微分方程组及初值为

$$\begin{cases} y'_0 = -21y_0 + 19y_1 - 20y_2, & y_0(0) = 1.0 \\ y'_1 = 19y_0 - 21y_1 + 20y_2, & y_1(0) = 0.0 \\ y'_2 = 40y_0 - 40y_1 - 40y_2, & y_2(0) = -1.0 \end{cases}$$

这是一个刚性方程组, 其解析解为

$$\begin{cases} y_0(t) = \frac{1}{2}e^{-2t} + \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_1(t) = \frac{1}{2}e^{-2t} - \frac{1}{2}e^{-40t}(\cos 40t + \sin 40t) \\ y_2(t) = -e^{-40t}(\cos 40t - \sin 40t) \end{cases}$$

用吉尔方法分以下四种情况求在区间 $[0, 1]$ 中的数值解。其中 $a = 0.0$, $b = 1.0$, $n = 3$, 且取 $k = 30$ 。

- (1) $h = 0.01$, $h_{\min} = 0.0001$, $h_{\max} = 0.1$, $\epsilon_{ps} = 0.001$;
- (2) $h = 0.01$, $h_{\min} = 0.0001$, $h_{\max} = 0.1$, $\epsilon_{ps} = 0.0001$;
- (3) $h = 0.01$, $h_{\min} = 0.00001$, $h_{\max} = 0.1$, $\epsilon_{ps} = 0.0001$;
- (4) $h = 0.01$, $h_{\min} = 0.00001$, $h_{\max} = 0.1$, $\epsilon_{ps} = 0.00001$ 。

主函数程序及计算微分方程组右端函数值的函数程序、计算雅可比矩阵的函数程序(文

件名: ggear 0.c)如下:

```
# include "stdio.h"
# include "ggear.c"
# include "brinv.c"
main()
{
    int i, j, k, jj;
    double a, b, hmax, h, y[3], t[30], z[3][30];
    static double hmin[4] = {0.0001, 0.0001, 0.00001, 0.00001};
    static double eps[4] = {0.001, 0.0001, 0.0001, 0.00001};
    a=0.0; b=1.0; h=0.01; hmax=0.1;
    for (k=0; k<=3; k++)
    {
        y[0]=1.0; y[1]=0.0; y[2]=-1.0;
        jj=ggear(a,b,hmin[k],hmax,h,eps[k],3,y,30,t,z);
        printf("\n");
        printf("h= %5.2f \n",h);
        printf("hmin= %8.5f \n",hmin[k]);
        printf("hmax= %8.5f \n",hmax);
        printf("eps= %8.5f \n",eps[k]);
        printf("\n");
        printf("jjs= %d \n",jj);
        printf("\n");
        for (i=0;i<=29;i++)
        {
            printf("t= %10.6f",t[i]);
            for (j=0;j<=2;j++)
                printf("y(%d)= %e",j,z[j][i]);
            printf("\n");
        }
        printf("\n");
    }

    void ggearf(t, y, n, d)
    int n;
    double t, y[ ], d[ ];
    {
        t = t; n = n;
        d[0] = -21.0 * y[0] + 19.0 * y[1] - 20.0 * y[2];
        d[1] = 19.0 * y[0] - 21.0 * y[1] + 20.0 * y[2];
        d[2] = 40.0 * y[0] - 40.0 * y[1] - 40.0 * y[2];
        return;
    }
}
```

```
void ggears(t, y, n, p)
int n;
double t, y[ ], p[3][3];
{
    t=t; n=n; y[0]=y[0];
    p[0][0]=-21.0; p[0][1]=19.0; p[0][2]=-20.0;
    p[1][0]=19.0; p[1][1]=-21.0; p[1][2]=20.0;
    p[2][0]=40.0; p[2][1]=-40.0; p[2][2]=-40.0;
    return;
}
```

运行结果为:

```
h= 0.01
hmin= 0.00010
hmax= 0.10
eps= 0.0010

jjs=1

t= 0.000000   y(0)= 1.00000e+00   y(1)= 0.00000e+00   y(2)= -1.00000e+00
t= 0.000690   y(0)= 9.98591e-01   y(1)= 3.15899e-05   y(2)= -9.46338e-01
t= 0.001380   y(0)= 9.96502e-01   y(1)= 7.44849e-04   y(2)= -8.94192e-01
t= 0.003234   y(0)= 9.88675e-01   y(1)= 4.88066e-03   y(2)= -7.59460e-01
t= 0.005088   y(0)= 9.76732e-01   y(1)= 1.31464e-02   y(2)= -6.35490e-01
t= 0.006942   y(0)= 9.61290e-01   y(1)= 2.49243e-02   y(2)= -5.22061e-01
t= 0.010099   y(0)= 9.28377e-01   y(1)= 5.16304e-02   y(2)= -3.52411e-01
t= 0.013256   y(0)= 8.89586e-01   y(1)= 8.42532e-02   y(2)= -2.10762e-01
t= 0.016413   y(0)= 8.47441e-01   y(1)= 1.20268e-01   y(2)= -9.48460e-02
t= 0.019569   y(0)= 8.03844e-01   y(1)= 1.57757e-01   y(2)= -2.18476e-03
t= 0.023568   y(0)= 7.48863e-01   y(1)= 2.05096e-01   y(2)= 8.58296e-02
t= 0.027567   y(0)= 6.96222e-01   y(1)= 2.50139e-01   y(2)= 1.45945e-01
t= 0.031565   y(0)= 6.47292e-01   y(1)= 2.91531e-01   y(2)= 1.83266e-01
t= 0.035564   y(0)= 6.02954e-01   y(1)= 3.28392e-01   y(2)= 2.02440e-01
t= 0.039562   y(0)= 5.63735e-01   y(1)= 3.60192e-01   y(2)= 2.07577e-01
t= 0.045876   y(0)= 5.12593e-01   y(1)= 3.99741e-01   y(2)= 1.95664e-01
t= 0.052190   y(0)= 4.73985e-01   y(1)= 4.26901e-01   y(2)= 1.69112e-01
t= 0.058503   y(0)= 4.46209e-01   y(1)= 4.43372e-01   y(2)= 1.36401e-01
t= 0.064817   y(0)= 4.27137e-01   y(1)= 4.51281e-01   y(2)= 1.03242e-01
t= 0.071131   y(0)= 4.14653e-01   y(1)= 4.52743e-01   y(2)= 7.30822e-02
t= 0.078731   y(0)= 4.05714e-01   y(1)= 4.48597e-01   y(2)= 4.32409e-02
t= 0.086331   y(0)= 4.00860e-01   y(1)= 4.40564e-01   y(2)= 2.11851e-02
t= 0.093931   y(0)= 3.98047e-01   y(1)= 4.30683e-01   y(2)= 6.31603e-03
t= 0.101532   y(0)= 3.95908e-01   y(1)= 4.20320e-01   y(2)= -2.65137e-03
t= 0.109132   y(0)= 3.93628e-01   y(1)= 4.10287e-01   y(2)= -7.19698e-03
t= 0.119089   y(0)= 3.89743e-01   y(1)= 3.98321e-01   y(2)= -8.72038e-03
t= 0.129046   y(0)= 3.84662e-01   y(1)= 3.87864e-01   y(2)= -7.57989e-03
t= 0.139003   y(0)= 3.78578e-01   y(1)= 3.78716e-01   y(2)= -5.52278e-03
t= 0.148960   y(0)= 3.71820e-01   y(1)= 3.70543e-01   y(2)= -3.49394e-03
t= 0.158917   y(0)= 3.64699e-01   y(1)= 3.63027e-01   y(2)= -1.88095e-03
```

h= 0.01

• 209 •

• 210 •

$h_{min} = 0.00010$
 $h_{max} = 0.10$
 $\epsilon_{ps} = 0.00010$

$jjs = 1$

$t = 0.000000$	$y(0) = 1.00000e+00$	$y(1) = 0.00000e+00$	$y(2) = -1.00000e+00$
$t = 0.000218$	$y(0) = 9.99707e-01$	$y(1) = -1.43222e-04$	$y(2) = -9.82702e-01$
$t = 0.000436$	$y(0) = 9.99341e-01$	$y(1) = -2.12843e-04$	$y(2) = -9.65556e-01$
$t = 0.001286$	$y(0) = 9.97381e-01$	$y(1) = 4.96717e-05$	$y(2) = -8.99883e-01$
$t = 0.002137$	$y(0) = 9.94389e-01$	$y(1) = 1.34717e-03$	$y(2) = -8.36513e-01$
$t = 0.002987$	$y(0) = 9.90432e-01$	$y(1) = 3.61242e-03$	$y(2) = -7.75437e-01$
$t = 0.004397$	$y(0) = 9.81900e-01$	$y(1) = 9.34551e-03$	$y(2) = -6.79183e-01$
$t = 0.005806$	$y(0) = 9.71188e-01$	$y(1) = 1.72665e-01$	$y(2) = -5.89113e-01$
$t = 0.007216$	$y(0) = 9.58607e-01$	$y(1) = 2.70648e-02$	$y(2) = -5.05108e-01$
$t = 0.008626$	$y(0) = 9.44427e-01$	$y(1) = 3.84696e-02$	$y(2) = -4.27027e-01$
$t = 0.010509$	$y(0) = 9.23405e-01$	$y(1) = 5.7956e-02$	$y(2) = -3.31659e-01$
$t = 0.012393$	$y(0) = 9.00503e-01$	$y(1) = 7.50160e-02$	$y(2) = -2.46140e-01$
$t = 0.014276$	$y(0) = 8.76207e-01$	$y(1) = 9.56443e-02$	$y(2) = -1.69995e-01$
$t = 0.016160$	$y(0) = 8.50950e-01$	$y(1) = 1.17247e-01$	$y(2) = -1.02715e-01$
$t = 0.019069$	$y(0) = 8.10926e-01$	$y(1) = 1.51654e-01$	$y(2) = -1.49844e-02$
$t = 0.021978$	$y(0) = 7.70725e-01$	$y(1) = 2.70725e-01$	$y(2) = 5.48703e-02$
$t = 0.024887$	$y(0) = 7.31274e-01$	$y(1) = 3.12300e-01$	$y(2) = 1.08895e-01$
$t = 0.027797$	$y(0) = 6.93289e-01$	$y(1) = 3.73285e-01$	$y(2) = 1.49844e-02$
$t = 0.030706$	$y(0) = 6.57317e-01$	$y(1) = 4.49087e-01$	$y(2) = 1.08895e-01$
$t = 0.034273$	$y(0) = 6.16518e-01$	$y(1) = 5.26335e-01$	$y(2) = 1.49087e-01$
$t = 0.037840$	$y(0) = 5.79765e-01$	$y(1) = 5.97976e-01$	$y(2) = 1.77348e-01$
$t = 0.041407$	$y(0) = 5.47237e-01$	$y(1) = 6.47348e-01$	$y(2) = 2.07108e-01$
$t = 0.044975$	$y(0) = 5.18920e-01$	$y(1) = 6.98575e-01$	$y(2) = 2.37230e-01$
$t = 0.048542$	$y(0) = 4.94658e-01$	$y(1) = 7.41282e-01$	$y(2) = 2.63933e-01$
$t = 0.052901$	$y(0) = 4.70134e-01$	$y(1) = 7.80558e-01$	$y(2) = 2.90469e-01$
$t = 0.057260$	$y(0) = 4.50670e-01$	$y(1) = 8.11240e-01$	$y(2) = 3.12891e-01$
$t = 0.061619$	$y(0) = 4.35577e-01$	$y(1) = 8.41124e-01$	$y(2) = 3.42891e-01$
$t = 0.065979$	$y(0) = 4.24148e-01$	$y(1) = 8.69413e-01$	$y(2) = 3.70890e-02$
$t = 0.070338$	$y(0) = 4.15701e-01$	$y(1) = 8.95058e-01$	$y(2) = 4.15701e-01$
$t = 0.075490$	$y(0) = 4.08704e-01$	$y(1) = 9.11616e-01$	$y(2) = 4.51161e-01$

$h = 0.01$
 $h_{min} = 0.00001$
 $h_{max} = 0.10$
 $\epsilon_{ps} = 0.00010$

$jjs = 1$

$t = 0.000000$	$y(0) = 1.00000e+00$	$y(1) = 0.00000e+00$	$y(2) = -1.00000e+00$
$t = 0.000218$	$y(0) = 9.99707e-01$	$y(1) = -1.43222e-04$	$y(2) = -9.82702e-01$
$t = 0.000436$	$y(0) = 9.99341e-01$	$y(1) = -2.12843e-04$	$y(2) = -9.65556e-01$
$t = 0.001286$	$y(0) = 9.97381e-01$	$y(1) = 4.96717e-05$	$y(2) = -8.99883e-01$
$t = 0.002137$	$y(0) = 9.94389e-01$	$y(1) = 1.34717e-03$	$y(2) = -8.36513e-01$
$t = 0.002987$	$y(0) = 9.90432e-01$	$y(1) = 3.61242e-03$	$y(2) = -7.75437e-01$
$t = 0.004397$	$y(0) = 9.81900e-01$	$y(1) = 9.34551e-03$	$y(2) = -6.79183e-01$

$t = 0.005806$	$y(0) = 9.71188e-01$	$y(1) = 1.72665e-02$	$y(2) = -5.89113e-01$
$t = 0.007216$	$y(0) = 9.58607e-01$	$y(1) = 2.70648e-02$	$y(2) = -5.05108e-01$
$t = 0.008626$	$y(0) = 9.44427e-01$	$y(1) = 3.84696e-02$	$y(2) = -4.27027e-01$
$t = 0.010509$	$y(0) = 9.23405e-01$	$y(1) = 5.7956e-02$	$y(2) = -3.31659e-01$
$t = 0.012393$	$y(0) = 9.00503e-01$	$y(1) = 7.50160e-02$	$y(2) = -2.46140e-01$
$t = 0.014276$	$y(0) = 8.76207e-01$	$y(1) = 9.56443e-02$	$y(2) = -1.69995e-01$
$t = 0.016160$	$y(0) = 8.50950e-01$	$y(1) = 1.17247e-01$	$y(2) = -1.02715e-01$
$t = 0.019069$	$y(0) = 8.10926e-01$	$y(1) = 1.51654e-01$	$y(2) = -1.49844e-02$
$t = 0.021978$	$y(0) = 7.70725e-01$	$y(1) = 2.01700e-01$	$y(2) = -1.08895e-01$
$t = 0.024887$	$y(0) = 7.31274e-01$	$y(1) = 2.52635e-01$	$y(2) = -1.49087e-01$
$t = 0.027797$	$y(0) = 6.93289e-01$	$y(1) = 3.12300e-01$	$y(2) = -1.77348e-01$
$t = 0.030706$	$y(0) = 6.57317e-01$	$y(1) = 3.73285e-01$	$y(2) = -2.07108e-01$
$t = 0.034273$	$y(0) = 6.16518e-01$	$y(1) = 4.37348e-01$	$y(2) = -2.37230e-01$
$t = 0.037840$	$y(0) = 5.79765e-01$	$y(1) = 4.98575e-01$	$y(2) = -2.63933e-01$
$t = 0.041407$	$y(0) = 5.47237e-01$	$y(1) = 5.52331e-01$	$y(2) = -2.90469e-01$
$t = 0.044975$	$y(0) = 5.18920e-01$	$y(1) = 5.95058e-01$	$y(2) = -3.12891e-01$
$t = 0.048542$	$y(0) = 4.94658e-01$	$y(1) = 6.35623e-01$	$y(2) = -3.42891e-01$
$t = 0.052901$	$y(0) = 4.70134e-01$	$y(1) = 6.73544e-01$	$y(2) = -3.70890e-01$
$t = 0.057260$	$y(0) = 4.50670e-01$	$y(1) = 7.11616e-01$	$y(2) = -4.09910e-01$
$t = 0.061619$	$y(0) = 4.35577e-01$	$y(1) = 7.49727e-01$	$y(2) = -4.47489e-01$
$t = 0.065979$	$y(0) = 4.24148e-01$	$y(1) = 7.87848e-01$	$y(2) = -4.85099e-01$
$t = 0.070338$	$y(0) = 4.15701e-01$	$y(1) = 8.25969e-01$	$y(2) = -5.23609e-01$
$t = 0.075490$	$y(0) = 4.08704e-01$	$y(1) = 8.64090e-01$	$y(2) = -5.61219e-01$

$t = 0.000000$	$y(0) = 1.00000e+00$	$y(1) = 0.00000e+00$	$y(2) = -1.00000e+00$
$t = 0.000069$	$y(0) = 9.99923e-01$	$y(1) = -6.13974e-05$	$y(2) = -9.94497e-01$
$t = 0.000138$	$y(0) = 9.99839e-01$	$y(1) = -1.15256e-04$	$y(2) = -9.89009e-01$
$t = 0.000531$	$y(0) = 9.99241e-01$	$y(1) = -3.02322e-04$	$y(2) = -9.57984e-01$
$t = 0.000924$	$y(0) = 9.98407e-01$	$y(1) = -2.53055e-04$	$y(2) = -9.27454e-01$
$t = 0.001317$	$y(0) = 9.97344e-01$	$y(1) = 2.55855e-05$	$y(2) = -8.97417e-01$
$t = 0.001956$	$y(0) = 9.95144e-01$	$y(1) = 9.51994e-04$	$y(2) = -8.49651e-01$
$t = 0.002595$	$y(0) = 9.92388e-01$	$y(1) = 2.43622e-03$	$y(2) = -8.03183e-01$
$t = 0.003234$	$y(0) = 9.89108e-01$	$y(1) = 4.4563e-03$	$y(2) = -7.58009e-01$
$t = 0.003872$	$y(0) = 9.85335e-01$	$y(1) = 6.95038e-03$	$y(2) = -7.14120e-01$
$t = 0.004810$	$y(0) = 9.78964e-01$	$y(1) = 1.14625e-02$	$y(2) = -6.52041e-01$
$t = 0.005747$	$y(0) = 9.71679e-01$	$y(1) = 1.68931e-02$	$y(2) = -5.92682e-01$
$t = 0.006684$	$y(0) = 9.63562e-01$	$y(1) = 2.31585e-02$	$y(2) = -5.36005e-01$
$t = 0.007621$	$y(0) = 9.54694e-01$	$y(1) = 3.01793e-02$	$y(2) = -4.81972e-01$
$t = 0.009414$	$y(0) = 9.35907e-01$	$y(1) = 4.54416e-02$	$y(2) = -3.85808e-01$
$t = 0.011207$	$y(0) = 9.15141e-01$	$y(1) = 6.26946e-02$	$y(2) = -2.98804e-01$
$t = 0.012999$	$y(0) = 8.92843e-01$	$y(1) = 8.14927e-02$	$y(2) = -2.20566e-01$
$t = 0.014792$	$y(0) = 8.69413e-01$	$y(1) = 1.01436e-01$	$y(2) = -1.50669e-01$
$t = 0.016585$	$y(0) = 8.45210e-01$	$y(1) = 1.22164e-01$	$y(2) = -8.86666e-02$
$t = 0.018378$	$y(0) = 8.20554e-01$	$y(1) = 1.43358e-01$	$y(2) = -3.40991e-02$

$t = 0.020171$	$y(0) = 7.95725e-01$	$y(1) = 1.64737e-01$	$y(2) = 1.35040e-02$
$t = 0.021963$	$y(0) = 7.70970e-01$	$y(1) = 1.86055e-01$	$y(2) = 5.46174e-02$
$t = 0.023756$	$y(0) = 7.46499e-01$	$y(1) = 2.07100e-01$	$y(2) = 8.97148e-02$
$t = 0.025549$	$y(0) = 7.22493e-01$	$y(1) = 2.27693e-01$	$y(2) = 1.19264e-01$
$t = 0.027342$	$y(0) = 6.99104e-01$	$y(1) = 2.47681e-01$	$y(2) = 1.43724e-01$
$t = 0.029134$	$y(0) = 6.76459e-01$	$y(1) = 2.66938e-01$	$y(2) = 1.63540e-01$
$t = 0.030927$	$y(0) = 6.54658e-01$	$y(1) = 2.85362e-01$	$y(2) = 1.79142e-01$
$t = 0.032720$	$y(0) = 6.33784e-01$	$y(1) = 3.02872e-01$	$y(2) = 1.90946e-01$
$t = 0.034513$	$y(0) = 6.13895e-01$	$y(1) = 3.19408e-01$	$y(2) = 1.99344e-01$
$t = 0.037424$	$y(0) = 5.83809e-01$	$y(1) = 3.44076e-01$	$y(2) = 2.06683e-01$

7.17 二阶微分方程边值问题的数值解法

一、功能

用有限差分法求二阶线性微分方程边值问题的数值解。

二、方法说明

设线性二阶微分方程的边值问题为

$$\begin{cases} u(x)y'' + v(x)y' + w(x)y = f(x) \\ y(a) = \alpha \\ y(b) = \beta \end{cases}$$

现要求未知函数 $y(x)$ 在区间 $[a, b]$ 上的 n 个(包括端点 $x=a$ 与 $x=b$)等距离散点上的近似解。其中 n 个等距离散点为:

$$x_i = a + i \cdot h, \quad i = 0, 1, \dots, n - 1$$

$$h = (b - a)/(n - 1)$$

显然, $y(x_0) = y(a) = \alpha$, $y(x_{n-1}) = y(b) = \beta$ 。

用中心差分近似代替 $y''(x_i)$ 与 $y'(x_i)$, 即

$$y'(x_i) \approx \frac{y_{i+1} - y_{i-1}}{2h}$$

$$y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

其中 $y_i = y(x_i)$ 。将它们代入原微分方程得

$$\frac{u(x_i)}{h^2}(y_{i+1} - 2y_i + y_{i-1}) + \frac{v(x_i)}{2h}(y_{i+1} - y_{i-1}) + w(x_i)y_i = f(x_i)$$

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

经整理后可得到如下关于 y_i ($i = 0, 1, \dots, n - 1$) 的方程组

$$\begin{cases} y_0 = \alpha \\ p_i y_{i-1} + q_i y_i + r_i y_{i+1} = d_i, \quad i = 1, 2, \dots, n - 2 \\ y_{n-1} = \beta \end{cases}$$

其中

$$\left. \begin{array}{l} p_i = u(x_i) - \frac{h}{2}v(x_i) \\ q_i = h^2w(x_i) - 2u(x_i) \\ r_i = u(x_i) + \frac{h}{2}v(x_i) \\ d_i = h^2f(x_i) \end{array} \right\} \quad i = 1, 2, \dots, n - 2$$

用矩阵表示为

$$\begin{bmatrix} q_0 & r_0 & & 0 & & & \\ p_1 & q_1 & r_1 & & & & \\ \vdots & \ddots & \ddots & \ddots & & & \\ & & p_{n-2} & q_{n-2} & r_{n-2} & & \\ 0 & & p_{n-1} & q_{n-1} & & y_{n-1} & \\ & & & & & y_{n-2} & \\ & & & & & \vdots & \\ & & & & & y_1 & \\ & & & & & y_0 & \\ & & & & & & d_0 \\ & & & & & & d_1 \\ & & & & & & \vdots \\ & & & & & & d_{n-2} \\ & & & & & & d_{n-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-2} \\ d_{n-1} \end{bmatrix}$$

其中 $q_0 = 1, r_0 = 0, d_0 = \alpha; p_{n-1} = 0, q_{n-1} = 1, d_{n-1} = \beta$ 。

此为三对角线方程组, 可用追赶法求解。

在实际计算时, 为了提高精度, 采用如下方法: 用步长 $h = (b - a)/(n - 1)$ 计算得到 $n - 2$ 个等距离散点上的一组近似解

$$y_i^{(h)} = y_i[a + i \cdot h], \quad i = 1, 2, \dots, n - 2$$

再用步长 $\frac{h}{2} = (b - a)/(2n - 2)$ 计算得到 $n - 2$ 个等距离散点上的另一组近似解

$$y_i^{(\frac{h}{2})} = y_i[a + i \cdot \frac{h}{2}], \quad i = 1, 2, \dots, n - 2$$

最后令

$$y_i = \frac{4y_i^{(\frac{h}{2})} - y_i^{(h)}}{3}, \quad i = 1, 2, \dots, n - 2$$

三、函数语句

void gdfte(a, b, ya, yb, n, y)

本函数要调用追赶法求解三对角线方程组的函数 atrde, 参看 1.5 节。

本函数还要调用计算二阶微分方程中的函数 $u(x), v(x), w(x), f(x)$ 的值的函数 gdftef, 由用户自编, 其形式如下:

```
void gdftef(x, z)
double x, z[4];
| z[0] = u(x)的表达式;
z[1] = v(x)的表达式;
z[2] = w(x)的表达式;
z[3] = f(x)的表达式;
return;
```

四、形参说明

a——双精度实型变量。求解区间的左端点。

b——双精度实型变量。求解区间的右端点。

ya——双精度实型变量。未知函数在求解区间左端点处的函数值 $y(a)$ 。

yb——双精度实型变量。未知函数在求解区间右端点处的函数值 $y(b)$ 。

n——整型变量。求解区间 $[a, b]$ 的等分点数(包括左、右端点 a, b)，即各等分点为

$$x_i = a + i \cdot h, \quad i = 0, 1, \dots, n - 1$$

其中 $h = (b - a)/(n - 1)$ 。

y——双精度实型一维数组，长度为 n。返回 n 个等距离散点上的未知函数值 $y(x_i)$ ($i = 0, 1, \dots, n - 1$)。

五、函数程序(文件名: gdfte.c)

六、例

设二阶微分方程边值问题为

$$\begin{cases} -y'' + \frac{2}{x^2}y = \frac{1}{x} \\ y(2) = 0, \quad y(3) = 0 \end{cases}$$

求解区间为 $[2, 3]$ ，等分数为 $n = 11$ (即步长 $h = 0.1$)。其中 $u(x) = -1, v(x) = 0, w(x)$

$$= \frac{2}{x^2}, f(x) = \frac{1}{x}.$$

主函数程序及计算 $u(x)、v(x)、w(x)、f(x)$ 值的函数程序(文件名 gdfte0.c)如下：

```
# include "stdio.h"
# include "gdfte.c"
# include "atrde.c"
main()
{
    int i;
    double a, b, ya, yb, y[11];
    a = 2.0; b = 3.0; ya = 0.0; yb = 0.0;
    gdfte(a, b, ya, yb, 11, y);
    printf("\n");
    for (i = 0; i <= 10; i++)
        printf("y(%2d) = %e\n", i, y[i]);
    printf("\n");
}
```

```
void gdftef(x, z)
double x, z[4];
{
    z[0] = -1.0; z[1] = 0.0;
    z[2] = 2.0/(x * x); z[3] = 1.0/x;
    return;
}
```

运行结果为：

```
y(0)=0.00000e+00
y(1)=1.86090e-02
y(2)=3.25359e-02
```

y(3)=4.20480e-02
y(4)=4.73684e-02
y(5)=4.86842e-02
y(6)=4.61538e-02
y(7)=3.99123e-02
y(8)=3.00752e-02
y(9)=1.67423e-02
y(10)=0.00000e+00

本问题的解析解为

$$y(x) = \left(19x - 5x^2 - \frac{36}{x} \right) / 38$$

第8章 拟合与逼近

8.1 最小二乘曲线拟合

一、功能

用最小二乘法求给定数据点的拟合多项式。

二、方法说明

设已知 n 个数据点 $(x_i, y_i) (i=0, 1, \dots, n-1)$, 求 $(m-1)$ 次最小二乘拟合多项式

$$P_{m-1}(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{m-1} x^{m-1}$$

其中 $m \leq n$ 且 $m \leq 20$ 。

设拟合多项式为各正交多项式 $Q_j(x) (j=0, 1, \dots, m-1)$ 的线性组合:

$$P_{m-1}(x) = c_0 Q_0(x) + c_1 Q_1(x) + \dots + c_{m-1} Q_{m-1}(x)$$

其中 $Q_j(x)$ 可以由以下递推公式来构造:

$$Q_0(x) = 1$$

$$Q_1(x) = (x - a_1)$$

$$Q_{j+1}(x) = (x - a_{j+1}) Q_j(x) - \beta_j Q_{j-1}(x), j = 1, 2, \dots, m-2$$

若设

$$d_j = \sum_{i=0}^{n-1} Q_j^2(x_i), j = 0, 1, \dots, m-1$$

则

$$\left. \begin{aligned} a_{j+1} &= \frac{1}{d_j} \sum_{i=0}^{n-1} x_i Q_j^2(x_i) \\ \beta_j &= d_j / d_{j-1} \end{aligned} \right\}, j = 0, 1, \dots, m-2$$

可以证明, 由上述递推构造的多项式函数组 $\{Q_j(x)\} (j=0, 1, \dots, m-1)$ 是互相正交的。根据最小二乘原理, 可得

$$c_j = \frac{1}{d_j} \sum_{i=0}^{n-1} y_i Q_j(x_i), j = 0, 1, \dots, m-1$$

最后可以化成一般的 $m-1$ 次多项式

$$P_{m-1}(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{m-1} x^{m-1}$$

由于拟合多项式的次数越高, 其拟合精度未必越高, 因此, 在本函数中允许的拟合次数最高为 19(即 m 的最大值为 20)。

具体计算步骤如下。

(1) $Q_0(x) = 1$, 可得

$$b_0 = 1, d_0 = n, c_0 = \frac{1}{d_0} \sum_{i=0}^{n-1} y_i, \alpha = \frac{1}{d_1} \sum_{i=0}^{n-1} x_i$$

$$a_0 = c_0 b_0$$

(2) $Q_1(x) = (x - \alpha)$, 可得

$$t_0 = -\alpha, t_1 = 1$$

$$d_1 = \sum_{i=0}^{n-1} Q_1^2(x_i), c_1 = \frac{1}{d_1} \sum_{i=0}^{n-1} y_i Q_1(x_i)$$

$$\alpha = \frac{1}{d_1} \sum_{i=0}^{n-1} x_i Q_1^2(x_i), \beta = d_1 / d_0$$

$$a_0 = a_0 + c_1 t_0, a_1 = c_1 t_1$$

(3) 对于 $j = 2, 3, \dots, m-1$ 做以下各步:

$$\begin{aligned} Q_j(x) &= (x - \alpha) Q_{j-1}(x) - \beta Q_{j-2}(x) \\ &= (x - \alpha)(t_{j-1} x^{j-1} + \dots + t_1 x + t_0) \\ &\quad - \beta(b_{j-2} x^{j-2} + \dots + b_1 x + b_0) \\ &\stackrel{\text{令}}{=} s_j x^j + s_{j-1} x^{j-1} + \dots + s_1 x + s_0 \end{aligned}$$

其中 $s_i (i=0, 1, \dots, j)$ 由以下递推公式计算:

$$\left\{ \begin{array}{l} s_j = t_{j-1} \\ s_{j-1} = -\alpha t_{j-1} + t_{j-2} \\ s_k = -\alpha t_k + t_{k-1} - \beta b_k, k = j-2, \dots, 1 \\ s_0 = -\alpha t_0 - \beta t_1 \end{array} \right.$$

再计算

$$d_j = \sum_{i=0}^{n-1} Q_j^2(x_i)$$

$$c_j = \sum_{i=0}^{n-1} y_i Q_j(x_i) / d_j$$

$$\alpha = \sum_{i=0}^{n-1} x_i Q_j^2(x_i) / d_j$$

$$\beta = d_j / d_{j-1}$$

由此可以计算相应的 a_i :

$$\left\{ \begin{array}{l} a_j = c_j s_j \\ a_k = a_k + c_j s_k, k = j-1, \dots, 1, 0 \end{array} \right.$$

且

$$\left\{ \begin{array}{l} t_j = s_j \\ b_k = t_k, t_k = s_k, k = j-1, \dots, 1, 0 \end{array} \right.$$

在实际计算过程中, 为了防止运算溢出, x_i 用

$$x_i^* = x_i - \bar{x}, i = 0, 1, \dots, n-1$$

代替, 其中

$$\bar{x} = \sum_{i=0}^{n-1} x_i / n$$

此时,拟合多项式的形式为

$$P_{m-1}(x) = a_0 + a_1(x - \bar{x}) + a_2(x - \bar{x})^2 + \cdots + a_{m-1}(x - \bar{x})^{m-1}$$

三、函数语句

```
void hpir1 (x, y, n, a, m, dt)
```

四、形参说明

x——双精度实型一维数组,长度为n。存放给定n个数据点的X坐标。

y——双精度实型一维数组,长度为n。存放给定n个数据点的Y坐标。

n——整型变量。给定数据点的个数。

a——双精度实型一维数组,长度为m。返回m-1次拟合多项式的m个系数。

m——整型变量。拟合多项式的项数,即拟合多项式的最高次数为m-1。要求m≤n且m≤20。若m>n或m>20,则本函数自动按m=min{n,20}处理。

dt——双精度实型一维数组,长度为3。其中:dt(0)返回拟合多项式与数据点误差的平方和;dt(1)返回拟合多项式与数据点误差的绝对值之和;dt(2)返回拟合多项式与数据点误差绝对值的最大值。

五、函数程序(文件名:hpir1.c)

六、例

设给定函数

$$f(x) = x - e^{-x}$$

从x₀=0开始,取步长h=0.1的20个数据点,求五次最小二乘拟合多项式

$$P_5(x) = a_0 + a_1(x - \bar{x}) + a_2(x - \bar{x})^2 + \cdots + a_5(x - \bar{x})^5$$

其中

$$\bar{x} = \frac{1}{20} \sum_{i=0}^{19} x_i = 0.95$$

主函数程序(文件名:hpir10.c)如下:

```
#include "stdio.h"
#include "hpir1.c"
#include "math.h"
main()
{ int i;
  double x[20], y[20], a[6], dt[3];
  for (i = 0; i < 20; i++)
    { x[i] = 0.1 * i;
      y[i] = x[i] - exp(-x[i]);
    }
  hpir1(x, y, 20, a, 6, dt);
  printf("\n");
```

```
for (i = 0; i < 5; i++)
  printf("a(%2d) = %e\n", i, a[i]);
printf("\n");
for (i = 0; i < 2; i++)
  printf("dt(%2d) = %e\n", i, dt[i]);
printf("\n");
```

运行结果为:

```
a(0) = 5.63248e-01
a(1) = 1.38675e+00
a(2) = -1.93134e-01
a(3) = 6.44035e-02
a(4) = -1.68412e-02
a(5) = 3.34429e-03
```

```
dt(0) = 1.80174e-09  dt(1) = 1.68505e-04  dt(2) = 1.53940e-05
```

8.2 切比雪夫曲线拟合

一、功能

给定n个数据点,求切比雪夫(Chebyshev)意义下的最佳拟合多项式。

二、方法说明

设给定n个数据点

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

其中x₀<x₁<…<x_{n-1}。求m-1次(m<n且m≤20)多项式

$$P_{m-1}(x) = a_0 + a_1x + \cdots + a_{m-1}x^{m-1}$$

使得在n个给定点上的偏差最大值为最小,即

$$\max_{0 \leq i \leq n-1} |P_{m-1}(x_i) - y_i| = \min$$

其计算步骤如下:

从给定的n个点中选取m+1个不同点u₀, u₁, …, u_m组成初始参考点集。

设在初始点集u₀, u₁, …, u_m上,参考多项式Φ(x)的偏差为h,即参考多项式Φ(x)在初始点集上的取值为

$$\Phi(u_i) = f(u_i) + (-1)^i h, \quad i = 0, 1, \dots, m$$

且Φ(u_i)的各阶差商是h的线性函数。

由于Φ(x)为m-1次多项式,其m阶差商等于零,由此可以求出h。再根据Φ(u_i)的各阶差商,由牛顿插值公式可求出Φ(x):

$$\Phi(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}x^{m-1}$$

令

$$hh = \max_{0 \leq i \leq n-1} |\Phi(x_i) - y_i|$$

若 $hh = h$, 则 $\Phi(x)$ 即为所求的拟合多项式。

若 $hh > h$, 则用达到偏差最大值的点 x_j 代替点集 $\{u_i\} (i = 0, 1, \dots, m)$ 中离 x_j 最近且具有与

$$\Phi(x_i) = y_i$$

的符号相同的点, 从而构造一个新的参考点集。用这个新的参考点集重复以上过程, 直到最大逼近误差等于参考偏差为止。

三、函数语句

```
void hchir(x, y, n, a, m)
```

四、形参说明

x —双精度实型一维数组, 长度为 n 。存放 n 个数据点的 X 坐标。

y —双精度实型一维数组, 长度为 n 。存放 n 个数据点的 Y 坐标。

n —整型变量。给定数据点的个数。

a —双精度实型一维数组, 长度为 $m + 1$ 。其中前 m 个元素($a(0)$ 到 $a(m - 1)$)返回 $m - 1$ 次拟合多项式

$$P_{m-1}(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}$$

的系数 $a_i (i = 0, 1, \dots, m - 1)$; 最后一个元素 $a(m)$ 返回拟合多项式 $P_{m-1}(x)$ 的偏差最大值。若 $a(m)$ 为负值, 说明在迭代过程中参考偏差不再增大, 其绝对值为当前选择的参考偏差。

m —整型变量。拟合多项式的项数, 即拟合多项式的最高次数为 $m - 1$ 。要求 $m < n$ 且 $m \leq 20$ 。若 $m \geq n$ 或 $m > 20$, 则在本函数中自动取 $m = \min\{20, n - 1\}$ 。

五、函数程序(文件名:hchir.c)

六、例

取函数 $f(x) = \arctan x$ 在区间 $[-1, 1]$ 上的 101 个点

$$x_i = -1.0 + i \cdot 0.02, i = 0, 1, \dots, 100$$

上的函数值 $y_i = f(x_i) (i = 0, 1, \dots, 100)$ 。根据此 101 个数据点构造切比雪夫意义下的五次拟合多项式

$$P_5(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$$

主函数程序(文件名:hchir 0.c)如下:

```
#include "math.h"
#include "stdio.h"
#include "hchir.c"

main()
{
    int i;
    double x[101], y[101], a[7];
    for (i = 0; i <= 100; i++)
        . . .
}
```

```
{   x[i] = -1.0 + 0.02 * i;
    y[i] = atan(x[i]);
}
hchir(x, y, 101, a, 6);
printf("\n");
for (i = 0; i <= 5; i++)
    printf("a(%2d) = %e\n", i, a[i]);
printf("\n");
printf("MAX(p-f) = %e\n", a[6]);
printf("\n");
}
```

运行结果为:

```
a(0) = -1.11022e-16
a(1) = 9.95364e-01
a(2) = 1.66533e-16
a(3) = -2.88716e-01
a(4) = -1.66533e-16
a(5) = 7.93575e-02
```

```
MAX(p-f) = 6.07072e-04
```

8.3 最佳一致逼近的里米兹方法

一、功能

用里米兹(Remez)方法求给定函数的最佳一致逼近多项式。

二、方法说明

若函数 $f(x)$ 在区间 $[a, b]$ 上的 $n - 1$ 次最佳一致逼近多项式为

$$P_{n-1}(x) = p_0 + p_1x + p_2x^2 + \dots + p_{n-1}x^{n-1}$$

则存在 $n + 1$ 个点的交错点组 $\{x_i\}$ 满足

$$f(x_i) - P_{n-1}(x_i) = (-1)^i \mu, \quad i = 0, 1, \dots, n$$

或

$$f(x_i) - P_{n-1}(x_i) = (-1)^{i+1} \mu, \quad i = 0, 1, \dots, n$$

其中

$$\mu = \max_{x \in [a, b]} |f(x) - P_{n-1}(x)|$$

求函数 $f(x)$ 在区间 $[a, b]$ 上的 $n - 1$ 次最佳一致逼近多项式

$$P_{n-1}(x) = p_0 + p_1x + p_2x^2 + \dots + p_{n-1}x^{n-1}$$

的里米兹方法如下。

1. 在区间 $[a, b]$ 上取 n 次切比雪夫(Chebyshev)多项式的交错点组

$$x_k = \frac{1}{2} \left[b + a + (b - a) \cos \frac{n - k}{n} \pi \right], \quad k = 0, 1, \dots, n$$

作为初始参考点集。

2. 以参考点集 $\{x_i\}$ ($i = 0, 1, \dots, n$) 构造一个参考多项式

$$P_{n-1}(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_{n-1} x^{n-1}$$

满足

$$P_{n-1}(x_i) - f(x_i) = (-1)^i \mu, \quad i = 0, 1, \dots, n$$

由于 $P_{n-1}(x)$ 为 $n-1$ 次多项式, 在 $(n+1)$ 个点 $\{x_i\}$ ($i = 0, 1, \dots, n$) 上的 n 阶差商为零, 由此可以确定出参考偏差 μ 。

然后根据 $P_{n-1}(x)$ 在 $(n+1)$ 个点 $\{x_i\}$ ($i = 0, 1, \dots, n$) 上的各阶差商, 利用牛顿插值公式确定出参考多项式 $P_{n-1}(x)$ 的各系数 p_0, p_1, \dots, p_{n-1} 。

3. 找出使函数 $|f(x) - P_{n-1}(x)|$ 在区间 $[a, b]$ 上取最大值的点 x^* , 并按如下原则替换原参考点集 $\{x_i\}$ ($i = 0, 1, \dots, n$) 中的某一点:

(1) 若 x^* 在 a 与 x_0 之间, 且 $f(x_0) - P_{n-1}(x_0)$ 与 $f(x^*) - P_{n-1}(x^*)$ 同号, 则将 x^* 代替 x_0 , 构成新点集

$$\{x^*, x_1, \dots, x_n\}$$

否则新点集为

$$\{x^*, x_0, x_1, \dots, x_{n-1}\}$$

(2) 若 x^* 在 x_n 与 b 之间, 且 $f(x_n) - P_{n-1}(x_n)$ 与 $f(x^*) - P_{n-1}(x^*)$ 同号, 则将 x^* 代替 x_n , 构成新点集

$$\{x_0, x_1, \dots, x_{n-1}, x^*\}$$

否则取新点集 $\{x_1, x_2, \dots, x_n, x^*\}$

(3) 若 x^* 在 x_i 与 x_{i+1} ($i = 0, 1, \dots, n-1$) 之间, 且 $f(x_i) - P_{n-1}(x_i)$ 与 $f(x^*) - P_{n-1}(x^*)$ 同号, 则将 x^* 替换 x_i , 否则将 x^* 替换 x_{i+1} , 构成新点集。

重复 2 与 3, 直到相邻两次求得的参考偏差接近相等为止。此时, 最后获得的参考多项式 $P_{n-1}(x)$ 即为近似的 $n-1$ 次最佳一致逼近多项式。

三、函数语句

```
void hremz(a, b, p, n, eps)
```

本函数要调用计算函数 $f(x)$ 值的函数 $hremzf$, 由用户自编, 其形式如下:

```
double hremzf(x)
double x;
double y;
y = f(x)的表达式;
return(y);
|
```

四、形参说明

a —双精度实型变量。区间左端点值。

b —双精度实型变量。区间右端点值。

p —双精度实型一维数组, 长度为 $n+1$ 。其中前 n 个元素返回 $n-1$ 次最佳一致逼近多项式 $P_{n-1}(x)$ 的 n 个系数 p_0, p_1, \dots, p_{n-1} ; 最后一个元素 $p(n)$ 返回 $P_{n-1}(x)$ 的偏差绝对值 μ 。

n —整型变量。 $n-1$ 次最佳一致逼近多项式的项数。要求 $n \leq 20$ 。若 $n > 20$, 本函数自动取 $n = 20$ 。

eps —双精度实型变量。控制精度要求。一般取 $10^{-10} \sim 10^{-35}$ 之间的数。

五、函数程序(文件名:hremz.c)

六、例

求函数 $f(x) = e^x$ 在区间 $[-1, 1]$ 上的三次最佳一致逼近多项式及其偏差的绝对值。

其中 $a = -1.0, b = 1.0, n = 4$ 。取 $eps = 10^{-10}$ 。

主函数程序及计算函数 $f(x)$ 值的函数程序(文件名:hremz 0.c)如下:

```
# include "math.h"
# include "stdio.h"
# include "hremz.c"
main()
| int i;
double a, b, eps, p[5];
a = -1.0; b = 1.0; eps = 1.0e-10;
hremz(a, b, p, 4, eps);
printf("\n");
for (i = 0; i <= 3; i++)
    printf("p[%d] = %e\n", i, p[i]);
printf("\n");
printf("MAX(p-f) = %e\n", p[4]);
printf("\n");
|
```

double hremzf(x)

```
double x;
double y;
y = exp(x);
return(y);
|
```

运行结果为:

```
p(0) = 9.94594e-01
p(1) = 9.95683e-01
p(2) = 5.42974e-01
p(3) = 1.79518e-01
```

MAX(p-f) = 5.51304e-0.3

8.4 矩形域的最小二乘曲面拟合

一、功能

用最小二乘法求矩形域上 $n \times m$ 个数据点的拟合曲面。

二、方法说明

设已知矩形区域内 $n \times m$ 个网点 (x_i, y_j) ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$) 上的函数值 z_{ij} , 求最小二乘拟合多项式

$$f(x, y) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} a_{ij} x^i y^j$$

首先, 固定 y , 对 x 构造 m 个最小二乘拟合多项式

$$g_j(x) = \sum_{k=0}^{p-1} \lambda_{kj} \varphi_k(x), j = 0, 1, \dots, m-1$$

其中各 $\varphi_k(x)$ ($k = 0, 1, \dots, p-1$) 为互相正交的多项式, 并由以下递推公式构造:

$$\varphi_0(x) = 1$$

$$\varphi_1(x) = x - \alpha_0$$

$$\varphi_{k+1}(x) = (x - \alpha_k) \varphi_k(x) - \beta_k \varphi_{k-1}(x), k = 1, 2, \dots, p-1$$

若令

$$d_k = \sum_{i=0}^{n-1} \varphi_k^2(x_i), k = 0, 1, \dots, p-1$$

则有

$$\alpha_k = \sum_{i=0}^{n-1} x_i \varphi_k^2(x_i) / d_k, k = 0, 1, \dots, p-1$$

$$\beta_k = d_k / d_{k-1}, k = 1, 2, \dots, p-1$$

根据最小二乘原理可得

$$\lambda_{kj} = \sum_{i=0}^{n-1} z_{ij} \varphi_k(x_i) / d_k, j = 0, 1, \dots, m-1$$
$$k = 0, 1, \dots, p-1$$

然后, 再构造 y 的最小二乘拟合多项式

$$h_k(y) = \sum_{l=0}^{q-1} \mu_{kl} \psi_l(y), k = 0, 1, \dots, p-1$$

其中各 $\psi_l(y)$ ($l = 0, 1, \dots, q-1$) 也为互相正交的多项式, 并由以下递推公式构造:

$$\psi_0(y) = 1$$

$$\psi_1(y) = y - \alpha'_0$$

$$\psi_{l+1}(y) = (y - \alpha'_l) \psi_l(y) - \beta'_l \psi_{l-1}(y), l = 1, 2, \dots, q-1$$

若令

$$\delta_l = \sum_{j=0}^{m-1} \psi_l^2(y_j), l = 0, 1, \dots, q-1$$

则有

$$\alpha'_l = \sum_{j=0}^{m-1} y_j \psi_l^2(y_j) / \delta_l, l = 0, 1, \dots, q-1$$

$$\beta'_l = \delta_l / \delta_{l-1}, l = 1, 2, \dots, q-1$$

根据最小二乘原理可得

$$\mu_{kl} = \sum_{j=0}^{m-1} \lambda_{kj} \psi_l(y_j) / \delta_l, k = 0, 1, \dots, p-1$$
$$l = 0, 1, \dots, q-1$$

最后可得二元函数的拟合多项式为

$$f(x, y) = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} \mu_{kl} \varphi_k(x) \psi_l(y)$$

再转换成标准的多项式形式

$$f(x, y) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} a_{ij} x^i y^j$$

在实际计算过程中, 为了防止运算溢出, x_i 与 y_j 分别用

$$x_i^* = x_i - \bar{x}, i = 0, 1, \dots, n-1$$

$$y_j^* = y_j - \bar{y}, j = 0, 1, \dots, m-1$$

代替。其中

$$\bar{x} = \sum_{i=0}^{n-1} x_i / n, \bar{y} = \sum_{j=0}^{m-1} y_j / m$$

此时, 二元拟合多项式的形式为

$$f(x, y) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} a_{ij} (x - \bar{x})^i (y - \bar{y})^j$$

三、函数语句

void hpir2(x, y, z, n, m, a, p, q, dt)

四、形参说明

x——双精度实型一维数组, 长度为 n 。存放给定数据点的 n 个 X 坐标。

y——双精度实型一维数组, 长度为 m 。存放给定数据点的 m 个 Y 坐标。

z——双精度实型二维数组, 体积为 $n \times m$ 。存放矩形区域内 $n \times m$ 个网点 (x_i, y_j) ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$) 上的函数值 z_{ij} 。

n——整型变量。 X 坐标个数。

m——整型变量。 Y 坐标个数。

a——双精度实型二维数组, 体积为 $p \times q$ 。返回二元拟合多项式

$$f(x, y) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} a_{ij} (x - \bar{x})^i (y - \bar{y})^j$$

的各系数 a_{ij} ($i = 0, 1, \dots, p-1; j = 0, 1, \dots, q-1$)。

p ——整型变量。拟合多项式中 x 的最高次数加 1, 即 x 的最高次为 $p-1$ 。要求 $p \leq n$ 且 $p \leq 20$, 若不满足这个条件, 在本函数中自动取 $p = \min\{n, 20\}$ 。

q ——整型变量。拟合多项式中 y 的最高次数加 1, 即 y 的最高次为 $q-1$ 。要求 $q \leq m$ 且 $q \leq 20$, 若不满足这个条件, 在本函数中自动取 $q = \min\{m, 20\}$ 。

dt ——双精度实型一维数组, 长度为 3。其中: $dt(0)$ 返回拟合多项式与数据点误差的平方和, 即

$$dt(0) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} [f(x_i, y_j) - z_{ij}]^2$$

$dt(1)$ 返回拟合多项式与数据点误差的绝对值之和, 即

$$dt(1) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} |f(x_i, y_j) - z_{ij}|$$

$dt(2)$ 返回拟合多项式与数据点误差绝对值的最大值, 即

$$dt(2) = \max_{\substack{0 \leq i \leq n-1 \\ 0 \leq j \leq m-1}} |f(x_i, y_j) - z_{ij}|$$

五、函数程序(文件名:hpir2.c)

六、例

设二元函数为 $z(x, y) = e^{x^2 - y^2}$, 取矩形区域内 11×21 个网点

$$\begin{cases} x_i = 0.2i, & i = 0, 1, \dots, 10 \\ y_j = 0.1j, & j = 0, 1, \dots, 20 \end{cases}$$

上的函数值 $z(x_i, y_j)$ ($i = 0, 1, \dots, 10; j = 0, 1, \dots, 20$)。由这些数据点构造一个最小二乘拟合多项式

$$f(x, y) = \sum_{i=0}^5 \sum_{j=0}^4 a_{ij}(x - \bar{x})^i(y - \bar{y})^j$$

其中

$$\bar{x} = \sum_{i=0}^{10} x_i / 11 = 1.0$$

$$\bar{y} = \sum_{j=0}^{20} y_j / 21 = 1.0$$

并分别计算此拟合多项式与各网点上的函数值之差的平方之和 $dt(0)$ 、绝对值之和 $dt(1)$ 及绝对值最大者 $dt(2)$ 。

主函数程序(文件名:hpir20.c)如下:

```
#include "math.h"
#include "stdio.h"
#include "hpir2.c"

main()
{
    int i, j;
    double x[11], y[21], z[11][21], a[6][5], dt[3];
    for (i = 0; i < 10; i++) x[i] = 0.2 * i;
```

```
for (i = 0; i <= 20; i++) y[i] = 0.1 * i;
for (i = 0; i <= 10; i++)
    for (j = 0; j <= 20; j++)
        z[i][j] = exp(x[i] * x[i] - y[j] * y[j]);
hpir2(x, y, z, 11, 21, a, 6, 5, dt);
printf("\n");
printf("MAT A(i, j) IS: \n");
for (i = 0; i <= 5; i++)
    for (j = 0; j <= 4; j++)
        printf("%e ", a[i][j]);
    printf("\n");
    printf("\n");
for (i = 0; i <= 2; i++)
    printf("dt(%d) = %e ", i, dt[i]);
printf("\n");
```

运行结果为:

```
MAT A(i, j) IS:
1.11924e+00 -2.22009e+00 9.97902e-01 7.38116e-01 -5.81947e-01
2.25202e+00 -4.46704e+00 2.00788e+00 1.48516e+00 -1.17094e+00
9.16373e-01 -1.81769e+00 8.17030e-01 6.04331e=01 -4.76468e-01
1.34836e+00 -2.67457e+00 1.20219e+00 8.89219e-01 -7.01080e-01
8.19701e+00 -1.62593e+01 7.30837e+00 5.40577e+00 -4.26203e+00
6.30257e+00 -1.25016e+01 5.61931e+00 4.15642e+00 -3.27702e+00
```

```
dt(0) = 5.80139e+00 dt(1) = 2.53761e+01 dt(2) = 5.55972e-01
```

第9章 数据处理与回归分析

9.1 随机样本分析

一、功能

根据给定的一维随机样本，要求：

- (1) 计算算术平均值、方差及标准差。
- (2) 按高斯分布计算出在各给定区间上近似的理论样本点数。
- (3) 打印经验直方图。

二、方法说明

设给定随机变量 x 的 n 个样本点值 $x_i (i = 0, 1, \dots, n-1)$ 。

(1) 计算样本参数值

$$\text{随机样本算术平均值 } \bar{x} = \sum_{i=0}^{n-1} x_i / n$$

$$\text{样本的方差 } s = \sum_{i=0}^{n-1} (x_i - \bar{x})^2 / n$$

$$\text{样本标准差 } t = \sqrt{s}$$

(2) 按高斯分布计算出给定各区间的近似理论样本点数

设随机变量 x 的起始值为 x_0 , 区间长度为 h , 则第 i 个区间的中点为

$$x_i^* = x_0 + (i - 0.5)h, \quad i = 1, 2, \dots$$

在第 i 个区间上, 按高斯分布所应有的近似理论样本点数为

$$F_i = \frac{n}{\sqrt{2\pi}s} \exp\left(-\frac{(x_i^* - \bar{x})^2}{2s}\right) \cdot h$$

(3) 打印经验直方图

在直方图上方打印样本点数 n , 直方图中随机变量的起始值 x_0 、等区间长度值 h 、区间总数 m , 随机变量样本的算术平均值 xa 、方差 s 及标准差 t 。

在打印的直方图中: 左起第一列为从小到大打印各区间的中点值; 第二列打印随机样本中落在对应区间中的实际点数;

右边是直方图本身。各区间对应行上的符号“X”的个数代表样本中随机变量值落在该区间中的点数, 而“*”号所占的序数则为按高斯分布计算得到的近似理论点数。

在直方图的下方打印直方图的比例 k , 即在直方图中, 每一个符号表示 k 个点。

三、函数语句

```
void irhis(x, n, x0, h, m, l, dt, g, q)
```

四、形参说明

x ——双精度实型一维数组, 长度为 n 。存放随机变量的 n 个样本点值。

n ——整型变量。随机样本点数。

$x0$ ——双精度实型变量。直方图中随机变量的起始值。

h ——双精度实型变量。直方图中随机变量等区间长度值。

m ——整型变量。直方图中区间总数。

l ——整型变量。输入标志。若 $l = 0$, 则不需打印直方图; 若 $l \neq 0$, 则需打印直方图。

dt ——双精度实型一维数组, 长度为 3。其中 $dt(0)$ 返回随机样本的算术平均值; $dt(1)$ 返回随机样本的方差 s ; $dt(2)$ 返回随机样本的标准差。

g ——整型一维数组, 长度为 m 。返回 m 个区间的按高斯分布所应有的近似理论样本点数。

q ——整型一维数组, 长度为 m 。返回落在 m 个区间中的每一个区间上的随机样本实际点数。

五、函数程序(文件名:irhis.c)

六、例

给定随机变量的 100 个样本点(参看主函数程序), 打印出直方图。其中 $x0 = 192.000$, $h = 2.00000$, $m = 10$, $l \neq 0$ 。

主函数程序(文件名:irhis 0.c)如下:

```
#include "irhis.c"
main()
{
    int g[10], q[10];
    double dt[3], x0, h;
    static double x[100] = {
        193.199, 195.673, 195.757, 196.051, 196.092, 196.596,
        196.579, 196.763, 196.847, 197.267, 197.392, 197.477,
        198.189, 193.850, 198.944, 199.070, 199.111, 199.153,
        199.237, 199.698, 199.572, 199.614, 199.824, 199.908,
        200.188, 200.160, 200.243, 200.285, 200.453, 200.704,
        200.746, 200.830, 200.872, 200.914, 200.956, 200.998,
        200.998, 201.123, 201.208, 201.333, 201.375, 201.543,
        201.543, 201.584, 201.711, 201.878, 201.919, 202.004,
        202.004, 202.088, 202.172, 202.172, 202.297, 202.339,
        202.381, 202.507, 202.591, 202.716, 202.633, 202.884,
        203.051, 203.052, 203.094, 203.094, 203.177, 203.178,
        203.219, 203.764, 203.765, 203.848, 203.890, 203.974,
        204.184, 204.267, 204.352, 204.352, 204.729, 205.106,
        205.148, 205.231, 205.357, 205.400, 205.483, 206.070,
        206.112, 206.154, 206.155, 206.615, 206.657, 206.993
    };
}
```

```

207.243, 207.621, 208.124, 208.375, 208.502, 208.628,
208.670, 208.711, 210.012, 211.394};

x0 = 192.0; h = 2.0;
irhis(x, 100, x0, h, 10, dt, g, q);
}

```

运行结果为：

n = 100

x0 = 1.92000e + 02 h = 2.00000e + 00 n = 10

```

xa = 2.02230e + 02 s = 1.29867e + 01 t = 3.60371e + 00

1.93000e + 02      2  xx
1.95000e + 02      2  xx *
1.97000e + 02      9  xxxxxxxx *
1.99000e + 02      11  xxxxxxxxxxxx *
2.01000e + 02      23  xxxxxxxxxxxxxxxxxxxxx *
2.03000e + 02      25  xxxxxxxxxxxxxxxxxxxxxx *
2.05000e + 02      11  xxxxxxxxxxxx *
2.07000e + 02      9  xxxxxxxxx *
2.09000e + 02      6  xxx * xx
2.11000e + 02      2  x *

```

k = 1

9.2 一元线性回归分析

一、功能

给定 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$)，用直线 $y = ax + b$ 作回归分析。

二、方法说明

设随机变量 y 随自变量 x 变化。给定 n 组观测数据 (x_i, y_i) , $i = 0, 1, \dots, n-1$ ，用直线 $y = ax + b$ 作回归分析。其中 a, b 为回归系数。

为确定回归系数 a, b 通常采用最小二乘法，即要使

$$Q = \sum_{i=0}^{n-1} [y_i - (ax_i + b)]^2$$

达到最小。根据极值原理， a 与 b 满足下列方程

$$\frac{\partial Q}{\partial a} = 2 \sum_{i=0}^{n-1} [y_i - (ax_i + b)](-x_i) = 0$$

$$\frac{\partial Q}{\partial b} = 2 \sum_{i=0}^{n-1} [y_i - (ax_i + b)](-1) = 0$$

从而解得

$$a = \frac{\sum_{i=0}^{n-1} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=0}^{n-1} (x_i - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

其中

$$\bar{x} = \sum_{i=0}^{n-1} x_i / n, \quad \bar{y} = \sum_{i=0}^{n-1} y_i / n$$

最后可以计算出以下几个量：

$$(1) \text{ 偏差平方和 } Q = \sum_{i=0}^{n-1} [y_i - (ax_i + b)]^2$$

$$(2) \text{ 平均标准偏差 } s = \sqrt{Q/n}$$

$$(3) \text{ 回归平方和 } p = \sum_{i=0}^{n-1} [(ax_i + b) - \bar{y}]^2$$

$$(4) \text{ 最大偏差 } umax = \max_{0 \leq i \leq n-1} |y_i - (ax_i + b)|$$

$$(5) \text{ 最小偏差 } umin = \min_{0 \leq i \leq n-1} |y_i - (ax_i + b)|$$

$$(6) \text{ 偏差平均值 } u = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - (ax_i + b)|$$

三、函数语句

void isqt1(x, y, n, a, dt)

四、形参说明

x——双精度实型一维数组，长度为 n 。存放自变量 x 的 n 个取值。

y——双精度实型一维数组，长度为 n 。存放与自变量 x 的 n 个取值相对应的随机变量 y 的观测值。

n——整型变量。观测点数。

a——双精度实型一维数组，长度为 2。其中 $a(0)$ 返回回归系数 b , $a(1)$ 返回回归系数 a 。

dt——双精度实型一维数组，长度为 6。其中 $dt(0)$ 返回偏差平方和 Q , $dt(1)$ 返回平均标准偏差 s , $dt(2)$ 返回回归平方和 p , $dt(3)$ 返回最大偏差 $umax$, $dt(4)$ 返回最小偏差 $umin$, $dt(5)$ 返回偏差平均值 u 。

五、函数程序(文件名:isqt1.c)

六、例

给定 11 个观测值如下：

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	2.75	2.84	2.965	3.01	3.20	3.25	3.38	3.43	3.55	3.66	3.74

求回归系数 a 与 b 、偏差平方和 q 、平均标准偏差 s 、回归平方和 p 、偏差最大值 $umax$ 、偏差最小值 $umin$ 、偏差平均值 u 。

主函数程序(文件名:isqt10.c)如下:

```
#include "stdio.h"
#include "isqt1.c"
main()
{ int i;
double dt[6], a[2];
static double x[11] = { 0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
                      0.6, 0.7, 0.8, 0.9, 1.0 };
static double y[11] = { 2.75, 2.84, 2.965, 3.01, 3.20,
                      3.25, 3.38, 3.43, 3.55, 3.66, 3.74 };
isqt1(x, y, 11, a, dt);
printf("\n");
printf("a = %e    b = %e\n", a[1], a[0]);
printf("\n");
printf("q = %e  s = %e  p = %e\n", dt[0], dt[1], dt[2]);
printf("\n");
printf("u max = %e u min = %e  u = %e\n", dt[3], dt[4], dt[5]);
printf("\n");
}
```

运行结果为:

```
a = 1.00045e+00  b = 2.75205e+00
q = 5.86795e-03  s = 2.30965e-02  p = 1.10100e+00
umax = 4.77727e-02  umin = 2.04545e-03  u = 1.74298e-02
```

9.3 多元线性回归分析

一、功能

根据随机变量 y 及自变量 x_0, x_1, \dots, x_{m-1} 的 n 组观测值 $(x_{0i}, x_{1i}, \dots, x_{(m-1)i}, y_i)$ ($i = 0, 1, \dots, n-1$) 作线性回归分析。

二、方法说明

设随机变量 y 及 m 个自变量 x_0, x_1, \dots, x_{m-1} 。给定 n 组观测数据 $(x_{0i}, x_{1i}, \dots, x_{(m-1)i}, y_i)$ ($i = 0, 1, \dots, n-1$)，用线性表达式

$$y = a_0 x_0 + a_1 x_1 + \dots + a_{m-1} x_{m-1} + a_m$$

对观测数据进行回归分析。其中 $a_0, a_1, \dots, a_{m-1}, a_m$ 为回归系数。

与一元线性回归分析一样(见 10.2 节的方法说明)，根据最小二乘原理，为使

$$q = \sum_{i=0}^{n-1} [y_i - (a_0 x_{0i} + a_1 x_{1i} + \dots + a_{m-1} x_{(m-1)i} + a_m)]^2$$

达到最小，回归系数 $a_0, a_1, \dots, a_{m-1}, a_m$ 应满足下列方程组

$$(CC^T) \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix} = C \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix}$$

其中

$$C = \begin{bmatrix} x_{00} & x_{01} & x_{02} & \cdots & x_{0, n-1} \\ x_{10} & x_{11} & x_{12} & \cdots & x_{1, n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m-1, 0} & x_{m-1, 1} & x_{m-1, 2} & \cdots & x_{m-1, n-1} \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

采用乔里斯基(Cholesky)分解法解出回归系数。

为了衡量回归效果，还要计算以下五个量。

$$(1) \text{ 偏差平方和 } q = \sum_{i=0}^{n-1} [y_i - (a_0 x_{0i} + a_1 x_{1i} + \dots + a_{m-1} x_{(m-1)i} + a_m)]^2$$

$$(2) \text{ 平均标准偏差 } s = \sqrt{q/n}$$

$$(3) \text{ 复相关系数 } r = \sqrt{1 - q/t}$$

$$\text{其中 } t = \sum_{i=0}^{n-1} (y_i - \bar{y})^2, \text{ 而 } \bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$$

当 r 接近于 1 时，说明相对误差 q/t 接近于零，线性回归效果好。

$$(4) \text{ 偏相关系数 } v_j = \sqrt{1 - q_j/q}, j = 0, 1, \dots, m-1$$

其中

$$q_j = \sum_{i=0}^{n-1} [y_i - (a_m + \sum_{k=0, k \neq j}^{m-1} a_k x_{ki})]^2$$

当 v_j 越大时，说明 x_j 对于 y 的作用越显著，此时不可把 x_j 删除。

$$(5) \text{ 回归平方和 } u = \sum_{i=0}^{n-1} [\bar{y} - (a_0 x_{0i} + a_1 x_{1i} + \dots + a_{m-1} x_{(m-1)i} + a_m)]^2$$

三、函数语句

void isqt2 (x, y, m, n, a, dt, v)

本函数要调用平方根法求解对称正定方程组的函数 chol，参看 1.8 节。

四、形参说明

x ——双精度实型二维数组,体积为 $m \times n$ 。每一列存放 m 个自变量的一组观测值,即

$$(x_{0i}, x_{1i}, \dots, x_{m-1,i})^T, i = 0, 1, \dots, n - 1$$

y ——双精度实型一维数组,长度为 n 。存放随机变量 y 的 n 个观测值。

m ——整型变量。自变量个数。

n ——整型变量。观测数据的组数。

a ——双精度实型一维数组,长度为 $m + 1$ 。返回回归系数 $a_0, a_1, \dots, a_{m-1}, a_m$ 。

dt ——双精度实型一维数组,长度为 4。基中 $dt(0)$ 返回偏差平方和 q , $dt(1)$ 返回平均标准偏差 s , $dt(2)$ 返回复相关系数 r , $dt(3)$ 返回回归平方和 u 。

v ——双精度实型一维数组,长度为 m 。返回 m 个自变量的偏相关系数。

五、函数程序(文件名:isqt 2.c)

六、例

对随机变量 y 及自变量 x_0, x_1, x_2 的下列五组观测数据作多元线性回归分析。

i	x_{0i}	x_{1i}	x_{2i}	y_i
0	1.1	2.0	3.2	10.1
1	1.0	2.0	3.2	10.2
2	1.2	1.8	3.0	10.0
3	1.1	1.9	2.9	10.1
4	0.9	2.1	2.9	10.0

主函数程序(文件名:isqt 20.c)如下:

```
# include "stdio.h"
# include "isqt2.c"
# include "achol.c"
main()
{
    int i;
    double a[4], v[3], dt[4];
    static double x[3][5] = { {1.1, 1.0, 1.2, 1.1, 0.9},
                            {2.0, 2.0, 1.8, 1.9, 2.1}, {3.2, 3.2, 3.0, 2.9, 2.9} };
    static double y[5] = {10.1, 10.2, 10.0, 10.1, 10.0};
    isqt2(x, y, 3, 5, a, dt, v);
    printf("\n");
    for (i = 0; i <= 3; i++)
        printf("a(%2d) = %e\n", i, a[i]);
    printf("\n");
    printf("q = %e s = %e r = %e\n", dt[0], dt[1], dt[2]);
}
```

```

printf("\n");
for (i = 0; i <= 2; i++)
    printf("v(%2d) = %e\n", i, v[i]);
printf("\n");
printf("u = %e\n", dt[3]);
printf("\n");
}

```

运行结果为:

```

a(0) = -8.00000e-01
a(1) = -7.00000e-01
a(2) = 5.00000e-01
a(3) = 1.07800e+01

q = 1.20000e-02  s = 4.89898e-02  r = 7.55929e-01

v(0) = 9.98351e-01
v(1) = 9.99365e-01
v(2) = 9.99482e-01

u = 1.60000e-02

```

9.4 逐步回归分析

一、功能

对多元线性回归进行因子筛选,最后给出一定显著性水平下各因子均为显著的回归方程中的诸回归系数、偏回归平方和、估计的标准偏差、复相关系数及 F-检验值、各回归系数的标准偏差、应变量条件期望值的估计值及残差。

二、方法说明

设 n 个自变量为 $x_j (j = 0, 1, \dots, n - 1)$, 应变量为 y 。有 k 个观测点为

$$(x_{i0}, x_{i1}, \dots, x_{in-1}, y_i), \quad i = 0, 1, \dots, k - 1$$

根据最小二乘原理, y 的估计值为

$$\hat{y} = b_{i_0}x_{i_0} + b_{i_1}x_{i_1} + \dots + b_{i_l}x_{i_l} + b_n$$

其中 $0 \leq i_0 < i_1 < \dots < i_l \leq n - 1$, 且各 $x_{it} (t = 0, 1, \dots, l)$ 是从 n 个自变量 $x_j (j = 0, 1, \dots, n - 1)$ 中按一定显著性水平筛选出的统计检验为显著的因子。其筛选过程如下。

(1) 首先作出 $(n + 1) \times (n + 1)$ 的规格化的系数初始相关阵

$$R = \begin{bmatrix} r_{00} & r_{01} & \cdots & r_{0, n-1} & r_{0y} \\ r_{10} & r_{11} & \cdots & r_{1, n-1} & r_{1y} \\ \vdots & \vdots & & \vdots & \vdots \\ r_{n-1, 0} & r_{n-1, 1} & \cdots & r_{n-1, n-1} & r_{n-1, y} \\ r_{yy} & r_{y1} & \cdots & r_{y, n-1} & r_{yy} \end{bmatrix}$$

矩阵中各元素为

$$r_{ij} = \frac{d_{ij}}{d_i d_j} = \frac{\sum_{t=0}^{k-1} (x_{ti} - \bar{x}_i)(x_{tj} - \bar{x}_j)}{\sqrt{\sum_{t=0}^{k-1} (x_{ti} - \bar{x}_i)^2} \cdot \sqrt{\sum_{t=0}^{k-1} (x_{tj} - \bar{x}_j)^2}}$$

$$i, j = 0, 1, \dots, n-1, n$$

其中 n 对应 y 。式中

$$\bar{x}_i = \sum_{t=0}^{k-1} x_{ti} / k, i = 0, 1, \dots, n-1, n$$

(2) 计算偏回归平方和 $V_i = \frac{r_{iy}^2}{r_{ii}}, i = 0, 1, \dots, n-1$

(3) 若 $V_i < 0$, 则对应的 x_i 是已被选入回归方程的因子。

从所有 $V_i < 0$ 的 V_i 中选出 $V_{\min} = \min |V_i|$, 其对应的因子为 x_{\min} 。然后检验因子 x_{\min} 的显著性。若

$$\frac{\varphi V_{\min}}{r_{yy}} < F_2$$

则剔除因子 x_{\min} , 并对系数相关阵 R 进行该因子的消元变换。转(2)。

(4) 若 $V_i > 0$, 则对应的 x_i 为尚待选入的因子。

从所有 $V_i > 0$ 的 V_i 中选出 $V_{\max} = \max |V_i|$, 其对应的因子为 x_{\max} 。然后检验因子 x_{\max} 的显著性。若

$$\frac{(\varphi - 1)V_{\max}}{r_{yy} - V_{\max}} \geq F_1$$

则因子 x_{\max} 应选入, 并对系数相关阵 R 进行该因子的消元变换。转(2)。

上述过程一直进行到无因子可剔可选为止。

在以上步骤中, φ 为相应的残差平方和的自由度。 F_1 及 F_2 均是 F -分布值, 它们取决于观测点数、已选入的因子数及取舍显著性水平 α 。通常取 $F_1 > F_2$, 当选入单个因子的显著性水平取为 α 时, 则可以从 F -分布表中取 $m = 1$, 观测点数为 n 时的 F_α 为 F_2 , 而取 $m = 1$, 观测点数为 $n-1$ 时的 F_α 为 F_1 。

当要剔除或选入某个因子 x_l 时, 均需对系数相关阵 R 进行消元变换, 其算法如下:

$$r_{ij} = r_{ij} - \frac{r_{il}}{r_{ll}} r_{il}, i, j = 0, 1, \dots, n; i, j \neq l$$

$$r_{lj} = r_{lj} / r_{ll}, j = 0, 1, \dots, n; j \neq l$$

$$r_{ll} = -r_{ll} / r_{ll}, i = 0, 1, \dots, n; i \neq l$$

$$r_{ll} = 1 / r_{ll}$$

当筛选结束时, 就可得出规范化回归方程的各回归系数 $b_0, b_1, b_2, \dots, b_n$, 其中值为 0 的系数表示对应的自变量可剔除。

回归模型的各有关值由下列各式计算:

· 选入回归方程的各因子的回归系数 $b_i = \frac{d_y}{d_i} r_{iy}, i = 0, 1, \dots, n-1$

- 回归方程的常数项 $b_n = \bar{y} - \sum_{i=0}^{n-1} b_i \bar{x}_i$
- 各因子的偏回归平方和 $V_i = r_{iy} r_{yi} / r_{ii}, i = 0, 1, \dots, n-1$
- 估计的标准偏差 $s = d_y \sqrt{r_{yy}/\varphi}$
- 各回归系数的标准偏差 $s_i = s \sqrt{r_{ii}/d_i}, i = 0, 1, \dots, n-1$
- 复相关系数 $C = \sqrt{1 - r_{yy}}$
- F -检验值 $F = \frac{\varphi(1 - r_{yy})}{(k - \varphi - 1)r_{yy}}$
- 残差平方和 $q = d_y^2 r_{yy}$
- 应变量条件期望值的估计值 $e_i = b_n + \sum_{j=0}^{n-1} b_j x_{ij}, i = 0, 1, \dots, k-1$
- 残差 $\delta_i = y_i - e_i, i = 0, 1, \dots, k-1$

本函数适用于自变量个数较多, 且观测点较多的问题。

三、函数语句

void isqt3(n, k, x, f1, f2, eps, xx, b, v, s, dt, ye, yr, r)

四、形参说明

n ——整型变量。自变量 x 的个数。

k ——整型变量。观测点数。

x ——双精度实型二维数组, 体积为 $k \times (n+1)$, 其中前 n 列存放自变量因子 $x_i (i = 0, 1, \dots, n-1)$ 的 k 次观测值。最后一列存放应变量 y 的 k 次观测值。

$f1$ ——双精度实型变量。置欲选入因子时显著性检验的 F -分布值。

$f2$ ——双精度实型变量。置欲剔除因子时显著性检验的 F -分布值。

eps ——双精度实型变量。置防止系数相关阵退化的判据。

xx ——双精度实型一维数组, 长度为 $n+1$ 。其中前 n 个分量返回 n 个自变量因子的算术平均值 $\bar{x}_i (i = 0, 1, \dots, n-1)$; 第 $n+1$ 个分量(即 $xx(n)$)返回应变量 y 的算术平均值 \bar{y} 。

b ——双精度实型一维数组, 长度为 $n+1$ 。返回回归方程中各因子的回归系数及常数项 b_0, b_1, \dots, b_n 。

v ——双精度实型一维数组, 长度为 $n+1$ 。其中前 n 个分量返回各因子的偏回归平方和 $v_i (i = 0, 1, \dots, n-1)$; 最后一个分量(即 $v(n)$)返回残差平方和 q 。

s ——双精度实型一维数组, 长度为 $n+1$ 。其中前 n 个分量返回各因子回归系数的标准偏差 $s_i (i = 0, 1, \dots, n-1)$; 最后一个分量(即 $s(n)$)返回估计的标准偏差 s 。

dt ——双精度实型一维数组, 长度为 2。其中 $dt(0)$ 返回复相关系数 c ; $dt(1)$ 返回 F -检验值。

ye ——双精度实型一维数组, 长度为 k 。返回因变量条件期望值的 k 个估计值(对应于 k 个观测值) $e_i (i = 0, 1, \dots, k-1)$ 。

yr ——双精度实型一维数组, 长度为 k 。返回因变量的 k 个观测值的残差 $\delta_i (i = 0, 1, \dots, k-1)$ 。

$\dots, k-1$)。

r ——双精度实型二维数组, 体积为 $(n+1) \times (n+1)$ 。返回最终的规格化的系数相关阵 R。

五、函数程序(文件名:isqt 3.c)

六、例

设四个自变量为 x_0, x_1, x_2, x_3 , 因变量为 y。13 个观测点值如下:

k	x_0	x_1	x_2	x_3	y
0	7.0	26.0	6.0	60.0	78.5
1	1.0	29.0	15.0	52.0	74.3
2	11.0	56.0	8.0	20.0	104.3
3	11.0	31.0	8.0	47.0	87.6
4	7.0	52.0	6.0	33.0	95.9
5	11.0	55.0	9.0	22.0	109.2
6	3.0	71.0	17.0	6.0	102.7
7	1.0	31.0	22.0	44.0	72.5
8	2.0	54.0	18.0	22.0	93.1
9	21.0	47.0	4.0	26.0	115.9
10	1.0	40.0	23.0	34.0	83.8
11	11.0	66.0	9.0	12.0	113.3
12	10.0	68.0	8.0	12.0	109.4

对于不同的 F_1 与 F_2 值进行逐步回归分析:

- (1) 当取 $\alpha=0.25$ 时, 查 F-分布表得 $F_1=1.46, F_2=1.45$;
- (2) 当取 $\alpha=0.05$ 时, 查 F-分布表得 $F_1=4.75, F_2=4.67$;
- (3) 当取 $\alpha=0.01$ 时, 查 F-分布表得 $F_1=9.33, F_2=9.07$ 。

主函数程序(文件名:isqt 30.c)如下:

```
# include "stdio.h"
# include "isqt3.c"
main()
{ int i, j, k;
double eps, xx[5], b[5], v[5], s[5], ye[13], yr[13];
double r[5][5], dt[2];
static double x[13][5] = {
    {7.0, 26.0, 6.0, 60.0, 78.5},
    {1.0, 29.0, 15.0, 52.0, 74.3},
    {11.0, 56.0, 8.0, 20.0, 104.3},
    {11.0, 31.0, 8.0, 47.0, 87.6},
    {7.0, 52.0, 6.0, 33.0, 95.9},
    {11.0, 55.0, 9.0, 22.0, 109.2},
```

```
{3.0, 71.0, 17.0, 6.0, 102.7},
{1.0, 31.0, 22.0, 44.0, 72.5},
{2.0, 54.0, 18.0, 22.0, 93.1},
{21.0, 47.0, 4.0, 26.0, 115.9},
{1.0, 40.0, 23.0, 34.0, 83.8},
{11.0, 66.0, 9.0, 12.0, 113.3},
{10.0, 68.0, 8.0, 12.0, 109.4}};

static double f1[3] = {1.46, 4.75, 9.33};
static double f2[3] = {1.45, 4.67, 9.07};
eps = 1.0e-30;
for (k = 0; k <= 2; k++)
{ isqt3(4, 13, x, f1[k], f2[k], eps, xx, b, v, s, dt, ye, yr, r);
printf("\n");
printf("f1 = %e f2 = %e\n", f1[k], f2[k]);
printf("\n");
printf("original x(i) and y values: \n");
for (i = 0; i <= 12; i++)
{ for (j = 0; j <= 3; j++)
printf("x[%d][%d] = %6.2f", i, j, x[i][j]);
printf("y = %6.2f\n", x[i][4]);
}
printf("\n");
printf("mean of x(i) and y: \n");
for (i = 0; i <= 3; i++)
printf("x(%d) = %6.3f", i, xx[i]);
printf("y = %6.3f\n", xx[4]);
printf("\n");
printf("regression coeffi b(i): \n");
for (i = 0; i <= 4; i++)
printf("b(%d) = %9.6f ", i, b[i]);
printf("\n");
printf("standard partial sum of square of \n");
printf("regression for x(i) and sum of \n");
printf("square of residuals: \n");
for (i = 0; i <= 3; i++)
printf("v(%d) = %e ", i, v[i]);
printf("\n");
printf("q = %e\n", v[4]);
printf("\n");
printf("standard deviation of regression \n");
printf("coeffi and regression equation: \n");
for (i = 0; i <= 3; i++)
printf("s(%d) = %e ", i, s[i]);
```

```

printf("\n");
printf("s = %e\n", s[4]);
printf("\n");
printf("multi-correlation coeffi c is: %e\n", dt[0]);
printf("\n");
printf("the f value= %e\n", dt[1]);
printf("\n");
printf("estimated values and residuals: \n");
for (i = 0; i < = 12; i++)
{
    printf("ye(%d) = %e yr(%d) = %e\n",
           i, ye[i], i, yr[i]);
}
printf("\n");
printf("matrix r:\n");
for (i = 0; i < = 4; i++)
{
    for (j = 0; j < = 4; j++)
        printf("%e ", r[i][j]);
    printf("\n");
}
printf("\n");

```

运行结果为：

f1 = 1.46000e + 00 f2 = 1.45000e + 00

original x(i) and y values:

x(0) = 7.00	x(1) = 26.00	x(2) = 6.00	x(3) = 60.00	y = 78.50
x(0) = 1.00	x(1) = 29.00	x(2) = 15.00	x(3) = 52.00	y = 74.30
x(0) = 11.00	x(1) = 56.00	x(2) = 8.00	x(3) = 20.00	y = 104.30
x(0) = 11.00	x(1) = 31.00	x(2) = 8.00	x(3) = 47.00	y = 87.60
x(0) = 7.00	x(1) = 52.00	x(2) = 6.00	x(3) = 33.00	y = 95.90
x(0) = 11.00	x(1) = 55.00	x(2) = 9.00	x(3) = 22.00	y = 109.20
x(0) = 3.00	x(1) = 71.00	x(2) = 17.00	x(3) = 6.00	y = 102.70
x(0) = 1.00	x(1) = 31.00	x(2) = 22.00	x(3) = 44.00	y = 72.50
x(0) = 2.00	x(1) = 54.00	x(2) = 18.00	x(3) = 22.00	y = 93.10
x(0) = 21.00	x(1) = 47.00	x(2) = 4.00	x(3) = 26.00	y = 115.90
x(0) = 1.00	x(1) = 40.00	x(2) = 23.00	x(3) = 34.00	y = 83.80
x(0) = 11.00	x(1) = 66.00	x(2) = 9.00	x(3) = 12.00	y = 113.30
x(0) = 10.00	x(1) = 68.00	x(2) = 8.00	x(3) = 12.00	y = 109.40

mean of x(i) and y:

x(0) = 7.462 x(1) = 48.154 x(2) = 11.769 x(3) = 30.000 y = 95.423

```

regression coeffi b(i):
b(0) = 1.451938 b(1) = 0.416110 b(2) = 0.000000 b(3) = - 0.236540 b(4) = 71.648307
standard partial sum of square of
regression for x(i) and sum of
square of residuals:
v(0) = - 3.02275e - 01 v(1) = - 9.86440e - 03 v(2) = 4.01692e - 05 v(3) = - 3.65708e - 03
q = 4.79727e + 01
standard deviation of regression
coeffi and regression equation:
s(0) = 1.16998e - 01 s(1) = 1.85610e - 01 s(2) = 0.00000e + 00 s(3) = 1.73288e - 01
s = 2.30874e + 00
multi-correlation coeffi c is: 9.91128e - 01
the f value = 1.66832e + 02
estimated values and residuals:
ye(0) = 7.84383e + 01 yr(0) = 6.16864e - 02
ye(1) = 7.28673e + 01 yr(1) = 1.43266e + 00
ye(2) = 1.06191e + 02 yr(2) = - 1.89097e + 00
ye(3) = 8.94016e + 01 yr(3) = - 1.80164e + 00
ye(4) = 9.56438e + 01 yr(4) = 2.56247e - 01
ye(5) = 1.05302e + 02 yr(5) = 3.89822e + 00
ye(6) = 1.04129e + 02 yr(6) = - 1.42867e + 00
ye(7) = 7.55919e + 01 yr(7) = - 3.09188e + 00
ye(8) = 9.18182e + 01 yr(8) = 1.28177e + 00
ye(9) = 1.15546e + 02 yr(9) = 3.53883e - 01
ye(10) = 8.17023e + 01 yr(10) = 2.09773e + 00
ye(11) = 1.12244e + 02 yr(11) = 1.05561e + 00
ye(12) = 1.11625e + 02 yr(12) = - 2.22467e + 00
matrix r:
1.06633e + 00 2.04390e - 01 - 8.93654e - 01 4.60588e - 01 5.67737e - 01
2.04390e - 01 1.87803e + 01 - 2.24227e + 00 1.83226e + 01 4.30414e - 01
8.93654e - 01 2.24227e + 00 2.13363e - 02 2.37143e + 00 9.25778e - 04
4.60588e - 01 1.83226e + 01 - 2.37143e + 00 1.89401e + 01 2.63183e - 01
- 5.67737e - 01 - 4.30414e - 01 9.25778e - 04 2.63183e - 01 1.76645e - 02

```

f1 = 4.75000e + 00 f2 = 4.67000e + 00

original x(i) and y values:

```
x(0) = 7.00 x(1) = 26.00 x(2) = 6.00 x(3) = 60.00 y = 78.50
x(0) = 1.00 x(1) = 29.00 x(2) = 15.00 x(3) = 52.00 y = 74.30
x(0) = 11.00 x(1) = 56.00 x(2) = 8.00 x(3) = 20.00 y = 104.30
x(0) = 11.00 x(1) = 31.00 x(2) = 8.00 x(3) = 47.00 y = 87.60
x(0) = 7.00 x(1) = 52.00 x(2) = 6.00 x(3) = 33.00 y = 95.90
x(0) = 11.00 x(1) = 55.00 x(2) = 9.00 x(3) = 22.00 y = 109.20
x(0) = 3.00 x(1) = 71.00 x(2) = 17.00 x(3) = 6.00 y = 102.70
x(0) = 1.00 x(1) = 31.00 x(2) = 22.00 x(3) = 44.00 y = 72.50
x(0) = 2.00 x(1) = 54.00 x(2) = 18.00 x(3) = 22.00 y = 93.10
x(0) = 21.00 x(1) = 47.00 x(2) = 4.00 x(3) = 26.00 y = 115.90
x(0) = 1.00 x(1) = 40.00 x(2) = 23.00 x(3) = 34.00 y = 83.80
x(0) = 11.00 x(1) = 66.00 x(2) = 9.00 x(3) = 12.00 y = 113.30
x(0) = 10.00 x(1) = 68.00 x(2) = 8.00 x(3) = 12.00 y = 109.40
```

mean of x(i) and y:

```
x(0) = 7.462 x(1) = 48.154 x(2) = 11.769 x(3) = 30.000 y = 95.423
```

regression coeffi b(i):

```
b(0) = 1.468306 b(1) = 0.662250 b(2) = 0.000000 b(3) = 0.000000 b(4) = 52.577349
```

standard partial sum of square of

regression for x(i) and sum of

square of residuals:

```
v(0) = -3.12410e-01 v(1) = -4.44730e-01 v(2) = 3.60630e-03 v(3) = 3.65708e-03
```

```
q = 5.79045e+01
```

standard deviation of regression

coeffi and regression equation:

```
s(0) = 1.21301e-01 s(1) = 4.58547e-02 s(2) = 0.00000e+00 s(3) = 0.00000e+00
```

```
s = 2.40634e+00
```

multi-correlation coeffi c is: 9.89282e-01

the f value = 2.29504e+02

estimated values and residuals:

```
ye(0) = 8.00740e+01 yr(0) = -1.57400e+00
ye(1) = 7.32509e+01 yr(1) = 1.04908e+00
ye(2) = 1.05815e+02 yr(2) = -1.51474e+00
ye(3) = 8.92585e+01 yr(3) = -1.65848e+00
ye(4) = 9.72925e+01 yr(4) = -1.39251e+00
ye(5) = 1.05152e+02 yr(5) = 4.04751e+00
```

```
ye(6) = 1.04002e+02 yr(6) = -1.30205e+00
ye(7) = 7.45754e+01 yr(7) = -2.07542e+00
ye(8) = 9.12755e+01 yr(8) = 1.82451e+00
ye(9) = 1.14538e+02 yr(9) = 1.36246e+00
ye(10) = 8.05357e+01 yr(10) = 3.26433e+00
ye(11) = 1.12437e+02 yr(11) = 8.62756e-01
ye(12) = 1.12293e+02 yr(12) = -2.89344e+00
```

matrix r:

1.05513e+00	-2.41181e-01	-8.35985e-01	-2.43182e-02	5.74137e-01
-2.41181e-01	1.05513e+00	5.18466e-02	-9.67396e-01	6.85017e-01
8.35985e-01	-5.18466e-02	3.18256e-01	-1.25207e-01	3.38781e-02
2.43182e-02	9.67396e-01	-1.25207e-01	5.27981e-02	-1.38956e-02
-5.74137e-01	-6.85017e-01	3.38781e-02	-1.38956e-02	2.13216e-02

f1 = 9.33000e+00 f2 = 9.07000e+00

original x(i) and y values:

```
x(0) = 7.00 x(1) = 26.00 x(2) = 6.00 x(3) = 60.00 y = 78.50
x(0) = 1.00 x(1) = 29.00 x(2) = 15.00 x(3) = 52.00 y = 74.30
x(0) = 11.00 x(1) = 56.00 x(2) = 8.00 x(3) = 20.00 y = 104.30
x(0) = 11.00 x(1) = 31.00 x(2) = 8.00 x(3) = 47.00 y = 87.60
x(0) = 7.00 x(1) = 52.00 x(2) = 6.00 x(3) = 33.00 y = 95.90
x(0) = 11.00 x(1) = 55.00 x(2) = 9.00 x(3) = 22.00 y = 109.20
x(0) = 3.00 x(1) = 71.00 x(2) = 17.00 x(3) = 6.00 y = 102.70
x(0) = 1.00 x(1) = 31.00 x(2) = 22.00 x(3) = 44.00 y = 72.50
x(0) = 2.00 x(1) = 54.00 x(2) = 18.00 x(3) = 22.00 y = 93.10
x(0) = 21.00 x(1) = 47.00 x(2) = 4.00 x(3) = 26.00 y = 115.90
x(0) = 1.00 x(1) = 40.00 x(2) = 23.00 x(3) = 34.00 y = 83.80
x(0) = 11.00 x(1) = 66.00 x(2) = 9.00 x(3) = 12.00 y = 113.30
x(0) = 10.00 x(1) = 68.00 x(2) = 8.00 x(3) = 12.00 y = 109.40
```

mean of x(i) and y:

```
x(0) = 7.462 x(1) = 48.154 x(2) = 11.769 x(3) = 30.000 y = 95.423
```

regression coeffi b(i):

```
b(0) = 1.439958 b(1) = 0.000000 b(2) = 0.000000 b(3) = -0.613954 b(4) = 103.097382
```

standard partial sum of square of

regression for x(i) and sum of

square of residuals:

```
v(0) = -2.97929e-01 v(1) = 9.86440e-03 v(2) = 8.81004e-03 v(3) = -4.38523e-01
```

```
q = 7.47621e+01
```

```

standard deviation of regression
coeffi and regression equation:
s(0) = 1.38417e-01 s(1) = 0.00000e+00 s(2) = 0.00000e+00 s(3) = 4.86446e-02
s = 2.73427e+00

```

multi-correlation coeffi c is: 9.86139e-01

the f value = 1.76627e + 02

estimated values and residuals:

ye(0) = 7.63399e+01	yr(0) = 2.16013e+00
ye(1) = 7.26118e+01	yr(1) = 1.68825e+00
ye(2) = 1.06658e+02	yr(2) = -2.35785e+00
ye(3) = 9.00811e+01	yr(3) = -2.48110e+00
ye(4) = 9.29166e+01	yr(4) = 2.98338e+00
ye(5) = 1.05430e+02	yr(5) = 3.77006e+00
ye(6) = 1.03734e+02	yr(6) = -1.03353e+00
ye(7) = 7.75234e+01	yr(7) = -5.02338e+00
ye(8) = 9.24703e+01	yr(8) = 6.29682e-01
ye(9) = 1.17374e+02	yr(9) = -1.47371e+00
ye(10) = 8.36629e+01	yr(10) = 1.37083e-01
ye(11) = 1.11569e+02	yr(11) = 1.73052e+00
ye(12) = 1.10130e+02	yr(12) = -7.29521e-01

matrix r:

1.06411e+00	-1.08832e-02	-8.69251e-01	2.61179e-01	5.63052e-01
1.08832e-02	5.32473e-02	-1.19395e-01	9.75626e-01	2.29184e-02
8.69251e-01	-1.19395e-01	2.89051e-01	1.83816e-01	-5.04633e-02
2.61179e-01	-9.75626e-01	-1.83816e-01	1.06411e+00	-6.83107e-01
-5.63052e-01	2.29184e-02	5.04633e-02	6.83107e-01	2.75290e-02

9.5 半对数数据相关

一、功能

对于给定的 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$) 用

$$y = bt^{ax}, t > 0$$

作拟合。

二、方法说明

设给定 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$)，且 $y_i > 0$ ，用函数

$$y = bt^{ax}, t > 0$$

进行拟合。为了求拟合参数，两边取对数，即

$$\log y = \log b + ax$$

令

$$\tilde{y} = \tilde{a}\tilde{x} + \tilde{b}$$

其中

$$\tilde{y} = \log y, \tilde{x} = x, \tilde{a} = a, \tilde{b} = \log b$$

此时，问题化为对 n 个数据点 $(\tilde{x}_i, \tilde{y}_i)$ 作线性拟合，求出 \tilde{a} 与 \tilde{b} 后，就可以得到

$$a = \tilde{a}, b = t^{\tilde{b}}$$

关于一元线性拟合参看 9.2 节的方法说明。

三、函数语句

void ilog1(n, x, y, t, a)

四、形参说明

n——整型变量。数据点数。

x, y——均为双精度实型一维数组，长度为 n 。存放 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$)，要求所有的 $y_i > 0$ 。

t——双精度实型变量。指数函数的底，要求 $t > 0$ 。

a——双精度实型一维数组，长度为 7。返回拟合函数的参数以及各种统计量，其意义如下：

a(0)：拟合函数 $y = bt^{ax}$ 中的 b ；

a(1)：拟合函数 $y = bt^{ax}$ 中的 a ；

a(2)：偏差平方和 q ，即 $q = \sum_{i=0}^{n-1} (y_i - bt^{ax})^2$ ；

a(3)：平均标准偏差 s ，即 $s = \sqrt{q/n}$ ；

a(4)：最大偏差 $umax$ ，即 $umax = \max_{0 \leq i \leq n-1} |y_i - bt^{ax}|$ ；

a(5)：最小偏差 $umin$ ，即 $umin = \min_{0 \leq i \leq n-1} |y_i - bt^{ax}|$ ；

a(6)：偏差平均值 u ，即 $u = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - bt^{ax}| / n$ 。

五、函数程序(文件名:ilog1.c)

六、例

给定 12 个数据点如下：

x	0.96	0.94	0.92	0.90	0.88	0.86	0.84	0.82	0.80	0.78	0.76	0.74
y	558.0	313.0	174.0	97.0	55.8	31.3	17.4	9.70	5.58	3.13	1.74	1.00

用函数 $y = b10^{ax}$ 进行拟合，并求偏差平方和 q 、平均标准偏差 s 、最大偏差 $umax$ 、最小偏差 $umin$ 、偏差平均值 u 。

主函数程序(文件名:ilog 10.c)如下:

```
#include "stdio.h"
#include "ilog1.c"
main()
{ int n;
double t, a[7];
static double x[12] = {0.96, 0.94, 0.92, 0.90, 0.88,
0.86, 0.84, 0.82, 0.80, 0.78, 0.76, 0.74};
static double y[12] = {558.0, 313.0, 174.0, 97.0, 55.8,
31.3, 17.4, 9.70, 5.58, 3.13, 1.74, 1.00};
t = 10.0; n = 12;
ilog1(n, x, y, t, a);
printf("\n");
printf("a = %13.7e      b = %13.7e      \n", a[1], a[0]);
printf("\n");
printf("q = %13.7e      s = %13.7e      \n", a[2], a[3]);
printf("\n");
printf("umax = %13.7e  umin = %13.7e  u = %13.7e  \n",
a[4], a[5], a[6]);
printf("\n");
}
```

运行结果为:

```
a = 1.249270e+01    b = 5.619315e-10
q = 3.271371e+01    s = 1.651103e+00
umax = 5.064987e+00, umin = 1.119425e-02  u = 8.630600e-01
```

9.6 对数数据相关

一、功能

对于给定的 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$) 用函数

$$y = bx^a$$

作拟合。

二、方法说明

设给定 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$)，且所有的 $x_i, y_i > 0$ ，用函数

$$y = bx^a, x, y > 0$$

进行拟合。

为了求拟合参数 a 与 b ，两边取对数，即

$$\ln y = \ln b + a \ln x$$

令

$$\tilde{y} = \ln y, \tilde{x} = \ln x, \tilde{a} = a, \tilde{b} = \ln b$$

其中

$$\tilde{y} = \ln y, \tilde{x} = \ln x, \tilde{a} = a, \tilde{b} = \ln b$$

此时，问题化为对 n 个数据点 $(\tilde{x}_i, \tilde{y}_i)$ 作线性拟合，求出 \tilde{a} 与 \tilde{b} 后，就可以得到

$$a = \tilde{a}, b = e^{\tilde{b}}$$

关于一元线性拟合参看 9.2 节的方法说明。

三、函数语句

void ilog2(n, x, y, a)

四、形参说明

n ——整型变量。数据点数。

x, y ——均为双精度实型一维数组，长度为 n 。存放 n 个数据点 (x_i, y_i) ($i = 0, 1, \dots, n-1$)，要求所有的 $x_i, y_i > 0$ 。

a ——双精度实型一维数组，长度为 7。返回拟合参数以及各种统计量，其意义如下：

$a(0)$: 拟合函数 $y = bx^a$ 中的 b ；

$a(1)$: 拟合函数 $y = bx^a$ 中的 a ；

$a(2)$: 偏差平方和 q ，即 $q = \sum_{i=0}^{n-1} (y_i - bx_i^a)^2$ ；

$a(3)$: 平均标准偏差 s ，即 $s = \sqrt{q/n}$ ；

$a(4)$: 最大偏差 $umax$ ，即 $umax = \max_{0 \leq i \leq n-1} |y_i - bx_i^a|$ ；

$a(5)$: 最小偏差 $umin$ ，即 $umin = \min_{0 \leq i \leq n-1} |y_i - bx_i^a|$ ；

$a(6)$: 偏差平均值 u ，即 $u = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - bx_i^a|$ 。

五、函数程序(文件名:ilog 2.c)

六、例

给定 10 个数据点如下：

x	0.1	1.0	3.0	5.0	8.0	10.0	20.0	50.0	80.0	100.0
y	0.1	0.9	2.5	4.0	6.3	7.8	14.8	36.0	54.0	67.0

用函数 $y = bx^a$ 进行拟合，并求偏差平方和 q 、平均标准偏差 s 、最大偏差 $umax$ 、最小偏差 $umin$ 、偏差平均值 u 。

主函数程序(文件名:ilog 20.c)如下：

```

#include "stdio.h"
#include "ilog2.c"
main()
{
    int n;
    double a[7];
    static double x[10] = {0.1, 1.0, 3.0, 5.0, 8.0, 10.0,
                          20.0, 50.0, 80.0, 100.0};
    static double y[10] = {0.1, 0.9, 2.5, 4.0, 6.3, 7.8,
                          14.8, 36.0, 54.0, 67.0};

    n = 10;
    ilog2(n, x, y, a);
    printf("\n");
    printf("a = %13.7e      b = %13.7e      \n", a[1], a[0]);
    printf("\n");
    printf("q = %13.7e      s = %13.7e      \n", a[2], a[3]);
    printf("\n");
    printf("umax = %13.7e  umin = %13.7e  u = %13.7e      \n",
           a[4], a[5], a[6]);
    printf("\n");
}

```

运行结果为：

```

a = 9.415757e-01  b = 8.857726e-01
q = 1.786596e+00  s = 4.226814e-01
umax = 8.561686e-01  umin = 1.331988e-03  u = 2.506153e-01

```

第 10 章 极值问题

10.1 一维极值连分式法

一、功能

用连分式法求目标函数 $f(x)$ 的极值点。

二、方法说明

设目标函数 $f(x)$ 在区间 $[a, b]$ 内连续且单峰(或单谷)。 $f(x)$ 的极值点为函数

$$y(x) = \frac{d[f(x)]}{dx}$$

的零点。

设 $y(x)$ 的反函数为 $x = \varphi(y)$, 且用如下连分式表示

$$x = \varphi(y) = b_0 + \frac{y - y_0}{b_1} + \frac{y - y_1}{b_2} + \cdots + \frac{y - y_{i-1}}{b_i} + \cdots$$

令 $y=0$, 即得 $y(x)$ 的零点(即 $f(x)$ 的极值点)为

$$\tilde{x} = b_0 - \frac{y_0}{b_1} - \frac{y_1}{b_2} - \cdots - \frac{y_{i-1}}{b_i} - \cdots$$

由此可得计算 $f(x)$ 的极值点(即 $y(x)$ 的零点)的过程如下。

在区间 $[a, b]$ 内选取三个初始点 x_0, x_1, x_2 , 并分别计算得相应的导数值

$$y_0 = f'(x_0), y_1 = f'(x_1), y_2 = f'(x_2)$$

再利用连分式插值法计算得系数 b_0, b_1 和 b_2 。如此, 又可以求得一个新的点

$$x_3 = b_0 - \frac{y_0}{b_1} - \frac{y_1}{b_2}$$

一般来说, 假设已经取得点列 $(x_0, y_0), (x_1, y_1), \dots, (x_{i-1}, y_{i-1})$, 并由此确定了系数 b_0, b_1, \dots, b_{i-1} 。则可以求出新的点

$$x_i = b_0 - \frac{y_0}{b_1} - \frac{y_1}{b_2} - \cdots - \frac{y_{i-2}}{b_{i-1}}$$

及 $y_i = f'(x_i)$

再由如下递推关系计算出新的 b_i :

$$b_0 = x_0$$

$$b_{0i} = x_i$$

$$b_{j+1,i} = (y_i - y_j)/(b_{ji} - b_j), j = 0, 1, \dots, i-1$$

$$b_i = b_{ii}$$

以上过程一直进行到对于某个 x_i 满足

$$|y_i| < \epsilon$$

为止。此时 x_i 即为所求的极值点 \tilde{x} 。

求出极值点后，取一个很小的 Δx ，若不等式

$$f(\tilde{x} + \Delta x) - 2f(\tilde{x}) + f(\tilde{x} - \Delta x) \leq 0$$

成立，则点 \tilde{x} 为极大值点；反之为极小值点。

三、函数语句

```
void jmax1(x, eps, k, js)
```

本函数要调用计算目标函数 $f(x)$ 值与导数 $f'(x)$ 值的函数 jmax1f，由用户自编，其形式为：

```
void jmax1f(x, y)
double x, y[2];
{ y[0] = f(x)的表达式;
  y[1] = f'(x)的表达式;
  return;
}
```

四、形参说明

x ——双精度实型一维数组，长度为 2。其中 $x(0)$ 存放极值点初值，返回时存放极值点 \tilde{x} ； $x(1)$ 返回极值点 \tilde{x} 处的函数值。

ϵ ——双精度实型变量。控制精度要求。一般取 $10^{-10} \sim 10^{-35}$ 之间的数。

k ——整型变量。允许迭代次数的最大值。

js ——整型一维数组，长度为 2。其中 $js(0)$ 返回实际迭代次数； $js(1)$ 返回标志；若 $js(0) = k$ ，则有可能没有满足精度要求，返回的极值点只作为参考。若 $js(1) < 0$ ，则说明返回的极值点为极小值点；若 $js(1) > 0$ ，则说明返回的极值点为极大值点。

五、函数程序(文件名:jmax 1.c)

六、例

用连分式法计算目标函数

$$f(x) = (x - 1)(10 - x)$$

的极值点与极值点处的函数值。取初值 $x(0) = 1.0$, $\epsilon = 10^{-10}$, 最大迭代次数 $k = 10$ 。

主函数程序及计算 $f(x), f'(x)$ 的函数程序(文件名:jmax 10.c)如下：

```
# include "stdio.h"
# include "jmax1.c"
main()
{ int js[2];
  double eps, x[2];
  eps = 1.0e-10; x[0] = 1.0;
```

```
jmax1(x, eps, 10, js);
printf("\n");
printf("js(0) = %3d\n", js[0]);
if (js[1]<0 printf("MIN: ");
else printf("MAX: ");
printf("x = %e  f(x) = %e\n", x[0], x[1]);
printf("\n");
```

```
void jmax1f(x, y)
double x, y[2];
{ y[0] = (x - 1.0) * (10.0 - x);
  y[1] = -2.0 * x + 11.0;
  return;
}
```

运行结果为：

```
js(0) = 1
MAX: x = 5.50000e+00  f(x) = 2.02500e+01
```

10.2 n 维极值连分式法

一、功能

用连分式法计算多元函数的极值点与极值点处的函数值。

二、方法说明

设 n 元函数

$$Z = f(x_0, x_1, \dots, x_{n-1})$$

单峰(或单谷)，则 $f(x_0, x_1, \dots, x_{n-1})$ 的极值点为方程组

$$\frac{\partial f}{\partial x_i} = 0, \quad i = 0, 1, \dots, n-1$$

的一组实数解。

利用连分式法，轮流求某个方向 x_i 上的极值点(其余 $n-1$ 个变量保持当前值不变)。即对于 $i = 0, 1, \dots, n-1$ ，分别求函数

$$y(x_i) = \frac{\partial f}{\partial x_i}$$

的零点(具体方法见 10.1 节的方法说明)。

反复进行上述过程，直到满足

$$\sum_{i=0}^{n-1} \left| \frac{\partial f}{\partial x_i} \right| < \epsilon$$

为止。

三、函数语句

```
void jmaxn(x, n, eps, k, js)
```

本函数要调用计算目标函数 $f(x_0, x_1, \dots, x_{n-1})$ 以及各偏导数值的函数 jmaxnf, 由用户自编, 其形式为:

```
double jmaxnf(x, n, j)
int n, j;
double x[ ];
{ double y;
switch(j)
{ case 0: y = f(x0, x1, ..., xn-1) 的表达式; break;
case 1: y =  $\frac{\partial f}{\partial x_0}$  的表达式; break;
:
case n: y =  $\frac{\partial f}{\partial x_{n-1}}$  的表达式; break;
default: {}
}
return (y);
}
```

四、形参说明

x——双精度实型一维数组, 长度为 $n + 1$ 。其中 $x(0), x(1), \dots, x(n - 1)$ 存放初值, 返回时存放极值点的 n 个坐标。 $x(n)$ 返回极值点处的函数值。

n——整型变量。自变量个数。

eps——双精度实型变量。控制精度要求。

k——整型变量。允许最大迭代次数。

js——整型一维数组, 长度为 2。其中 $js(0)$ 返回实际迭代代数; $js(1)$ 返回标志: 若 $js(0) = k$, 则有可能未满足精度要求, 返回的极值点只作为参考。若 $js(1) < 0$, 则说明返回的为极小值点; 若 $js(1) > 0$, 则说明返回的为极大值点; 若 $js(1) = 0$, 则说明返回的为鞍点。

五、函数程序(文件名:jmaxn.c)

六、例

用连分式法求二元函数

$$z = (x_0 - 1)^2 + (x_1 + 2)^2 + 2$$

的极值点与极值点处的函数值。

取初值 $x_0 = 0.0, x_1 = 0.0, eps = 10^{-6}$, 最大迭代次数 $k = 10$ 。

主函数程序及计算各偏导数值的函数程序(文件名:jmaxn0.c)如下:

```
#include "stdio.h"
#include "jmaxn.c"
main()
{ int js[2];
double eps, x[3];
eps = 0.000001; x[0] = 0.0; x[1] = 0.0;
jmaxn(x, 2, eps, 10, js);
```

```
printf("\n");
printf("js(0) = %3d\n", js[0]);
printf("x(0) = %e\n", x[0]);
printf("x(1) = %e\n", x[1]);
if (js[1] < 0.0) printf("MIN: ");
if (js[1] > 0.0) printf("MAX: ");
printf("z = %e\n", x[2]);
printf("\n");

double jmaxnf(x, n, j)
int n, j;
double x[ ];
{ double y;
n = n;
switch(j)
{ case 0: y = (x[0] - 1.0) * (x[0] - 10.0)
+ (x[1] + 2.0) * (x[1] + 2.0) + 2.0;
break;
case 1: y = 2.0 * (x[0] - 1.0); break;
case 2: y = 2.0 * (x[1] + 2.0); break;
default: {}
}
return(y);
}
```

运行结果为:

```
js(0) = 2
x(0) = 1.00000e+00
x(1) = -2.00000e+00
MIN: z = 2.00000e+00
```

10.3 不等式约束线性规划问题

一、功能

求解不等式约束条件下的线性规划问题。

二、方法说明

设给定 m 阶 n 维不等式约束条件

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + \dots + a_{0,n-1}x_{n-1} \leq b_0 \\ a_{10}x_0 + a_{11}x_1 + \dots + a_{1,n-1}x_{n-1} \leq b_1 \\ \vdots \\ a_{m-1,0}x_0 + a_{m-1,1}x_1 + \dots + a_{m-1,n-1}x_{n-1} \leq b_{m-1} \end{cases}$$

且 $x_j \geq 0, j = 0, 1, \dots, n - 1$

求一组 $(x_0, x_1, \dots, x_{n-1})$ 值, 使目标函数

$$f = \sum_{j=0}^{n-1} c_j x_j$$

达到极小值。

如果要求极大值, 则只要令 $\tilde{f} = -f$, 此时就化为求目标函数

$$\tilde{f} = -\sum_{j=0}^{n-1} c_j x_j$$

的极小值。

引进 m 个非负松弛变量 $x_n, x_{n+1}, \dots, x_{n+m-1}$, 则上述问题化为:

寻找 X 值, 使满足

$$AX = B \quad (1)$$

$$X \geq 0 \quad (2)$$

且使目标函数

$$f = C^T X \quad (3)$$

达到极小值。其中

$$X = (x_0, x_1, \dots, x_{n-1}, x_n, \dots, x_{n+m-1})^T$$

$$B = (b_0, b_1, \dots, b_{m-1})^T$$

$$C = (c_0, c_1, \dots, c_{n-1}, 0, \dots, 0)^T$$

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} & 1 & 0 & \cdots & 0 \\ a_{10} & a_{11} & \cdots & a_{1,n-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

称满足(1)和(2)的解为容许解, 其中正分量的个数不多于 m 个的容许解称为基本解, 而使(3)取极小值的解称为最优解。最优解必在基本解中。

寻找最优解的过程如下。

假定已得到一个基本解, 其正分量个数为 m , 设分别为 $x_{i_0}, x_{i_1}, \dots, x_{i_{m-1}}$, 且与之对应的矩阵 A 中的列向量 $P_{i_0}, P_{i_1}, \dots, P_{i_{m-1}}$ 为线性无关, 这组向量称为基底向量。对于矩阵 A 中的每一列向量均可用基底向量的线性组合表示, 即

$$A = PD$$

其中

$$P = [P_{i_0}, P_{i_1}, \dots, P_{i_{m-1}}]$$

$$D = \begin{bmatrix} d_{00} & d_{01} & \cdots & d_{0,n+m-1} \\ d_{10} & d_{11} & \cdots & d_{1,n+m-1} \\ \vdots & \vdots & & \vdots \\ d_{m-1,0} & d_{m-1,1} & \cdots & d_{m-1,n+m-1} \end{bmatrix}$$

而组合系数矩阵 D 可用下式计算

$$D = P^{-1} A$$

$$\text{令 } z_j = c_{i_0} d_{0j} + c_{i_1} d_{1j} + \cdots + c_{i_{m-1}} d_{m-1,j}, \quad j = 0, 1, \dots, n+m-1$$

如果对于所有的 $j (j = 0, 1, \dots, n+m-1)$ 满足

$$z_j - c_j \leq 0$$

则取 X 为

$$(x_{i_0}, x_{i_1}, \dots, x_{i_{m-1}})^T$$

而 X 的其余 n 个分量均为 0。此时, X 即为最优解。

否则, 选择对应于

$$\max_{0 \leq j \leq n+m-1} (z_j - c_j) = z_k - c_k$$

的向量 P_k 进入基底向量组, 而将对应于

$$\min_{0 \leq l \leq m-1} (x_{i_l} / d_{lk}) = x_{i_l} / d_{lk}$$

$$d_{lk} > 0$$

的向量 P_l 从基底向量组中消去。这样, 对应新的基底, 其目标函数值比原先的下降了。如果所有的 $d_{lk} \leq 0 (l = 0, 1, \dots, m-1)$, 则说明目标函数值无界。

重复以上过程, 直至求出最优解, 或者确定目标函数值无界为止。

在上述的每一步中, 新的解可用下式计算:

$$(x_{i_0}, x_{i_1}, \dots, x_{i_{m-1}})^T = P^{-1} B$$

基底向量组的初值取单位矩阵, 即

$$P = I_m$$

也就是说, 取初始解为

$$X = (0, 0, \dots, 0, b_0, b_1, \dots, b_{m-1})^T$$

三、函数语句

int jlplq(a, b, c, m, n, x)

本函数返回一个整型标志值。若返回的标志值为 0, 则表示目标函数值无界; 若返回的标志值不为 0, 则表示正常返回。

本函数要调用全选主元高斯-约当法求逆的函数 brinv, 参看 2.3 节。

本函数还要调用实矩阵相乘的函数 brmul, 参看 2.1 节。

四、形参说明

a——双精度实型二维数组, 体积为 $m \times (m+n)$ 。存放不等式约束条件中左端系数矩阵及 m 阶单位矩阵, 其存放格式为:

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} & 1 & 0 & \cdots & 0 \\ a_{10} & a_{11} & \cdots & a_{1,n-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

b——双精度实型一维数组, 长度为 m 。存放不等式约束条件的右端项 b_0, b_1, \dots, b_{m-1} 。

c——双精度实型一维数组, 长度为 $m+n$ 。存放目标函数中的系数, 其中后 m 个分量

为0,即存放顺序为

$$c = (c_0, c_1, \dots, c_{n-1}, 0, \dots, 0)$$

m——整型变量。不等式约束条件的个数。

n——整型变量。变量个数。

x——双精度实型一维数组,长度为 $m+n$ 。其中前 n 个分量返回目标函数 f 的极小值点的 n 个坐标(即最优解),第 $n+1$ 个分量(即 $x(n)$)返回目标函数 f 的极小值。其余为本函数的工作单元。

五、函数程序(文件名:jlplq.c)

六、例

设不等式约束条件为

$$\begin{cases} x_0 + 2x_1 + 7x_2 \leq 10 \\ x_0 + 4x_1 + 13x_2 \leq 18 \\ 2x_1 + 8x_2 \leq 13 \\ x_0, x_1, x_2 \geq 0 \end{cases}$$

求目标函数 $f = 4x_0 + 9x_1 + 26x_2$ 的极大值。

化为极小值问题后的目标函数为 $\bar{f} = -4x_0 - 9x_1 - 26x_2$

在本例中, $m = 3$, $n = 3$, $m+n = 6$ 。且

$$A = \begin{bmatrix} 1 & 2 & 7 & 1 & 0 & 0 \\ 1 & 4 & 13 & 0 & 1 & 0 \\ 0 & 2 & 8 & 0 & 0 & 1 \end{bmatrix}$$

$$B = (10, 18, 13)^T$$

$$C = (-4, -9, -26, 0, 0, 0)^T$$

$$X = (x_0, x_1, x_2, x_3, x_4, x_5)^T$$

主函数程序(文件名:jlplq 0, c)如下:

```
#include "stdio.h"
#include "jlplq.c"
#include "brinv.c"
#include "brmul.c"
main()
{
    int i;
    double x[6];
    static double a[3][6] = { {1.0, 2.0, 7.0, 1.0, 0.0, 0.0}, 
                            {1.0, 4.0, 13.0, 0.0, 1.0, 0.0}, {0.0, 2.0, 8.0, 0.0, 0.0, 1.0} };
    static double b[3] = {10.0, 18.0, 13.0};
    static double c[6] = {-4.0, -9.0, -26.0, 0.0, 0.0, 0.0};
    i = jlplq(a, b, c, 3, x);
    printf("\n");
    if (i != 0)
        for (i = 0; i <= 5; i++)
            printf("%2d = %e\n", i, x[i]);
}
```

```
printf("%2d = %e\n", i, x[i]);
printf("\n");
```

运行结果为:

```
x(0) = 2.00000e+00
x(1) = 4.00000e+00
x(2) = 0.00000e+00
x(3) = -4.40000e+01
x(4) = 0.00000e+00
x(5) = 5.00000e+00
```

10.4 求 n 维极值的单纯形调优法

一、功能

用单纯形调优法求解无约束条件下的 n 维极值问题。

二、方法说明

设具有 n 个变量的目标函数为 $J = f(x_0, x_1, \dots, x_{n-1})$

单纯形调优法求目标函数 J 的极小值点的迭代过程如下:

(1) 在 n 维变量空间中确定一个由 $n+1$ 个顶点所构成的初始单纯形

$$X_{(i)} = (x_{0i}, x_{1i}, \dots, x_{n-1i}), i = 0, 1, \dots, n$$

并计算在每一个顶点上的函数值

$$f_{(i)} = f(X_{(i)}), i = 0, 1, \dots, n$$

(2) 确定

$$f_{(R)} = f(X_{(R)}) = \max_{0 \leq i \leq n} f_{(i)}$$

$$f_{(G)} = f(X_{(G)}) = \max_{\substack{0 \leq i \leq n \\ i \neq R}} f_{(i)}$$

$$f_{(L)} = f(X_{(L)}) = \min_{0 \leq i \leq n} f_{(i)}$$

其中 $X_{(R)}$ 称为最坏点。

(3) 求出最坏点 $X_{(R)}$ 的对称点 $X_T = 2X_F - X_{(R)}$, 其中 $X_F = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq R}}^n X_{(i)}$

(4) 确定新的顶点替代原顶点,从而构成新的单纯形。替代的原则如下:

若 $f(X_T) < f_{(L)}$, 则需要由下式将 X_T 扩大为 X_E :

$$X_E = (1 + \mu)X_T - \mu X_F$$

其中 μ 称为扩张系数,一般取 $1.2 < \mu < 2.0$ 。在这种情况下,如果 $f(X_E) < f_{(L)}$, 则 $X_E \Rightarrow X_{(R)}$, $f(X_E) \Rightarrow f_{(R)}$; 否则, $X_T \Rightarrow X_{(R)}$, $f(X_T) \Rightarrow f_{(R)}$ 。

若 $f(X_T) \leq f_{(G)}$, 则 $X_T \Rightarrow X_{(R)}$, $f(X_T) \Rightarrow f_{(R)}$ 。

若 $f(X_T) \geq f_{(G)}$ 。如果 $f(X_T) > f_{(R)}$, 则 $X_{(T)} \Rightarrow X_{(R)}$, $f(X_T) \Rightarrow f_{(R)}$ 。然后由下式将

X_T 缩小为 X_E :

$$X_E = \lambda X_{(R)} + (1 - \lambda) X_F$$

其中 λ 称为收缩系数, 一般取 $0.0 < \lambda < 1.0$ 。在这种情况下, 如果 $f(X_E) > f_{(R)}$, 则新的单形的 $n+1$ 个顶点为

$$X_{(i)} = \frac{1}{2}(X_{(i)} + X_{(L)}), i = 0, 1, \dots, n$$

且计算 $f_{(i)} = f(X_{(i)})$, $i = 0, 1, \dots, n$; 否则 $X_E \Rightarrow X_{(R)}$, $f(X_E) \Rightarrow f(R)$ 。

重复(2)~(4), 直到单形中各顶点距离小于预先给定的精度要求为止。

如果实际问题中需要求极大值, 则只要令目标函数为

$$\tilde{J} = -J = -f(x_0, x_1, \dots, x_{n-1})$$

即可。此时, \tilde{J} 的极小值的绝对值即为 J 的极大值。

三、函数语句

```
int jjsim(n, d, u, v, x, eps, k, xx, f)
```

本函数返回实际迭代次数。若实际迭代次数等于允许的最大迭代次数 k , 则有可能未达到精度要求, 返回的极值点只作为参考。

本函数要调用计算目标函数 $f(x_0, x_1, \dots, x_{n-1})$ 值的函数 $jjsimf$, 由用户自编, 其形式为:

```
double jjsimf(x, n)
int n;
double x[];
| double y;
y = f(x_0, x_1, \dots, x_{n-1}) 的表达式;
return (y);
|
```

四、形参说明

n ——整型变量。变量个数。

d ——双精度实型变量。初始单形中任意两顶点间的距离。

u ——双精度实型变量。存放扩张系数 μ , 一般取 $1.2 < \mu < 2.0$ 。

v ——双精度实型变量。存放收缩系数 λ , 一般取 $0.0 < \lambda < 1.0$ 。

x ——双精度实型一维数组, 长度为 $n+1$ 。其中前 n 个分量返回极小值点的 n 个坐标, 最后一个分量(即 $x(n)$)返回极小值。

eps ——双精度实型变量。控制精度要求。

k ——整型变量。允许的最大迭代次数。

xx ——双精度实型二维数组, 体积为 $n \times (n+1)$ 。返回最后单形的 $n+1$ 个顶点的坐标

$$(x_{0i}, x_{1i}, \dots, x_{n-1,i}), i = 0, 1, \dots, n$$

f ——双精度实型一维数组, 长度为 $n+1$ 。返回最后单形的 $n+1$ 个顶点上的目标函

数值。

五、函数程序(文件名:jjsim.c)

六、例

用单形调优法求目标函数

$$J = 100(x_1 - x_0^2) + (1 - x_0)^2$$

的极小值点与极小值。

取 $eps = 10^{-30}$, $d = 1.0$, $\mu = 1.6$, $\lambda = 0.4$, $k = 200$ 。

主函数程序及计算目标函数值的函数程序(文件名:jjsim 0.c)如下:

```
#include "math.h"
#include "stdio.c"
#include "jjsim.c"
main()
{ int i;
  double d, u, v, eps, x[3], xx[2][3], f[3];
  d = 1.0; u = 1.6; v = 0.4; eps = 1.0e-30;
  i = jjsim(2, d, u, v, eps, 200, xx, f);
  printf("\n");
  printf("i = %d\n", i);
  printf("\n");
  for (i = 0; i <= 2; i++)
    printf("x(0) = %e x(1) = %e f = %e\n",
           xx[0][i], xx[1][i], f[i]);
  printf("\n");
  for (i = 0; i <= 1; i++)
    printf("x(%d) = %e\n", i, xx[2][i]);
  printf("J = %e\n", x[2]);
  printf("\n");
```

```
double jjsimf(x, n)
int n;
double x[];
| double y;
n = n;
y = x[1] - x[0] * x[0]; y = 100.0 * y * y;
y = y + (1.0 - x[0]) * (1.0 - x[0]);
return (y);
|
```

运行结果为:

i = 87		
x(0) = 1.00000e+00	x(1) = 1.00000e+00	f = 2.44512e-16
x(0) = 1.00000e+00	x(1) = 1.00000e+00	f = 3.84621e-16
x(0) = 1.00000e+00	x(1) = 1.00000e+00	f = 1.93158e-16

$x(0) = 1.00000e + 00$
 $x(1) = 1.00000e + 00$
 $J = 4.09579e - 17$

10.5 求约束条件下 n 维极值的复形调优法

一、功能

用复形调优法求解等式与不等式约束条件下的 n 维极值问题。

二、方法说明

设多变量目标函数为 $J = f(x_0, x_1, \dots, x_{n-1})$, n 个常量约束条件为 $a_i \leq x_i \leq b_i$, $i = 0, 1, \dots, n-1$, m 个函数约束条件为

$$C_j(x_0, x_1, \dots, x_{n-1}) \leq W_j(x_0, x_1, \dots, x_{n-1}) \leq D_j(x_0, x_1, \dots, x_{n-1})$$

$$j = 0, 1, \dots, m-1$$

求 n 维目标函数 J 的极小值点及极小值。

若实际问题中要求极大值, 只需令

$$\tilde{J} = -J = -f(x_0, x_1, \dots, x_{n-1})$$

\tilde{J} 的极小值的绝对值即为 J 的极大值。

复形调优法求目标函数 J 的极小值点的迭代过程如下:

复形共有 $2n$ 个顶点。假设给定初始复形中的第一个顶点坐标

$$X_{(0)} = (x_{00}, x_{10}, \dots, x_{n-1,0})$$

且此顶点坐标满足 n 个常数约束条件及 m 个函数约束条件。

(1) 在 n 维变量空间中再确定出初始复形的其余 $2n-1$ 个顶点。其方法如下:

利用伪随机数按常量约束条件产生第 j 个顶点 $X_{(j)} = (x_{0j}, x_{1j}, \dots, x_{n-1,j})$ ($j = 1, 2, \dots, 2n-1$) 中的各分量 x_{ij} ($i = 0, 1, \dots, n-1$), 即

$$x_{ij} = a_i + r(b_i - a_i), \quad i = 0, 1, \dots, n-1; j = 1, 2, \dots, 2n-1$$

其中 r 为 $[0, 1]$ 之间的一个伪随机数。

显然, 由上述方法产生的初始复形的各顶点满足常量约束条件。然后再检查它们是否符合函数约束条件, 如果不符合, 则需要作调整, 直到全部顶点均符合函数约束条件及常量约束条件为止。调整的原则为:

假设前 j 个顶点已满足所有的约束条件, 而第 $j+1$ 个顶点不满足约束条件, 则作如下调整变换($j = 1, 2, \dots, 2n-1$):

$$X_{(j+1)} = (X_{(j+1)} + T)/2$$

其中

$$T = \frac{1}{j} \sum_{k=1}^j X_{(k)}$$

这个过程一直作到满足所有约束条件为止。

初始复形的 $2n$ 个顶点确定以后, 计算各顶点处的目标函数值

$$f_{(j)} = f(X_{(j)}), \quad j = 0, 1, \dots, 2n-1$$

(2) 确定

$$f_{(R)} = f(X_{(R)}) = \max_{0 \leq i \leq 2n-1} f_{(i)}$$

$$f_{(G)} = f(X_{(G)}) = \max_{\substack{0 \leq i \leq 2n-1 \\ i \neq R}} f_{(i)}$$

其中 $X_{(R)}$ 称为最坏点。

(3) 计算最坏点 $X_{(R)}$ 的对称点

$$X_T = (1 + \alpha)X_F - \alpha X_{(R)}$$

其中

$$X_F = \frac{1}{2n-1} \sum_{\substack{i=0 \\ i \neq R}}^{2n-1} X_{(i)}$$

α 称为反射系数, 一般取 1.3 左右。

(4) 确定一个新的顶点替代最坏点 $X_{(R)}$ 以构成新的复形。其方法如下:

如果 $f(X_T) > f_{(G)}$, 则用下式修改 X_T :

$$X_T = (X_F + X_T)/2$$

直到 $f(X_T) \leq f_{(G)}$ 为止。

然后检查 X_T 是否满足所有约束条件。如果对于某个分量 $X_T(j)$ 不满足常量约束条件, 即如果 $X_T(j) < a_j$ 或 $X_T(j) > b_j$, 则令

$$X_T(j) = a_j + \delta \text{ 或 } X_T(j) = b_j - \delta$$

其中 δ 为很小的一个常数, 一般取 $\delta = 10^{-6}$ 。然后重复(4)。

如果 X_T 不满足函数约束条件, 则用下式修改 X_T :

$$X_T = (X_F + X_T)/2$$

重复(4)。

直到 $f(X_T) \leq f_{(G)}$ 且 X_T 满足所有约束条件为止。此时令

$$X_{(R)} = X_T, \quad f_{(R)} = f(X_T)$$

重复(2)~(4), 直到复形中各顶点距离小于预先给定的精度要求为止。

三、函数语句

int jcplx(n, m, a, b, alpha, eps, x, xx, k)

本函数返回实际迭代次数。若实际迭代次数等于 k , 则返回的极小值点可能不满足精度要求, 只作为参考。

本函数要调用以下两个函数。

(1) 计算目标函数值的函数 jcplxf, 由用户自编, 其形式如下:

```
double jcplxf(x, n)
int n;
double x[];
double y;
y = f(x_0, x_1, ..., x_{n-1}) 的表达式;
```

```

    return(y);
}

```

(2) 计算函数约束条件中的下限、上限及条件值的函数 jcplx, 由用户自编, 其形式如下:

```

void jcplx(n, m, x, c, d, w)
int n, m;
double x[], c[], d[], w[];
{c[0] = C0(x0, x1, ..., xn-1) 的表达式;
:
c[m-1] = Cm-1(x0, x1, ..., xn-1) 的表达式;
d[0] = D0(x0, x1, ..., xn-1) 的表达式;
:
d[m-1] = Dm-1(x0, x1, ..., xn-1) 的表达式;
w[0] = W0(x0, x1, ..., xn-1) 的表达式;
:
w[m-1] = Wm-1(x0, x1, ..., xn-1) 的表达式;
return;
}

```

四、形参说明

n——整型变量。变量个数。

m——整型变量。函数约束条件的个数。

a——双精度实型一维数组, 长度为 n。依次存放常量约束条件中变量 x_i (i=0, 1, ..., n-1) 的下界。

b——双精度实型一维数组, 长度为 n。依次存放常量约束条件中变量 x_i (i=0, 1, ..., n-1) 的上界。

alpha——双精度实型变量。反射系数 α, 一般取值为 1.3 左右。

eps——双精度实型变量。控制精度要求。

x——双精度实型一维数组, 长度为 n+1。其中前 n 个分量存放初始复形的第一个顶点坐标值(要求满足所有的约束条件), 返回时存放极小值点各坐标值。最后一个分量 x(n) 返回极小值。

xx——双精度实型二维数组, 体积为 (n+1) × 2n。其中前 n 行返回最后复形的 2n 个顶点坐标(一列作为一个顶点)

(x_{0i}, x_{1i}, ..., x_{n-1,i}), i = 0, 1, ..., 2n - 1

最后一行返回最后复形的 2n 个顶点上的目标函数值。

k——整型变量。允许的最大迭代次数。

五、函数程序(文件名: jcplx.c)

六、例

用复形调优法求目标函数

$$J = f(x_0, x_1) = -\frac{[9 - (x_0 - 3)^2]x_1^3}{27\sqrt{3}}$$

满足约束条件

$$\begin{cases} x_0 \geq 0 \\ x_1 \geq 0 \\ 0 \leq x_1 \leq \frac{x_0}{\sqrt{3}} \\ 0 \leq x_0 + \sqrt{3}x_1 \leq 6 \end{cases}$$

的极小值点及极小值。

取 $\text{eps} = 10^{-30}$, 反射系数 $\text{alpha} = 1.3$ 。最大迭代次数 $k = 200$ 。

初始复形的第一个顶点为 (0.0, 0.0)。

其中常量约束条件的下界为 $a_0 = 0.0, a_1 = 0.0$, 上界取 $b_0 = 10^{35}, b_1 = 10^{35}$; 函数约束条件的下限、上限及条件函数为:

$$C_0(x_0, x_1) = 0.0, C_1(x_0, x_1) = 0.0$$

$$D_0(x_0, x_1) = x_0 / \sqrt{3}, D_1(x_0, x_1) = 6.0$$

$$W_0(x_0, x_1) = x_1, W_1(x_0, x_1) = x_0 + \sqrt{3}x_1$$

主函数程序及计算目标函数值、计算函数约束条件中各值的函数程序(文件名: jcplx0.c)如下:

```

#include "stdio.h"
#include "jcplx.c"
main()
{
    int i;
    double alpha, eps, a[2], b[2], x[3], xx[3][4];
    x[0] = 0.0; x[1] = 0.0;
    a[0] = 0.0; a[1] = 0.0;
    b[0] = 1.0e+35; b[1] = b[0];
    eps = 1.0e-30; alpha = 1.3;
    i = jcplx(2, 2, a, b, alpha, eps, x, xx, 200);
    printf("\n");
    printf("i = %4d\n", i);
    printf("\n");
    for (i = 0; i < i; i++)
        printf("x(0) = %e x(1) = %e f = %e\n",
               xx[0][i], xx[1][i], xx[2][i]);
    printf("\n");
    for (i = 0; i < i; i++)
        printf("x(%2d) = %e\n", i, x[i]);
    printf("J = %e\n", x[2]);
    printf("\n");
}

```

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

• 264 •

```

double jcplx(x, n)
int n;
double x[];
{ double y;
  n = n;
  y = -(9.0 - (x[0] - 3.0) * (x[0] - 3.0));
  y = y * x[1] * x[1] * x[1] / (27.0 * sqrt(3.0));
  return(y);
}

#include "math.h"
void jcplxs(n, m, x, c, d, w)
int n, m;
double x[], c[], d[], w[];
{n = n; m = m;
  c[0] = 0.0; c[1] = 0.0;
  d[0] = x[0] / sqrt(3.0); d[1] = 6.0;
  w[0] = x[1]; w[1] = x[0] + x[1] * sqrt(3.0);
  return;
}

```

运行结果为：

```

i = 88

x(0) = 3.00000e+00  x(1) = 1.73205e+00  f = -1.00000e+00
x(0) = 3.00000e+00  x(1) = 1.73205e+00  f = -1.00000e+00
x(0) = 3.00000e+00  x(1) = 1.73205e+00  f = -1.00000e+00
x(0) = 3.00000e+00  x(1) = 1.73205e+00  f = -1.00000e+00

x(0) = 3.00000e+00
x(1) = 1.73205e+00
J = -1.00000e+00

```

本问题的理论极小值点为 $x_0 = 3.0$, $x_1 = \sqrt{3}$, 极小值为 -1.0 。

第 11 章 数学变换与滤波

11.1 傅里叶级数逼近

一、功能

根据函数 $f(x)$ 在区间 $[0, 2\pi]$ 上的 $2n+1$ 个等距点

$$x_i = \frac{2\pi}{2n+1}(i+0.5), i = 0, 1, \dots, 2n$$

处的函数值 $f_i = f(x_i)$, 求傅里叶(Fourier)级数

$$f(x) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} (a_k \cos kx + b_k \sin kx)$$

的前 $2n+1$ 个系数 a_k ($k = 0, 1, \dots, n$) 和 b_k ($k = 1, 2, \dots, n$) 的近似值。

二、方法说明

设函数 $f(x)$ 在区间 $[0, 2\pi]$ 上 $2n+1$ 个点

$$x_i = \frac{2\pi}{2n+1}(i+0.5), i = 0, 1, \dots, 2n$$

处的函数值 $f_i = f(x_i)$, 计算傅里叶级数的前 $2n+1$ 个系数

$$a_k (k = 0, 1, \dots, n) \text{ 和 } b_k (k = 1, 2, \dots, n)$$

的近似值的方法如下,

对于 $k = 0, 1, \dots, n$ 作如下运算:

(1) 按下列迭代公式计算 u_1 与 u_2

$$\begin{cases} u_{2n+2} = u_{2n+1} = 0 \\ u_j = f_i + 2\cos k\theta u_{j+1} - u_{j+2}, \quad j = 2n, 2n-1, \dots, 2, 1 \end{cases}$$

其中 $\theta = \frac{2\pi}{2n+1}$ 。计算 $\cos k\theta$ 与 $\sin k\theta$ 用如下递推公式

$$\cos k\theta = \cos \theta \cos(k-1)\theta - \sin \theta \sin(k-1)\theta$$

$$\sin k\theta = \sin \theta \cos(k-1)\theta + \cos \theta \sin(k-1)\theta$$

(2) 按下列公式计算 a_k 及 b_k

$$\begin{cases} a_k = \frac{2}{2n+1}(f_0 + u_1 \cos k\theta - u_2) \\ b_k = \frac{2}{2n+1} u_1 \sin k\theta \end{cases}$$

三、函数语句

void kfour(f, n, a, b)

四、形参说明

f——双精度实型一维数组,长度为 $2n+1$ 。存放区间 $[0, 2\pi]$ 上 $2n+1$ 个等距点

$$x_i = \frac{2\pi}{2n+1}(i + 0.5), i = 0, 1, \dots, 2n$$

处的函数值 $f(x_i)$ ($i = 0, 1, \dots, 2n$)。

n——整型变量。

a——双精度实型一维数组,长度为 $n+1$ 。返回傅里叶级数中的系数 a_k ($k = 0, 1, \dots, n$)。

b——双精度实型一维数组,长度为 $n+1$ 。返回傅里叶级数中的系数 b_k ($k = 0, 1, \dots, n$)。其中 $b_0 = 0$ 。

五、函数程序(文件名:kfour.c)

六、例

根据函数 $f(x) = x^2$ 在区间 $[0, 2\pi]$ 上101个等距点

$$x_i = \frac{2\pi}{101}(i + 0.5), i = 0, 1, \dots, 100$$

处的函数值 $f_i = x_i^2$, $i = 0, 1, \dots, 100$,求傅里叶级数的系数

$a_k(k = 0, 1, \dots, 50)$ 及 $b_k(k = 0, 1, \dots, 50)$

其中 $n = 50$ 。

主函数程序(文件名:kfour0.c)如下:

```
#include "stdio.h"
#include "kfour.c"

main()
{
    int i;
    double f[101], a[51], b[51], c, h;
    h = 6.283185306/101.0;
    for (i=0; i<=100; i++)
        c = (i+0.5)*h; f[i] = c*c;
    kfour(f, 50, a, b);
    printf("\n");
    for (i=0; i<=50; i++)
        printf("a(%3d) = %e    b(%3d) = %e\n", i, a[i], i, b[i]);
    printf("\n");
}
```

运行结果为:

a(0) = 2.63183e+01	b(0) = 0.00000e+00
a(1) = 3.60654e+00	b(1) = -1.26867e+01
a(2) = 6.06546e-01	b(2) = -6.33721e+00
a(3) = 5.09912e-02	b(3) = -4.21798e+00
a(4) = -1.43451e-01	b(4) = -3.15631e+00
a(5) = -2.33449e-01	b(5) = -2.51766e+00

a(6) = -2.82335e-01	b(6) = -2.09052e+00
a(7) = -3.11810e-01	b(7) = -1.78423e+00
a(8) = -3.30939e-01	b(8) = -1.55347e+00
a(9) = -3.44052e-01	b(9) = -1.37306e+00
a(10) = -3.53430e-01	b(10) = -1.22788e+00
a(11) = -3.60367e-01	b(11) = -1.10833e+00
a(12) = -3.65641e-01	b(12) = -1.00799e+00
a(13) = -3.69744e-01	b(13) = -9.22419e-01
a(14) = -3.72997e-01	b(14) = -8.48448e-01
a(15) = -3.75620e-01	b(15) = -7.83750e-01
a(16) = -3.77764e-01	b(16) = -7.26582e-01
a(17) = -3.79539e-01	b(17) = -6.75606e-01
a(18) = -3.81025e-01	b(18) = -6.29786e-01
a(19) = -3.82280e-01	b(19) = -5.88299e-01
a(20) = -3.83349e-01	b(20) = -5.50490e-01
a(21) = -3.84267e-01	b(21) = -5.15825e-01
a(22) = -3.85061e-01	b(22) = -4.83869e-01
a(23) = -3.85750e-01	b(23) = -4.54261e-01
a(24) = -3.86353e-01	b(24) = -4.26700e-01
a(25) = -3.86883e-01	b(25) = -4.00933e-01
a(26) = -3.87350e-01	b(26) = -3.76744e-01
a(27) = -3.87764e-01	b(27) = -3.53951e-01
a(28) = -3.88131e-01	b(28) = -3.32395e-01
a(29) = -3.88459e-01	b(29) = -3.11940e-01
a(30) = -3.88751e-01	b(30) = -2.92466e-01
a(31) = -3.89013e-01	b(31) = -2.73870e-01
a(32) = -3.89247e-01	b(32) = -2.56060e-01
a(33) = -3.89457e-01	b(33) = -2.38954e-01
a(34) = -3.89646e-01	b(34) = -2.22482e-01
a(35) = -3.89816e-01	b(35) = -2.06576e-01
a(36) = -3.89968e-01	b(36) = -1.91181e-01
a(37) = -3.90104e-01	b(37) = -1.76243e-01
a(38) = -3.90226e-01	b(38) = -1.61714e-01
a(39) = -3.90335e-01	b(39) = -1.47550e-01
a(40) = -3.90431e-01	b(40) = -1.33713e-01
a(41) = -3.90517e-01	b(41) = -1.20164e-01
a(42) = -3.90592e-01	b(42) = -1.06869e-01
a(43) = -3.90657e-01	b(43) = -9.37966e-02
a(44) = -3.90713e-01	b(44) = -8.09159e-02
a(45) = -3.90760e-01	b(45) = -6.81984e-02
a(46) = -3.90799e-01	b(46) = -5.56170e-02
a(47) = -3.90829e-01	b(47) = -4.31454e-02
a(48) = -3.90852e-01	b(48) = -3.07583e-02
a(49) = -3.90867e-01	b(49) = -1.84311e-02
a(50) = -3.90874e-01	b(50) = -6.13973e-03

11.2 快速傅里叶变换

一、功能

用FFT计算离散傅里叶(Fourier)变换。

二、方法说明

计算 n 个采样点

$$P = \{p_0, p_1, \dots, p_{n-1}\}$$

的离散傅里叶变换, 可以归结为计算多项式

$$F(x) = p_0 + p_1x + p_2x^2 + \dots + p_{n-1}x^{n-1}$$

在各 n 次单位根 $1, w, w^2, \dots, w^{n-1}$ 上的值, 即

$$F_0 = p_0 + p_1 + p_2 + \dots + p_{n-1}$$

$$F_1 = p_0 + p_1w + p_2w^2 + \dots + p_{n-1}w^{n-1}$$

$$F_2 = p_0 + p_1w^2 + p_2(w^2)^2 + \dots + p_{n-1}(w^2)^{n-1}$$

⋮

$$F_{n-1} = p_0 + p_1w^{n-1} + p_2(w^{n-1})^2 + \dots + p_{n-1}(w^{n-1})^{n-1}$$

其中

$$w = e^{-\frac{2\pi}{n}}$$

为 n 次单位元根。

若 n 是 2 的 k 次幂, 即 $n = 2^k (k > 0)$ 。则 $F(x)$ 可以分解为关于 x 的偶次幂和奇次幂两部分, 即

$$F(x) = p_0 + p_2x^2 + \dots + p_{n-2}x^{n-2} + x(p_1 + p_3x^2 + \dots + p_{n-1}x^{n-2})$$

若令

$$P_{even}(x^2) = p_0 + p_2x^2 + \dots + p_{n-2}x^{n-2}$$

$$P_{odd}(x^2) = p_1 + p_3x^2 + \dots + p_{n-1}x^{n-2}$$

则有

$$F(x) = P_{even}(x^2) + xP_{odd}(x^2)$$

并且有

$$F(-x) = P_{even}(x^2) - xP_{odd}(x^2)$$

由此可以看出, 为了求 F 在各 n 次单位根上的值, 只需求 P_{even} 和 P_{odd} 在 $1, w^2, \dots, (w^{(n/2)-1})^2$ 上的值就可以了。

而 P_{even} 和 P_{odd} 同样可以分解成关于 x^2 的偶次幂和奇次幂两部分。以此类推, 一直分解下去, 最后可归结为只需求二次单位根 1 及 -1 上的值。

在实际计算时, 可以将上述过程倒过来进行, 这就是 FFT 算法。

三、函数语句

```
void kkfft(pr, pi, n, k, fr, fi, l, il)
```

四、形参说明

pr——双精度实型一维数组, 长度为 n 。当 $l=0$ 时, 存放 n 个采样输入的实部, 返回时存放离散傅里叶变换的模; 当 $l=1$ 时, 存放傅里叶变换的 n 个实部, 返回时存放逆傅里叶变换的模。

pi——双精度实型一维数组, 长度为 n 。当 $l=0$ 时, 存放 n 个采样输入的虚部, 返回时存放离散傅里叶变换的幅角; 当 $l=1$ 时, 存放傅里叶变换的 n 个虚部, 返回时存放逆傅里

叶变换的幅角。其中幅角的单位为度。

n——整型变量。输入的点数。

k——整型变量。满足 $n = 2^k$ 。

fr——双精度实型一维数组, 长度为 n 。当 $l=0$ 时, 返回傅里叶变换的实部; 当 $l=1$ 时, 返回逆傅里叶变换的实部。

fi——双精度实型一维数组, 长度为 n 。当 $l=0$ 时, 返回傅里叶变换的虚部; 当 $l=1$ 时, 返回逆傅里叶变换的虚部。

l——整型变量。若 $l=0$, 表示要求本函数计算傅里叶变换; 若 $l=1$, 表示要求本函数计算逆傅里叶变换。

il——整型变量。若 $il=0$, 表示不要求本函数计算傅里叶变换或逆变换的模与幅角; 若 $il=1$, 表示要求本函数计算傅里叶变换或逆变换的模与幅角。

五、函数程序(文件名:kkfft.c)

六、例

设函数 $p(t) = e^{-t}, t \geq 0$, 取 $n = 64, k = 6$, 周期 $T = 6.4$, 步长为 $h = T/n = 0.1$ 。采样序列为

$$p_i = p((i + 0.5)h), i = 0, 1, \dots, 63$$

计算:

- (1) p_i 的离散傅里叶变换 f_i , 且求出 f_i 的模与幅角。即取 $l=0, il=1$ 。
- (2) f_i 的逆傅里叶变换 p_i 的模。即取 $l=1, il=1$ 。

主函数程序(文件名:kkfft0.c)如下:

```
#include "stdio.h"
#include "kkfft.c"
#include "math.h"
main()
{ int i, j;
  double pr[64], pi[64], fr[64], fi[64];
  for (i = 0; i < 63; i++)
    { pr[i] = exp(-0.1 * (i + 0.5)); pi[i] = 0.0; }
  printf("\n");
  for (i = 0; i < 15; i++)
    { for (j = 0; j < 3; j++)
        printf("%e ", pr[4 * i + j]);
      printf("\n");
    }
  printf("\n");
  kkfft(pr, pi, 64, 6, fr, fi, 0, 1);
  for (i = 0; i < 15; i++)
    { for (j = 0; j < 3; j++)
        printf("%e ", fr[4 * i + j]);
      printf("\n");
    }
}
```

```

printf("\n");
for (i=0; i<=15; i++)
    { for (j=0; j<=3; j++)
        printf("%e ",fi[4*i+j]);
    printf("\n");
}
printf("\n");
for (i=0; i<=15; i++)
    { for (j=0; j<=3; j++)
        printf("%e ",pr[4*i+j]);
    printf("\n");
}
printf("\n");
for (i=0; i<=15; i++)
    { for (j=0; j<=3; j++)
        printf("%e ",pi[4*i+j]);
    printf("\n");
}
printf("\n");
kkfft(fr,fi,64,6,pr,pi,1,1);
for (i=0; i<=15; i++)
    { for (j=0; j<=3; j++)
        printf("%e ",fr[4*i+j]);
    printf("\n");
}
printf("\n");

```

运行结果为：

序列 p_i ($i = 0, 1, \dots, 63$):

9.51229e-01	8.60708e-01	7.78801e-01	7.04688e-01
6.37628e-01	5.76950e-01	5.22046e-01	4.72367e-01
4.27415e-01	3.86741e-01	3.49938e-01	3.16637e-01
2.86505e-01	2.59240e-01	2.34570e-01	2.12248e-01
1.92050e-01	1.73774e-01	1.57237e-01	1.42274e-01
1.28735e-01	1.16484e-01	1.05399e-01	9.53692e-02
8.62936e-02	7.80817e-02	7.06512e-02	6.39279e-02
5.78443e-02	5.23397e-02	4.73589e-02	4.28521e-02
3.87742e-02	3.50844e-02	3.17456e-02	2.87246e-02
2.59911e-02	2.35177e-02	2.12797e-02	1.92547e-02
1.74224e-02	1.57644e-02	1.42642e-02	1.29068e-02
1.16786e-02	1.05672e-02	9.56160e-03	8.65170e-03
7.82838e-03	7.08341e-03	6.40933e-03	5.79940e-03
5.24752e-03	4.74815e-03	4.29630e-03	3.88746e-03
3.51752e-03	3.18278e-03	2.87990e-03	2.60584e-03
2.35786e-03	2.13348e-03	1.93045e-03	1.74675e-03

p_i 的傅里叶变换 f_i ($i = 0, 1, \dots, 63$) 的实部:

9.97923e+00	5.31844e+00	2.43865e+00	1.46438e+00
1.06110e+00	8.61244e-01	7.48900e-01	6.79837e-01
6.34482e-01	6.03157e-01	5.80650e-01	5.63957e-01
5.51251e-01	5.41370e-01	5.33549e-01	5.27264e-01
5.22149e-01	5.17944e-01	5.14456e-01	5.11543e-01
5.09098e-01	5.07039e-01	5.05301e-01	5.03836e-01
5.02604e-01	5.01574e-01	5.00723e-01	5.00030e-01
4.99482e-01	4.99067e-01	4.98775e-01	4.98603e-01
4.98546e-01	4.98603e-01	4.98775e-01	4.99067e-01
4.99482e-01	5.00030e-01	5.00723e-01	5.01574e-01
5.02604e-01	5.03836e-01	5.05301e-01	5.07039e-01
5.09098e-01	5.11543e-01	5.14456e-01	5.17944e-01
5.22149e-01	5.27264e-01	5.33549e-01	5.41370e-01
5.51251e-01	5.63957e-01	5.80650e-01	6.03157e-01
6.34482e-01	6.79837e-01	7.48900e-01	8.61244e-01
1.06110e+00	1.46438e+00	2.43865e+00	5.31844e+00

p_i 的傅里叶变换 f_i ($i = 0, 1, \dots, 63$) 的虚部:

0.00000e+00	-4.73967e+00	-3.82485e+00	-2.86773e+00
-2.23986e+00	-1.81854e+00	-1.52015e+00	-1.29842e+00
-1.12707e+00	-9.90376e-01	-8.78443e-01	-7.84768e-01
-7.04911e-01	-6.35744e-01	-5.75000e-01	-5.20998e-01
-4.72460e-01	-4.28403e-01	-3.88053e-01	-3.50793e-01
-3.16123e-01	-2.83634e-01	-2.52985e-01	-2.23890e-01
-1.96104e-01	-1.69417e-01	-1.43644e-01	-1.18622e-01
-9.42034e-02	-7.02539e-02	-4.66483e-02	-2.32683e-02
0.00000e+00	2.32683e-02	4.66483e-02	7.02539e-02
9.42034e-02	1.18622e-01	1.43644e-01	1.69417e-01
1.96104e-01	2.23890e-01	2.52985e-01	2.83634e-01
3.16123e-01	3.50793e-01	3.88053e-01	4.28403e-01
4.72460e-01	5.20998e-01	5.75000e-01	6.35744e-01
7.04911e-01	7.84768e-01	8.78443e-01	9.90376e-01
1.12707e+00	1.29842e+00	1.52015e+00	1.81854e+00
2.23986e+00	2.86773e+00	3.82485e+00	4.73967e+00

p_i 的傅里叶变换 f_i ($i = 0, 1, \dots, 63$) 的模:

9.97923e+00	7.12393e+00	4.53613e+00	3.21998e+00
2.47849e+00	2.01217e+00	1.69461e+00	1.46563e+00
1.29339e+00	1.15959e+00	1.05300e+00	9.66389e-01
8.94861e-01	8.35016e-01	7.84410e-01	7.41246e-01
7.04172e-01	6.72157e-01	6.44399e-01	6.20268e-01
5.99262e-01	5.80979e-01	5.65094e-01	5.51341e-01
5.39507e-01	5.29414e-01	5.20919e-01	5.13908e-01
5.08288e-01	5.03987e-01	5.00952e-01	4.99146e-01
4.98546e-01	4.99146e-01	5.00952e-01	5.03987e-01
5.08288e-01	5.13908e-01	5.20919e-01	5.29414e-01
5.39507e-01	5.51341e-01	5.65094e-01	5.80979e-01
5.99262e-01	6.20268e-01	6.44399e-01	6.72157e-01

7.04172e-01	7.41246e-01	7.84410e-01	8.35016e-01
8.94861e-01	9.66389e-01	1.05300e+00	1.15959e+00
1.29339e+00	1.46563e+00	1.69461e+00	2.01217e+00
2.47849e+00	3.21998e+00	4.53613e+00	7.12393e+00

p_i 的傅里叶变换 f_i ($i = 0, 1, \dots, 63$) 的幅角(单位为度):

0.00000e+00	-4.17067e+01	-5.74792e+01	-6.29494e+01
-6.46513e+01	-6.46581e+01	-6.37729e+01	-6.23640e+01
-6.06228e+01	-5.86578e+01	-5.65353e+01	-5.42979e+01
-5.19741e+01	-4.95838e+01	-4.71414e+01	-4.46575e+01
-4.21400e+01	-3.95949e+01	-3.70272e+01	-3.44406e+01
-3.18381e+01	-2.92224e+01	-2.65954e+01	-2.39589e+01
-2.13145e+01	-1.86634e+01	-1.60068e+01	-1.33455e+01
-1.06807e+01	-8.01291e+00	-5.34308e+00	-2.67188e+00
0.00000e+00	2.67188e+00	5.34308e+00	8.01291e+00
1.06807e+01	1.33455e+01	1.60068e+01	1.86634e+01
2.13145e+01	2.39589e+01	2.65954e+01	2.92224e+01
3.18381e+01	3.44406e+01	3.70272e+01	3.95949e+01
4.21400e+01	4.46575e+01	4.71414e+01	4.95838e+01
5.19741e+01	5.42979e+01	5.65353e+01	5.86578e+01
6.06228e+01	6.23640e+01	6.37729e+01	6.46581e+01
6.46513e+01	6.29494e+01	5.74792e+01	4.17067e+01

f_i 的逆傅里叶变换 p_i ($i = 0, 1, \dots, 63$) 的模:

9.51229e-01	8.60708e-01	7.78801e-01	7.04688e-01
6.37628e-01	5.76950e-01	5.22046e-01	4.72367e-01
4.27415e-01	3.86741e-01	3.49938e-01	3.16637e-01
2.86505e-01	2.59240e-01	2.34570e-01	2.12248e-01
1.92050e-01	1.73774e-01	1.57237e-01	1.42274e-01
1.28735e-01	1.16484e-01	1.05399e-01	9.53692e-02
8.62936e-02	7.80817e-02	7.06512e-02	6.39279e-02
5.78443e-02	5.23397e-02	4.73589e-02	4.28521e-02
3.87742e-02	3.50844e-02	3.17456e-02	2.87246e-02
2.59911e-02	2.35177e-02	2.12797e-02	1.92547e-02
1.74224e-02	1.57644e-02	1.42642e-02	1.29068e-02
1.16786e-02	1.05672e-02	9.56160e-03	8.65170e-03
7.82838e-03	7.08341e-03	6.40933e-03	5.79940e-03
5.24752e-03	4.74815e-03	4.29630e-03	3.88746e-03
3.51752e-03	3.18278e-03	2.87990e-03	2.60584e-03
2.35786e-03	2.13348e-03	1.93045e-03	1.74675e-03

11.3 快速沃什变换

一、功能

计算给定序列的沃什(Walsh)变换序列。

二、方法说明

设给定序列为 p_0, p_1, \dots, p_{n-1} , 其中 $n = 2^k$ ($k \geq 1$)。则沃什变换定义为

$$x_i = \sum_{j=0}^{n-1} W_{ij}^{(n)} p_j, \quad i = 0, 1, \dots, n-1$$

沃什变换可以看作是矩阵与向量的乘积, 即

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = W^{(n)} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

其中

$$W^{(n)} = \begin{bmatrix} W_{00}^{(n)} & W_{01}^{(n)} & \cdots & W_{0,n-1}^{(n)} \\ W_{10}^{(n)} & W_{11}^{(n)} & \cdots & W_{1,n-1}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n-1,0}^{(n)} & W_{n-1,1}^{(n)} & \cdots & W_{n-1,n-1}^{(n)} \end{bmatrix}$$

是一个特殊的矩阵, 其元素值只有两个, +1 与 -1。 $W^{(n)}$ 中的各元素有以下递推关系:

$$W^{(1)} = W_{00}^{(1)} = 1$$

$$W_{2i,j}^{(2n)} = (W_{it}^{(n)}, (-1)^i W_{it}^{(n)}), \quad t = 0, 1, \dots, n-1$$

$$W_{2i+1,j}^{(2n)} = (W_{it}^{(n)}, (-1)^{i+1} W_{it}^{(n)}), \quad t = 0, 1, \dots, n-1$$

$$i = 0, 1, \dots, n-1$$

若利用通常的矩阵相乘的方法, 计算沃什变换序列需要作 $n(n-1)$ 次加减法。本函数采用与快速傅里叶变换类似的方法, 加减法总次数为 $n \log_2 n$ 。

三、函数语句

void kkfwf(p, n, k, x)

四、形参说明

p——双精度实型一维数组, 长度为 n 。存放长度为 $n = 2^k$ 的给定输入序列。

n——整型变量。输入序列的长度。

k——整型变量。满足 $n = 2^k$ 。

x——双精度实型一维数组, 长度为 n 。返回输入序列 p_i ($i = 0, 1, \dots, n-1$) 的沃什变换序列。

五、函数程序(文件名:kkfwf.c)

六、例

设输入序列为

$$p_i = i + 1, i = 0, 1, \dots, 7$$

计算 p_i 的沃什变换序列 x_i ($i = 0, 1 \dots, 7$)。其中 $k = 3$, $n = 8$ 。

主函数程序(文件名:kkfwt 0.c)如下:

```
# include "stdio.h"
# include "kkfwt.c"
main()
{ int i;
  double p[8], x[8];
  for (i = 0; i <= 7; i++) p[i] = i + 1;
  kkfwt(p, 8, 3, x);
  printf("\n");
  for (i = 0; i <= 7; i++)
    printf("x(%2d) = %e\n", i, x[i]);
  printf("\n");
}
```

运行结果为:

```
x(0) = 3.60000e+01
x(1) = -1.60000e+01
x(2) = 0.00000e+00
x(3) = -8.00000e+00
x(4) = 0.00000e+00
x(5) = 0.00000e+00
x(6) = 0.00000e+00
x(7) = -4.00000e+00
```

11.4 五点三次平滑

一、功能

用五点三次平滑公式对等距点上的观测数据进行平滑。

二、方法说明

设已知 n 个等距点 $x_0 < x_1 < \dots < x_{n-1}$ 上的观测(或实验)数据为 y_0, y_1, \dots, y_{n-1} , 则可以在每个数据点的前后各取两个相邻的点, 用三次多项式

$$y = a_0 + a_1x + a_2x^2 + a_3x^3$$

进行逼近。

根据最小二乘原理确定出系数 a_0, a_1, a_2, a_3 , 最后可得到五点三次平滑公式如下:

$$\bar{y}_{i-2} = \frac{1}{70}(69y_{i-2} + 4y_{i-1} - 6y_i + 4y_{i+1} - y_{i+2})$$

$$\bar{y}_{i-1} = \frac{1}{35}(2y_{i-2} + 27y_{i-1} + 12y_i - 8y_{i+1} + 2y_{i+2})$$

$$\bar{y}_i = \frac{1}{35}(-3y_{i-2} + 12y_{i-1} + 17y_i + 12y_{i+1} - 3y_{i+2})$$

$$\bar{y}_{i+1} = \frac{1}{35}(2y_{i-2} - 8y_{i-1} + 12y_i + 27y_{i+1} + 2y_{i+2})$$

$$\bar{y}_{i+2} = \frac{1}{70}(-y_{i-2} + 4y_{i-1} - 6y_i + 4y_{i+1} + 69y_{i+2})$$

其中 \bar{y}_i 表示 y_i 的平滑值。

对于开始两点和最后两点分别由上述第 1, 2, 与 4, 5 公式进行平滑。本方法要求数据点数 $n \geq 5$ 。

三、函数语句

```
void kkspt(n, y, yy)
```

四、形参说明

n —整型变量。给定等距观测点的个数。要求 $n \geq 5$ 。

y —双精度实型一维数组, 长度为 n 。 n 个等距观测点上的观测数据。

yy —双精度实型一维数组, 长度为 n 。返回 n 个等距观测点上的平滑结果。

五、函数程序(文件名:kkspt.c)

六、例

设 9 个等距观测点上的观测数据 y 为 54.0, 145.0, 227.0, 359.0, 401.0, 342.0, 259.0, 112.0, 65.0, 用五点三次平滑公式对此 9 个观测数据进行平滑。

主函数程序(文件名:kkspt0.c)如下:

```
# include "stdio.h"
# include "kkspt.c"
main()
{ int i;
  static double y[9] = {54.0, 145.0, 227.0, 359.0, 401.0,
                        342.0, 259.0, 112.0, 65.0};
  double yy[9];
  kkspt(9, y, yy);
  printf("\n");
  for (i = 0; i <= 8; i++)
    printf("y(%2d) = %e      yy(%2d) = %e\n", i, y[i], i, yy[i]);
  printf("\n");
}
```

运行结果为:

y(0) = 5.40000e+01	yy(0) = 5.68429e+01
y(1) = 1.45000e+02	yy(1) = 1.33629e+02
y(2) = 2.27000e+02	yy(2) = 2.44057e+02
y(3) = 3.59000e+02	yy(3) = 3.47943e+02
y(4) = 4.01000e+02	yy(4) = 3.93457e+02
y(5) = 3.42000e+02	yy(5) = 3.52029e+02
y(6) = 2.59000e+02	yy(6) = 2.41514e+02
y(7) = 1.12000e+02	yy(7) = 1.23657e+02
y(8) = 6.50000e+01	yy(8) = 6.20857e+01

11.5 离散随机线性系统的卡尔曼滤波

一、功能

对时间离散点上的采样数据进行卡尔曼(Kalman)滤波。

二、方法说明

设 n 维线性动态系统与 m 维线性观测系统由下列差分方程组描述：

$$\begin{cases} X_k = \Phi_{k,k-1}X_{k-1} + W_{k-1}, & k = 1, 2, \dots \\ Y_k = H_kX_k + V_k \end{cases}$$

其中 X_k 为 n 维向量，表示系统在第 k 时刻的状态。

$\Phi_{k,k-1}$ 是一个 $n \times n$ 阶矩阵，称为系统的状态转移矩阵，它反映了系统从第 $k-1$ 个采样时刻的状态到第 k 个采样时刻的状态的变换。

W_k 是一个 n 维向量，表示在第 k 时刻作用于系统的随机干扰，称为模型噪声。为简单起见，一般假设 $\{W_k\}$ ，($k=1, 2, \dots$) 为高斯白噪声序列，具有已知的零均值和协方差阵 Q_k 。

Y_k 为 m 维的观测向量。

H_k 为 $m \times n$ 阶的观测矩阵，表示了从状态量 X_k 到观测量 Y_k 的转换。

V_k 为 m 维的观测噪声，同样假设 $\{V_k\}$ ($k=1, 2, \dots$) 为高斯白噪声序列，具有已知的零均值和协方差阵 R_k 。

经推导(推导过程略)，可得到如下滤波的递推公式：

$$\begin{aligned} G_k &= P_k H_k^T [H_k P_k H_k^T + R_k]^{-1} \\ \tilde{X}_k &= \Phi_{k,k-1} \tilde{X}_{k-1} + G_k [Y_k - H_k \Phi_{k,k-1} \tilde{X}_{k-1}] \\ C_k &= (I - G_k H_k) P_k \\ P_{k+1} &= \Phi_{k+1,k} C_k \Phi_{k+1,k}^T + Q_k \end{aligned}$$

式中 Q_k 为 $n \times n$ 阶的模型噪声 W_k 的协方差阵；

R_k 为 $m \times m$ 阶的观测噪声 V_k 的协方差阵；

G_k 为 $n \times m$ 阶的增益矩阵；

\tilde{X}_k 为 n 维向量，第 k 时刻经滤波后的估值；

C_k 为 $n \times n$ 阶的估计误差协方差阵。

根据上述公式，可以从 $\tilde{X}_0 = E[X_0]$ ， P_0 (给定)出发，利用已知的矩阵 Q_k ， R_k ， H_k ， $\Phi_{k,k-1}$ 以及 k 时刻的观测值 Y_k ，递推地算出每个时刻的状态估计 \tilde{X}_k ($k=1, 2, \dots$)。

如果线性系统是定常的，则有 $\Phi_{k,k-1} = \Phi$ ， $H_k = H$ ，即它们都是常阵；如果模型噪声 W_k 和观测噪声 V_k 都是平稳随机序列，则 Q_k 和 R_k 都是常阵。在这种情况下，常增益的离散卡尔曼滤波是渐近稳定的。

三、函数语句

int klman(n, m, k, f, q, r, h, y, x, p, g)

本函数返回一个整型标志。若返回标志值为 0，则说明求增益矩阵过程中求逆失败，并在求逆函数中输出信息“err * * not inv”；若返回标志值不为 0，则说明正常返回。

本函数要调用全选主元实矩阵求逆的函数 brinv，参看 2.3 节。

四、形参说明

n ——整型变量。动态系统的维数。

m ——整型变量。观测系统的维数。

k ——整型变量。观测序列的长度。

f ——双精度实型二维数组，体积为 $n \times n$ 。系统状态转移矩阵。

q ——双精度实型二维数组，体积为 $n \times n$ 。模型噪声 W_k 的协方差阵。

r ——双精度实型二维数组，体积为 $m \times m$ 。观测噪声 V_k 的协方差阵。

h ——双精度实型二维数组，体积为 $m \times n$ 。观测矩阵。

y ——双精度实型二维数组，体积为 $k \times m$ 。观测向量序列。其中 $y(i, j)$ ($i=0, 1, \dots, k-1$ ； $j=0, 1, \dots, m-1$) 表示第 i 时刻的观测向量的第 j 个分量。

x ——双精度实型二维数组，体积为 $k \times n$ 。其中 $x(0, j)$ ($j=0, 1, \dots, n-1$) 存放给定的初值，其余各行返回状态向量估值序列。 $x(i, j)$ ($i=0, 1, \dots, k-1$ ； $j=0, 1, \dots, n-1$) 表示第 i 时刻的状态向量估值的第 j 个分量。

p ——双精度实型二维数组，体积为 $n \times n$ 。存放初值 P_0 ，返回时存放最后时刻的估计误差协方差阵。

g ——双精度实型二维数组，体积为 $n \times m$ 。返回最后时刻的稳定增益矩阵。

五、函数程序(文件名:klman.c)

六、例

设信号源运动方程为 $s(t) = 5 - 2t + 3t^2 + v(t)$ ，其中 $v(t)$ 是一个均值为零、方差为 0.25 的高斯白噪声。

状态向量为 $X_k = (s, s', s'')^T$ ，并取初值

$$X_0 = (0, 0, 0)^T$$

状态转移矩阵为

$$F = \Phi_{k,k-1} = \begin{bmatrix} 1 & T & \frac{T^2}{2} \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}$$

其中 T 为采样间隔，在本例中取 $T=0.05$ ，即

$$F = \begin{bmatrix} 1 & 0.05 & 0.00125 \\ 0 & 1 & 0.05 \\ 0 & 0 & 1 \end{bmatrix}$$

动态系统维数为 $n=3$ ，观测系统维数为 $m=1$ 。

模型噪声协方差阵取为

$$Q = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$$

观测矩阵为 $H = (1, 0, 0)$ 。

观测噪声协方差阵为 $R = 0.25$ 。

初始估计误差协方差阵取为

$$\hat{P}_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

取观测向量序列长度为 $k = 200$ 。

对输出的状态向量序列每隔 5 个时刻打印一次。

主函数程序(文件名:klman 0.c)如下:

```
#include "stdio.h"
#include "klman.c"
#include "brinv.c"
main()
{
    int i, j, js;
    void keklm5();
    double p[3][3], x[200][3], y[200][1], g[3][1], t, s;
    static double f[3][3] = {{1.0, 0.05, 0.00125},
                           {0.0, 1.0, 0.05}, {0.0, 0.0, 1.0}};
    static double q[3][3] = {{0.25, 0.0, 0.0},
                           {0.0, 0.25, 0.0}, {0.0, 0.0, 0.25}};
    static double r[1][1] = {0.25};
    static double h[1][3] = {1.0, 0.0, 0.0};
    for (i = 0; i <= 2; i++)
        for (j = 0; j <= 2; j++) p[i][j] = 0.0;
    for (i = 0; i < 199; i++)
        for (j = 0; j <= 2; j++) x[i][j] = 0.0;
    keklm5(200, y);
    for (i = 0; i < 199; i++)
    {
        t = 0.05 * i;
        y[i][0] = 5.0 - 2.0 * t + 3.0 * t * t + y[i][0];
    }
    js = klman(3, 1, 200, f, q, r, h, y, x, p, g);
    if (js != 0)
    {
        printf("\n");
        printf(" t      s      y\n");
        printf("x(0)      x(1)      x(2)      \n");
        for (i = 0; i < 199; i += 5)
        {
            t = 0.05 * i; s = 5.0 - 2.0 * t + 3.0 * t * t;
            printf("%6.2f %e %e %e %e\n",
                   t, s, y[i][0], x[i][0], x[i][1], x[i][2]);
        }
        printf("\n");
    }
}
```

```
static void keklm5(n, y)
int n;
double y[];
int i, j, m;
double s, w, v, r, t;
s = 65536.0; w = 2053.0; v = 13849.0; r = 0.0;
for (i = 0; i <= n - 1; i++)
{
    to = 0.0;
    for (j = 0; j <= 11; j++)
    {
        r = w * r + v; m = r / s; r = r - m * s; t = t + r / s;
        y[i] = 0.5 * (t - 6.0);
    }
}
return;
```

其中函数 keklm5() 为产生均值为零、方差为 0.5^2 的高斯白噪声序列。

运行结果为:(其中:t 为采样时刻值, s 为真值, y 为迭加有高斯白噪声的采样值,
 $x(0), x(1), x(2)$ 为状态向量各分量的估值。)

t	s	y	x(0)	x(1)	x(2)
0.00	5.00000e+00	4.45366e+00	0.00000e+00	0.00000e+00	0.00000e+00
0.25	4.68750e+00	5.79460e+00	5.50856e+00	7.60676e-02	4.84784e-03
0.50	4.75000e+00	4.76250e+00	4.72830e+00	2.17265e-02	5.15474e-03
0.75	5.18750e+00	5.13860e+00	5.10419e+00	1.30927e-01	3.07622e-02
1.00	6.00000e+00	6.20415e+00	5.94252e+00	5.10189e-01	1.55950e-01
1.25	7.18750e+00	5.74040e+00	6.21737e+00	5.81884e-01	1.51985e-01
1.50	8.75000e+00	9.02861e+00	8.74250e+00	2.16299e+00	7.78554e-01
1.75	1.06875e+01	1.03500e+01	1.00942e+01	2.97607e+00	1.06118e+00
2.00	1.30000e+01	1.34859e+01	1.30311e+01	5.00237e+00	1.86290e+00
2.25	1.56875e+01	1.62175e+01	1.61832e+01	7.10706e+00	2.64510e+00
2.50	1.87500e+01	1.93260e+01	1.90708e+01	8.71490e+00	3.10305e+00
2.75	2.21875e+01	2.20927e+01	2.18146e+01	9.95167e+00	3.33751e+00
3.00	2.60000e+01	2.57989e+01	2.57681e+01	1.21210e+01	4.00507e+00
3.25	3.01875e+01	3.02258e+01	3.02154e+01	1.43706e+01	4.61651e+00
3.50	3.47500e+01	3.46546e+01	3.46969e+01	1.62706e+01	4.99304e+00
3.75	3.96875e+01	4.03666e+01	4.01881e+01	1.87946e+01	5.63874e+00
4.00	4.50000e+01	4.41431e+01	4.44858e+01	1.96295e+01	5.35001e+00
4.25	5.06875e+01	5.07653e+01	5.07502e+01	2.21203e+01	5.91025e+00
4.50	5.67500e+01	5.75145e+01	5.71867e+01	2.43034e+01	6.25666e+00
4.75	6.31875e+01	6.31718e+01	6.31480e+01	2.55870e+01	6.12990e+00
5.00	7.00000e+01	6.95186e+01	6.96816e+01	2.70223e+01	6.06045e+00
5.25	7.71875e+01	7.73361e+01	7.73050e+01	2.92205e+01	6.40472e+00
5.50	8.47500e+01	8.49056e+01	8.51194e+01	3.11391e+01	6.56671e+00
5.75	9.26875e+01	9.20082e+01	9.23158e+01	3.19670e+01	6.14893e+00
6.00	1.01000e+02	1.01925e+02	1.01513e+02	3.45542e+01	6.69288e+00
6.25	1.09688e+02	1.09438e+02	1.09423e+02	3.53232e+01	6.25664e+00

6.50	1.18750e+02	1.18328e+02	1.18296e+02	3.67581e+01	6.20321e+00
6.75	1.28188e+02	1.27376e+02	1.27812e+02	3.83698e+01	6.19835e+00
7.00	1.38000e+02	1.37863e+02	1.37813e+02	4.01287e+01	6.30017e+00
7.25	1.48188e+02	1.48071e+02	1.48002e+02	4.16663e+01	6.28210e+00
7.50	1.58750e+02	1.58282e+02	1.58346e+02	4.29461e+01	6.12585e+00
7.75	1.69688e+02	1.69275e+02	1.69189e+02	4.43603e+01	6.05294e+00
8.00	1.81000e+02	1.80833e+02	1.80914e+02	4.62460e+01	6.20345e+00
8.25	1.92688e+02	1.92736e+02	1.92655e+02	4.77988e+01	6.21242e+00
8.50	2.04750e+02	2.04766e+02	2.04750e+02	4.93236e+01	6.20868e+00
8.75	2.17188e+02	2.18205e+02	2.17801e+02	5.14332e+01	6.50329e+00
9.00	2.30000e+02	2.29333e+02	2.29441e+02	5.16644e+01	5.81859e+00
9.25	2.43188e+02	2.42432e+02	2.42840e+02	5.33595e+01	5.90891e+00
9.50	2.56750e+02	2.55783e+02	2.55953e+02	5.43816e+01	5.65976e+00
9.75	2.70688e+02	2.70666e+02	2.70645e+02	5.66564e+01	6.06082e+00

11.6 α - β - γ 滤波

一、功能

对等间隔的量测数据进行滤波估值。

二、方法说明

设一个过程的量测数据为 $X^*(t)$, 且

$$X^*(t) = X(t) + \eta(t)$$

其中: $X(t)$ 为有用信号的准确值; $\eta(t)$ 是均值为零的白噪声过程, 即有

$$E\{\eta(t)\} = 0$$

$$E\{\eta(t)\eta(\tau)\} = \gamma\delta(t - \tau)$$

采用 α - β - γ 滤波方法对量测数据序列 X^* 进行估值的计算公式如下。

由上时刻对本时刻的一步预测估值公式为:

$$\tilde{X}_{n+1/n} = \tilde{X}_n + \tilde{X}'_n T + \tilde{X}''_n (T^2/2)$$

$$\tilde{X}'_{n+1/n} = \tilde{X}'_n + \tilde{X}''_n T$$

$$\tilde{X}''_{n+1/n} = \tilde{X}''_n$$

本时刻的滤波估值公式为

$$\tilde{X}_{n+1} = \tilde{X}_{n+1/n} + \alpha(X_{n+1}^* - \tilde{X}_{n+1/n})$$

$$\tilde{X}'_{n+1} = \tilde{X}'_{n+1/n} + \frac{\beta}{T}(X_{n+1}^* - \tilde{X}_{n+1/n})$$

$$\tilde{X}''_{n+1} = \tilde{X}''_{n+1/n} + \frac{2\gamma}{T^2}(X_{n+1}^* - \tilde{X}_{n+1/n})$$

其中: X_{n+1}^* 为本时刻的量测值;

$\tilde{X}_{n+1/n}$ 为上时刻对本时刻的位置一步预测估值;

$\tilde{X}'_{n+1/n}$ 为上时刻对本时刻的速度一步预测估值;

$\tilde{X}''_{n+1/n}$ 为上时刻对本时刻的加速度一步预测估值;

\tilde{X}_{n+1} 为本时刻的位置滤波估值;

\tilde{X}'_{n+1} 为本时刻的速度滤波估值;

\tilde{X}''_{n+1} 为本时刻的加速度滤波估值;

T 为采样间隔;

α, β, γ 为滤波器结构的参数。

三、函数语句

void kkabg(n, x, t, a, b, c, y)

四、形参说明

n——整型变量。量测数据的点数。

x——双精度实型一维数组, 长度为 n。n 个等间隔点上的量测值。

t——双精度实型变量。采样周期。

a——双精度实型变量。滤波器结构参数 α 。

b——双精度实型变量。滤波器结构参数 β 。

c——双精度实型变量。滤波器结构参数 γ 。

y——双精度实型一维数组, 长度为 n。返回 n 个等间隔点上的滤波估值。

五、函数程序(文件名:kkabg.c)

六、例

设准确信号为 $z(t) = 3t^2 - 2t + 5$, 叠加上一个均值为零、方差为 0.5^2 的正态分布白噪声后, 以周期 $T = 0.04$ 采样 250 个点。

取 $\alpha = 0.271$, $\beta = 0.0285$, $\gamma = 0.0005$, 对此 250 个采样点进行滤波估值。

主函数程序(文件名:kkabg 0.c)如下:

```
#include "stdio.h"
# include "kkabg.c"
main()
{
    int i;
    void kfabg5();
    double x[250], y[250], z[250];
    double a, b, c, dt, t;
    a = 0.271; b = 0.0285; c = 0.0005; dt = 0.04;
    printf("\n");
    kfabg5(250, y);
    for (i = 0; i < 249; i++)
        t = (i + 1) * dt;
        z[i] = 3.0 * t * t - 2.0 * t + 5.0;
        x[i] = z[i] + y[i];
    }
    kkabg(250, x, dt, a, b, c, y);
    for (i = 0; i < 249; i = i + 5)
    }
```

```

    | t = (i + 1) * dt;
    printf("t = %.2f x(t) = %e y(t) = %e z(t) = %e\n",
           t, x[i], y[i], z[i]);
    |
    printf("\n");
}

static void kfabc5(n, y)
int n;
double y[];
| int i, j, m;
double s, w, v, r, t;
s = 65536.0; w = 2053.0; v = 13849.0; r = 0.0;
for (i = 0; i <= n - 1; i++)
    | t = 0.0;
    for (j = 0; j < 11; j++)
        | r = w * r + v; m = r / s; r = r - m * s; t = t + r / s;
        y[i] = 0.5 * (t - 6.0);
    |
    return;
}

```

其中函数 kfabc5()为产生均值为零、方差为 0.5² 的正态分布白噪声序列。

运行结果为:(其中:t 为采样时刻值, x(t)为迭加上白噪声后的信号值, y(t)为滤波估值, z(t)为准确信号值。)

t = 0.04	x(t) = 4.37846e+00	y(t) = 1.18656e+00	z(t) = 4.92480e+00
t = 0.24	x(t) = 5.79990e+00	y(t) = 4.51626e+00	z(t) = 4.69280e+00
t = 0.44	x(t) = 4.71330e+00	y(t) = 4.77019e+00	z(t) = 4.70080e+00
t = 0.64	x(t) = 4.89990e+00	y(t) = 5.25536e+00	z(t) = 4.94880e+00
t = 0.84	x(t) = 5.64095e+00	y(t) = 5.76577e+00	z(t) = 5.43680e+00
t = 1.04	x(t) = 4.71770e+00	y(t) = 6.20442e+00	z(t) = 6.16480e+00
t = 1.24	x(t) = 7.41141e+00	y(t) = 7.53452e+00	z(t) = 7.13280e+00
t = 1.44	x(t) = 8.00332e+00	y(t) = 8.32977e+00	z(t) = 8.34080e+00
t = 1.64	x(t) = 1.02747e+01	y(t) = 1.00646e+01	z(t) = 9.78880e+00
t = 1.84	x(t) = 1.20068e+01	y(t) = 1.20243e+01	z(t) = 1.14768e+01
t = 2.04	x(t) = 1.39808e+01	y(t) = 1.38508e+01	z(t) = 1.34048e+01
t = 2.24	x(t) = 1.54780e+01	y(t) = 1.53646e+01	z(t) = 1.55728e+01
t = 2.44	x(t) = 1.77797e+01	y(t) = 1.75315e+01	z(t) = 1.79808e+01
t = 2.64	x(t) = 2.06671e+01	y(t) = 2.03259e+01	z(t) = 2.06288e+01
t = 2.84	x(t) = 2.34214e+01	y(t) = 2.30730e+01	z(t) = 2.35168e+01
t = 3.04	x(t) = 2.73239e+01	y(t) = 2.64157e+01	z(t) = 2.66448e+01
t = 3.24	x(t) = 2.91559e+01	y(t) = 2.94384e+01	z(t) = 3.00128e+01
t = 3.44	x(t) = 3.36986e+01	y(t) = 3.32519e+01	z(t) = 3.36208e+01
t = 3.64	x(t) = 3.82333e+01	y(t) = 3.74397e+01	z(t) = 3.74688e+01
t = 3.84	x(t) = 4.15411e+01	y(t) = 4.13792e+01	z(t) = 4.15568e+01
t = 4.04	x(t) = 4.54034e+01	y(t) = 4.57688e+01	z(t) = 4.58848e+01
t = 4.24	x(t) = 5.06014e+01	y(t) = 5.05272e+01	z(t) = 5.04528e+01
t = 4.44	x(t) = 5.54164e+01	y(t) = 5.57505e+01	z(t) = 5.52608e+01

t = 4.64	x(t) = 5.96295e+01	y(t) = 6.06647e+01	z(t) = 6.03088e+01
t = 4.84	x(t) = 6.65221e+01	y(t) = 6.61841e+01	z(t) = 6.55968e+01
t = 5.04	x(t) = 7.08755e+01	y(t) = 7.15091e+01	z(t) = 7.11248e+01
t = 5.24	x(t) = 7.64707e+01	y(t) = 7.70208e+01	z(t) = 7.68928e+01
t = 5.44	x(t) = 8.20892e+01	y(t) = 8.32875e+01	z(t) = 8.29008e+01
t = 5.64	x(t) = 8.90121e+01	y(t) = 8.94037e+01	z(t) = 8.91488e+01
t = 5.84	x(t) = 9.55208e+01	y(t) = 9.57576e+01	z(t) = 9.56368e+01
t = 6.04	x(t) = 1.01896e+02	y(t) = 1.02262e+02	z(t) = 1.02365e+02
t = 6.24	x(t) = 1.08920e+02	y(t) = 1.08923e+02	z(t) = 1.09333e+02
t = 6.44	x(t) = 1.16373e+02	y(t) = 1.16353e+02	z(t) = 1.16541e+02
t = 6.64	x(t) = 1.24037e+02	y(t) = 1.23517e+02	z(t) = 1.23989e+02
t = 6.84	x(t) = 1.31693e+02	y(t) = 1.31055e+02	z(t) = 1.31677e+02
t = 7.04	x(t) = 1.40622e+02	y(t) = 1.39207e+02	z(t) = 1.39605e+02
t = 7.24	x(t) = 1.47106e+02	y(t) = 1.46836e+02	z(t) = 1.47773e+02
t = 7.44	x(t) = 1.55425e+02	y(t) = 1.55390e+02	z(t) = 1.56181e+02
t = 7.64	x(t) = 1.63861e+02	y(t) = 1.63839e+02	z(t) = 1.64829e+02
t = 7.84	x(t) = 1.73696e+02	y(t) = 1.73214e+02	z(t) = 1.73717e+02
t = 8.04	x(t) = 1.82710e+02	y(t) = 1.82652e+02	z(t) = 1.82845e+02
t = 8.24	x(t) = 1.92684e+02	y(t) = 1.92306e+02	z(t) = 1.92213e+02
t = 8.44	x(t) = 2.01401e+02	y(t) = 2.02024e+02	z(t) = 2.01821e+02
t = 8.64	x(t) = 2.11640e+02	y(t) = 2.12177e+02	z(t) = 2.11669e+02
t = 8.84	x(t) = 2.21685e+02	y(t) = 2.22391e+02	z(t) = 2.21757e+02
t = 9.04	x(t) = 2.31814e+02	y(t) = 2.32803e+02	z(t) = 2.32085e+02
t = 9.24	x(t) = 2.41811e+02	y(t) = 2.43234e+02	z(t) = 2.42653e+02
t = 9.44	x(t) = 2.53456e+02	y(t) = 2.54300e+02	z(t) = 2.53461e+02
t = 9.64	x(t) = 2.65031e+02	y(t) = 2.65446e+02	z(t) = 2.64509e+02
t = 9.84	x(t) = 2.75816e+02	y(t) = 2.76467e+02	z(t) = 2.75797e+02

第 12 章 特殊函数

12.1 伽马函数

一、功能

计算实变量 x 的伽马(Gamma)函数值

$$\Gamma(x) = \int_0^{\infty} e^{-t} t^{x-1} dt \quad (x > 0)$$

二、方法说明

当 $2 < x \leq 3$ 时, $\Gamma(x)$ 用下列切比雪夫(Chebyshev)多项式逼近

$$\Gamma(x) = \sum_{i=0}^{10} a_i (x-2)^{10-i}$$

其中

$$\begin{aligned} a_0 &= 0.0000677106, \quad a_1 = -0.0003442342 \\ a_2 &= 0.0015397681, \quad a_3 = -0.0024467480 \\ a_4 &= 0.0109736958, \quad a_5 = -0.0002109075 \\ a_6 &= 0.0742379071, \quad a_7 = 0.0815782188 \\ a_8 &= 0.4118402518, \quad a_9 = 0.4227843370 \\ a_{10} &= 1.0 \end{aligned}$$

当 $0 < x \leq 2$ 时, 利用公式

$$\Gamma(x) = \frac{1}{x} \Gamma(x+1), \quad 1 < x \leq 2$$

或

$$\Gamma(x) = \frac{1}{x(x+1)} \Gamma(x+2), \quad 0 < x \leq 1$$

当 $x > 3$ 时, 利用公式

$$\Gamma(x) = (x-1)(x-2)\cdots(x-i)\Gamma(x-i)$$

直到满足 $2 < (x-i) \leq 3$ 为止。

利用伽马函数可以计算贝塔(Beta)函数

$$B(x, y) = B(y, x) = \int_0^1 t^{x-1} (1-t)^{y-1} dt, \quad x, y > 0$$

其计算公式为

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

三、函数语句

double lgam1(x)

本函数返回一个双精度型的伽马函数值。

四、形参说明

x ——双精度型变量。自变量值。要求 $x > 0$ 。若 $x \leq 0$, 本函数返回函数值 -1, 且输出信息“err * * x ≤ 0!”。

五、函数程序(文件名:lgam1.c)

六、例

计算 0.5 到 5.0 之间每隔 0.5 的伽马函数值以及贝塔函数值 $B(1.5, 2.5)$ 。

主函数程序(文件名:lgam10.c)如下:

```
#include "stdio.h"
#include "lgam1.c"

main()
{
    int i;
    double x, y;
    printf("\n");
    for (i = 1; i <= 10; i++)
        | x = 0.5 * i; y = lgam1(x);
        | printf("x = %6.3f gamma(x) = %e\n", x, y);
    |
    printf("\n");
    y = lgam1(1.5) * lgam1(2.5) / lgam1(4.0);
    printf("B(1.5, 2.5) = %e\n", y);
    printf("\n");
}
```

运行结果为:

```
x = 0.500 gamma(x) = 1.77245e+00
x = 1.000 gamma(x) = 1.00001e+00
x = 1.500 gamma(x) = 8.86227e-01
x = 2.000 gamma(x) = 1.00001e+00
x = 2.500 gamma(x) = 1.32934e+00
x = 3.000 gamma(x) = 2.00002e+00
x = 3.500 gamma(x) = 3.32335e+00
x = 4.000 gamma(x) = 6.00006e+00
x = 4.500 gamma(x) = 1.16317e+01
x = 5.000 gamma(x) = 2.40002e+01
```

$B(1.5, 2.5) = 1.96348e-01$

• 285 •

• 286 •

12.2 不完全伽马函数

一、功能

计算不完全伽马函数(Incomplete Gamma function)的值。

二、方法说明

不完全伽马函数定义为

$$\Gamma(\alpha, x) = \frac{P(\alpha, x)}{\Gamma(\alpha)}, \alpha > 0, x > 0 \quad (1)$$

其中

$$P(\alpha, x) = \int_0^x e^{-t} t^{\alpha-1} dt$$

对于不完全伽马函数有

$$\Gamma(\alpha, 0) = 0, \Gamma(\alpha, \infty) = 1$$

不完全伽马函数也可表示为

$$\Gamma(\alpha, x) = 1 - \frac{Q(\alpha, x)}{\Gamma(\alpha)}, \alpha > 0, x > 0 \quad (2)$$

其中

$$Q(\alpha, x) = \int_x^\infty e^{-t} t^{\alpha-1} dt$$

(2) 式中的 $Q(\alpha, x)/\Gamma(\alpha)$ 也称为余不完全伽马函数。

当 $x < \alpha + 1.0$ 时, 用(1)式计算。其中 $P(\alpha, x)$ 用如下级数计算:

$$P(\alpha, x) = e^{-x} x^\alpha \sum_{k=0}^{\infty} \frac{\Gamma(\alpha)}{\Gamma(\alpha + 1 + k)} x^k$$

当 $x \geq \alpha + 1.0$ 时, 用(2)式计算。其中 $Q(\alpha, x)$ 用如下连分式计算

$$Q(\alpha, x) = e^{-x} x^\alpha \varphi(\alpha, x)$$
$$\varphi(\alpha, x) = \frac{1}{x + \frac{1 - \alpha}{1 + \frac{1}{x + \frac{2 - \alpha}{1 + \frac{2}{x + \dots + \frac{n - \alpha}{1 + \frac{n}{x + \dots}}}}}}$$

三、函数语句

double lgam2(a, x)

本函数返回一个双精度实型的不完全伽马函数值。

本函数要调用计算伽马函数值的函数 lgam1, 参看 12.1 节。

四、形参说明

a——双精度实型变量。自变量 a 的值。要求 $a > 0$ 。当 $a \leq 0$ 时, 本函数返回的函数值为 -1.0, 且输出信息“err * * a<=0!”。

x——双精度实型变量。自变量值。要求 $x > 0$ 。当 $x < 0$ 时, 本函数返回的函数值为 -1.0, 且输出信息“err * * x<0!”。

五、函数程序(文件名:lgam2.c)

六、例

计算当 $\alpha = 0.5, 5.0, 50.0$ 时, x 分别取 0.1, 1.0, 10.0 时的不完全伽马函数值 $\Gamma(\alpha, x)$ 。

主函数程序(文件名:lgam20.c)如下:

```
# include "stdio.h"
# include "lgam2.c"
# include "lgam1.c"
main()
{ int i, j;
  double y, s, t;
  static double a[3] = {0.5, 5.0, 50.0};
  static double x[3] = {0.1, 1.0, 10.0};
  printf("\n");
  for (i = 0; i < 2; i++)
    for (j = 0; j < 2; j++)
      { s = a[i]; t = x[j];
        y = lgam2(s, t);
        printf("gamma(%4.1f, %4.1f) = %e\n", a[i], x[j], y);
      }
  printf("\n");
}
```

运行结果为:

```
gamma( 0.5, 0.1) = 3.45279e-01
gamma( 0.5, 1.0) = 8.42701e-01
gamma( 0.5, 10.0) = 9.99992e-01
gamma( 5.0, 0.1) = 7.66773e-08
gamma( 5.0, 1.0) = 3.65981e-03
gamma( 5.0, 10.0) = 9.70748e-01
gamma( 50.0, 0.1) = 2.98087e-115
gamma( 50.0, 1.0) = 1.23374e-65
gamma( 50.0, 10.0) = 1.85471e-19
```

12.3 误差函数

一、功能

计算实变量 x 的误差函数值

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

二、方法说明

误差函数 $\text{erf}(x)$ 具有下列极限值及对称性:

$$\text{erf}(0) = 0, \text{erf}(\infty) = 1, \text{erf}(-x) = -\text{erf}(x)$$

当 $x \geq 0$ 时, 可以用不完全伽马函数计算误差函数, 即

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt = 1 - \text{erf}(x)$$

利用误差函数可以计算余误差函数

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt = 1 - \text{erf}(x)$$

也可以计算正态概率积分

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt = \frac{1}{2} + \frac{1}{2} \text{erf}\left(\frac{x}{\sqrt{2}}\right)$$

三、函数语句

double lerrf(x)

本函数返回一个双精度实型的误差函数值。

本函数要调用不完全伽马函数 lgam2, 参看 12.2 节。

四、形参说明

x——双精度实型变量。自变量值。

五、函数程序(文件名:lerrf.c)

六、例

计算 0 到 2 之间间隔为 0.05 的误差函数值。

主函数程序(文件名:lerrf 0.c)如下:

```
# include "stdio.h"
# include "lerrf.c"
# include "lgam2.c"
# include "lgam1.c"
main()
{
    int i, j;
    double x, y;
    printf("\n");
    for (i = 0; i <= 9; i++)
        for (j = 0, j <= 3; j++)
            x = 0.05 * (4.0 * i + j), y = lerrf(x);
            printf("erf(%4.2f) = %8.6f\n", x, y);
    printf("\n");
    x = 2.0, y = lerrf(x);
```

```
printf("erf(%4.2f) = %8.6f\n", x, y);
printf("\n");
```

运行结果为:

```
erf(0.00) = 0.000000 erf(0.05) = 0.056372 erf(0.10) = 0.112463 erf(0.15) = 0.167996
erf(0.20) = 0.222703 erf(0.25) = 0.276326 erf(0.30) = 0.328627 erf(0.35) = 0.379382
erf(0.40) = 0.428392 erf(0.45) = 0.475482 erf(0.50) = 0.520500 erf(0.55) = 0.563323
erf(0.60) = 0.603856 erf(0.65) = 0.642029 erf(0.70) = 0.677801 erf(0.75) = 0.711156
erf(0.80) = 0.742101 erf(0.85) = 0.770668 erf(0.90) = 0.796908 erf(0.95) = 0.820891
erf(1.00) = 0.842701 erf(1.05) = 0.862436 erf(1.10) = 0.880205 erf(1.15) = 0.896124
erf(1.20) = 0.910314 erf(1.25) = 0.922900 erf(1.30) = 0.934008 erf(1.35) = 0.943762
erf(1.40) = 0.952285 erf(1.45) = 0.959695 erf(1.50) = 0.966105 erf(1.55) = 0.971623
erf(1.60) = 0.976348 erf(1.65) = 0.980376 erf(1.70) = 0.983790 erf(1.75) = 0.986672
erf(1.80) = 0.989091 erf(1.85) = 0.991111 erf(1.90) = 0.992790 erf(1.95) = 0.994179
erf(2.00) = 0.995322
```

12.4 第一类整数阶贝塞耳函数

一、功能

计算实变量 x 的第一类整数阶贝塞耳(Bessel)函数值 $J_n(x)$ 。

二、方法说明

第一类贝塞耳函数的定义为

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(n+k)!} \left(\frac{x}{2}\right)^{2k}$$

其中 n 为非负整数。其积分表达式为

$$J_n(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \cos(nt - xs \sin t) dt$$

第一类整数阶贝塞耳函数具有下列递推关系

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x) \quad (1)$$

$J_0(x)$ 与 $J_1(x)$ 的计算公式如下:

当 $|x| < 8.0$ 时, $J_0(x) = \frac{A(y)}{B(y)}$, 其中 $y = x^2$, 且

$$A(y) = a_0 + a_1 y + a_2 y^2 + a_3 y^3 + a_4 y^4 + a_5 y^5$$

$$B(y) = b_0 + b_1 y + b_2 y^2 + b_3 y^3 + b_4 y^4 + b_5 y^5$$

其系数分别为

$$a_0 = 5.7568490574 \times 10^{10}, \quad a_1 = -1.3362590354 \times 10^{10}$$

$$a_2 = 6.516196407 \times 10^8, \quad a_3 = -1.121442418 \times 10^7$$

$$a_4 = 7.739233017 \times 10^4, \quad a_5 = -1.849052456 \times 10^2$$

$$b_0 = 5.7568490411 \times 10^{10}, \quad b_1 = 1.029532985 \times 10^9$$

$$\begin{aligned} b_2 &= 9.494680718 \times 10^6, & b_3 &= 5.927264853 \times 10^4 \\ b_4 &= 2.678532712 \times 10^2, & b_5 &= 1.0 \\ J_1(x) &= x \frac{C(y)}{D(y)} \end{aligned}$$

其中 $y = x * x$, 且

$$\begin{aligned} C(y) &= c_0 + c_1y + c_2y^2 + c_3y^3 + c_4y^4 + c_5y^5 \\ D(y) &= d_0 + d_1y + d_2y^2 + d_3y^3 + d_4y^4 + d_5y^5 \end{aligned}$$

其系数分别为:

$$\begin{aligned} c_0 &= 7.2362614232 \times 10^{10}, & c_1 &= -7.895059235 \times 10^9 \\ c_2 &= 2.423968531 \times 10^8, & c_3 &= -2.972611439 \times 10^6 \\ c_4 &= 1.570448260 \times 10^4, & c_5 &= -3.016036606 \times 10 \\ d_0 &= 1.44725228443 \times 10^{11}, & d_1 &= 2.300535178 \times 10^9 \\ d_2 &= 1.858330474 \times 10^7, & d_3 &= 9.944743394 \times 10^4 \\ d_4 &= 3.769991397 \times 10^2, & d_5 &= 1.0 \end{aligned}$$

当 $|x| \geq 8.0$ 时, 令 $z = 8.0 / |x|$, $y = z * z$,

$$\text{则 } J_0(x) = \sqrt{\frac{2}{\pi|x|}} [E(y)\cos\theta - zF(y)\sin\theta]$$

其中 $\theta = |x| - \frac{\pi}{4}$, 且

$$\begin{aligned} E(y) &= e_0 + e_1y + e_2y^2 + e_3y^3 + e_4y^4 \\ F(y) &= f_0 + f_1y + f_2y^2 + f_3y^3 + f_4y^4 \end{aligned}$$

其系数分别为:

$$\begin{aligned} e_0 &= 1.0, & e_1 &= -0.1098628627 \times 10^{-2} \\ e_2 &= 0.2734510407 \times 10^{-4}, & e_3 &= -0.2073370639 \times 10^{-5} \\ e_4 &= 0.2093887211 \times 10^{-6} \\ f_0 &= -0.1562499995 \times 10^{-1}, & f_1 &= 0.1430488765 \times 10^{-3} \\ f_2 &= -0.6911147651 \times 10^{-5}, & f_3 &= 0.7621095161 \times 10^{-6} \\ f_4 &= -0.934935152 \times 10^{-7} \end{aligned}$$

而且

$$\begin{aligned} J_1(x) &= \sqrt{\frac{2}{\pi|x|}} [G(y)\cos\theta - zH(y)\sin\theta], \quad x > 0 \\ J_1(-x) &= -J_1(x) \end{aligned}$$

其中 $\theta = |x| - \frac{3\pi}{4}$, 且

$$\begin{aligned} G(y) &= g_0 + g_1y + g_2y^2 + g_3y^3 + g_4y^4 \\ H(y) &= h_0 + h_1y + h_2y^2 + h_3y^3 + h_4y^4 \end{aligned}$$

其系数分别为:

$$g_0 = 1.0, \quad g_1 = 0.183105 \times 10^{-2}$$

$$\begin{aligned} g_2 &= -0.3516396496 \times 10^{-4}, & g_3 &= 0.2457520174 \times 10^{-5} \\ g_4 &= -0.240337019 \times 10^{-6}, & & \\ h_0 &= 0.4687499995 \times 10^{-1}, & h_1 &= -0.2002690873 \times 10^{-3} \\ h_2 &= 0.8449199096 \times 10^{-5}, & h_3 &= -0.88228987 \times 10^{-6} \\ h_4 &= 0.105787412 \times 10^{-6} \end{aligned}$$

当 $n \geq 2$ 时, 如果 $|x| > n$, 则可用递推公式(1)进行递推计算; 否则用下列递推关系

$$J_{n-1}(x) = \frac{2n}{x} J_n - J_{n+1}(x)$$

三、函数语句

double lssl1(n, x)

本函数返回一个双精度实型函数值 $J_n(x)$ 。

四、形参说明

n——整型变量。第一类贝塞耳函数的阶数。 $n \geq 0$ 。当 $n < 0$ 时, 本函数按 $|n|$ 计算。
x——双精度实型变量。自变量值。

五、函数程序(文件名:lssl1.c)

六、例

计算 $n = 0, 1, 2, 3, 4, 5$ 时, $x = 0.05, 0.5, 5.0, 50.0$ 时的 $J_n(x)$ 。

主函数程序(文件名:lssl10.c)如下:

```
#include "stdio.h"
#include "lssl1.c"
main()
{ int n, i;
  double x, y;
  printf("\n");
  for (n=0; n<=5; n++)
  { x=0.05;
    for (i=1; i<=4; i++)
    { y=lssl1(n, x);
      printf("n=%d x=%6.3f J(n, x)=%12.7f\n", n, x, y);
      x=x*10.0;
    }
    printf("\n");
  }
}
```

运行结果为:

$n=0$	$x=0.050$	$J(n, x)=0.9993751$
$n=0$	$x=0.500$	$J(n, x)=0.9384698$
$n=0$	$x=5.000$	$J(n, x)=-0.1775968$

n = 0	x = 50.000	J(n, x) =	0.0558123
n = 1	x = 0.050	J(n, x) =	0.0249922
n = 1	x = 0.500	J(n, x) =	0.2422685
n = 1	x = 5.000	J(n, x) =	-0.3275791
n = 1	x = 50.000	J(n, x) =	-0.0975118
n = 2	x = 0.050	J(n, x) =	0.0003124
n = 2	x = 0.500	J(n, x) =	0.0306040
n = 2	x = 5.000	J(n, x) =	0.0465651
n = 2	x = 50.000	J(n, x) =	-0.0597128
n = 3	x = 0.050	J(n, x) =	0.0000026
n = 3	x = 0.500	J(n, x) =	0.0025637
n = 3	x = 5.000	J(n, x) =	0.3648312
n = 3	x = 50.000	J(n, x) =	0.0927348
n = 4	x = 0.050	J(n, x) =	0.0000000
n = 4	x = 0.500	J(n, x) =	0.0001607
n = 4	x = 5.000	J(n, x) =	0.3912324
n = 4	x = 50.000	J(n, x) =	0.0708410
n = 5	x = 0.050	J(n, x) =	0.0000000
n = 5	x = 0.500	J(n, x) =	0.0000081
n = 5	x = 5.000	J(n, x) =	0.2611405
n = 5	x = 50.000	J(n, x) =	-0.0814002

12.5 第二类整数阶贝塞耳函数

一、功能

计算实变量 x 的第二类整数阶贝塞耳(Bessel)函数值 $Y_n(x)$ 。

二、方法说明

第二类整数阶贝塞耳函数具有如下递推关系

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x), \quad x \geq 0$$

这个递推公式是稳定的。

$$\text{当 } x < 8.0 \text{ 时, } Y_0(x) = \frac{A(y)}{B(y)} + \frac{2}{\pi} J_0(x) \ln x$$

其中 $y = x^2$, 且

$$A(y) = a_0 + a_1y + a_2y^2 + a_3y^3 + a_4y^4 + a_5y^5$$

$$B(y) = b_0 + b_1y + b_2y^2 + b_3y^3 + b_4y^4 + b_5y^5$$

其系数分别为:

$$\begin{aligned} a_0 &= -2.957821389 \times 10^9, & a_1 &= 7.062834065 \times 10^9 \\ a_2 &= -5.123598036 \times 10^8, & a_3 &= 1.087988129 \times 10^7 \\ a_4 &= -8.632792757 \times 10^4, & a_5 &= 2.284622733 \times 10^2 \\ b_0 &= 4.0076544269 \times 10^{10}, & b_1 &= 7.452499648 \times 10^8 \\ b_2 &= 7.189466438 \times 10^6, & b_3 &= 4.744726470 \times 10^4 \end{aligned}$$

$$b_4 = 2.261030244 \times 10^2, \quad b_5 = 1.0$$

而

$$Y_1(x) = x \frac{C(y)}{D(y)} + \frac{2}{\pi} \left[J_1(x) \ln x - \frac{1}{x} \right]$$

其中 $y = x^2$, 且

$$C(y) = c_0 + c_1y + c_2y^2 + c_3y^3 + c_4y^4 + c_5y^5$$

$$D(y) = d_0 + d_1y + d_2y^2 + d_3y^3 + d_4y^4 + d_5y^5 + d_6y^6$$

其系数分别为:

$$\begin{aligned} c_0 &= -4.900604943 \times 10^{12}, & c_1 &= 1.275274390 \times 10^{12} \\ c_2 &= -5.153438139 \times 10^{10}, & c_3 &= 7.349264551 \times 10^8 \\ c_4 &= -4.237922726 \times 10^6, & c_5 &= 8.511937935 \times 10^3 \\ d_0 &= 2.499580570 \times 10^{13}, & d_1 &= 4.244419664 \times 10^{11} \\ d_2 &= 3.733650367 \times 10^9 & d_3 &= 2.245904002 \times 10^7 \\ d_4 &= 1.020426050 \times 10^5, & d_5 &= 3.549632885 \times 10^2 \\ d_6 &= 1.0 & & \end{aligned}$$

当 $x \geq 8.0$ 时, 令 $z = 8.0/x, y = z * z$

$$\text{则 } Y_0(x) = \sqrt{\frac{2}{\pi x}} [E(y) \sin \theta + zF(y) \cos \theta]$$

其中 $\theta = x - \frac{\pi}{4}$, 且

$$E(y) = e_0 + e_1y + e_2y^2 + e_3y^3 + e_4y^4$$

$$F(y) = f_0 + f_1y + f_2y^2 + f_3y^3 + f_4y^4$$

其系数分别为:

$$\begin{aligned} e_0 &= 1.0, & e_1 &= -0.1098628627 \times 10^{-2} \\ e_2 &= 0.2734510407 \times 10^{-4}, & e_3 &= -0.2073370639 \times 10^{-5} \\ e_4 &= 0.2093887211 \times 10^{-6} & & \\ f_0 &= -0.1562499995 \times 10^{-1}, & f_1 &= 0.1430488765 \times 10^{-3} \\ f_2 &= -0.6911147651 \times 10^{-5} & f_3 &= 0.7621095161 \times 10^{-6} \\ f_4 &= -0.934945152 \times 10^{-7} & & \end{aligned}$$

而且

$$Y_1(x) = \sqrt{\frac{2}{\pi x}} [G(y) \sin \theta + zH(y) \cos \theta]$$

其中 $\theta = x - \frac{3\pi}{4}$, 且

$$G(y) = g_0 + g_1y + g_2y^2 + g_3y^3 + g_4y^4$$

$$H(y) = h_0 + h_1y + h_2y^2 + h_3y^3 + h_4y^4$$

其系数分别为:

$$g_0 = 1.0, \quad g_1 = 0.183105 \times 10^{-2}$$

```

 $g_2 = -0.3516396496 \times 10^{-4}, \quad g_3 = 0.2457520174 \times 10^{-5}$ 
 $g_4 = -0.240337019 \times 10^{-6}$ 
 $h_0 = 0.4687499995 \times 10^{-1}, \quad h_1 = -0.2002690873 \times 10^{-3}$ 
 $h_2 = 0.8449199096 \times 10^{-5}, \quad h_3 = -0.88228987 \times 10^{-6}$ 
 $h_4 = 0.105787412 \times 10^{-6}$ 

```

三、函数语句

double lssl2(n, x)

本函数返回一个双精度实型的函数值 $Y_n(x)$ 。

本函数要调用第一类整数阶贝塞耳函数 lssl1, 参看 12.4 节。

四、形参说明

n——整型变量。第二类贝塞耳函数的阶数。n≥0。当 n<0 时, 本函数按|n|计算。

x——双精度实型变量。自变量值。x≥0。当 x<0 时, 本函数按|x|计算。

五、函数程序(文件名:lssl2.c)

六、例

计算 n=0, 1, 2, 3, 4, 5 时, x=0.05, 0.5, 5.0, 50.0 时的 $Y_n(x)$ 。

主函数程序(文件名:lssl20.c)如下:

```

#include "stdio.h"
#include "lssl2.c"
#include "lssl1.c"
main()
{
    int n, i;
    double x, y;
    printf("n");
    for (n=0;n<=5; n++)
    {
        x=0.05;
        for (i=1; i<=4; i++)
        {
            y=lssl2(n, x);
            printf("n = %d x = %6.3f Y(n, x) = %e\n", n, x, y);
            x=x*10.0;
        }
    }
    printf("\n");
}

```

运行结果为:

```

n = 0 x = 0.050 Y(n, x) = -1.9793e+00
n = 0 x = 0.500 Y(n, x) = -4.44519e-01
n = 0 x = 5.000 Y(n, x) = -3.08518e-01
n = 0 x = 50.000 Y(n, x) = -9.80650e-02

```

```

n = 1 x = 0.050 Y(n, x) = -1.27899e+01
n = 1 x = 0.500 Y(n, x) = -1.47147e+00
n = 1 x = 5.000 Y(n, x) = 1.47863e-01
n = 1 x = 50.000 Y(n, x) = -5.67957e-02
n = 2 x = 0.050 Y(n, x) = -5.09615e+02
n = 2 x = 0.500 Y(n, x) = -5.44137e+00
n = 2 x = 5.000 Y(n, x) = 3.67663e-01
n = 2 x = 50.000 Y(n, x) = -9.57932e-02
n = 3 x = 0.050 Y(n, x) = -4.07564e+04
n = 3 x = 0.500 Y(n, x) = -4.20595e+01
n = 3 x = 5.000 Y(n, x) = 1.46267e-01
n = 3 x = 50.000 Y(n, x) = 6.44591e-02
n = 4 x = 0.050 Y(n, x) = -4.89026e+06
n = 4 x = 0.500 Y(n, x) = -4.99273e+02
n = 4 x = 5.000 Y(n, x) = -1.92142e-01
n = 4 x = 50.000 Y(n, x) = -8.80581e-02
n = 5 x = 0.050 Y(n, x) = -7.82401e+08
n = 5 x = 0.500 Y(n, x) = -7.94630e+03
n = 5 x = 5.000 Y(n, x) = -4.53695e-01
n = 5 x = 50.000 Y(n, x) = -7.85484e-02

```

12.6 变型第一类整数阶贝塞耳函数

一、功能

计算实变量 x 的变型第一类整数阶贝塞耳(Bessel)函数值 $I_n(x)$ 。

二、方法说明

变型第一类整数阶贝塞耳函数表示为

$$I_n(x) = (-j)^n J_n(jx)$$

其中 n 为非负整数, j 为虚数(即 $\sqrt{-1}$), $J_n(jx)$ 为纯虚变量(jx)的第一类贝塞耳函数。

$I_0(x)$ 与 $I_1(x)$ 的计算公式如下。

当 |x| < 3.75 时, 令 $y = \left(\frac{x}{3.75}\right)^2$, 则

$$I_0(x) = a_0 + a_1 y + a_2 y^2 + a_3 y^3 + a_4 y^4 + a_5 y^5 + a_6 y^6$$

$$I_1(x) = x(b_0 + b_1 y + b_2 y^2 + b_3 y^3 + b_4 y^4 + b_5 y^5 + b_6 y^6)$$

其系数分别为:

$a_0 = 1.0,$	$a_1 = 3.5156229$
$a_2 = 3.0899424,$	$a_3 = 1.2067492$
$a_4 = 2.659732 \times 10^{-1},$	$a_5 = 3.60768 \times 10^{-2}$
$a_6 = 4.5813 \times 10^{-3}$	
$b_0 = 5.0 \times 10^{-1},$	$b_1 = 8.7890594 \times 10^{-1}$
$b_2 = 5.1498869 \times 10^{-1},$	$b_3 = 1.5084934 \times 10^{-1}$

$$b_4 = 2.658773 \times 10^{-2}, \quad b_5 = 3.01532 \times 10^{-3}$$

$$b_6 = 3.2411 \times 10^{-4}$$

当 $|x| \geq 3.75$ 时, 令 $y = \frac{3.75}{|x|}$, 则

$$I_0(x) = \frac{e^{|x|}}{\sqrt{|x|}} C(y)$$

其中

$$C(y) = c_0 + c_1y + c_2y^2 + c_3y^3 + \cdots + c_8y^8$$

$$\text{且 } I_1(|x|) = \frac{e^{|x|}}{\sqrt{|x|}} D(y), I_1(-|x|) = -I_1(|x|)$$

其中

$$D(y) = d_0 + d_1y + d_2y^2 + \cdots + d_8y^8$$

其系数分别为:

$$\begin{aligned} c_0 &= 3.9894228 \times 10^{-1}, & c_1 &= 1.328592 \times 10^{-2} \\ c_2 &= 2.25319 \times 10^{-3}, & c_3 &= -1.57565 \times 10^{-3} \\ c_4 &= 9.16281 \times 10^{-3}, & c_5 &= -2.057706 \times 10^{-2} \\ c_6 &= 2.635537 \times 10^{-2}, & c_7 &= -1.647633 \times 10^{-2} \\ c_8 &= 3.92377 \times 10^{-3} \\ d_0 &= 3.9894228 \times 10^{-1}, & d_1 &= -3.988024 \times 10^{-2} \\ d_2 &= -3.62018 \times 10^{-3}, & d_3 &= 1.63801 \times 10^{-3} \\ d_4 &= -1.031555 \times 10^{-2}, & d_5 &= 2.282967 \times 10^{-2} \\ d_6 &= -2.895312 \times 10^{-2}, & d_7 &= 1.787654 \times 10^{-2} \\ d_8 &= -4.20059 \times 10^{-3} \end{aligned}$$

当 $n \geq 2$ 时, 变型第一类贝塞耳函数具有下列递推关系

$$I_{n+1}(x) = -\frac{2n}{x} I_n(x) + I_{n-1}(x)$$

但这个递推公式是不稳定的。实际计算时, 用如下递推关系

$$I_{n-1}(x) = \frac{2n}{x} I_n(x) + I_{n+1}(x)$$

三、函数语句

double lssl3(n, x)

本函数返回一个双精度实型函数值 $I_n(x)$ 。

四、形参说明

n ——整型变量。变型贝塞耳函数的阶数。 $n \geq 0$ 。当 $n < 0$ 时, 本函数按 $|n|$ 处理。

x ——双精度实型变量。自变量值。

五、函数程序(文件名:lssl3.c)

六、例

计算 $n=0, 1, 2, 3, 4, 5$ 时, $x=0.05, 0.5, 5.0, 50.0$ 时的 $I_n(x)$ 。

主函数程序(文件名:lssl30.c)如下:

```
#include "stdio.h"
#include "lssl3.c"
main()
{
    int n, i;
    double x, y;
    printf("\n");
    for (n = 0; n <= 5; n++)
    {
        x = 0.05;
        for (i = 1; i <= 4; i++)
        {
            y = lssl3(n, x);
            printf("n=%d x=%e I(n,x)=%e\n", n, x, y);
            x = x * 10.0;
        }
        printf("\n");
    }
}
```

运行结果为:

$n=0$	$x=0.050$	$I(n,x)=1.00063e+00$
$n=0$	$x=0.500$	$I(n,x)=1.06348e+00$
$n=0$	$x=5.000$	$I(n,x)=2.72399e+01$
$n=0$	$x=50.000$	$I(n,x)=2.93255e+20$
$n=1$	$x=0.050$	$I(n,x)=2.50078e-02$
$n=1$	$x=0.500$	$I(n,x)=2.57894e-01$
$n=1$	$x=5.000$	$I(n,x)=2.43356e+01$
$n=1$	$x=50.000$	$I(n,x)=2.90308e+20$
$n=2$	$x=0.050$	$I(n,x)=3.12565e-04$
$n=2$	$x=0.500$	$I(n,x)=3.19061e-02$
$n=2$	$x=5.000$	$I(n,x)=1.75056e+01$
$n=2$	$x=50.000$	$I(n,x)=2.81647e+20$
$n=3$	$x=0.050$	$I(n,x)=2.60457e-06$
$n=3$	$x=0.500$	$I(n,x)=2.64511e-03$
$n=3$	$x=5.000$	$I(n,x)=1.03312e+01$
$n=3$	$x=50.000$	$I(n,x)=2.67776e+20$
$n=4$	$x=0.050$	$I(n,x)=1.62781e-08$
$n=4$	$x=0.500$	$I(n,x)=1.64806e-04$
$n=4$	$x=5.000$	$I(n,x)=5.10823e+00$
$n=4$	$x=50.000$	$I(n,x)=2.49510e+20$
$n=5$	$x=0.050$	$I(n,x)=8.13887e-11$
$n=5$	$x=0.500$	$I(n,x)=8.22317e-06$
$n=5$	$x=5.000$	$I(n,x)=2.15797e+00$

$n = 5 \quad x = 50.000 \quad I(n, x) = 2.27855e + 20$

12.7 变型第二类整数阶贝塞耳函数

一、功能

计算实变量 x 的变型第二类整数阶贝塞耳(Bessel)函数值 $K_n(x)$ 。

二、方法说明

变型第二类整数阶贝塞耳函数表示为

$$K_n(x) = \frac{\pi}{2} j^{n+1} [J_n(jx) + j Y_n(jx)], \quad x > 0$$

其中 n 为非负整数, j 为虚数(即 $\sqrt{-1}$, $J_n(jx)$ 为纯虚变量(jx)的第一类贝塞耳函数, $Y_n(jx)$ 为纯虚变量(jx)的第二类贝塞耳函数。

$K_0(x)$ 与 $K_1(x)$ 的计算公式如下:

当 $x \leq 2.0$ 时, 令 $y = x^2/4.0$, 则:

$$K_0(x) = A(y) - I_0(x) \ln\left(\frac{x}{2}\right)$$

$$K_1(x) = \frac{1}{x} B(y) + I_1(x) \ln\left(\frac{x}{2}\right)$$

其中 $I_0(x), I_1(x)$ 分别为变型第一类零阶、一阶贝塞耳函数, 而

$$A(y) = a_0 + a_1y + a_2y^2 + a_3y^3 + a_4y^4 + a_5y^5 + a_6y^6$$

$$B(y) = b_0 + b_1y + b_2y^2 + b_3y^3 + b_4y^4 + b_5y^5 + b_6y^6$$

其系数分别为:

$a_0 = -0.57721566,$	$a_1 = 0.42278420$
$a_2 = 0.23069756,$	$a_3 = 0.3488590 \times 10^{-1}$
$a_4 = 0.262698 \times 10^{-2},$	$a_5 = 0.10750 \times 10^{-3}$
$a_6 = 0.74 \times 10^{-5}$	
$b_0 = 1.0,$	$b_1 = 0.15443144$
$b_2 = -0.67278579,$	$b_3 = -0.18156897$
$b_4 = -0.1919402 \times 10^{-1},$	$b_5 = -0.110404 \times 10^{-2}$
$b_6 = -0.4686 \times 10^{-4}$	

当 $x > 2.0$ 时, 令 $y = 2.0/x$, 则:

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} C(y)$$

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} D(y)$$

其中

$$C(y) = c_0 + c_1y + c_2y^2 + c_3y^3 + c_4y^4 + c_5y^5 + c_6y^6$$

$$D(y) = d_0 + d_1y + d_2y^2 + d_3y^3 + d_4y^4 + d_5y^5 + d_6y^6$$

其系数分别为:

$c_0 = 0.125331414 \times 10^1,$	$c_1 = -0.7832358 \times 10^{-1}$
$c_2 = 0.2189568 \times 10^{-1},$	$c_3 = -0.1062446 \times 10^{-1}$
$c_4 = 0.587872 \times 10^{-2},$	$c_5 = -0.251540 \times 10^{-2}$
$c_6 = 0.53208 \times 10^{-3}$	
$d_0 = 0.125331414 \times 10^1,$	$d_1 = 0.23498619$
$d_2 = -0.3655620 \times 10^{-1},$	$d_3 = 0.1504268 \times 10^{-1}$
$d_4 = -0.780353 \times 10^{-2},$	$d_5 = 0.325614 \times 10^{-2}$
$d_6 = -0.68245 \times 10^{-3}$	

当 $n \geq 2$ 时, 用递推关系式

$$K_{n+1}(x) = \frac{2n}{x} K_n(x) + K_{n-1}(x)$$

三、函数语句

double lssl4(n, x)

本函数返回一个双精度实型函数值 $K_n(x)$ 。

本函数要调用变型第一类贝塞耳函数 lssl3, 参看 12.6 节。

四、形参说明

n ——整型变量。变型贝塞耳函数的阶数。 $n \geq 0$ 。当 $n < 0$ 时, 本函数按 $|n|$ 处理。

x ——双精度实型变量。自变量值。 $x > 0$ 。当 $x < 0$ 时, 本函数按 $|x|$ 处理。

五、函数程序(文件名:lssl4.c)

六、例

计算 $n = 0, 1, 2, 3, 4, 5$ 时, $x = 0.05, 0.5, 5.0, 50.0$ 时的 $K_n(x)$ 。

主函数程序(文件名:lssl4.c)如下:

```
#include "stdio.h"
#include "lssl4.c"
#include "lssl3.c"
main()
{
    int n, i;
    double x, y;
    printf("\n");
    for (n = 0; n <= 5; n++)
    {
        x = 0.05;
        for (i = 1; i <= 4; i++)
        {
            y = lssl4(n, x);
            printf("n = %d x = %6.3f K(n, x) = %e\n", n, x, y);
            x = x * 10.0;
        }
    }
}
```

```

    }
    printf("\n");
}

```

运行结果为：

```

n = 0   x = 0.050   K(n, x) = 3.11423e+00
n = 0   x = 0.500   K(n, x) = 9.24419e-01
n = 0   x = 5.000   K(n, x) = 3.69110e-03
n = 0   x = 50.000  K(n, x) = 3.41017e-23
n = 1   x = 0.050   K(n, x) = 1.99097e+01
n = 1   x = 0.500   K(n, x) = 1.65644e+00
n = 1   x = 5.000   K(n, x) = 4.04461e-03
n = 1   x = 50.000  K(n, x) = 3.44410e-23
n = 2   x = 0.050   K(n, x) = 7.99501e+02
n = 2   x = 0.500   K(n, x) = 7.55018e+00
n = 2   x = 5.000   K(n, x) = 5.30894e-03
n = 2   x = 50.000  K(n, x) = 3.54793e-23
n = 3   x = 0.050   K(n, x) = 6.39800e+04
n = 3   x = 0.500   K(n, x) = 6.20579e+01
n = 3   x = 5.000   K(n, x) = 8.29177e-03
n = 3   x = 50.000  K(n, x) = 3.72794e-23
n = 4   x = 0.050   K(n, x) = 7.67840e+06
n = 4   x = 0.500   K(n, x) = 7.52245e+02
n = 4   x = 5.000   K(n, x) = 1.52591e-02
n = 4   x = 50.000  K(n, x) = 3.99528e-23
n = 5   x = 0.050   K(n, x) = 1.22861e+09
n = 5   x = 0.500   K(n, x) = 1.20980e+04
n = 5   x = 5.000   K(n, x) = 3.27063e-02
n = 5   x = 50.000  K(n, x) = 4.36718e-23

```

12.8 不完全贝塔函数

一、功能

计算不完全贝塔(Beta)函数值 $B_x(a, b)$ 。

二、方法说明

不完全贝塔函数的定义为

$$B_x(a, b) = \frac{1}{B(a, b)} \int_0^x t^{a-1} (1-t)^{b-1} dt$$

其中 $a, b > 0, 0 \leq x \leq 1$, $B(a, b)$ 为贝塔函数, 即

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

不完全贝塔函数具有下列对称关系及极限值

$$B_x(a, b) = 1 - B_{1-x}(b, a) \quad (1)$$

$$B_0(a, b) = 0, B_1(a, b) = 1$$

不完全贝塔函数可以用下列连分式表示

$$B_x(a, b) = \frac{x^a (1-x)^b}{a \cdot B(a, b)} \varphi(x)$$

$$\varphi(x) = \frac{1}{1 + \frac{d_1}{1 + \frac{d_2}{1 + \dots}}} \quad (2)$$

其中

$$\left. \begin{aligned} d_{2k-1} &= -\frac{(a+k)(a+b+k)x}{(a+2k)(a+2k+1)} \\ d_{2k} &= -\frac{k(b-k)x}{(a+2k-1)(a+2k)} \end{aligned} \right\} k = 1, 2, \dots$$

当 $x < \frac{a+1}{a+b+2}$ 时, 连分式(2)的收敛速度很快。而当 $x > \frac{a+1}{a+b+2}$ 时, 可以用对称关系(1)来计算。

三、函数语句

double lbeta(a, b, x)

本函数返回一个双精度实型函数值 $B_x(a, b)$ 。

本函数要调用伽马函数 lgam1, 参看 12.1 节。

四、形参说明

a ——双精度实型变量。不完全贝塔函数的参数。 $a > 0$ 。当 $a \leq 0$ 时, 本函数输出信息“err * * a<=0!”，并返回函数值 -1。

b ——双精度实型变量。不完全贝塔函数的参数。 $b > 0$ 。当 $b \leq 0$ 时, 本函数输出信息“err * * b<=0!”，并返回函数值 -1。

x ——双精度实型变量。不完全贝塔函数的自变量。 $0 \leq x \leq 1$ 。当 $x < 0$ 或 $x > 1$ 时, 本函数输出信息“err * * x<0 or x>1!”，并返回值 10^{37} 。

五、函数程序(文件名: lbeta.c)

六、例

计算 (a, b) 为 $(0.5, 0.5), (0.5, 5.0), (1.0, 3.0), (5.0, 0.5), (8.0, 10.0)$ 时, x 取值为 $0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ 时的不完全贝塔函数值 $B_x(a, b)$ 。

主函数程序(文件名: lbeta 0.c)如下:

```

#include "stdio.h"
#include "lbeta.c"
#include "lgam1.c"
main()
{
    int i, j;
    double x, a0, b0, y;
    static double a[5] = { 0.5, 0.5, 1.0, 5.0, 8.0 };

```

```

static double b[5] = { 0.5, 5.0, 3.0, 0.5, 10.0 };
printf("\n");
x=0.0;
for (j=0; j<=5; j++)
    printf("x= %4.2f\n",x);
    for (i=0; i<=4; i++)
        a0=a[i]; b0=b[i];
        y=lbeta(a0,b0,x);
        printf("B(%4.2f, %4.2f) = %12.7f\n",a0,b0,y);
    }
    x=x+0.2;
}
printf("\n");

```

运行结果为：

```

x=0.00
B(0.50, 0.50) = 0.000000
B(0.50, 5.00) = 0.000000
B(1.00, 3.00) = 0.000000
B(5.00, 0.50) = 0.000000
B(8.00,10.00) = 0.000000

x=0.20
B(0.50, 0.50) = 0.2739886
B(0.50, 5.00) = 0.9165408
B(1.00, 3.00) = 0.3635302
B(5.00, 0.50) = 0.0000819
B(8.00,10.00) = 0.0095850

x=0.40
B(0.50, 0.50) = 0.3652567
B(0.50, 5.00) = 0.9790393
B(1.00, 3.00) = 0.8560014
B(5.00, 0.50) = 0.0027123
B(8.00,10.00) = 0.2282132

x=0.60
B(0.50, 0.50) = 0.6347433
B(0.50, 5.00) = 0.9972877
B(1.00, 3.00) = 0.9476369
B(5.00, 0.50) = 0.0209607
B(8.00,10.00) = 0.9291743

x=0.80
B(0.50, 0.50) = 0.7260114
B(0.50, 5.00) = 0.9999181
B(1.00, 3.00) = 0.9926155
B(5.00, 0.50) = 0.5609660
B(8.00,10.00) = 0.9995458

x=1.00

```

B(0.50, 0.50) =	1.0000000
B(0.50, 5.00) =	1.0000000
B(1.00, 3.00) =	1.0000000
B(5.00, 0.50) =	1.0000000
B(8.00,10.00) =	1.0000000

12.9 正态分布函数

一、功能

计算随机变量 x 的正态分布函数 $P(a, \sigma, x)$ 值。

二、方法说明

正态分布函数的定义为

$$P(a, \sigma, x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-a)^2}{2\sigma^2}} dt$$

其中 a 为随机变量的数学期望(平均值); $\sigma > 0$, σ^2 为随机变量的方差。

正态分布函数可以用误差函数来计算, 即

$$P(a, \sigma, x) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{x-a}{\sqrt{2}\sigma}\right)$$

三、函数语句

double lgass(a, d, x)

本函数返回一个双精度实型函数值 $P(a, \sigma, x)$ 。

本函数要调用误差函数 lerrf, 参看 12.3 节。误差函数又要调用不完全伽马函数 lgam2, 参看 12.2 节。

四、形参说明

a——双精度实型变量。数学期望值。

d——双精度实型变量。 d^2 为方差值。 $d > 0$ 。

x——双精度实型变量。随机变量值。

五、函数程序(文件名:lgass.c)

六、例

计算当 (a, σ) 为 $(-1.0, 0.5), (3.0, 15.0)$ 时, x 取值为 $-10.0, -5.0, 0.0, 5.0, 10.0$ 时的正态分布函数值 $P(a, \sigma, x)$ 。

主函数程序(文件名:lgass0.c)如下:

```

#include "stdio.h"
#include "lgass.c"
#include "lerrf.c"
#include "lgam2.c"

```

```

#include "lgam1.c"
main()
{
    int i, j;
    double a0, d0, x, y;
    static double a[2] = { -1.0, 3.0 };
    static double d[2] = { 0.5, 15.0 };
    printf("\n");
    for (i = 0; i <= 1; i++)
    {
        a0 = a[i]; d0 = d[i]; x = -10.0;
        for (j = 0; j <= 4; j++)
        {
            y = lgass(a0, d0, x);
            printf("p(%4.2f, %4.2f, %4.2f) = %e\n", a0, d0, x, y);
            x = x + 5.0;
        }
    }
    printf("\n");
}

```

运行结果为：

```

P(-1.00, 0.50, -10.00) = 0.00000e+00
P(-1.00, 0.50, -5.00) = 6.10623e-16
P(-1.00, 0.50, 0.00) = 9.77250e-01
P(-1.00, 0.50, 5.00) = 1.00000e+00
P(-1.00, 0.50, 10.00) = 1.00000e+00
P( 3.00, 15.00, -10.00) = 1.93062e-01
P( 3.00, 15.00, -5.00) = 2.96901e-01
P( 3.00, 15.00, 0.00) = 4.20740e-01
P( 3.00, 15.00, 5.00) = 5.53035e-01
P( 3.00, 15.00, 10.00) = 6.79631e-01

```

12.10 t-分布函数

一、功能

计算 t -分布函数 $P(t, n)$ 。

二、方法说明

t -分布又称 Student-分布, 它定义为

$$P(t, n) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n\pi}\Gamma\left(\frac{n}{2}\right)} \int_{-t}^t \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} dx$$

其中 t 为随机变量, n 为自由度, 且 $t \geq 0$ 。它的极限值为

$$P(0, n) = 0, P(\infty, n) = 1$$

t -分布函数可以用不完全贝塔函数表示, 即

$$P(t, n) = 1 - B_{\frac{n}{n+t}}\left(\frac{n}{2}, 0.5\right)$$

三、函数语句

```
double lstdt(t, n)
```

本函数返回一个双精度实型函数值 $P(t, n)$ 。

本函数要调用不完全贝塔函数 lbeta, 参看 12.8 节。不完全贝塔函数又要调用伽马函数 lgam1, 参看 12.1 节。

四、形参说明

t ——双精度实型变量。随机变量值。 $t \geq 0$ 。

n ——整型变量。自由度。

五、函数程序(文件名:lstdt.c)

六、例

计算 $n = 1, 2, 3, 4$ 时, $t = 0.5, 5.0$ 时的 t -分布函数值 $P(t, n)$ 。

主函数程序(文件名:lstdt0.c)如下

```

#include "stdio.h"
#include "lstdt.c"
#include "lbeta.c"
#include "lgam1.c"
main()
{
    int n;
    double t, y;
    printf("\n");
    for (n = 1; n <= 5; n++)
    {
        t = 0.5; y = lstdt(t, n);
        printf("p(%4.2f, %d) = %e\n", t, n, y);
        t = 5.0; y = lstdt(t, n);
        printf("p(%4.2f, %d) = %e\n", t, n, y);
    }
    printf("\n");
}

```

运行结果为：

```

P(0.50, 1) = 2.73989e-01
P(5.00, 1) = 8.75927e-01
P(0.50, 2) = 3.11108e-01
P(5.00, 2) = 9.63219e-01
P(0.50, 3) = 3.25943e-01
P(5.00, 3) = 9.85174e-01
P(0.50, 4) = 3.33855e-01
P(5.00, 4) = 9.92855e-01
P(0.50, 5) = 3.38772e-01
P(5.00, 5) = 9.96117e-01

```

12.11 χ^2 -分布函数

一、功能

计算 χ^2 变量的分布函数值 $P(\chi^2, n)$ 。

二、方法说明

χ^2 -分布函数的定义为

$$P(\chi^2, n) = \frac{1}{2^{n/2}\Gamma\left(\frac{n}{2}\right)} \int_0^{\chi^2} t^{(n/2)-1} e^{-t/2} dt$$

其中 n 为自由度, 且 $\chi^2 \geq 0$ 。它的极限值为

$$P(0, n) = 0, P(\infty, n) = 1$$

χ^2 -分布函数可以用不完全伽马函数来计算, 即

$$P(\chi^2, n) = \Gamma\left(\frac{n}{2}, \frac{\chi^2}{2}\right)$$

三、函数语句

double lchii(x, n)

本函数返回一个双精度实型函数值 $P(\chi^2, n)$ 。

本函数要调用不完全伽马函数 lgam2, 参看 12.2 节。不完全伽马函数又要调用伽马函数 lgam1, 参看 12.1 节。

四、形参说明

x——双精度实型变量。 χ^2 变量值。

n——整型变量。自由度。

五、函数程序(文件名:lchii.c)

六、例

计算 $n = 1, 2, 3, 4, 5$ 时, $\chi^2 = 0.5, 5.0$ 时的 χ^2 -分布函数值 $P(\chi^2, n)$ 。

主函数程序(文件名:lchii0.c)如下:

```
#include "stdio.h"
#include "lchii.c"
#include "lgam2.c"
#include "lgam1.c"

main()
{
    int n;
    double t, y;
    printf("\n");
    for (n = 1; n <= 5; n++)
    {
        t = 0.5;
        y = lchii(t, n);
        printf("P(%4.2f, %d) = %e\n", t, n, y);
        t = 5.0;
        y = lchii(t, n);
        printf("P(%4.2f, %d) = %e\n", t, n, y);
    }
}
```

```
{
    t = 0.5;
    y = lchii(t, n);
    printf("P(%4.2f, %d) = %e\n", t, n, y);
    t = 5.0;
    y = lchii(t, n);
    printf("P(%4.2f, %d) = %e\n", t, n, y);
}
```

运行结果为:

```
P(0.50, 1) = 5.20500e-01
P(5.00, 1) = 9.74653e-01
P(0.50, 2) = 2.21197e-01
P(5.00, 2) = 9.17916e-01
P(0.50, 3) = 8.11086e-02
P(5.00, 3) = 8.28203e-01
P(0.50, 4) = 2.64988e-02
P(5.00, 4) = 7.12695e-01
P(0.50, 5) = 7.87671e-03
P(5.00, 5) = 5.84120e-01
```

12.12 F-分布函数

一、功能

计算随机变量 F 的 F -分布函数值 $P(F, n_1, n_2)$ 。

二、方法说明

随机变量 F 的 F -分布函数定义为

$$P(F, n_1, n_2) = \frac{\Gamma\left(\frac{n_1 + n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)} n_1^{n_1/2} n_2^{n_2/2} \int_F^\infty \frac{t^{n_1/2-1}}{(n_2 + n_1 t)^{(n_1+n_2)/2}} dt$$

其中随机变量 $F \geq 0$, n_1 与 n_2 为自由度。它的极限值为

$$P(0, n_1, n_2) = 1, P(\infty, n_1, n_2) = 0$$

F -分布函数可以用不完全贝塔函数计算, 即

$$P(F, n_1, n_2) = B_{n_2/(n_2+n_1 F)}\left(\frac{n_2}{2}, \frac{n_1}{2}\right)$$

三、函数语句

double lffff(f, n1, n2)

本函数返回一个双精度实型函数值 $P(F, n_1, n_2)$ 。

本函数要调用不完全贝塔函数 lbeta, 参看 12.8 节。不完全贝塔函数又要调用伽马函数 lgam1, 参看 12.1 节。

四、形参说明

f——双精度实型变量。随机变量 F 的取值。 $F \geq 0$ 。

n1——整型变量。自由度。

n2——整型变量。自由度。

五、函数程序(文件名:lfff.c)

六、例

计算(n_1, n_2)分别为(2, 3)、(5, 10)时, 随机变量 F 的取值为 3.50, 9.0 时的 F-分布函数值 $P(F, n_1, n_2)$ 。

主函数程序(文件名:lfff0.c)如下:

```
#include "stdio.h"
#include "lfff.c"
#include "lbeta.c"
#include "lgam1.c"

main()
{
    int n1, n2, i;
    double y, f;
    static int n[2] = {2, 5};
    static int m[2] = {3, 10};
    printf("\n");
    for (i = 0; i <= 1; i++)
    {
        n1 = n[i]; n2 = m[i]; f = 3.5;
        y = lfff(f, n1, n2);
        printf("p(%4.2f, %d, %d) = %e\n", f, n1, n2, y);
        f = 9.0; y = lfff(f, n1, n2);
        printf("p(%4.2f, %d, %d) = %e\n", f, n1, n2, y);
    }
    printf("\n");
}
```

运行结果为:

```
P(3.50, 2, 3) = 1.38025e-01
P(9.00, 2, 3) = 5.02706e-02
P(3.50, 5, 10) = 3.57164e-02
P(9.00, 5, 10) = 1.68888e-03
```

12.13 正弦积分

一、功能

计算正弦积分值。

二、方法说明

正弦积分的定义为

$$Si(x) = \int_0^x \frac{\sin t}{t} dt, \quad x > 0$$

本函数采用勒让德-高斯求积公式计算该积分。

三、函数语句

```
double lsinn(x)
```

本函数返回一个双精度实型正弦积分值。

四、形参说明

x——双精度实型变量。自变量值, 要求 $x > 0$ 。

五、函数程序(文件名:lsinn.c)

六、例

计算自变量 x 从 0.5 开始、每隔 2.0 的 10 个正弦积分值。

主函数程序(文件名:lsinn0.c)如下:

```
#include "stdio.h"
#include "lsinn.c"
main()
{
    int i;
    double x, y;
    printf("\n");
    for (i = 0; i <= 9; i++)
    {
        x = 0.5 + i * 2; y = lsinn(x);
        printf("x = %4.2f Si(x) = %10.7f\n", x, y);
    }
    printf("\n");
}
```

运行结果为:

```
x = 0.50      Si(x) = 0.4931074
x = 2.50      Si(x) = 1.7785202
x = 4.50      Si(x) = 1.6541404
x = 6.50      Si(x) = 1.4217943
x = 8.50      Si(x) = 1.6295971
x = 10.50     Si(x) = 1.6229407
x = 12.50     Si(x) = 1.4923370
x = 14.50     Si(x) = 1.5907229
x = 16.50     Si(x) = 1.6156262
x = 18.50     Si(x) = 1.5212824
```

12.14 余弦积分

一、功能

计算余弦积分值。

二、方法说明

余弦积分的定义为

$$Ci(x) = - \int_x^{\infty} \frac{\cos t}{t} dt, \quad x > 0$$

计算余弦积分的公式为

$$Ci(x) = \gamma + \ln x - \int_0^x \frac{1 - \cos t}{t} dt$$

其中

$$\gamma = 0.57721566490153286060651$$

为欧拉(Euler)常数。

本函数采用勒让德-高斯求积公式计算积分

$$\int_0^x \frac{1 - \cos t}{t} dt$$

三、函数语句

double lcoss(x)

本函数返回一个双精度实型余弦积分值。

四、形参说明

x——双精度实型变量。自变量值, 要求 $x > 0$ 。

五、函数程序(文件名:lcoss.c)

六、例

计算自变量 x 从 0.5 开始, 每隔 2.0 的 10 个余弦积分值。

主函数程序(文件名:lcoss_0.c)如下:

```
#include "stdio.h"
#include "lcoss.c"

main()
{
    int i;
    double x, y;
    printf("\n");
    for (i = 0; i <= 9; i++)
    {
        x = 0.5 + i * 2.0;
        y = lcoss(x);
        printf("x = %4.2f Ci(x) = %10.7f\n", x, y);
    }
}
```

```
printf("\n");
```

运行结果为:

x = 0.50	Ci(x) = -0.1777841
x = 2.50	Ci(x) = 0.2858712
x = 4.50	Ci(x) = -0.1934911
x = 6.50	Ci(x) = 0.0111015
x = 8.50	Ci(x) = 0.0994314
x = 10.50	Ci(x) = -0.0782840
x = 12.50	Ci(x) = -0.0114084
x = 14.50	Ci(x) = 0.0655370
x = 16.50	Ci(x) = -0.0403075
x = 18.50	Ci(x) = -0.0211074

12.15 指数积分

一、功能

计算指数积分值。

二、方法说明

指数积分的定义为

$$Ei(x) = - \int_x^{\infty} \frac{e^{-t}}{t} dt, \quad x > 0$$

或

$$Ei(x) = \int_{-\infty}^x \frac{e^t}{t} dt, \quad x < 0$$

计算指数积分的公式为

$$Ei(x) = \gamma + \ln x + \int_0^x \frac{e^{-t} - 1}{t} dt, \quad x > 0$$

其中

$$\gamma = 0.57721566490153286060651$$

为欧拉(Euler)常数。

本函数采用勒让德-高斯求积公式计算积分

$$\int_0^x \frac{e^{-t} - 1}{t} dt$$

三、函数语句

double lepp(x)

本函数返回一个双精度实型指数积分值。

四、形参说明

x——双精度实型变量。自变量值, 要求 $x > 0$ 。

五、函数程序(文件名:lexpp.c)

六、例

计算自变量 x 以 0.05 开始,每隔 0.2 的 10 个指数积分值。

主函数程序(文件名:lexpp 0.c)如下:

```
# include "stdio.h"
# include "lexpp.c"
main()
{ int i;
  double x, y;
  printf("\n");
  for (i = 0; i <= 9; i++)
    { x = 0.05 + 0.2 * i; y = lexpp(x);
      printf("x= %4.2f Ei(x)= %10.7f \n", x, y);
    }
  printf("\n");
}
```

运行结果为:

```
x = 0.05   Ei(x) = - 2.4678985
x = 0.25   Ei(x) = - 1.0442826
x = 0.45   Ei(x) = - 0.6253313
x = 0.65   Ei(x) = - 0.4115170
x = 0.85   Ei(x) = - 0.2840193
x = 1.05   Ei(x) = - 0.2018728
x = 1.25   Ei(x) = - 0.1464134
x = 1.45   Ei(x) = - 0.1077774
x = 1.65   Ei(x) = - 0.0802476
x = 1.85   Ei(x) = - 0.0602950
```

12.16 第一类椭圆积分

一、功能

计算第一类椭圆积分。

二、方法说明

第一类椭圆积分的定义为

$$F(k, \varphi) = \int_0^{\varphi} \frac{1}{\sqrt{1 - k^2 \sin^2 \theta}} d\theta, 0 \leq k \leq 1$$

当 $\varphi = \frac{\pi}{2}$ 时, $F\left(k, \frac{\pi}{2}\right)$ 称为第一类完全椭圆积分。

当 $|\varphi| > \frac{\pi}{2}$ 时, 第一类椭圆积分有如下关系

$$F(k, n\pi \pm \varphi) = 2n F\left(k, \frac{\pi}{2}\right) \pm F(k, \varphi)$$

本函数采用勒让德-高斯求积公式计算上述积分。

三、函数语句

```
double lelp1(k, f)
```

本函数返回一个双精度实型的第一类椭圆积分值。

四、形参说明

k ——双精度实型变量。要求 $0 \leq k \leq 1$ 。

f ——双精度实型变量。椭圆积分中的 φ 。

五、函数程序(文件名:lelp 1.c)

六、例

取 $k = 0.5$ 与 1.0 , φ_i 取 $\varphi_i = \frac{\pi}{18}i$ ($i = 0, 1, \dots, 10$), 计算第一类椭圆积分值。

主函数程序(文件名:lelp10.c)如下:

```
# include "stdio.h"
# include "lelp1.c"
main()
{ int i;
  double f, k, y;
  printf("\n");
  for (i = 0; i <= 10; i++)
    { f = i * 3.1415926 / 18.0;
      k = 0.5; y = lelp1(k, f);
      printf("F(%4.2f, %8.6f) = %e \n", k, f, y);
      k = 1.0; y = lelp1(k, f);
      printf("F(%4.2f, %8.6f) = %e \n", k, f, y);
    }
  printf("\n");
}
```

运行结果为:

$F(0.50, 0.000000) = 0.00000e + 00$	$F(1.00, 0.000000) = 0.00000e + 00$
$F(0.50, 0.174533) = 1.74754e - 01$	$F(1.00, 0.174533) = 1.75426e - 01$
$F(0.50, 0.349066) = 3.50819e - 01$	$F(1.00, 0.349066) = 3.56378e - 01$
$F(0.50, 0.523599) = 5.29429e - 01$	$F(1.00, 0.523599) = 5.49306e - 01$
$F(0.50, 0.698132) = 7.11647e - 01$	$F(1.00, 0.698132) = 7.62910e - 01$
$F(0.50, 0.872665) = 8.98245e - 01$	$F(1.00, 0.872665) = 1.01068e + 00$
$F(0.50, 1.047198) = 1.08955e + 00$	$F(1.00, 1.047198) = 1.31696e + 00$
$F(0.50, 1.221730) = 1.28530e + 00$	$F(1.00, 1.221730) = 1.73542e + 00$
$F(0.50, 1.396263) = 1.48455e + 00$	$F(1.00, 1.396263) = 2.43625e + 00$
$F(0.50, 1.570796) = 1.68575e + 00$	$F(1.00, 1.570796) = 1.38106e + 01$

$F(0.50, 1.745329) = 1.88695e + 00$ $F(1.00, 1.745329) = 2.51850e + 01$

12.17 第二类椭圆积分

一、功能

计算第二类椭圆积分。

二、方法说明

第二类椭圆积分的定义为

$$E(k, \varphi) = \int_0^{\varphi} \sqrt{1 - k^2 \sin^2 \theta} d\theta, \quad 0 \leq k \leq 1$$

当 $\varphi = \frac{\pi}{2}$ 时, $E\left(k, \frac{\pi}{2}\right)$ 称为第二类完全椭圆积分。

当 $|\varphi| > \frac{\pi}{2}$ 时, 第二类椭圆积分有如下关系

$$E(k, n\pi \pm \varphi) = 2nE\left(k, \frac{\pi}{2}\right) \pm E(k, \varphi)$$

本函数采用勒让德-高斯求积公式计算上述积分。

三、函数语句

```
double lelp2(k, f)
```

本函数返回一个双精度实型的第二类椭圆积分值。

四、形参说明

k ——双精度实型变量。要求 $0 \leq k \leq 1$ 。

f ——双精度实型变量。椭圆积分中的 φ 。

五、函数程序(文件名:lelp 2.c)

六、例

取 $k = 0.5$ 与 1.0 , φ 取 $\varphi_i = \frac{\pi}{18}i$ ($i = 0, 1, \dots, 10$), 计算第二类椭圆积分值。

主函数程序(文件名:lelp 20.c)如下:

```
#include "stdio.h"
#include "lelp2.c"
main()
{
    int i;
    double f, k, y;
    printf("\n");
    for (i = 0; i <= 10; i++)
    {
        f = i * 3.1415926 / 18.0;
        k = 0.5; y = lelp2(k, f);
    }
}
```

```
printf("E( %4.2f, %8.6f) = %e      ", k, f, y);
k = 1.0; y = lelp2(k, f);
printf("E( %4.2f, %8.6f) = %e\n", k, f, y);
|
```

运行结果为:

E(0.50, 0.00000) = 0.00000e + 00	E(1.00, 0.00000) = 0.00000e + 00
E(0.50, 0.174533) = 1.74312e - 01	E(1.00, 0.174533) = 1.73648e - 01
E(0.50, 0.349066) = 3.47329e - 01	E(1.00, 0.349066) = 3.42020e - 01
E(0.50, 0.523599) = 5.17882e - 01	E(1.00, 0.523599) = 5.00000e - 01
E(0.50, 0.698132) = 6.85060e - 01	E(1.00, 0.698132) = 6.42788e - 01
E(0.50, 0.872665) = 8.48317e - 01	E(1.00, 0.872665) = 7.66044e - 01
E(0.50, 1.047198) = 1.00756e + 00	E(1.00, 1.047198) = 8.66025e - 01
E(0.50, 1.221730) = 1.16318e + 00	E(1.00, 1.221730) = 9.39693e - 01
E(0.50, 1.396263) = 1.31606e + 00	E(1.00, 1.396263) = 9.84808e - 01
E(0.50, 1.570796) = 1.46746e + 00	E(1.00, 1.570796) = 1.00000e + 00
E(0.50, 1.745329) = 1.61887e + 00	E(1.00, 1.745329) = 1.01519e + 00

第 13 章 随机数的产生

13.1 0 到 1 之间均匀分布的一个随机数

一、功能

产生 0 到 1 之间均匀分布的一个随机数。

二、方法说明

设 $m = 2^{16}$, 产生 0 到 1 之间均匀分布随机数的公式如下:

$$\begin{cases} r_i = \text{mod}(2053r_{i-1} + 13849, m), & i = 1, 2, \dots \\ p_i = r_i/m \end{cases}$$

其中 r_i 为随机数种子, p_i 为第 i 个随机数。

三、函数语句

```
double mrnd1(r)
```

本函数返回一个双精度实型的 0 到 1 之间均匀分布的一个随机数。

四、形参说明

r —— 双精度实型变量指针。该指针指向的单元存放随机数的种子; 返回一个新值。

五、函数程序(文件名:mrnd1.c)

六、例

连续产生 10 个 0 到 1 之间均匀分布的随机数。 r 的初值取 5.0。

主函数程序(文件名:mrnd10.c)如下:

```
#include "stdio.h"
#include "mrnd1.c"

main()
{ int i;
  double r;
  r = 5.0;
  printf("\n");
  for (i = 0; i <= 9; i++)
    printf("%10.7f\n", mrnd1(&r));
  printf("\n");
}
```

运行结果为:

```
0.3679504
0.6135712
0.8729248
0.3259430
0.3722839
0.5102386
0.7312622
0.4926300
0.5807190
0.4274139
```

13.2 0 到 1 之间均匀分布的随机数序列

一、功能

产生 0 到 1 之间均匀分布的随机数序列。

二、方法说明

同 13.1 节。

三、函数语句

```
void mrnds(r, p, n)
```

四、形参说明

r —— 双精度实型变量指针。该指针指向的单元存放随机数的种子; 返回一个新值。

p —— 双精度实型一维数组, 长度为 n 。返回随机数序列 p_i ($i = 0, 1, \dots, n - 1$)。

n —— 整型变量。随机数序列的长度。

五、函数程序(文件名:mrnds.c)

六、例

产生 50 个 0 到 1 之间均匀分布的随机数。 r 初值取为 1.0。

主函数程序(文件名:mrnds0.c)如下:

```
#include "stdio.h"
#include "mrnds.c"

main()
{ int i, j;
  double p[50], r;
  r = 1.0;
  mrnds(&r, p, 50);
  printf("\n");
  for (i = 0; i <= 9; i++)
    for (j = 0; j <= 4; j++)
      printf("%10.7f ", p[5 * i + j]);
}
```

```

    printf("\n");
}
printf("\n");

```

运行结果为：

0.2426453	0.3620453	0.4902954	0.7877960	0.5565491
0.8065643	0.0878906	0.6507721	0.2464294	0.1309662
0.0848999	0.5108185	0.9216614	0.3821259	0.7156982
0.5398102	0.4416199	0.8569183	0.4646606	0.1596222
0.9156799	0.1022186	0.0661621	0.0421295	0.7032166
0.9149017	0.5045776	0.1092072	0.4136047	0.3418427
0.0142822	0.5327301	0.9062195	0.6799164	0.0796509
0.7345734	0.2904358	0.4759979	0.4350586	0.3866119
0.9256287	0.5269623	0.0648804	0.4107208	0.4211731
0.8796844	0.2034912	0.9787750	0.6364441	0.8310394

13.3 任意区间内均匀分布的一个随机整数

一、功能

产生给定区间 $[a, b]$ 内均匀分布的一个随机整数。

二、方法说明

首先产生在区间 $[0, s]$ 内均匀分布的随机整数。其计算公式如下：

$$\begin{cases} r_i = \text{mod}(5r_{i-1}, 4m) \\ p_i = \text{int}(r_i/4) \end{cases}$$

其中初值为 $r_0 \geq 1$ 的奇数(随机整数种子), $s = b - a + 1$, $m = 2^k$, $k = [\log_2 s] + 1$ 。

然后将每个随机整数加上 a , 即得到实际需要的随机整数。

三、函数语句

int mrab1(a, b, r)

本函数返回一个在区间 $[a, b]$ 内均匀分布的随机整数。

四、形参说明

a, b ——均为整型变量。随机整数所在区间的左、右端点。

r ——整型变量指针。该指针指向的单元中存放随机整数的种子(应为大于等于 1 的奇数); 返回一个新值。

五、函数程序(文件名:mrab1.c)

六、例

产生 50 个在区间 $[101, 200]$ 内均匀分布的随机整数。取初值为 $r = 5$ 。

主函数程序(文件名:mrab10.c)如下：

```

#include "stdio.h"
#include "mrab1.c"
main()
{
    int i, j, r;
    r = 5;
    printf("\n");
    for (i = 0; i <= 4; i++)
        for (j = 0; j <= 9; j++)
            printf("%d ", mrab1(101, 200, &r));
    printf("\n");
}

```

运行结果为：

107	132	129	114	167	176	190	163	156	121
200	150	180	113	162	151	110	147	105	122
120	197	198	135	144	189	158	131	124	170
191	168	181	118	187	148	130	119	192	173
115	172	175	165	166	171	196	193	178	103

13.4 任意区间内均匀分布的随机整数序列

一、功能

产生给定区间 $[a, b]$ 内均匀分布的随机整数序列。

二、方法说明

同 13.3 节。

三、函数语句

void mrabs(a, b, r, p, n)

四、形参说明

a, b ——均为整型变量。随机整数所在区间的左、右端点。

r ——整型变量指针。该指针指向的单元中存放随机整数的种子初值(应为大于等于 1 的奇数); 返回一个新值。

p ——整型一维数组, 长度为 n 。返回随机整数序列 p_i ($i = 0, 1, \dots, n - 1$)。

n ——整型变量。随机整数序列的长度。

五、函数程序(文件名:mrabs.c)

六、例

产生 50 个在区间 $[100, 300]$ 内均匀分布的随机整数, 取初值 $r = 1$ 。

320 ·

主函数程序(文件名:mrabs 0.c)如下:

```
# include "stdio.h"
# include "mrabs.c"
main()
{ int i, j, p[50], r;
r = 1;
printf("\n");
mrabs(100, 300, &r, p, 50);
for (i=0; i<=4; i++)
{ for (j=0; j<=9; j++)
    printf("%d ", p[10*i+j]);
    printf("\n");
}
printf("\n");
}
```

运行结果为:

101	106	131	256	113	166	175	220	189	290
283	248	222	199	277	218	179	240	289	278
223	204	109	146	232	249	247	197	227	224
209	134	271	188	285	258	123	216	169	190
295	117	186	275	208	129	246	172	205	114

13.5 任意均值与方差的一个正态分布随机数

一、功能

产生给定均值与方差的正态分布的随机数。

二、方法说明

产生均值为 μ 、方差为 σ^2 的正态分布随机数 y 的计算公式为:

$$y = \mu + \sigma \frac{\left(\sum_{i=0}^{n-1} rnd_i - \frac{n}{2} \right)}{\sqrt{n/12}}$$

其中 n 足够大。通常取 $n=12$ 时, 其近似程度已是相当好了, 此时有

$$y = \mu + \sigma \left(\sum_{i=0}^{11} rnd_i - 6 \right)$$

其中 rnd_i 为 0 到 1 之间均匀分布的随机数。

三、函数语句

```
double mgrn1(u, g, r)
```

本函数返回一个双精度实型的正态分布随机数。

四、形参说明

u —— 双精度实型变量。正态分布的均值 μ 。

g —— 双精度实型变量。正态分布的方差 $\sigma^2 = g^2$ 。

r —— 双精度实型变量指针。该指针指向的单元存放随机数种子初值; 返回一个新值。

五、函数程序(文件名:mgrn1.c)

六、例

产生 50 个均值为 1.0、方差为 1.5² 的正态分布随机数。随机数种子初值取 $r=5.0$ 。

主函数程序(文件名:mgrn10.c)如下:

```
# include "stdio.h"
# include "mgrn1.c"
main()
{ int i, j;
double u, g, r;
r = 5.0; u = 1.0; g = 1.5
printf("\n");
for (i=0; i<=9; i++)
{ for (j=0; j<=4; j++)
    printf("%10.7lf ", mg rn1(u, g, &r));
    printf("\n");
}
printf("\n");
}
```

运行结果为:

1.2386322	-1.1779938	0.5128021	1.9047699	1.5916595
1.1672211	-0.7747955	1.8593597	1.6634369	1.7311859
0.6563568	-1.4673004	-0.0460358	-0.9860992	0.3062592
3.9247894	-0.5367584	3.5153656	-0.3250885	0.0356293
0.1912689	0.2355804	0.2623138	0.3652191	3.6380463
1.1745453	2.0684662	0.4135590	2.3035736	4.8322601
-0.9066315	0.1806488	2.1878510	2.2087250	1.8370209
-0.3335114	3.2908783	0.8039398	1.2994232	-1.1289215
4.1126556	0.6179047	0.4805756	0.7944183	-1.3468170
1.6506195	2.3804779	2.4365082	1.9124603	0.9020844

13.6 任意均值与方差的正态分布随机数序列

一、功能

产生给定均值 μ 与方差 σ^2 的正态分布随机数序列。

二、方法说明

同 13.5 节。

三、函数语句

```
void mgrnns(u, g, r, n, a)
```

• 321 •

• 322 •

四、形参说明

u——双精度实型变量。正态分布的均值 μ 。
g——双精度实型变量。正态分布的方差 $\sigma^2 = g^2$ 。
r——双精度实型变量指针。该指针指向的单元存放随机数种子初值；返回一个新值。
n——整型变量。随机数序列的长度。
a——双精度实型一维数组，长度为 n。返回正态分布随机数序列。

五、函数程序(文件名:mgrns.c)

六、例

产生 50 个均值为 1.0、方差为 1.5^2 的正态分布随机数。取随机数种子初值为 $r = 3.0$ 。

主函数程序(文件名:mgrns 0.c)如下：

```
# include "stdio.h"
# include "mgrns.c"

main()
{ int i, j;
  double u, g, r, a[50];
  r = 3.0; u = 1.0; g = 1.5;
  printf("\n");
  mgrns(u, g, &r, 50, a);
  for (i = 0; i <= 9; i++)
    { for (j = 0; j <= 4; j++)
        printf("%10.7f ", a[i * 5 + j]);
      printf("\n");
    }
  printf("\n");
}
```

运行结果为：

-0.4124298	0.3672333	-0.0425568	3.7019501	1.9445038
1.5288544	-1.2012482	2.0979462	-2.7298126	0.1592255
-0.3911896	2.4626923	0.5646210	-0.7416534	2.3876190
1.7961884	1.3278046	0.3262177	2.6351776	1.5984344
1.0597382	3.3628387	-1.1485138	3.3694305	1.2604218
2.3682098	1.5365448	1.1091766	0.4298553	3.3423309
0.1903534	2.3176727	1.5680389	1.7852020	0.8129120
0.9949188	0.1749725	2.1968231	1.9042206	1.6409149
0.7506561	3.0771942	0.4642792	2.7556610	0.2950897
-1.5736847	0.9930878	2.8391571	1.8082733	3.2441864

第 14 章 多项式与连分式函数的计算

14.1 一维多项式求值

一、功能

计算多项式

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

在指定点 x 处的函数值。

二、方法说明

首先将多项式表达成如下嵌套形式：

$$p(x) = ((\cdots((a_{n-1}x + a_{n-2})x + a_{n-3})x + \cdots + a_1)x + a_0)$$

然后从里往外一层一层地进行计算。其递推计算公式如下：

$$u_{n-1} = a_{n-1}$$

$$u_k = u_{k+1}x + a_k, k = n-2, \dots, 1, 0$$

最后得到的 u_0 即是多项式值 $p(x)$ 。

三、函数语句

double npolyv(a, n, x)

本函数返回一个双精度实型多项式值 $p(x)$ 。

四、形参说明

a——双精度实型一维数组，长度为 n。存放 n-1 次多项式的 n 个系数 $a_{n-1}, a_{n-2}, \dots, a_0$ 。

n——整型变量。多项式的项数，其最高次数为 n-1。

x——双精度实型变量。指定的自变量值。

五、函数程序(文件名:npolyv.c)

六、例

计算多项式

$$p(x) = 2x^6 - 5x^5 + 3x^4 + x^3 - 7x^2 + 7x - 20$$

在 $x = \pm 0.9, \pm 1.1, \pm 1.3$ 处的函数值。

主函数程序(文件名:npolyv 0.c)如下：

```
# include "stdio.h"
```

• 324 •

```

#include "nplyv.c"
main()
{
    int i;
    static double a[7] = {-20.0, 7.0, -7.0, 1.0,
                         3.0, -5.0, 2.0};
    static double x[6] = {0.9, -0.9, 1.1, -1.1, 1.3, -1.3};
    printf("\n");
    for (i = 0; i <= 5; i++)
        printf("x(%d) = %5.2lf p(%d) = %13.7e\n",
               i, x[i], i, nplyv(a, 7, x[i]));
    printf("\n");
}

```

运行结果为

```

x(0) = 0.90 p(0) = -1.856227e+01
x(1) = -0.90 p(1) = -2.671537e+01
x(2) = 1.10 p(2) = -1.955613e+01
x(3) = -1.10 p(3) = -2.151303e+01
x(4) = 1.30 p(4) = -2.087573e+01
x(5) = -1.30 p(5) = -6.340432e+00

```

14.2 一维多项式多组求值

一、功能

利用系数预处理法对多项式

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

进行多组求值。其中 $n = 2^k$ ($k \geq 1$)。

二、方法说明

首先将多项式变为首一多项式，即令

$$T(x) = p(x)/a_{n-1} = x^{n-1} + t_{n-2}x^{n-2} + \cdots + t_1x + t_0.$$

其中 $t_k = a_k/a_{n-1}$ ($k = n-2, \dots, 1, 0$)。

然后对首一多项式 $T(x)$ 的系数进行预处理。其预处理过程如下：

将 $T(x)$ 分解为如下形式：

$$T(x) = (x^j + b)q(x) + r(x)$$

其中 $j = 2^{k-1}$, $q(x)$ 与 $r(x)$ 均为 $2^{k-1}-1$ 次的首一多项式。 $b, q(x)$ 与 $r(x)$ 按以下规则确定：

多项式中的中项系数减 1 即为 b , 即

$$b = t_2k - 1 - 1$$

多项式左半部分除以 x^j 即为 $q(x)$, 即

$$q(x) = x^{s-1} + q_{s-2}x^{s-2} + \cdots + q_1x + q_0$$

其中 $s = 2^{k-1}$, $q_i = t_{i+s}$ ($i = s-2, \dots, 1, 0$)。

多项式右半部分减去 b 与 $q(x)$ 的乘积即为 $r(x)$, 即

$$r(x) = x^{s-1} + r_{s-2}x^{s-2} + \cdots + r_1x + r_0$$

其中 $r_i = t_i - bq_i$ ($i = s-2, \dots, 1, 0$)。

由于 $q(x)$ 与 $r(x)$ 还是首一多项式, 因此, 它们也可以按照上述方法分别进行分解。这个过程一直作到 $q(x)$ 与 $r(x)$ 的次数为 1 为止。在分解过程中, 每次分解后的系数仍放在 $T(x)$ 的各系数的存储单元中, 最后就得到 $T(x)$ 经分解处理后的系数。

首一多项式 $T(x)$ 的系数经预处理后, 就可以用这些预处理后的系数对不同的 x 求函数值。这种方法特别适用于对多个 x 进行求值, 减少求值中的乘法次数。

如果原来的多项式不满足 $n = 2^k$ 这个条件, 则可以添加系数为 0 的各项及系数为 1 的最高次项。

三、函数语句

```
void nplys(a, n, x, m, p)
```

四、形参说明

a —— 双精度实型一维数组, 长度为 n 。存放 $n-1$ 次多项式的系数 $a_{n-1}, a_{n-2}, \dots, a_1, a_0$ 。返回时该数组将被破坏。

n —— 整型变量。多项式的项数, 其最高次数为 $n-1$ 。

x —— 双精度实型一维数组, 长度为 m 。存放 m 个自变量值。

m —— 整型变量。给定自变量的个数。

p —— 双精度实型一维数组, 长度为 m 。返回与给定 n 个自变量值对应的多项式值, 即

$$p_k = p(x_k), k = 0, 1, \dots, m-1$$

五、函数程序(文件名:nplys.c)

六、例

(1) 利用系数预处理法计算多项式

$$p(x) = 2x^6 + 5x^5 + 3x^4 + x^3 - 7x^2 + 7x - 20$$

在 $x = \pm 0.9, \pm 1.1, \pm 1.3$ 处的函数值。

主函数程序(文件名:nplys 0, c)如下：

```

#include "stdio.h"
#include "nplys.c"
main()
{
    int i;
    double p[6];
    static double a[7] = {-20.0, 7.0, -7.0, 1.0,
                         3.0, -5.0, 2.0};
    static double x[6] = {0.9, -0.9, 1.1, -1.1, 1.3, -1.3};
    nplys(a, 7, x, 6, p);
    printf("\n");
}

```

```

for (i = 0; i <= 5; i++)
    printf("x(%d) = %5.2f      p(%d) = %13.7e\n",
           i, x[i], i, p[i]);
    printf("\n");
}

```

运行结果为：

```

x(0) = 0.90      p(0) = -1.856227e+01
x(1) = -0.90     p(1) = -2.671537e+01
x(2) = 1.10       p(2) = -1.955613e+01
x(3) = -1.10      p(3) = -2.151303e+01
x(4) = 1.30       p(4) = -2.087573e+01
x(5) = -1.30      p(5) = -6.340432e+00

```

(2) 利用系数预处理法计算多项式

$$p(x) = \sum_{k=1}^{16} kx^{k-1}$$

在 $x = \pm 0.9, \pm 1.1, \pm 1.3$ 处的函数值。

主函数程序(文件名:nplys 1.c)如下：

```

#include "stdio.h"
#include "nplys.c"
main()
{
    int i;
    double p[6];
    static double a[16] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
                          8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0};
    static double x[6] = {0.9, -0.9, 1.1, -1.1, 1.3, -1.3};
    nplys(a, 16, x, 6, p);
    printf("\n");
    for (i = 0; i <= 5; i++)
        printf("x(%d) = %5.2f      p(%d) = %13.7e\n",
               i, x[i], i, p[i]);
    printf("\n");
}

```

运行结果为：

```

x(0) = 0.90      p(0) = 5.182148e+01
x(1) = -0.90     p(1) = -1.334760e+00
x(2) = 1.10       p(2) = 3.756984e+02
x(3) = -1.10      p(3) = -3.582450e+01
x(4) = 1.30       p(4) = 2.820648e+03
x(5) = -1.30      p(5) = -4.752882e+02

```

14.3 二维多项式求值

一、功能

计算二维多项式

$$p(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} x^i y^j$$

在给定点 (x, y) 处的函数值。

二、方法说明

将二维多项式变形如下：

$$\begin{aligned} p(x, y) &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} x^i y^j \\ &= \sum_{i=0}^{m-1} \left[\sum_{j=0}^{n-1} (a_{ij} x^i) y^j \right] \end{aligned}$$

令

$$S_i = \sum_{j=0}^{n-1} (a_{ij} x^i) y^j, \quad i = 0, 1, \dots, m-1$$

则计算 S_i 的递推公式如下：

$$\begin{aligned} u_{n-1} &= a_{i, n-1} x^i \\ u_j &= u_{j+1} y + a_{ij} x^i, \quad j = n-2, \dots, 1, 0 \end{aligned}$$

其中最后的 u_0 即为 S_i 。

最后将所有的 S_i ($i = 0, 1, \dots, m-1$) 累加，即

$$p(x, y) = \sum_{i=0}^{m-1} S_i$$

三、函数语句

double nbply(a, m, n, x, y)

本函数返回一个双精度实型函数值 $p(x, y)$ 。

四、形参说明

a —— 双精度实型二维数组，体积为 $m \times n$ 。存放二维多项式的系数 a_{ij} ($i = 0, 1, \dots, m-1; j = 0, 1, \dots, n-1$)。

m —— 整型变量。自变量 x 的最高次数为 $m-1$ 。

n —— 整型变量。自变量 y 的最高次数为 $n-1$ 。

x, y —— 均为双精度实型变量。给定的一对自变量值。

五、函数程序(文件名:nbply.c)

六、例

计算二维多项式

$$p(x, y) = \sum_{i=0}^3 \sum_{j=0}^4 a_{ij} x^i y^j$$

在 $(0.6, -1.3)$ 处的函数值。其中系数矩阵为

$$A = \begin{bmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 \\ 6.0 & 7.0 & 8.0 & 9.0 & 10.0 \\ 11.0 & 12.0 & 13.0 & 14.0 & 15.0 \\ 16.0 & 17.0 & 18.0 & 19.0 & 20.0 \end{bmatrix}$$

主函数程序(文件名:nbply 0.c)如下:

```
# include "stdio.h"
# include "nbply.c"
main()
{ double z;
static double a[4][5] = {{1.0, 2.0, 3.0, 4.0, 5.0},
{6.0, 7.0, 8.0, 9.0, 10.0},
{11.0, 12.0, 13.0, 14.0, 15.0},
{16.0, 17.0, 18.0, 19.0, 20.0}};
printf("\n");
z = nbply(a, 4, 5, 0.6, -1.3);
printf(" p(0.60, -1.30) = %13.7e\n", z);
printf("\n");
}
```

运行结果为:

p(0.60, -1.30) = 3.966554e+01

14.4 复系数多项式求值

一、功能

计算复系数多项式

$$p(z) = a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \dots + a_1z + a_0$$

在给定复数 z 时的函数值。

二、方法说明

同 14.1 节, 只是改成复数运算。

三、函数语句

void ncpoly(ar, ai, n, x, y, u, v)

本函数要调用复数乘法的函数 ocmul, 参看 15.1 节。

四、形参说明

ar, ai —— 均为双精度实型一维数组, 长度均为 n 。分别存放多项式系数的实部与虚部。

n —— 整型变量。多项式的项数, 其最高次数为 $n-1$ 。

x, y —— 均为双精度实型变量。给定复数 z 的实部与虚部, 即 $z = x + jy$ ($j = \sqrt{-1}$)。

u, v —— 均为双精度实型变量指针。用于返回多项式值 $p(z)$ 的实部与虚部。

五、函数程序(文件名:ncpoly.c)

六、例

计算复系数多项式

$$p(z) = (2 + j2)z^3 + (1 + j)z^2 + (2 + j)z + (2 + j)$$

当 $z = 1 + j$ 时的函数值。

主函数程序(文件名:ncpoly 0.c)如下:

```
# include "stdio.h"
# include "ncpoly.c"
# include "ocmul.c"
main()
{ double x, y, u, v;
static double ar[4] = {2.0, 2.0, 1.0, 2.0};
static double ai[4] = {1.0, 1.0, 1.0, 2.0};
printf("\n");
x = 1.0; y = 1.0;
ncpoly(ar, ai, 4, x, y, &u, &v);
printf("p(1.0 + j1.0) = %10.7lf + j %10.7lf\n", u, v);
printf("\n");
}
```

运行结果为:

p(1.0 + j1.0) = -7.0000000 + j 6.0000000

14.5 多项式相乘

一、功能

求两个多项式

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \dots + p_1x + p_0$$

$$Q(x) = q_{n-1}x^{n-1} + q_{n-2}x^{n-2} + \dots + q_1x + q_0$$

的乘积多项式

$$\begin{aligned} S(x) &= P(x)Q(x) \\ &= s_{m+n-2}x^{m+n-2} + \dots + s_2x^2 + s_1x + s_0 \end{aligned}$$

二、方法说明

乘积多项式 $s(x)$ 中的各系数按如下公式计算:

$$S_k = 0, k = 0, 1, \dots, m+n-2$$

$$s_{i+j} = s_{i+j} + p_iq_j, i = 0, 1, \dots, m-1; j = 0, 1, \dots, n-1$$

三、函数语句

```
void npmul(p, m, q, n, s, k)
```

四、形参说明

p ——双精度实型一维数组, 长度为 m 。存放多项式 $P(x)$ 的系数。

m ——整型变量。多项式 $P(x)$ 的项数, 其最高次数为 $m - 1$ 。

q ——双精度实型一维数组, 长度为 n 。存放多项式 $Q(x)$ 的系数。

n ——整型变量。多项式 $Q(x)$ 的项数, 其最高次数为 $n - 1$ 。

s ——双精度实型一维数组, 长度为 k 。返回乘积多项式 $S(x)$ 的系数。

k ——整型变量。乘积多项式 $S(x)$ 的项数, 其最高次数为 $k - 1$ 。其中 $k = m + n - 1$ 。

五、函数程序(文件名:npmul.c)

六、例

计算下列两个多项式

$$P(x) = 3x^5 - x^4 + 2x^3 + 5x^2 - 6x + 4$$

$$Q(x) = 2x^3 - 6x^2 + 3x + 2$$

的乘积多项式 $S(x) = P(x)Q(x)$ 。其中 $m = 6, n = 4, k = 9$ 。

主函数程序(文件名:npmul.c)如下:

```
# include "stdio.h"
# include "npmul.c"
main()
{
    int i;
    static double p[6] = {4.0, -6.0, 5.0, 2.0, -1.0, 3.0};
    static double q[4] = {2.0, 3.0, -6.0, 2.0};
    double s[9];
    npmul(p, q, 4, s, 9);
    printf("\n");
    for (i = 0; i <= 8; i++)
        printf("s(%d) = %13.7e\n", i, s[i]);
    printf("\n");
}
```

运行结果为:

```
s(0) = 8.000000e+00
s(1) = 0.000000e+00
s(2) = -3.200000e+01
s(3) = 6.300000e+01
s(4) = -3.800000e+01
s(5) = 1.000000e+00
s(6) = 1.900000e+01
s(7) = -2.000000e+01
s(8) = 6.000000e+00
```

14.6 多项式相除

一、功能

求多项式

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \cdots + p_1x + p_0$$

被多项式

$$Q(x) = q_{n-1}x^{n-1} + q_{n-2}x^{n-2} + \cdots + q_1x + q_0$$

除得的商多项式 $S(x)$ 与余多项式 $R(x)$ 。

二、方法说明

本函数采用综合除法求商多项式 $s(x)$ 中的各系数。

设商多项式 $s(x)$ 的最高次数为 $k = m - n$, 则 $s(x)$ 的系数由如下过程来确定 ($i = 0, 1, \dots, k$):

$$s_{k-i} = p_{m-1-i}/q_{n-1}$$

$$p_j = p_j - s_{k-i}q_{j+i-k}, j = m - i - 1, \dots, k - i$$

最后的 p_0, p_1, \dots, p_{n-2} 即为余多项式的系数 r_0, r_1, \dots, r_{n-2} 。

三、函数语句

```
void npdiv(p, m, q, n, s, k, r, l)
```

四、形参说明

p ——双精度实型一维数组, 长度为 m 。存放多项式 $P(x)$ 的系数。返回时将被破坏。

m ——整型变量。多项式 $P(x)$ 的项数, 其最高次数为 $m - 1$ 。

q ——双精度实型一维数组, 长度为 n 。存放多项式 $Q(x)$ 的系数。

n ——整型变量。多项式 $Q(x)$ 的项数, 其最高次数为 $n - 1$ 。

s ——双精度实型一维数组, 长度为 k 。返回商多项式 $S(x)$ 的系数。

k ——整型变量。商多项式 $S(x)$ 的项数, 其最高次数为 $k - 1$ 。其中 $k = m - n + 1$ 。

r ——双精度实型一维数组, 长度为 l 。返回余多项式 $R(x)$ 的系数。

l ——整型变量。余多项式 $R(x)$ 的项数, 其最高次数为 $l - 1$ 。其中 $l = n - 1$ 。

五、函数程序(文件名:npdiv.c)

六、例

求多项式

$$P(x) = 3x^4 + 6x^3 - 3x^2 - 5x + 8$$

被多项式

$$Q(x) = 2x^2 - x + 1$$

除得的商多项式 $S(x)$ 与余多项式 $R(x)$ 。其中 $m = 5, n = 3, k = 3, l = 2$ 。

主函数程序(文件名:npdiv 0.c)如下:

```
#include "stdio.h"
#include "npdiv.c"
main()
{ int i;
static double p[5] = {8.0, -5.0, -3.0, 6.0, 3.0};
static double q[3] = {1.0, -1.0, 2.0};
double s[3], r[2];
npdiv(p, q, 3, s, 3, r, 2);
printf("\n");
for (i=0; i<=2; i++)
    printf(" s(%d) = %13.7e\n", i, s[i]);
printf("\n");
for (i=0; i<=1; i++)
    printf(" r(%d) = %13.7e\n", i, r[i]);
printf("\n");
}
```

运行结果为:

```
s(0) = -3.750000e-01
s(1) = 3.750000e+00
s(2) = 1.500000e+00

r(0) = 8.375000e+00
r(1) = -9.125000e+00
```

14.7 复系数多项式相乘

一、功能

求两个复系数多项式

$$P(z) = p_{m-1}z^{m-1} + p_{m-2}z^{m-2} + \dots + p_1z + p_0$$

$$Q(z) = q_{n-1}z^{n-1} + q_{n-2}z^{n-2} + \dots + q_1z + q_0$$

的乘积多项式

$$S(z) = s_{m+n-2}z^{m+n-2} + \dots + s_1z + s_0$$

其中所有的系数 p_i ($i = 0, 1, \dots, m-1$), q_i ($i = 0, 1, \dots, n-1$), s_i ($i = 0, 1, \dots, m+n-2$) 均为复数。

二、方法说明

同 14.5 节, 但所有运算为复数运算。

三、函数语句

```
void ncmul(pr, pi, m, qr, qi, n, sr, si, k)
```

本函数要调用复数乘法的函数 dcmul, 参看 15.1 节。

四、形参说明

pr, pi——均为双精度实型一维数组, 长度均为 m 。分别存放多项式 $P(z)$ 系数的实部与虚部。

m ——整型变量。多项式 $P(z)$ 的项数, 其最高次数为 $m-1$ 。

qr, qi——均为双精度实型一维数组, 长度均为 n 。分别存放多项式 $Q(z)$ 系数的实部与虚部。

n ——整型变量。多项式 $Q(z)$ 的项数, 其最高次数为 $n-1$ 。

sr, si——均为双精度实型一维数组, 长度均为 k 。返回乘积多项式 $S(z)$ 系数的实部与虚部。

k ——整型变量。乘积多项式 $S(z)$ 的项数, 其最高次数为 $k-1$ 。其中 $k = m+n-1$ 。

五、函数程序(文件名:ncmul.c)

六、例

求下列两个多项式

$$\begin{aligned} P(z) = & (3+j2)z^5 + (-1-j)z^4 + (2+j)z^3 + (5-j4)z^2 \\ & + (-6+j3)z + (4+j2) \end{aligned}$$

$$Q(z) = (2+j)z^3 + (-6-j4)z^2 + (3+j2)z + (2+j)$$

的乘积多项式 $S(z)$ 。其中 $m=6, n=4, k=9$ 。

主函数程序(文件名:ncmul 0.c)如下:

```
#include "stdio.h"
#include "ncmul.c"
#include "ocmul.c"
main()
{ int i;
static double pr[6] = {4.0, -6.0, 5.0, 0.2, 0, -1.0, 3.0};
static double pi[6] = {2.0, 3.0, -4.0, 1.0, -1.0, 2.0};
static double qr[4] = {2.0, 3.0, -6.0, 2.0};
static double qi[4] = {1.0, 2.0, -4.0, 1.0};
double sr[9], si[9];
ncmul(pr, pi, 6, qr, qi, 4, sr, si, 9);
printf("\n");
for (i=0; i<=8; i++)
    printf(" s(%d) = %13.7e + j %13.7e\n", i, sr[i], si[i]);
printf("\n");
}
```

运行结果为:

```
s(0) = 6.000000e+00 + j 8.000000e+00
s(1) = -7.000000e+00 + j 1.400000e+01
s(2) = -2.600000e+01 + j -3.400000e+01
s(3) = 8.000000e+01 + j 1.600000e+01
```

```

s(4) = -5.800000e+01 + j 8.000000e+00
s(5) = 9.000000e+00 + j -1.500000e+01
s(6) = 1.000000e+01 + j 2.600000e+01
s(7) = -1.100000e+01 + j -2.700000e+01
s(8) = 4.000000e+00 + j 7.000000e+00

```

14.8 复系数多项式相除

一、功能

求复系数多项式

$$P(z) = p_{m-1}z^{m-1} + p_{m-2}z^{m-2} + \cdots + p_1z + p_0$$

被复系数多项式

$$Q(z) = q_{n-1}z^{n-1} + q_{n-2}z^{n-2} + \cdots + q_1z + q_0$$

除得的商多项式 $S(z)$ 与余多项式 $R(z)$ 。

二、方法说明

同 14.6 节, 但所有运算为复数运算。

三、函数语句

```
void ncdiv(pr, pi, m, qr, qi, n, sr, si, k, rr, ri, l)
```

本函数要调用复数乘法的函数 ocmul, 参看 15.1 节; 还要调用复数除法的函数 ocdiv, 参看 15.2 节。

四、形参说明

pr, pi ——均为双精度实型一维数组, 长度均为 m 。分别存放多项式 $P(z)$ 系数的实部与虚部。返回时均将被破坏。

m ——整型变量。多项式 $P(z)$ 的项数, 其最高次数为 $m-1$ 。

qr, qi ——均为双精度实型一维数组, 长度均为 n 。分别存放多项式 $Q(z)$ 系数的实部与虚部。

n ——整型变量。多项式 $Q(z)$ 的项数, 其最高次数为 $n-1$ 。

sr, si ——均为双精度实型一维数组, 长度均为 k 。返回商多项式 $S(z)$ 系数的实部与虚部。

k ——整型变量。商多项式 $S(z)$ 的项数, 其最高次数为 $k-1$ 。其中 $k = m-n+1$ 。

rr, ri ——均为双精度实型一维数组, 长度均为 l 。返回余多项式 $R(z)$ 系数的实部与虚部。

l ——整型变量。余多项式 $R(z)$ 的项数, 其最高次数为 $l-1$ 。其中 $l = n-1$ 。

五、函数程序(文件名:ncdiv.c)

六、例

求复系数多项式

$$P(z) = (3-j)z^4 + (6-j5)z^3 + (-3+j4)z^2 + (-5+j4)z + (8+j3)$$

被复系数多项式

$$Q(z) = (2+j2)z^2 + (-1-j3)z + (1+j2)$$

除得的商多项式 $S(z)$ 与余多项式 $R(z)$ 。其中 $m=5, n=3, k=3, l=2$ 。

主函数程序(文件名:ncdiv 0.c)如下:

```

#include "stdio.h"
#include "ncdiv.c"
#include "ocmul.c"
#include "ocdiv.c"
main()
{
    int i;
    static double pr[5] = {8.0, -5.0, -3.0, 6.0, 3.0};
    static double pi[5] = {3.0, 4.0, 4.0, -5.0, -1.0};
    static double qr[3] = {1.0, -1.0, 2.0};
    static double qi[3] = {2.0, -3.0, 2.0};
    double sr[3], si[3], rr[2], ri[2];
    ncdiv(pr, pi, 5, qr, qi, 3, sr, si, 3, rr, ri, 2);
    printf("\n");
    for (i = 0; i <= 2; i++)
        printf(" s(%d) = %13.7e + j %13.7e\n", i, sr[i], si[i]);
    printf("\n");
    for (i = 0; i <= 1; i++)
        printf(" r(%d) = %13.7e + j %13.7e\n", i, rr[i], ri[i]);
    printf("\n");
}

```

运行结果为:

```

s(0) = 2.625000e+00 + j -5.000000e-01
s(1) = 1.250000e+00 + j -3.500000e+00
s(2) = 5.000000e-01 + j -1.000000e+00
r(0) = 4.375000e+00 + j -1.750000e+00
r(1) = -9.125000e+00 + j 1.237500e+01

```

14.9 函数连分式的计算

一、功能

计算函数连分式

$$\varphi(x) = b_0 + \frac{x - x_0}{b_1 + \frac{x - x_1}{b_2 + \cdots + \frac{x - x_{n-2}}{b_{n-1}}}}$$

在指定点处的函数值。

二、方法说明

从最后一节开始, 逐节往前递推计算。即

$$\begin{cases} u = b_{n-1} \\ u = b_k + (x - x_k)/u, k = n-2, \dots, 1, 0 \end{cases}$$

最后的 u 就是连分式的值。

三、函数语句

double nfpqv(x, b, n, t)

本函数返回一个双精度实型函数值 $\varphi(t)$ 。

四、形参说明

x ——双精度实型一维数组，长度为 n 。存放函数连分式中的各结点值 x_0, x_1, \dots, x_{n-1} ，其中 x_{n-1} 为任意。

b ——双精度实型一维数组，长度为 n 。存放函数连分式中的参数 b_0, b_1, \dots, b_{n-1} 。

n ——整型变量。函数连分式的节数。

t ——双精度实型变量。自变量值。

五、函数程序(文件名:nfpqv.c)

六、例

计算函数连分式

$$\varphi(x) = 1 + \frac{x-1}{3 + \frac{x-2}{-1 + \frac{x-3}{2 + \frac{x-4}{5 + \frac{x-5}{-8 + \frac{x-6}{11}}}}}}$$

当 $x = 0.0$ 与 $x = 3.5$ 时的函数值。

主函数程序(文件名:nfpqv 0.c)如下：

```
#include "stdio.h"
#include "nfpqv.c"

main()
{
    static double x[7] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 0.0};
    static double b[7] = {1.0, 3.0, -1.0, 2.0, 5.0, -8.0, 11.0};
    double t;
    printf("\n");
    t = 0.0;
    printf("t = %3.11f y = %13.7e\n", t, nfpqv(x, b, 7, t));
    printf("\n");
    t = 3.5;
    printf("t = %3.11f y = %13.7e\n", t, nfpqv(x, b, 7, t));
    printf("\n");
}
```

运行结果为：

```
t = 0.0      y = 7.221742e-01
t = 3.5      y = 3.588987e+00
```

第 15 章 复数运算

15.1 复数乘法

一、功能

计算两个复数的乘积

$$e + jf = (a + jb)(c + jd)$$

其中 $j = \sqrt{-1}$ 。

二、方法说明

通常，计算两个复数的乘积需要四次实数乘法。本函数只需要三次乘法，其算法如下。

设

$$e + jf = (a + jb)(c + jd)$$

令

$$p = ac, q = bd, s = (a + b)(c + d)$$

则有

$$e = p - q, f = s - p - q$$

三、函数语句

void ocmul(a, b, c, d, e, f)

四、形参说明

a, b ——均为双精度实型变量。表示复数 $a + jb$ 。

c, d ——均为双精度实型变量。表示复数 $c + jd$ 。

e, f ——均为双精度实型变量指针。用于返回复数的乘积 $e + jf = (a + jb)(c + jd)$ 。

五、函数程序(文件名:ocmul.c)

六、例

计算下列两个复数的乘积

$$(-1.3 + j4.5)(7.6 + j3.6)$$

主函数程序(文件名:ocmul 0.c)如下：

```
#include "stdio.h"
#include "ocmul.c"

main()
```

• 338 •

```

    | double e, f;
    | ocmul(-1.3, 4.5, 7.6, 3.6, &e, &f);
    | printf("\n");
    | printf(" e + jf = %10.7f + j %10.7f\n", e, f);
    | printf("\n");
}

```

运行结果为

```
e + jf = -26.0800000 + j 29.5200000
```

15.2 复数除法

一、功能

计算两个复数的商

$$e + jf = (a + jb)/(c + jd)$$

其中 $j = \sqrt{-1}$ 。

二、方法说明

本函数采用如下算法：

设

$$e + jf = (a + jb)/(c + jd)$$

令

$$p = ac, q = -bd, s = (a + b)(c - d), w = c^2 + d^2$$

则有

$$e = (p - q)/w, f = (s - p - q)/w$$

三、函数语句

```
void ocdiv(a, b, c, d, e, f)
```

四、形参说明

a, b——均为双精度实型变量。表示复数 $a + jb$ 。

c, d——均为双精度实型变量。表示复数 $c + jd$ 。

e, f——均为双精度实型指针变量。用于返回两个复数的商 $e + jf = (a + jb)/(c + jd)$ 。

五、函数程序(文件名:ocdiv.c)

六、例

计算下列两个复数的商

$$(-1.3 + j4.5)/(7.6 - j3.6)$$

主函数程序(文件名:ocdiv 0.c)如下：

```

    # include "stdio.h"
    # include "ocdiv.c"
    main()
    | double e, f;
    | ocdv(-1.3, 4.5, 7.6, -3.6, &e, &f);
    | printf("\n");
    | printf(" e + jf = %10.7f + j %10.7f\n", e, f);
    | printf("\n");
}

```

运行结果为

```
e + jf = -0.3687783 + j 0.4174208
```

15.3 复数乘幂

一、功能

计算复变量的整数次幂，即 $(x + jy)^n$ 。其中 n 为整数， $j = \sqrt{-1}$ 。

二、方法说明

设给定复数为

$$z = x + jy$$

化为

$$z = r(\cos\theta + j\sin\theta)$$

其中 $r = \sqrt{x^2 + y^2}$, $\theta = \text{Arctg} \frac{y}{x}$ 。则

$$\begin{aligned} z^n &= (x + jy)^n \\ &= r^n(\cos n\theta + j\sin n\theta) \\ &= u + jv \end{aligned}$$

其中

$$u = r^n \cos n\theta, v = r^n \sin n\theta$$

三、函数语句

```
void opowr(x, y, n, u, v)
```

四、形参说明

x, y——均为双精度实型变量。表示复数 $z = x + jy$ 。

n——整型变量。幂的次数。

u, v——均为双精度实型变量指针。用于返回 $(x + jy)^n$ 的实部与虚部。

五、函数程序(文件名:opowr.c)

六、例

计算当 $n = -3, 0, 3$ 时的 $(2 + j2)^n$ 。

主函数程序(文件名:opowr 0.c)如下:

```
# include "stdio.h"
# include "opowr.c"
main()
{ int n;
  double ,x,y,u,v;
  x = 2.0; y = 2.0;
  printf("\n");
  for (n = -3; n <= 3; n = n + 3)
    { opowr(x,y,n,&u,&v);
      printf("n = %3d      %13.7e + j %13.7e\n",n,u,v);
    }
  printf("\n");
}
```

运行结果为

```
n = -3      -3.125000e-02 + j -3.125000e-02
n = 0       1.000000e+00 + j 0.000000e+00
n = 3       -1.600000e+01 + j 1.600000e+01
```

15.4 复数的 N 次方根

一、功能

计算复变量的 n 次方根, 即 $(x + jy)^{\frac{1}{n}}$ 。其中 n 为正整数, $j = \sqrt{-1}$ 。

二、方法说明

设给定复数为

$$z = x + jy$$

化为

$$z = r(\cos\theta + j\sin\theta)$$

其中 $r = \sqrt{x^2 + y^2}$, $\theta = \text{Arctg} \frac{y}{x}$ 。则

$$\begin{aligned} z^{\frac{1}{n}} &= (x + jy)^{\frac{1}{n}} \\ &= r^{\frac{1}{n}} \left[\cos \frac{2k\pi + \theta}{n} + j \sin \frac{2k\pi + \theta}{n} \right] = u_k + jv_k, k = 0, 1, \dots, n-1 \end{aligned}$$

其中

$$u_k = r^{\frac{1}{n}} \cos \frac{2k\pi + \theta}{n}, v_k = r^{\frac{1}{n}} \sin \frac{2k\pi + \theta}{n}$$

三、函数语句

```
void ontrt(x, y, n, u, v)
```

四、形参说明

x, y —— 均为双精度实型变量。表示复数 $z = x + jy$ 。

n —— 整型变量。要求 $n > 0$ 。

u, v —— 均为双精度实型一维数组, 长度均为 n 。返回复数 $z = x + jy$ 的 n 次方根的 n 个值。

五、函数程序(文件名:ontrt.c)

六、例

计算复数 $1 + j$ 的 5 次方根。

主函数程序(文件名:ontrt 0.c)如下:

```
# include "stdio.h"
# include "ontrt.c"
main()
{ int n;
  double ,x,y,u[5],v[5];
  x = 1.0; y = 1.0;
  ontrt(x,y,5,u,v);
  printf("\n");
  for (n = 0; n <= 4; n++)
    printf("n = %3d      %13.7e + j %13.7e\n",
           n,u[n],v[n]);
  printf("\n");
}
```

运行结果为

```
n = 0      1.058578e+00 + j 1.676623e-01
n = 1      1.676623e-01 + j 1.058578e+00
n = 2      -9.549571e-01 + j 4.865750e-01
n = 3      -7.578583e-01 + j -7.578582e-01
n = 4      4.865749e-01 + j -9.549572e-01
```

15.5 复数指数

一、功能

计算复变量的指数, 即 e^{x+jy} 。其中 $j = \sqrt{-1}$ 。

二、方法说明

设复数为

$$z = x + jy$$

则

$$e^z = e^{x+jy} = e^x(\cos y + j \sin y) \\ = u + jv$$

其中 $u = e^x \cos y, v = e^x \sin y$

三、函数语句

void ocexp(x, y, u, v)

四、形参说明

x, y —— 均为双精度实型变量。表示复数 $z = x + jy$ 。
 u, v —— 均为双精度实型变量指针。用于返回 e^{x+jy} 值的实部与虚部。

五、函数程序(文件名:ocexp.c)

六、例

计算 e^{1+j4} 的值。

主函数程序(文件名:ocexp 0.c)如下:

```
# include "stdio.h"
# include "ocexp.c"
main()
{ double x, y, u, v;
  x = 1.0; y = 4.0;
  ocexp(x, y, &u, &v);
  printf("\n");
  printf(" %13.7e + j %13.7e\n", u, v);
  printf("\n");
}
```

运行结果为

$-1.776788e+00 + j -2.057202e+00$

15.6 复数对数

一、功能

计算复变量的自然对数, 即 $\ln(x + jy)$ 。其中 $j = \sqrt{-1}$ 。

二、方法说明

设给定的复数为

$$z = x + jy$$

则

$$\ln z = \ln(x + jy) = \ln \sqrt{x^2 + y^2} + j \operatorname{Arctg} \frac{y}{x}$$

$$= u + jv$$

$$\text{其中 } u = \ln \sqrt{x^2 + y^2}, v = \operatorname{Arctg} \frac{y}{x}$$

三、函数语句

void oclog(x, y, u, v)

四、形参说明

x, y —— 均为双精度实型变量。表示复数 $z = x + jy$ 。
 u, v —— 均为双精度实型变量指针。用于返回 $\ln(x + jy)$ 的实部与虚部。

五、函数程序(文件名:oclog.c)

六、例

计算 $\ln(1 + j4)$ 。

主函数程序(文件名:oclog 0.c)如下:

```
# include "stdio.h"
# include "oclog.c"
main()
{ double x, y, u, v;
  x = 1.0; y = 4.0;
  oclog(x, y, &u, &v);
  printf("\n");
  printf(" %13.7e + j %13.7e\n", u, v);
  printf("\n");
}
```

运行结果为

$1.416607e+00 + j 1.325818e+00$

15.7 复数正弦

一、功能

计算复变量的正弦值, 即 $\sin(x + jy)$ 。其中 $j = \sqrt{-1}$ 。

二、方法说明

设复数为

$$z = x + jy$$

则

$$\begin{aligned} \sin z &= \sin(x + jy) = \sin x \cos(jy) + \cos x \sin(jy) \\ &= \sin x \left(\frac{e^y + e^{-y}}{2} \right) + j \cos x \left(\frac{e^y - e^{-y}}{2} \right) = u + jv \end{aligned}$$

其中 $u = \sin x \left(\frac{e^y + e^{-y}}{2} \right)$, $v = \cos x \left(\frac{e^y - e^{-y}}{2} \right)$

三、函数语句

void ocsin(x, y, u, v)

四、形参说明

x, y——均为双精度实型变量。表示复数 $z = x + jy$ 。

u, v——均为双精度实型变量指针。用于返回 $\sin(x + jy)$ 的实部与虚部。

五、函数程序(文件名:ocsin.c)

六、例

计算 $\sin(1 + j4)$ 。

主函数程序(文件名:ocsin 0.c)如下:

```
#include "stdio.h"
#include "ocsin.c"
main()
{ double x, y, u, v;
  x = 1.0; y = 4.0;
  ocsin(x, y, &u, &v);
  printf("%13.7e + j %13.7e\n", u, v);
  printf("\n");
}
```

运行结果为

2.297909e+01 + j 1.474481e+01

15.8 复数余弦

一、功能

计算复变量的余弦值, 即 $\cos(x + jy)$ 。其中 $j = \sqrt{-1}$ 。

二、方法说明

设给定的复数为

$$z = x + jy$$

则

$$\begin{aligned}\cos z &= \cos(x + jy) = \cos x \cos(jy) - \sin x \sin(jy) \\ &= \cos x \left(\frac{e^y + e^{-y}}{2} \right) - j \sin x \left(\frac{e^y - e^{-y}}{2} \right) \\ &= u + jv\end{aligned}$$

其中

$$u = \cos x \left(\frac{e^y + e^{-y}}{2} \right), v = -\sin x \left(\frac{e^y - e^{-y}}{2} \right)$$

三、函数语句

void occos(x, y, u, v)

四、形参说明

x, y——均为双精度实型变量。表示复数 $z = x + jy$ 。

u, v——均为双精度实型变量指针。用于返回 $\cos(x + jy)$ 的实部与虚部。

五、函数程序(文件名:occos.c)

六、例

计算 $\cos(1 + j4)$ 。

主函数程序(文件名:occos 0.c)如下:

```
#include "stdio.h"
#include "occos.c"
main()
{ double x, y, u, v;
  x = 1.0; y = 4.0;
  occos(x, y, &u, &v);
  printf("%13.7e + j %13.7e\n", u, v);
  printf("\n");
}
```

运行结果为

1.475470e+01 + j -2.296367e+01

15.9 复数作图

一、功能

在复平面上打印输出给定的复数点集。

二、方法说明

设给定 n 个复数

$$z_k = x_k + jy_k, k = 0, 1, \dots, n-1$$

其中 $j = \sqrt{-1}$ 。

在打印纸上建立一个直角坐标系 XOY , 其 X 值与 Y 值的范围均为 $[-35, 35]$, 即所有复数点均打印在宽为 71 列、长为 71 行的打印纸上, 坐标原点在其中心。为了充分利用 71

行、71 列的打印幅面，在本函数中，选取所有复数点中 X 绝对值最大的点打印在第 1 列(负值)或 71 列(正值)上，Y 绝对值最大的点打印在第 1 行(正值)或第 71 行(负值)上，其余各值将按同样比例进行放大或缩小，本函数返回 X 值与 Y 值的比例因子。

三、函数语句

```
void oplot(n, x, y, xc, yc, c, xd, yd)
```

四、形参说明

n —— 整型变量。给定复数点的个数。

x, y —— 均为双精度实型一维数组，长度均为 n。存放 n 个复数点的实部与虚部。

xc —— 字符型变量。打印 X 轴所用的字符。

yc —— 字符型变量。打印 Y 轴所用的字符。

c —— 字符型变量。打印复数点所用的字符。

xd —— 双精度实型变量指针。用于返回打印复数点时 X 值的比例系数。

yd —— 双精度实型变量指针。用于返回打印复数点时 Y 值的比例系数。

五、函数程序(文件名:oplot.c)

六、例

在复平面上打印输出下列 50 个复数点：

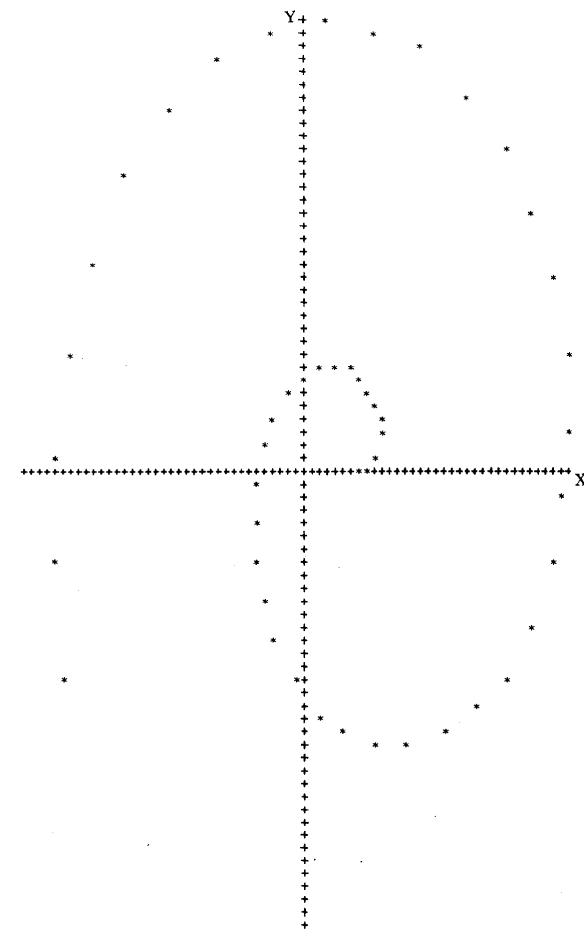
$$2.0 + \varphi e^{j\varphi}$$

其中 $\varphi = 0.2k$ ($k = 0, 1, \dots, 49$)。X 轴与 Y 轴用字符“+”，复数点用字符“*”。

主函数程序(文件名:oplot 0.c)如下：

```
# include "stdio.h"
# include "oplot.c"
main()
{ char xc, yc, c;
  int n, k;
  double x[50], y[50], fi, xd, yd;
  n = 50; xc = '+'; yc = '+'; c = '*';
  for (k = 0; k <= n - 1; k++)
  { fi = 0.2 * k;
    x[k] = 2.0 + fi * cos(fi);
    y[k] = fi * sin(fi);
  }
  oplot(n, x, y, xc, yc, c, &xd, &yd);
}
```

运行结果为



第 16 章 排序

16.1 整数冒泡排序

一、功能

用冒泡排序法将一个整型无序序列排序成有序(非递减)序列。

二、方法说明

冒泡排序的过程如下：

从前到后扫描序列，比较相邻两个项目的大小，若发现逆序则进行交换，最后使最大者换到序列的最后；然后再从后到前扫描剩下的序列，比较相邻两个项目的大小，若发现逆序则进行交换，最后使最小者换到序列的最前面。

对剩下的序列重复上述过程，直到剩下的序列为空止。

三、函数语句

void pibub(p, n)

四、形参说明

p——整型数组的指针。指示待排序序列(为顺序存储)的起始位置。

n——整型变量。待排序序列的长度。

五、函数程序(文件名:pibub.c)

六、例

产生 100 到 300 之间的 100 个随机整数，然后对其中第 31 个随机数开始的后 50 个进行排序。在主函数中要调用产生随机整数的函数 mrabs，参看 13.4 节。

主函数程序(文件名:pibub 0.c)如下：

```
# include "stdio.h"
# include "pibub.c"
# include "mrabs.c"
main()
{
    int i, j, p[100], r0, * r, * s;
    r0 = 5; r = &r0; s = p + 30;
    mrabs(100, 300, r, p, 100);
    printf("\n");
    for (i = 0; i < = 9; i++)
        for (j = 0; j < = 9; j++)
```

第二篇 非数值计算

```

    printf("%d", p[10 * i + j]);
    printf("\n");
}
printf("\n");
pibub(s, 50);
for (i = 0; i <= 9; i++)
{
    for (j = 0; j <= 9; j++)
        printf("%d", p[10 * i + j]);
    printf("\n");
}
printf("\n");

```

运行结果为:(前半部分为随机整数序列,后半部分为排序后的结果。)

106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
134	271	188	285	258	123	216	169	190	295
117	186	275	208	129	246	172	205	114	171
200	292	293	298	192	102	111	156	125	226
219	184	265	158	135	276	213	154	115	176
225	214	159	140	267	168	185	270	183	260
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108

106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
102	111	114	115	117	123	125	129	134	135
140	154	156	158	159	168	169	171	172	176
183	184	185	186	188	190	192	200	205	208
213	214	216	219	225	226	246	258	260	265
267	270	271	275	276	285	292	293	295	298
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108

16.2 实数冒泡排序

一、功能

用冒泡排序法将一个实型无序序列排序成有序(非递减)序列。

二、方法说明

同 16.1 节。

三、函数语句

void prpub(p, n)

四、形参说明

p——双精度实型数组的指针。指示待排序序列(为顺序存储)的起始位置。

n——整型变量。待排序序列的长度。

五、函数程序(文件名:prpub.c)

六、例

产生 100 到 300 之间的 100 个实型随机数,然后对第 31 个随机数开始的后 50 个随机数用冒泡法进行排序。本主函数要调用产生 0 到 1 之间随机数序列的函数 mrnds,参看 13.2 节。

主函数程序(文件名:prpub 0.c)如下:

```

#include "stdio.h"
#include "prpub.c"
#include "mrnds.c"
main()
{
    int i, j;
    double p[100], r0, *r, *s;
    r0 = 5; r = &r0; s = p + 30;
    mrnds(r, p, 100);
    for (i = 0; i <= 99; i++)
        p[i] = 100.0 + 200.0 * p[i];
    printf("\n");
    for (i = 0; i <= 9; i++)
    {
        for (j = 0; j <= 9; j++)
            printf("%7.3f", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");
    prpub(s, 50);
    for (i = 0; i <= 9; i++)
    {
        for (j = 0; j <= 9; j++)
            printf("%7.3f", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");
}

```

运行结果为:(前半部分为待排序序列,后半部分为排序结果。)

173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
147.546	154.996	248.492	197.226	147.144	127.982	188.422	271.872	195.374	144.131
144.171	225.620	239.111	137.820	187.616	217.844	275.232	293.423	240.631	257.919
150.610	245.316	275.092	205.222	162.122	277.872	112.872	169.125	256.641	125.467
225.848	109.006	231.042	172.476	135.895	234.238	132.202	253.275	214.886	204.196
256.995	252.237	284.698	128.256	152.295	103.738	217.206	165.793	214.978	192.154

```

134.283 126.181 291.919 151.859 207.806 268.796 279.993 267.227 259.650 103.012
126.074 272.635 161.688 188.205 227.039 152.457 235.797 133.749 229.761 141.068

173.590 222.714 274.585 165.189 174.457 202.048 246.252 198.526 216.144 185.483
238.428 134.402 170.526 132.394 247.986 157.193 159.479 252.158 121.802 101.273
154.877 204.147 156.750 251.016 178.595 297.739 299.707 140.799 102.509 292.307
103.738 109.006 112.872 125.467 127.982 128.256 132.202 135.895 137.820 144.131
144.171 147.144 147.546 150.610 152.295 154.996 162.122 165.793 169.125 172.476
187.616 188.422 192.154 195.374 197.226 204.196 205.222 214.886 214.978 217.206
217.844 225.620 225.848 231.042 234.238 239.111 240.631 245.316 248.492 252.237
253.275 256.641 256.995 257.919 271.872 275.092 275.232 277.872 284.698 293.423
134.283 126.181 291.919 151.859 207.806 268.796 279.993 267.227 259.650 103.012
126.074 272.635 161.688 188.205 227.039 152.457 235.797 133.749 229.761 141.068

```

16.3 字符冒泡排序

一、功能

用冒泡法将一个无序字符序列(即字符串)整序成有序(非递减)序列。

二、方法说明

同 16.1 节。

三、函数语句

```
void pcbub(p, n)
```

四、形参说明

p——字符串的指针。指示待排序字符串的起始位置。

n——整型变量。待排序字符串的长度。

五、函数程序(文件名:pcbub.c)

六、例

将一个字符串(见主函数程序)的第 3 个字符开始的后 17 个字符用冒泡排序法进行排序。

主函数程序(文件名:pcbub 0.c)如下:

```

#include "stdio.h"
#include "pcbub.c"
main()
{
    char *s;
    static char p[] = {'a', 's', 't', 'b', 'x', 'e', 'f', 'h', 'i', 'g', 'p', 'q', 'a', 'i', 'f', 'j', 'c', 'b', 'a', 'z',
                      'y', 'x'};
    s = p + 2;
    printf("\n");
    printf("%s\n", p);
}

```

```

printf("\n");
pcbub(s, 17);
printf("%s\n", p);
printf("\n");

```

运行结果为:(第一行为原字符序列,第二行为排序结果。)

```
astbxefhigpqajfcbazyx
```

```
asaabbceffghiijpqtxzyx
```

16.4 字符串冒泡排序

一、功能

用冒泡排序法将一组字符串序列整序成有序(非递减)序列。

二、方法说明

同 16.1 节。

三、函数语句

```
void phbub(p, n, k, m)
```

四、形参说明

p——指向字符串指针的一维数组,长度为 n。p(i)(i=0,1,...,n-1)指向字符串序列中的第 i 个字符串。

n——整型变量。指针数组 p 的长度,即字符串序列中字符串个数。

k——整型变量。待排序子序列的起始位置。要求 k ≥ 0。

m——整型变量。待排序子序列的终止位置。要求 m ≤ n - 1。

五、函数程序(文件名:phbub.c)

六、例

用冒泡法对单词序列(见主函数程序)进行排序。

主函数程序(文件名:phbub 0.c)如下:

```

#include "stdio.h"
#include "phbub.c"
main()
{
    int i;
    static char *p[10] = {"main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub"};
    printf("\n");
    for (i = 0; i <= 9; i++)
        printf("%s, ", p[i]);
    printf("\n");
}

```

```

phbub(p,10,0,9);
printf("\n");
for (i=0; i<=9; i++)
    printf("%s ", p[i]);
printf("\n");
printf("\n");
}

```

运行结果为:(第一行为原单词序列,第二行为排序结果。)

```

main,gou,zhao,lin,wang,zhang,li,zhen,ma,sub,
gou,li,lin,ma,main,sub,wang,zhang,zhao,zhen,

```

16.5 整数快速排序

一、功能

用快速排序法将整型无序序列整序成有序(非递减)序列。

二、方法说明

通过一次分割,将无序序列分成两部分,其中前一部分的元素值均不大于后一部分的元素值。然后对每一部分利用同样的方法进行分割。这个过程一直做到每一个子序列的长度小于某个值 m 为止。

对序列 p 的分割过程如下:

首先,在序列的第一个、中间一个及最后一个元素中选取中项,设为 $p(k)$,并将 $p(k)$ 赋给 t ,再将序列中的第一个元素移到 $p(k)$ 的位置上。然后设置两个指针 i 和 j 分别指向序列的起始和最后的位置。反复作以下两步,直到 $i=j$ 为止,此时将 t 移到 $p(i)$ 的位置上:

(1) 将 j 逐渐减小,并逐次比较 $p(j)$ 与 t ,直到发现一个 $p(j) < t$ 为止,将 $p(j)$ 移到 $p(i)$ 的位置上;

(2) 将 i 逐渐增大,并逐次比较 $p(i)$ 与 t ,直到发现一个 $p(i) > t$ 为止,将 $p(i)$ 移到 $p(j)$ 的位置上。

三、函数语句

```
void piqck(p, n)
```

四、形参说明

p —整型数组的指针。指示待排序序列的起始位置。

n —整型变量。待排序列的长度。

本函数要调用整数冒泡排序函数 pibub,参看 16.1 节。

五、函数程序(文件名:piqck.c)

六、例

产生 100 到 300 之间的 100 个随机整数,然后对其中第 31 个开始的后 50 个随机数用

快速排序法进行排序。本主函数要调用产生随机整数序列的函数 mrabs,参看 13.4 节。

主函数程序(文件名:piqck0.c)如下:

```

# include "stdio.h"
# include "piqck.c"
# include "mrabs.c"
# include "pibub.c"
main()
{
    int i, j, p[100], r0, *r, *s;
    r0 = 5; r = &r0; s = p + 30;
    mrabs(100, 300, r, p, 100);
    printf("\n");
    for (i=0; i<=9; i++)
        for (j=0; j<=9; j++)
            printf("%d ", p[10*i+j]);
    printf("\n");
}
printf("\n");
piqck(s, 50);
for (i=0; i<=9; i++)
    for (j=0; j<=9; j++)
        printf("%d ", p[10*i+j]);
printf("\n");
}
printf("\n");

```

运行结果为:(前半部分为随机整数序列,后半部分为排序结果。)

106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
134	271	188	285	258	123	216	169	190	295
117	186	275	208	129	246	172	205	114	171
200	292	293	298	192	102	111	156	125	226
219	184	265	158	135	276	213	154	115	176
225	214	159	140	267	168	185	270	183	260
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108
106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
102	111	114	115	117	123	125	129	134	135
140	154	156	158	159	168	169	171	172	176
183	184	185	186	188	190	192	200	205	208
213	214	216	219	225	226	246	258	260	265
267	270	271	275	276	285	292	293	295	298
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108

16.6 实数快速排序

一、功能

用快速排序法将一个实型无序序列整序成有序(非递减)序列。

二、方法说明

同 16.5 节。

三、函数语句

```
void prqck(p, n)
```

四、形参说明

p——双精度实型数组的指针。指示待排序序列的起始位置。

n——整型变量。待排序序列的长度。

本函数要调用实数冒泡排序函数 prbub, 参看 16.2 节。

五、函数程序(文件名:prqck.c)

六、例

产生 100 到 300 之间的 100 个实型随机数, 然后对第 31 个开始的后 50 个随机数用快速排序法进行排序。本主函数要调用产生 0 到 1 之间随机数序列的函数 mrnds, 参看 13.2 节。

主函数程序(文件名:prqck 0.c)如下:

```
# include "stdio.h"
# include "prqck.c"
# include "mrnds.c"
# include "prbub.c"
main()
| int i, j;
double p[100], r0, *r, *s;
r0 = 5; r = &r0; s = p + 30;
mrnds(r, p, 100);
for (i = 0; i <= 99; i++)
    p[i] = 100.0 + 200.0 * p[i];
printf("\n");
for (i = 0; i <= 9; i++)
    for (j = 0; j <= 9; j++)
        printf("%7.3f", p[10 * i + j]);
printf("\n");
printf("\n");
prqck(s, 50);
```

```
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%7.3f", p[10 * i + j]);
            printf("\n");
    printf("\n");
```

运行结果为:(前半部分为随机数序列, 后半部分为排序结果。)

173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
147.546	154.996	248.492	197.226	147.144	127.982	188.422	271.872	195.374	144.131
144.171	225.620	239.111	137.820	187.616	217.844	275.232	293.423	240.631	257.919
150.610	245.316	275.092	205.222	162.122	277.872	112.872	169.125	256.641	125.467
225.848	109.006	231.042	172.476	135.895	234.238	132.202	253.275	214.886	204.196
256.995	252.237	284.698	128.256	152.295	103.738	217.206	165.793	214.978	192.154
134.283	126.181	291.919	151.895	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068
173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
103.738	109.006	112.872	125.467	127.982	128.256	132.202	135.895	137.820	144.131
144.171	147.144	147.546	150.610	152.295	154.996	162.122	165.793	169.125	172.476
187.616	188.422	192.154	195.374	197.226	204.196	205.222	214.886	214.978	217.206
217.844	225.620	225.848	231.042	234.238	239.111	240.631	245.316	248.492	252.237
253.275	256.641	256.995	257.919	271.872	275.092	275.232	277.872	284.698	293.423
134.283	126.181	291.919	151.895	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068

16.7 字符快速排序

一、功能

用快速排序法将一个无序字符序列(即字符串)整序成有序(非递减)序列。

二、方法说明

同 16.5 节。

三、函数语句

```
void pcqck(p, n)
```

四、形参说明

p——字符串的指针。指示待排序字符串的起始位置。

n——整型变量。待排序字符串的长度。

本函数要调用字符串冒泡排序函数 pcpub, 参看 16.3 节。

五、函数程序(文件名:pcqck.c)

六、例

从给定字符串(见主函数程序)的第 3 个字符开始的后 17 个字符用快速排序法进行排序。

主函数程序(文件名:pcqck 0.c)如下:

```
#include "stdio.h"
#include "pcqck.c"
#include "pcpub.c"
main()
{
    char * s;
    static char p[] = {'a', 's', 't', 'b', 'x', 'e', 'f', 'h', 'i', 'g', 'p', 'q', 'a', 'i', 'f', 'j', 'c', 'b', 'a', 'z', 'y', 'x'};
    s = p + 2;
    printf("\n");
    printf("%s\n", p);
    printf("\n");
    pcqck(s, 17);
    printf("%s\n", p);
    printf("\n");
}
```

运行结果为:(第一行为字符序列, 第二行为排序结果。)

```
astbxeftghipqajcbazyx
```

```
asaabbceffghiijpqtxzyx
```

16.8 字符串快速排序

一、功能

用快速排序法将一组字符串序列整序成有序(非递减)序列。

二、方法说明

同 16.5 节。

三、函数语句

```
void phqck(p, n, k, m)
```

四、形参说明

p——指向字符串指针的一维数组, 长度为 n。p(i)(i=0, 1, ..., n-1)指向字符串序列中的第 i 个字符串。

n——整型变量。指针数组 p 的长度, 即字符串序列中字符串个数。

k——整型变量。待排序子序列的起始位置。要求 k ≥ 0。

m——整型变量。待排序子序列的终止位置。要求 m ≤ n - 1。

本函数要调用字符串冒泡排序函数 phpub, 参看 16.4 节。

五、函数程序(文件名:phqck.c)

六、例

用快速排序法对给定单词序列(见主函数程序)进行排序。

主函数程序(文件名:phqck 0.c)如下:

```
#include "stdio.h"
#include "phqck.c"
#include "phpub.c"
main()
{
    int i, j;
    static char * p[18] = {"main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub", "china",
                          "beijing", "liang", "juan", "yan", "teacher", "student", "qssort"};
    printf("\n");
    for (j = 0; j <= 1; j++)
        for (i = 0; i <= 8; i++)
            printf("%s, ", p[9 * j + i]);
    printf("\n");
    phqck(p, 18, 0, 17);
    printf("\n");
    for (j = 0; j <= 1; j++)
        for (i = 0; i <= 8; i++)
            printf("%s, ", p[9 * j + i]);
    printf("\n");
    printf("\n");
}
```

运行结果为:(前两行为原单词序列, 后两行为排序结果。)

```
main, gou, zhao, lin, wang, zhang, li, zhen, ma,
sub, china, beijing, liang, juan, yan, teacher, student, qssort,
```

```
beijing, china, gou, juan, li, liang, lin, ma, main,
qssort, student, sub, teacher, wang, yan, zhang, zhao, zhen,
```

16.9 整数希尔排序

一、功能

用希尔(Shell)排序法将一个整型无序序列整序成有序(非递减)序列。

二、方法说明

希尔排序的基本思想是：将整个无序序列分割成若干小的子序列分别进行插入排序。子序列的分割方法为：将相隔某个增量 h 的元素构成一个子序列。在排序过程中，逐次减小这个增量，最后当 h 减到 1 时，进行一次插入排序，排序就完成。在本函数中，增量序列取 $h_i = 2^i - 1, 1 \leq i \leq \lceil \log_2 n \rceil$ 其中 n 为待排序序列的长度。

三、函数语句

```
void pishl(p, n)
```

四、形参说明

p——整型数组的指针。指示待排序序列的起始位置。

n——整型变量。待排序序列的长度。

五、函数程序(文件名:pishl.c)

六、例

产生 100 到 300 之间的 100 个随机整数，然后对其中第 31 个开始的后 50 个随机数用希尔排序法进行排序。本主函数要调用产生随机整数序列的函数 mrabs，参看 13.4 节。

主函数程序(文件名:pishl0.c)如下：

```
#include "stdio.h"
#include "pishl.c"
#include "mrabs.c"

main()
{
    int i, j, p[100], r0, *r, *s;
    r0 = 5; r = &r0; s = p + 30;
    mrabs(100, 300, r, p, 100);
    printf("\n");
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%d ", p[10 * i + j]);
    printf("\n");
    pishl(s, 50);
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%d ", p[10 * i + j]);
    printf("\n");
    printf("\n");
}
```

运行结果为：(前半部分为随机数序列，后半部分为排序结果。)

106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
134	271	188	285	258	123	216	169	190	295
117	186	275	208	129	246	172	205	114	171
200	292	293	298	192	102	111	156	125	226
219	184	265	158	135	276	213	154	115	176
225	214	159	140	267	168	185	270	183	260
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108

16.10 实数希尔排序

一、功能

用希尔(Shell)排序法将一个实型无序序列整序成有序(非递减)序列。

二、方法说明

同 16.9 节。

三、函数语句

```
void prshl(p, n)
```

四、形参说明

p——双精度实型数组的指针。指示待排序序列的起始位置。

n——整型变量。待排序序列的长度。

五、函数程序(文件名:prshl.c)

六、例

产生 100 到 300 之间的 100 个实型随机数，然后对其中第 31 个开始的后 50 个随机数用希尔排序法进行排序。在本主函数中要调用产生 0 到 1 之间随机数序列的函数 mrnd，参看 13.2 节。

主函数程序(文件名:prshl0.c)如下：

```

# include "stdio.h"
# include "prshl.c"
# include "mrnds.c"
main()
{
    int i, j;
    double p[100], r0, * r, * s;
    r0 = 5; r = &r0; s = p + 30;
    mrnds(r, p, 100);
    for (i = 0; i <= 99; i++)
        p[i] = 100.0 + 200.0 * p[i];
    printf("\n");
    for (i = 0; i <= 9; i++)
    {
        for (j = 0; j <= 9; j++)
            printf("%7.3f ", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");
    prshl(s, 50);
    for (i = 0; i <= 9; i++)
    {
        for (j = 0; j <= 9; j++)
            printf("%7.3f ", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");
}

```

运行结果为:(前半部分为随机数序列,后半部分为排序结果。)

173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
147.546	154.996	248.492	197.226	147.144	127.982	188.422	271.872	195.374	144.131
144.171	225.620	239.111	137.820	187.616	217.844	275.232	293.423	240.631	257.919
150.610	245.316	275.092	205.222	162.122	277.872	112.872	169.125	256.641	125.467
225.848	109.006	231.042	172.476	135.895	234.238	132.202	253.275	214.886	204.196
256.995	252.237	284.698	128.256	152.295	103.738	217.206	165.793	214.978	192.154
134.283	126.181	291.919	151.859	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068
173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
103.738	109.006	112.872	125.467	127.982	128.256	132.202	135.895	137.820	144.131
144.171	147.144	147.546	150.610	152.295	154.996	162.122	165.793	169.125	172.476
187.616	188.422	192.154	195.374	197.226	204.196	205.222	214.886	214.978	217.206
217.844	225.620	225.848	231.042	234.238	239.111	240.631	245.316	248.492	252.237
253.275	256.641	256.995	257.919	271.872	275.092	275.232	277.872	284.698	293.423
134.283	126.181	291.919	151.859	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068

16.11 字符希尔排序

一、功能

用希尔(Shell)排序法将一个无序字符序列(即字符串)整序成有序(非递减)序列。

二、方法说明

同 16.9 节。

三、函数语句

```
void pcshl(p, n)
```

四、形参说明

p——字符串的指针。指示待排序字符串的起始位置。

n——整型变量。待排序字符串的长度。

五、函数程序(文件名:pcshl.c)

六、例

从给定字符串(见主函数程序)的第 3 个字符开始的后 17 个字符用希尔排序法进行排序。

主函数程序(文件名:pcshl 0.c)如下:

```

#include "stdio.h"
#include "pcshl.c"
main()
{
    char * s;
    static char p[] = {'a', 's', 't', 'b', 'x', 'f', 'h', 'i', 'g', 'p', 'q', 'a', 'i', 'f', 'j', 'c', 'b', 'a', 'z', 'y', 'x'};
    s = p + 2;
    printf("\n");
    printf("%s\n", p);
    printf("\n");
    pcshl(s, 17);
    printf("%s\n", p);
    printf("\n");
}

```

运行结果为:(第一行为原字符序列,第二行为排序结果。)

```
astbxefhigpqajfcbazyx
asaabbceffghijpqtxzyx
```

16.12 字符串希尔排序

一、功能

用希尔(Shell)排序法将一组字符串序列整序成有序(非递减)序列。

二、方法说明

同 16.9 节。

三、函数语句

```
void phshl(p, n, k, m)
```

四、形参说明

p——指向字符串指针的一维数组, 长度为 n。p(i)(i=0, 1, ..., n-1)指向字符串序列中的第 i 个字符串。

n——整型变量。指针数组 p 的长度, 即字符串序列中字符串个数。

k——整型变量。待排序子序列的起始位置。要求 k ≥ 0。

m——整型变量。待排序子序列的终止位置。要求 m ≤ n - 1。

五、函数程序(文件名:phshl.c)

六、例

用希尔排序法对给定的单词序列(见主函数程序)进行排序。

主函数程序(文件名:phshl 0.c)如下:

```
# include "stdio.h"
# include "phshl.c"
main()
{
    int i, j;
    static char * p[18] = {"main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub", "china",
                          "beijing", "liang", "juan", "yan", "teacher", "student", "qssort"};
    printf("\n");
    for (j = 0; j <= 1; j++)
        for (i = 0; i <= 8; i++)
            printf("%s, ", p[9 * j + i]);
    printf("\n");

    phshl(p, 18, 0, 17);
    printf("\n");
    for (j = 0; j <= 1; j++)
        for (i = 0; i <= 8; i++)
            printf("%s, ", p[9 * j + i]);
    printf("\n");
}
```

```
printf("\n");
```

运行结果为:(前两行为原单词序列, 后两行为排序结果。)

```
main, gou, zhao, lin, wang, zhang, li, zhen, ma,
sub, china, beijing, liang, juan, yan, teacher, student, qssort,
beijing, china, gou, juan, li, liang, lin, ma, main,
qssort, student, sub, teacher, wang, yan, zhang, zhao, zhen,
```

16.13 整数堆排序

一、功能

用堆(Heap)排序法将一个整型无序序列整序成有序(非递减)序列。

二、方法说明

堆的定义:具有 n 个元素的序列(h₁, h₂, ..., h_n), 当且仅当满足

$$\begin{cases} h_i \geq h_{2i}, \\ h_i \geq h_{2i+1} \end{cases}, i = 1, 2, \dots, \frac{n}{2}$$

时称之为堆。

由堆的这个定义可以看出, 堆顶元素(即第一个元素)必为最大项。堆排序的过程如下:

首先, 将一个无序序列建成一个堆; 然后, 对于 n 到 2 的每一个 j, 作如下操作:

- (1) 将第一个与第 j 个元素互相交换;
- (2) 将第一个到第 j - 1 个元素重新调整为堆。

三、函数语句

```
void pihap(p, n)
```

四、形参说明

p——整型数组的指针。指示待排序序列的起始位置。

n——整型变量。待排序序列的长度。

五、函数程序(文件名:pihap.c)

六、例

产生 100 到 300 之间的 100 个随机整数, 然后对其中第 31 个开始的后 50 个随机数用堆排序法进行排序。本主函数要调用产生随机整数序列的函数 mrabs, 参看 13.4 节。

主函数程序(文件名:pihap 0.c)如下:

```
# include "stdio.h"
# include "pihap.c"
```

```

# include "mrabs.c"
main()
{
    int i, j, p[100], r0, * r, * s;
    r0 = 5; r = &r0; s = p + 30;
    mrabs(100, 300, r, p, 100);
    printf("\n");
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%d ", p[10 * i + j]);
    printf("\n");
}
printf("\n");
pihap(s, 50);
for (i = 0; i <= 9; i++)
    for (j = 0; j <= 9; j++)
        printf("%d ", p[10 * i + j]);
    printf("\n");
}
printf("\n");

```

运行结果为:(前半部分为随机整数序列,后半部分为排序结果。)

106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
134	271	188	285	258	123	216	169	190	295
117	186	275	208	129	246	172	205	114	171
200	292	293	298	192	102	111	156	125	226
219	184	265	158	135	276	213	154	115	176
225	214	159	140	267	168	185	270	183	260
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108
106	131	256	113	166	175	220	189	290	283
248	222	199	277	218	179	240	289	278	223
204	109	146	232	249	247	197	227	224	209
102	111	114	115	117	123	125	129	134	135
140	154	156	158	159	168	169	171	172	176
183	184	185	186	188	190	192	200	205	208
213	214	216	219	225	226	246	258	260	265
267	270	271	275	276	285	292	293	295	298
133	266	163	160	145	207	124	221	194	152
105	126	231	244	122	211	144	182	255	108

16.14 实数堆排序

一、功能

用堆(Heap)排序法将一个实型无序序列整序成有序(非递减)序列。

二、方法说明

同 16.13 节。

三、函数语句

void prhap(p, n)

四、形参说明

p —— 双精度实型数组的指针。指示待排序序列的起始位置。

n —— 整型变量。待排序序列的长度。

五、函数程序(文件名:prhap.c)

六、例

产生 100 到 300 之间的 100 个实型随机数,然后对第 31 个开始的后 50 个随机数用堆排序法进行排序。本主函数要调用产生 0 到 1 之间随机数的函数 mrnds,参看 13.2 节。

主函数程序(文件名:prhap 0.c)如下:

```

# include "stdio.h"
# include "prhap.c"
# include "mrnds.c"
main()
{
    int i, j;
    double p[100], r0, * r, * s;
    r0 = 5; r = &r0; s = p + 30;
    mrnds(r, p, 100);
    for (i = 0; i <= 99; i++)
        p[i] = 100.0 + 200.0 * p[i];
    printf("\n");
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%7.3f ", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");
    prhap(s, 50);
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%7.3f ", p[10 * i + j]);
        printf("\n");
    }
    printf("\n");

```

运行结果为:(前半部分为随机数序列,后半部分为排序结果。)

173.590 222.714 274.585 165.189 174.457 202.048 246.252 198.526 216.144 158.483

238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
147.546	154.996	248.492	197.226	147.144	127.982	188.422	271.872	195.374	144.131
144.171	225.620	239.111	137.820	187.616	217.844	275.232	293.423	240.631	257.919
150.610	245.316	275.092	205.222	162.122	277.872	112.872	169.125	256.641	125.467
225.848	109.006	231.042	172.476	135.895	234.238	132.202	253.275	214.886	204.196
256.995	252.237	284.698	128.256	152.295	103.738	217.206	165.793	214.978	192.154
134.283	126.181	291.919	151.859	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068
173.590	222.714	274.585	165.189	174.457	202.048	246.252	198.526	216.144	185.483
238.428	134.402	170.526	132.394	247.986	157.193	159.479	252.158	121.802	101.273
154.877	204.147	156.750	251.016	178.595	297.739	299.707	140.799	102.509	292.307
103.738	109.006	112.872	125.467	127.982	128.256	132.202	135.895	137.820	144.131
144.171	147.144	147.546	150.610	152.295	154.996	162.122	165.793	169.125	172.476
187.616	188.422	192.154	195.374	197.226	204.196	205.222	214.886	214.978	217.206
217.844	225.620	225.848	231.042	234.238	239.111	240.631	245.316	248.492	252.237
253.275	256.641	256.995	257.919	271.872	275.092	275.232	277.872	284.698	293.423
134.283	126.181	291.919	151.859	207.806	268.796	279.993	267.227	259.650	103.012
126.074	272.635	161.688	188.205	227.039	152.457	235.797	133.749	229.761	141.068

16.15 字符堆排序

一、功能

用堆(Heap)排序法将一个无序字符序列(即字符串)整序成有序(非递减)序列。

二、方法说明

同 16.13 节。

三、函数语句

void pchap(p, n)

四、形参说明

p——字符串的指针。指示待排序字符串的起始位置。

n——整型变量。待排序字符串的长度。

五、函数程序(文件名:pchap.c)

六、例

从给定字符串(见主函数程序)中第 3 个开始的后 17 个字符用堆排序法进行排序。

主函数程序(文件名:pchap 0.c)如下:

```
# include "stdio.h"
# include "pchap.c"
main()
```

```
| char * s;
static char p[] = {'a','s','t','b','x','e','f','h','i','g','p','q','a','i','f','j','c','b','a','z',
'y','x'};
s = p + 2;
printf("\n");
printf("%s\n",p);
printf("\n");
pchap(s,17);
printf("%s\n",p);
printf("\n");
```

运行结果为:(第一行为原字符序列,第二行为排序结果。)

```
astbxefhigpqaijfcbazyx
asaabbceffghijpqtxzyx
```

16.16 字符串堆排序

一、功能

用堆(Heap)排序法将一组字符串序列整序成有序(非递减)序列。

二、方法说明

同 16.13 节。

三、函数语句

void phhap(p, n, k, m)

四、形参说明

p——指向字符串指针的一维数组,长度为 n。p(i)(i=0,1,...,n-1)指向字符串序列中的第 i 个字符串。

n——整型变量。指针数组 p 的长度,即字符串序列中字符串个数。

k——整型变量。待排序子序列的起始位置。要求 k ≥ 0。

m——整型变量。待排序子序列的终止位置。要求 m ≤ n - 1。

五、函数程序(文件名:phhap.c)

六、例

用堆排序法对给定的单词序列(见主函数程序)中第 3 个开始的后 15 个单词进行排序。

主函数程序(文件名:phhap 0.c)如下:

```
# include "stdio.h"
# include "phhap.c"
main()
```

```

int i, j;
static char * p[18] = {"main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub", "china",
                      "beijin", "liang", "juan", "yan", "teacher", "student", "qssort"};
printf("\n");
for (j = 0; j <= 1; j++)
    for (i = 0; i <= 8; i++)
        printf("%s, ", p[9 * j + i]);
printf("\n");
}
phap(p, 18, 2, 15);
printf("\n");
for (j = 0; j <= 1; j++)
    for (i = 0; i <= 8; i++)
        printf("%s, ", p[9 * j + i]);
printf("\n");
}
printf("\n");

```

运行结果为:(前两行为原单词序列,后两行为排序结果。)

```

main, gou, zhao, lin, wang, zhang, li, zhen, ma,
sub, china, beijin, liang, juan, yan, teacher, student, qssort,
main, gou, beijin, china, juan, li, liang, lin, ma,
sub, teacher, wang, yan, zhang, zhao, zhen, student, qssort,

```

16.17 关键字成员为整数的结构排序

一、功能

用堆(Heap)排序法对关键字成员为整数的结构体一维数组进行排序。

二、方法说明

堆排序的方法见 16.13 节的方法说明。

现针对结构排序的特点,另作以下几点说明。

(1) 在主函数外部对待排序结构体数组的结构体类型进行定义,并用常量 HEAPSORT 代替结构体类型定义中的

struct 结构体名

即结构体类型的定义格式为

```
#define HEAPSORT struct 结构体名
```

```
HEAPSORT
```

```
{结构体成员表列}
```

(2) 在主函数外部还需作如下说明:

```
#define KEY 关键字成员名
```

即用常量代替排序用的关键字成员名。

(3) 在主函数中用一个结构体指针数组,其中每一个指针元素指向结构体类型的各元素。在实际排序过程中,并不交换结构体类型的各元素,而只是交换指针数组中的各指针。因此,排序的最后结果,其结构体类型的各元素之间的存储顺序关系并没有改变,则只是按结构体指针数组中各指针元素顺序所指向的结构体类型元素关于关键字成员是有序的。

三、函数语句

```
void pikey(p, n, k, m)
```

四、形参说明

p —— 结构体指针数组,长度为 n。依次指向待排序结构体数组的各元素。

n —— 整型变量。待排序结构体数组的长度。

k —— 整型变量。结构体数组中待排序部分的起始下标。要求 k ≥ 0。

m —— 整型变量。结构体数组中待排序部分的终止下标。要求 m ≤ n - 1。

五、函数程序(文件名:pikey.c)

六、例

对下列结构体数组以成员 age(为整型)为关键字从第 2 个元素到第 9 个元素进行排序:

```

#define HEAPSORT struct student
HEAPSORT
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
} stu[10] =
{
    {101, "Zhang", 'M', 19, 95.6},
    {102, "Wang", 'F', 18, 92.4},
    {103, "Zhao", 'M', 19, 85.7},
    {104, "Li", 'M', 20, 96.3},
    {105, "Gou", 'M', 19, 90.2},
    {106, "Lin", 'M', 18, 91.5},
    {107, "Ma", 'F', 17, 98.7},
    {108, "Zhen", 'M', 21, 90.1},
    {109, "Xu", 'M', 19, 89.8},
    {110, "Ma", 'F', 18, 94.9},
}

```

主函数程序(文件名:pikey 0.c)如下:

```

#define HEAPSORT struct student
#define KEY age
HEAPSORT
{
    int num;
    char name[8];
    char sex;
    int age;
}

```

```

double score;
{
#include "stdio.h"
#include "pikey.c"
main()
{
    int i;
    static HEAPSORT stu[10] = {{101,"Zhang", 'M', 19, 95.6}, {102,"Wang", 'F', 18, 92.4}, {103,"Zhao", 'M', 19, 85.7}, {104,"Li", 'M', 20, 96.3}, {105,"Gou", 'M', 19, 90.2}, {106,"Lin", 'M', 18, 91.5}, {107,"Ma", 'F', 17, 98.7}, {108,"Zhen", 'M', 21, 90.1}, {109,"Xu", 'M', 19, 89.8}, {110,"Mao", 'F', 18, 94.9}};
    HEAPSORT * p[10];
    for (i=0; i<=9; i++) p[i] = &stu[i];
    printf("\n");
    printf("No. Name Sex Age Score\n");
    for (i=0; i<=9; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[i]).num, (*p[i]).name, (*p[i]).sex,
               (*p[i]).age, (*p[i]).score);
    printf("\n");
    pikey(p,10,1,8);
    printf("No. Name Sex Age Score\n");
    for (i=0; i<=9; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[i]).num, (*p[i]).name, (*p[i]).sex,
               (*p[i]).age, (*p[i]).score);
    printf("\n");
}

```

运行结果为:(前半部分为原序列,后半部分为排序结果。)

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
107	Ma	F	17	98.70
106	Lin	M	18	91.50
102	Wang	F	18	92.40
109	Xu	M	19	89.80
105	Gou	M	19	90.20
103	Zhao	M	19	85.70
104	Li	M	20	96.30
108	Zhen	M	21	90.10

110 Mao F 18 94.90

16.18 关键字成员为实数的结构排序

一、功能

用堆(Heap)排序法对关键字成员为实数的结构体一维数组进行排序。

二、方法说明

同 16.17 节。

三、函数语句

void prkey(p, n, k, m)

四、形参说明

p —— 结构体指针数组, 长度为 n。依次指示待排序结构体组数的各元素。

n —— 整型变量。待排序结构体数组的长度。

k —— 整型变量。结构体数组中待排序部分的起始下标。要求 $k \geq 0$ 。

m —— 整型变量。结构体数组中待排序部分的终止下标。要求 $m \leq n - 1$ 。

五、函数程序(文件名:prkey.c)

六、例

对下列结构体数组以成员 score(双精度实型)为关键字从第 2 个元素到第 9 个元素进行排序:

```

#define HEAPSORT struct student
HEAPSORT
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
} stu[10] =
{{101,"Zhang", 'M', 19, 95.6},
 {102,"Wang", 'F', 18, 92.4},
 {103,"Zhao", 'M', 19, 85.7},
 {104,"Li", 'M', 20, 96.3},
 {105,"Gou", 'M', 19, 90.2},
 {106,"Lin", 'M', 18, 91.5},
 {107,"Ma", 'F', 17, 98.7},
 {108,"Zhen", 'M', 21, 90.1},
 {109,"Xu", 'M', 19, 89.8},
 {110,"Mao", 'F', 18, 94.9}};

```

主函数程序(文件名:prkey 0.c)如下:

```
#define HEAPSORT struct student
#define KEY score
HEAPSORT
{ int num;
  char name[8];
  char sex;
  int age;
  double score;
}

#include "stdio.h"
#include "prkey.c"
main()
{ int i;
  static HEAPSORT stu[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
  HEAPSORT * p[10];
  for (i = 0; i <= 9; i++) p[i] = &stu[i];
  printf("\n");
  printf("No. Name Sex Age Score\n");
  for (i = 0; i <= 9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
           (*p[i]).num, (*p[i]).name, (*p[i]).sex,
           (*p[i]).age, (*p[i]).score);
  printf("\n");
  prkey(p, 10, 1, 8);
  printf("No. Name Sex Age Score\n");
  for (i = 0; i <= 9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
           (*p[i]).num, (*p[i]).name, (*p[i]).sex,
           (*p[i]).age, (*p[i]).score);
  printf("\n");
}
```

运行结果为:(前半部分为原序列,后半部分为排序结果。)

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60

103	Zhao	M	19	85.70
109	Xu	M	19	89.80
108	Zhen	M	21	90.10
105	Gou	M	19	90.20
106	Lin	M	18	91.50
102	Wang	F	18	92.40
104	Li	M	20	96.30
107	Ma	F	17	98.70
110	Mao	F	18	94.90

16.19 关键字成员为字符的结构排序

一、功能

用堆(Heap)排序法对关键字成员为字符的结构体一维数组进行排序。

二、方法说明

同 16.17 节。

三、函数语句

```
void pkkey(p, n, k, m)
```

四、形参说明

p —— 结构体指针数组, 长度为 n。依次指示待排序结构体数组的各元素。

n —— 整型变量。待排序结构体数组的长度。

k —— 整型变量。结构体数组中待排序部分的起始下标。要求 $k \geq 0$ 。

m —— 整型变量。结构体数组中待排序部分的终止下标。要求 $m \leq n - 1$ 。

五、函数程序(文件名:pkkey.c)

六、例

对下列结构体数组以成员 sex(字符型)为关键字从第 2 个元素到第 9 个元素进行排序:

```
#define HEAPSORT struct student
HEAPSORT
{ int num;
  char name[8];
  char sex;
  int age;
  double score;
} stu[10] =
{{101, "Zhang", 'M', 19, 95.6},
 {102, "Wang", 'F', 18, 92.4},
 {103, "Zhao", 'M', 19, 85.7},
 {104, "Li", 'M', 20, 96.3},
 {105, "Gou", 'M', 19, 90.2},
```

```

{106, "Lin", 'M', 18, 91.5},
{107, "Ma", 'F', 17, 98.7},
{108, "Zhen", 'M', 21, 90.1},
{109, "Xu", 'M', 19, 89.8},
{110, "Mao", 'F', 18, 94.9}};


```

主函数程序(文件名:pckey 0.c)如下:

```

#define HEAPSORT struct student
#define KEY sex
HEAPSORT
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
}

#include "stdio.h"
#include "pckey.c"
main()
{
    int i;
    static HEAPSORT stu[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
    HEAPSORT * p[10];
    for (i = 0; i <= 9; i++) p[i] = &stu[i];
    printf("\n");
    printf("No. Name Sex Age Score\n");
    for (i = 0; i <= 9; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
            (*p[i]).num, (*p[i]).name, (*p[i]).sex,
            (*p[i]).age, (*p[i]).score);
    printf("\n");
    pckey(p, 10, 1, 8);
    printf("No. Name Sex Age Score\n");
    for (i = 0; i <= 9; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
            (*p[i]).num, (*p[i]).name, (*p[i]).sex,
            (*p[i]).age, (*p[i]).score);
    printf("\n");
}

```

运行结果为:(前半部分为原序列,后半部分为排序结果。)

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70

No.	Name	Sex	Age	Score
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
107	Ma	F	17	98.70
102	Wang	F	18	92.40
104	Li	M	20	96.30
106	Lin	M	18	91.50
109	Xu	M	19	89.80
108	Zhen	M	21	90.10
105	Gou	M	19	90.20
103	Zhao	M	19	85.70
110	Mao	F	18	94.90

16.20 关键字成员为字符串的结构排序

一、功能

用堆(Heap)排序法对关键字成员为字符串的结构体一维数组进行排序。

二、方法说明

同 16.17 节。

三、函数语句

```
void phkey(p, n, k, m)
```

四、形参说明

p —— 结构体指针数组,长度为 n。依次指示待排序结构体数组的各元素。

n —— 整型变量。待排序结构体数组的长度。

k —— 整型变量。结构体数组中待排序部分的起始下标。要示 $k \geq 0$ 。

m —— 整型变量。结构体数组中待排序部分的终止下标。要示 $m \leq n - 1$ 。

五、函数程序(文件名:phkey.c)

六、例

对下列结构体数组以成员 name(字符串)为关键字从第 2 个元素到第 9 个元素进行排序:

```

#define HEAPSORT struct student
HEAPSORT
{
    int num;
    char name[8];
    char sex;
}

```

```

int age;
double score;
stu[10] = {
    {101, "Zhang", 'M', 19, 95.6},
    {102, "Wang", 'F', 18, 92.4},
    {103, "Zhao", 'M', 19, 85.7},
    {104, "Li", 'M', 20, 96.3},
    {105, "Gou", 'M', 19, 90.2},
    {106, "Lin", 'M', 18, 91.5},
    {107, "Ma", 'F', 17, 98.7},
    {108, "Zhen", 'M', 21, 90.1},
    {109, "Xu", 'M', 19, 89.8},
    {110, "Mao", 'F', 18, 94.9}};

```

主函数程序(文件名:phkey 0.c)如下:

```

#define HEAPSORT struct student
#define KEY name
HEAPSORT
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
}
#include "stdio.h"
#include "phkey.c"
main()
{
    int i;
    static HEAPSORT stu[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
    HEAPSORT * p[10];
    for (i = 0; i < 10; i++) p[i] = &stu[i];
    printf("\n");
    printf("No. Name Sex Age Score\n");
    for (i = 0; i < 10; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[i]).num, (*p[i]).name, (*p[i]).sex,
               (*p[i]).age, (*p[i]).score);
    printf("\n");
    phkey(p, 10, 1, 8);
    printf("No. Name Sex Age Score\n");
    for (i = 0; i < 10; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[i]).num, (*p[i]).name, (*p[i]).sex,
               (*p[i]).age, (*p[i]).score);
    printf("\n");
}

```

运行结果为:(前半部分为原序列,后半部分为排序结果。)

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

No.	Name	Sex	Age	Score
101	Zhang	M	19	95.60
105	Gou	M	19	90.20
104	Li	M	20	96.30
106	Lin	M	18	91.50
107	Ma	F	17	98.70
102	Wang	F	18	92.40
109	Xu	M	19	89.80
103	Zhao	M	19	85.70
108	Zhen	M	21	90.10
110	Mao	F	18	94.90

16.21 磁盘文件排序

一、功能

用快速排序法对磁盘随机文本文件进行排序。

二、方法说明

本函数要求待排序的随机文本文件的记录个数已知,并且每一个记录为同一结构体类型,其中的每一个成员为字符数组。

在主函数外部要求作如下说明:

define STU 结构体类型的变量名

define KEY 关键字成员名

define MUDISK struct 结构体名

MUDISK

{ char 成员名 1[X];

:

char KEY[X];

:

} STU;

其中 X 为常量。

三、函数语句

```
void pdisk(fp, n, k, m)
```

本函数用到两个函数：getkey 和 sp。其中函数 getkey 的作用是从文件中取出指定记录，并返回关键字成员的指针；函数 sp 的作用是实现文件中两个指定记录的交换。

四、形参说明

fp —— 文件型指针变量。指向待排序文件。要求待排序文件在主函数中已打开，并由主函数负责关闭。

n —— 长整型变量。待排序文件的记录个数。

k —— 长整型变量。待排序文件的排序起点记录号。要求 $k \geq 0$ 。

m —— 长整型变量。待排序文件的排序终止记录号。要求 $m \leq n - 1$ 。

五、函数程序(文件名:pdisk.c)

六、例

设学生情况登记表如下：

num	name	sex	age	score
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

按如下结构

```
char num[5];
char name[8];
char sex[3];
char age[4];
char score[7];
```

建立一个文件 student.dat。然后按成绩从第二个记录(即记录号为 1)到第九个记录(即记录号为 8)进行排序。分别输出排序前后的文件。

主函数程序(文件名:pdisk 0.c)如下：

```
#define STU stu
#define KEY score
#define MUDISK struct student
```

```

MUDISK
{
    char num[5];
    char name[8];
    char sex[3];
    char age[4];
    char KEY[7];
} STU;

#include "stdio.h"
#include "pdisk.c"

main()
{
    FILE *fp;
    long int n, k, m;
    MUDISK b[10];
    static MUDISK a[10] =
        {{"101", "Zhang", 'M', '19', "95.6"}, 
         {"102", "Wang", 'F', '18', "92.4"}, 
         {"103", "Zhao", 'M', '19', "85.7"}, 
         {"104", "Li", 'M', '20', "96.3"}, 
         {"105", "Gou", 'M', '19', "90.2"}, 
         {"106", "Lin", 'M', '18', "91.5"}, 
         {"107", "Ma", 'F', '17', "98.7"}, 
         {"108", "Zhen", 'M', '21', "90.1"}, 
         {"109", "Xu", 'M', '19', "89.8"}, 
         {"110", "Mao", 'F', '18', "94.9"}};

int i;
fp = fopen("student.dat", "w+");
for (i = 0; i < 9; i++)
    if (fwrite(&a[i], sizeof(stu), 1, fp) != 1)
        printf("file write error\n");
fclose(fp);
if ((fp = fopen("student.dat", "r+")) == NULL)
    printf("cannot open this file\n"); exit(0);
for (i = 0; i < 9; i++)
    fread(&b[i], sizeof(stu), 1, fp);
    printf("%-10s%-10s%-5s%-5s%-8s\n", b[i].num,
           b[i].name, b[i].sex, b[i].age, b[i].score);
}

printf("\n");
n = 10; k = 1; m = 8;
pdisk(fp, n, k, m);
fseek(fp, 0L, 0);
for (i = 0; i < = 9; i++)
    fread(&b[i], sizeof(stu), 1, fp);
    printf("%-10s%-10s%-5s%-5s%-8s\n", b[i].num,
           b[i].name, b[i].sex, b[i].age, b[i].score);
}

printf("\n");
fclose(fp);
}

```

运行结果为：(前半部分为原文件，后半部分为排序后的文件。)

101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

101	Zhang	M	19	95.6
103	Zhao	M	19	85.7
109	Xu	M	19	89.8
108	Zhen	M	21	90.1
105	Gou	M	19	90.2
106	Lin	M	18	91.5
102	Wang	F	18	92.4
104	Li	M	20	96.3
107	Ma	F	17	98.7
110	Mao	F	18	94.9

16.22 拓扑分类

一、功能

对用自然数编号的问题序列进行拓扑分类。

二、方法说明

设具有 n 个问题的问题序列用自然数进行编号,且 n 个问题之间存在如下关系:

$$R = \{(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)\}$$

其中 $1 \leq i_t, j_t \leq n$ ($1 \leq t \leq m$)。 (i_t, j_t) 表示:求解问题 j_t 之前要先求解问题 i_t 。称 i_t 为 j_t 的关于 R 的前件(简称 i_t 为 j_t 的前件), j_t 为 i_t 的关于 R 的后件(简称 j_t 为 i_t 的后件)。

n 个问题关于 R 的拓扑分类就是对 n 个问题的求解顺序。

如果 R 确实定义了一个半序关系,则总可以找到求解 n 个问题的拓扑分类序列。

为了寻找一个拓扑分类序列,首先要定义如下三个数据结构:

$F(1:n)$ 其中 $F(i)$ ($i=1, 2, \dots, n$) 用以存放编号 i 的前件个数。

$G(1:n)$ 其中 $G(i)$ ($i=1, 2, \dots, n$) 用以链接编号 i 关于关系 R 的所有后件的指针。

$S(1:n)$ 作为工作栈,用于存放所有没有前件的编号。

然后根据给定的关系 R 填写上述数据结构。

最后,通过逐步修改上述数据结构,找出拓扑分类序列。其过程如下:

(1) 输出栈顶元素,设为 i ,并将它从栈中删去;

(2) 将 i 的所有后件的前件个数减 1,如果对于某个后件的前件个数减为零时,将此后件压入栈中。

重复上述过程,直到栈空为止。

三、函数语句

```
void ptopo(n, r, m, p)
```

四、形参说明

n ——整型变量。问题的个数。

r ——整型二维数组,体积为 $m \times 2$ 。存放关系 R 。其中每一行为一个二元组。

m ——整型变量。关系 R 中二元组个数。

p ——整型一维数组,长度为 n 。返回拓扑分类序列。若某个元素 $i < 0$,则表示编号为 $|i|$ 的问题无法求解,即 $|i|$ 无法分类。

五、函数程序(文件名:ptopo.c)

六、例

设问题的集合为 $M = \{1, 2, 3, 4, 5, 6, 7\}$,分别寻找关于下列两个关系

$$R_1 = \{(3, 1), (3, 2), (4, 6), (4, 7), (5, 4), (5, 6), (5, 7), (6, 7)\}$$

与

$$R_2 = \{(3, 1), (3, 2), (2, 7), (4, 5), (5, 6), (6, 4)\}$$

的拓扑分类序列。

主函数程序(文件名:ptopo 0.c)如下:

```
# include "stdio.h"
# include "ptopo.c"
main()
{ int p[7], i;
  static int r1[8][2] = { {3,1}, {3,2}, {4,6}, {4,7},
                         {5,4}, {5,6}, {5,7}, {6,7} };
  static int r2[6][2] = { {3,1}, {3,2}, {2,7},
                         {4,5}, {5,6}, {6,4} };
  printf("\n");
  ptopo(7, r1, 8, p);
  for (i=0; i<=6; i++) printf("%d", p[i]);
  printf("\n\n");
  ptopo(7, r2, 6, p);
  for (i=0; i<=6; i++) printf("%d", p[i]);
  printf("\n\n");
}
```

运行结果为:(第一行为关于 R_1 的拓扑分类序列,第二行为关于 R_2 的拓扑分类序列。)

```
5 4 6 7 3 1 2
3 1 2 7 -4 -5 -6
```

第 17 章 查 找

17.1 关键字成员为整型、实型及字符的 结构体数组的顺序查找

一、功能

用顺序法在结构体数组中查找关键字值在某个指定区间内的元素。

二、方法说明

本函数要求在主函数外部作如下说明：

```
#define STRU struct student
#define TYPE double
#define KEY score
STRU
{ 结构体成员表列 }
```

三、函数语句

```
int qsech(p, k, m, a, b)
```

本函数返回关键字值在 $[a, b]$ 内的第一次发现的元素的下标。主函数中可以根据返回的这个下标继续向下查找。从而可以找出关键字值在 $[a, b]$ 内的所有元素。若在指定范围(即 $[k, m]$)内找不到关键字值在 $[a, b]$ 内的元素，则返回的函数值为 -1。

四、形参说明

p —— 结构体类型 STRU 的一维数组。

k —— 整型变量。在数组 p 中进行查找的起始下标。要求 $k \geq 0$ 。

m —— 整型变量。在数组 p 中进行查找的终止下标。要求 m 不大于数组 p 的最大下标。

a —— TYPE(整型、实型或字符型)型变量。存放查找的关键字值的下限值。

b —— TYPE(整型、实型或字符型)型变量。存放查找的关键字值的上限值。

五、函数程序(文件名:qsech.c)

六、例

在下列学生情况表中找出成绩在 90 分到 100 分之间的所有的学生情况：

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

主函数程序(文件名:qsech 0.c)如下：

```
#define STRU struct student
#define TYPE double
#define KEY score
STRU
{ int num;
  char name[8];
  char sex;
  int age;
  double score;
}

#include "stdio.h"
#include "qsech.c"

main()
{ int i, q;
  TYPE a, b;
  static STRU p[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
  a = 90.0; b = 100.0;
  i = 0;
  printf("\n");
  do
  { q = qsech(p, i, 9, a, b);
    if (q != -1)
      printf("%-5d%-10s%-4c%-6d%-7.2f\n", p[q].num, p[q].name, p[q].sex, p[q].age, p[q].score);
    i = q + 1;
  }
  while (q != -1);
  printf("\n");
}
```

运行结果为：

101	Zhang	M	19	95.60
102	Wang	F	18	92.40

104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
110	Mao	F	18	94.90

17.2 关键字成员为字符串的结构体数组的顺序查找

一、功能

用顺序法在结构体数组中查找关键字值(字符串)在某个指定区间内的元素。

二、方法说明

本函数要求在主函数外部作如下说明：

```
#define STRU struct student
#define KEY name
STRU
{ 结构体成员表列 }
```

三、函数语句

```
int qhsch(p, k, m, a, b)
```

本函数返回关键字值在 $[a, b]$ 内的第一次发现的元素的下标。主函数中可以根据返回的这个下标继续向下查找，从而可以找出关键字值在 $[a, b]$ 内的所有元素。若在指定范围(即下标在 k 与 m 之间)内找不到关键字值在 $[a, b]$ 内的元素，则返回的函数值为-1。

四、形参说明

p——结构体类型 STRU 的一维数组。

k——整型变量。在数组 p 中进行查找的起始下标。要求 $k \geq 0$ 。

m——整型变量。在数组 p 中进行查找的终止下标。要求 m 不大于数组 p 的最大下标。

a——字符串指针变量。指示查找关键字值的下限值。

b——字符串指针变量。指示查找关键字值的上限值。

五、函数程序(文件名:qhsch.c)

六、例

在下列学生情况表中找出姓为 Zhen 的学生情况：

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

主函数程序(文件名:qhsch 0.c)如下：

```
#define STRU struct student
#define KEY name
STRU
{ int num;
  char name[8];
  char sex;
  char age;
  double score;
}

#include "stdio.h"
#include "qhsch.c"
main()
{ int i, q;
  char aa[8], bb[8], * a = aa, * b = bb;
  static STRU p[10] = { {101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9} };
a = "Zhen"; b = a;
i = 0;
printf("\n");
do
{ q = qhsch(p, i, 9, a, b);
  if (q != -1)
    printf("%-5d%-10s%-4c%-6d%-7.2f\n", p[q].num, p[q].name, p[q].sex, p[q].age, p[q].score);
  i = q + 1;
}
while (q != -1);
printf("\n");
}
```

运行结果为：

108 Zhen M 21 90.10

17.3 磁盘随机文本文件的顺序查找

一、功能

用顺序法在磁盘随机文本文件中查找关键字值在某个指定区间内的记录。

二、方法说明

本函数要求磁盘随机文本文件中的每一个记录为同一结构体类型，并要在主函数外部作如下说明：

```
# define STU 结构体类型的变量名  
# define KEY 关键字成员名  
struct 结构体名  
{ 结构体成员表列 | STU;
```

其中结构体成员均为字符数组。

三、函数语句

```
int qdsch(fp, a, b)
```

本函数返回值为 0 时，说明未查到，fp 指向文件尾；若返回的函数值为 1，则说明发现了关键字在 $[a, b]$ 内的记录，并将该记录内容存放在结构体类型变量 STU 中，fp 指向下一个记录。主函数中可以由返回的 fp 继续向下查找，直到文件尾为止，从而可以找出关键字值在 $[a, b]$ 内的所有记录。

四、形参说明

fp——文件指针变量。指向查找的起始记录；返回时指向第一次发现的关键字值在指定区间 $[a, b]$ 内的记录的下一个记录(返回的函数值为 1 时)，或指向文件结尾(当返回的函数值为 0 时)。

a——字符串指针变量。指示查找关键字值的下限值。

b——字符串指针变量。指示查找关键字值的上限值。

五、函数程序(文件名:qdsch.c)

六、例

设有一学生情况表如下：

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5

107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

将此表写入文件 stu.dat 中，然后在此文件中找出成绩在 90 分~100 分的所有学生情况。

主函数程序(文件名:qdsch 0.c)如下：

```
# define STU stu  
# define KEY score  
struct student  
{ char num[5];  
    char name[8];  
    char sex[3];  
    char age[4];  
    char score[7];  
}; STU;  
# include "stdio.h"  
# include "qdsch.c"  
main()  
{ int i;  
    char aa[7], *a = aa, bb[7], *b = bb;  
    FILE *fp;  
    static struct student p[10] =  
    {{"101", "Zhang", "M", "19", "095.6"},  
     {"102", "Wang", "F", "18", "092.4"},  
     {"103", "Zhao", "M", "19", "085.7"},  
     {"104", "Li", "M", "20", "096.3"},  
     {"105", "Gou", "M", "19", "090.2"},  
     {"106", "Lin", "M", "18", "091.5"},  
     {"107", "Ma", "F", "17", "098.7"},  
     {"108", "Zhen", "M", "21", "090.1"},  
     {"109", "Xu", "M", "19", "089.8"},  
     {"110", "Mao", "F", "18", "094.9"}};  
    a = "090.0"; b = "100.0";  
    fp = fopen("stu.dat", "w+");  
    for (i = 0; i < = 9; i++)  
        if (fwrite(&p[i], sizeof(stu), 1, fp) != 1)  
            printf("file write error \n");  
    fclose (fp);  
    fp = fopen("stu.dat", "r+");  
    printf("\n");  
    i = 1;  
    do  
    { i = qdsch(fp, a, b);  
        if (i != 0)  
            printf("% - 10s % - 10s % - 5s % - 5s % - 8s \n", stu.num, stu.name, stu.sex, stu.age,  
                  stu.score);  
    }  
    while (i != 0);
```

```
printf("\n");
fclose(fp);
```

运行结果为：

```
101  Zhang  M  19  095.6
102  Wang   F  18  092.4
104  Li     M  20  096.3
105  Gou    M  19  090.2
106  Lin    M  18  091.5
107  Ma     F  17  098.7
108  Zhen   M  21  090.1
110  Mao    F  18  094.9
```

17.4 整型有序表的对分查找

一、功能

用对分查找法在一个整型有序一维数组中查找值在指定区间内的所有元素。

二、方法说明

设有序数组 p 的长度为 n , 被查元素为 x , 则对分查找的方法如下。

将 x 与数组 p 的中间项进行比较:

若中间项的值等于 x , 则说明查到, 查找结束;

若 x 小于中间项的值, 则在数组 p 的前半部分以相同的方法进行查找;

若 x 大于中间项的值, 则在数组 p 的后半部分以相同的方法进行查找。

这个过程一直进行到查找成功或子表长度为 0(说明数组中无此元素)为止。

三、函数语句

```
int qibsh(p, n, a, b, m)
```

四、形参说明

p —— 整型一维数组, 长度为 n 。存放整型有序表。

n —— 整型变量。有序表的长度。

a —— 整型变量。待查元素值的下限。

b —— 整型变量。待查元素值的上限。

m —— 整型变量指针。在该指针指向的变量中返回值在区间 $[a, b]$ 中的元素个数。若元素个数非零, 则本函数返回值在区间 $[a, b]$ 中的第一个元素的下标; 若元素个数为零, 则说明在有序表中没有值在区间 $[a, b]$ 中的元素, 本函数返回的下标值表示 $[a, b]$ 区间中的元素在有序表中应有位置(此信息可供插入元素之用)。

五、函数程序(文件名:qibsh.c)

六、例

产生 100 到 300 之间的 100 个随机整数, 然后用希尔排序法进行排序, 最后利用对分查找法输出值在 $[170, 195]$ 内的元素。

本主函数要调用产生随机整数序列的函数 mrabs, 参看 13.4 节。

本主函数要调用整数希尔排序的函数 pishl, 参看 16.9 节。

主函数程序(文件名:qibsh 0.c)如下:

```
# include "stdio.h"
# include "mrabs.c"
# include "pishl.c"
# include "qibsh.c"
main()
{
    int i, j, m0, * m;
    int p[100], r0, * r, a, b;
    r0 = 5; r = &r0; s = &m0;
    mrabs(100, 300, r, p, 100);
    pishl(p, 100);
    printf("\n");
    for (i = 0; i <= 9; i++)
        for (j = 0; j <= 9; j++)
            printf("%d ", p[10 * i + j]);
    printf("\n");
}
printf("\n");
a = 170; b = 195;
i = qibsh(p, 100, a, b, m);
printf("m = %d\n", m0);
printf("i = %d\n", i);
for (j = i; j <= i + m0 - 1; j++)
    printf("p(%d) = %d\n", j, p[j]);
printf("\n");
```

运行结果为:

```
102  105  106  108  109  111  113  114  115  117
122  123  124  125  126  129  131  133  134  135
140  144  145  146  152  154  156  158  159  160
163  166  168  169  171  172  175  176  179  182
183  184  185  186  188  189  190  192  194  197
199  200  204  205  207  208  209  211  213  214
216  218  219  220  221  222  223  224  225  226
227  231  232  240  244  246  247  248  249  255
256  258  260  265  266  267  270  271  275  276
277  278  283  285  289  290  292  293  295  298
```

$m = 15$

• 392 •

```
i = 34  
p(34) = 171  
p(35) = 172  
p(36) = 175  
p(37) = 176  
p(38) = 179  
p(39) = 182  
p(40) = 183  
p(41) = 184  
p(42) = 185  
p(43) = 186  
p(44) = 188  
p(45) = 189  
p(46) = 190  
p(47) = 192  
p(48) = 194
```

17.5 实型有序表的对分查找

一、功能

用对分查找法在一个实型有序一维数组中查找值在指定区间内的所有元素。

二、方法说明

同 17.4 节。

三、函数语句

```
int qrbsh(p, n, a, b, m)
```

四、形参说明

p —— 双精度实型一维数组, 长度为 n。存放实型有序表。

n —— 整型变量。有序表的长度。

a —— 双精度实型变量。待查元素值的下限。

b —— 双精度实型变量。待查元素值的上限。

m —— 整型变量指针。在该指针指向的变量中返回值在区间 [a, b] 中的元素个数。若元素个数非零, 则本函数返回值在区间 [a, b] 中的第一个元素的下标; 若元素个数为零, 则说明在有序表中没有值在区间 [a, b] 中的元素, 本函数返回的下标值表示 [a, b] 区间中的元素在有序表中应有的位置(此信息可供插入元素之用)。

五、函数程序(文件名:qrbsh.c)

六、例

产生 0 到 1 之间的 100 个实型随机数, 然后用希尔排序法进行排序, 最后利用对分查找法输出值在 [0.4, 0.5] 内的元素。

本主函数要调用产生 0 到 1 之间随机数序列的函数 mrnds, 参看 13.2 节。

本主函数还要调用实数希尔排序的函数 prshl, 参看 16.10 节。

主函数程序(文件名:qrbsh 0.c)如下:

```
# include "stdio.h"  
# include "mrnds.c"  
# include "prshl.c"  
# include "qrbsh.c"  
  
main()  
{ int i, j, m0, * m;  
    double p[100], r0, * r, a, b;  
    r0 = 5.0; r = &r0; m = &m0;  
    mrnds(r, p, 100);  
    prshl(p, 100);  
    printf("\n");  
    for (i = 0; i <= 19; i++)  
        for (j = 0; j <= 4; j++)  
            printf("%10.7f ", p[5 * i + j]);  
    printf("\n");  
    printf("\n");  
    a = 0.4; b = 0.5;  
    i = qrbsh(p, 100, a, b, m);  
    printf("m = %d\n", m0);  
    printf("i = %d\n", i);  
    for (j = i; j <= i + m0 - 1; j++)  
        printf("p(%d) = %10.7f\n", j, p[j]);  
    printf("\n");  
}
```

运行结果为:

0.0063629	0.0125427	0.0150604	0.0186920	0.0450287
0.0643616	0.1090088	0.1273346	0.1303711	0.1309052
0.1399078	0.1412811	0.1610107	0.1619720	0.1687469
0.1714172	0.1720123	0.1794739	0.1891022	0.2039948
0.2053375	0.2206573	0.2208557	0.2357178	0.2377319
0.2530518	0.2592926	0.2614746	0.2622833	0.2743835
0.2749786	0.2837524	0.2859650	0.2973938	0.3084412
0.3106079	0.3259430	0.3289642	0.3456268	0.3526306
0.3623810	0.3679504	0.3722839	0.3929749	0.4274139
0.4380798	0.4410248	0.4421082	0.4607697	0.4768677
0.4861298	0.4926300	0.5102386	0.5207367	0.5209808
0.5261078	0.5390320	0.5744324	0.5748901	0.5807190
0.5860291	0.5892181	0.6135712	0.6280975	0.6292419
0.6351929	0.6488037	0.6552124	0.6711884	0.6789856
0.6921387	0.6955566	0.7031555	0.7265778	0.7312622
0.7399292	0.7424622	0.7550812	0.7607880	0.7611847
0.7663727	0.7832031	0.7849731	0.7895966	0.7982483
0.8361359	0.8439789	0.8593597	0.8631744	0.8729248
0.8754578	0.8761597	0.8893585	0.8999634	0.9234924

```
0.9595947 0.9615326 0.9671173 0.9886932 0.9985352
```

```
m = 8  
i = 44  
p(44) = 0.4274139  
p(45) = 0.4380798  
p(46) = 0.4410248  
p(47) = 0.4421082  
p(48) = 0.4607697  
p(49) = 0.4768677  
p(50) = 0.4861298  
p(51) = 0.4926300
```

17.6 字符串有序表的对分查找

一、功能

用对分查找法在一个字符串有序序列中查找值在指定区间内的所有字符串。

二、方法说明

同 17.4 节。

三、函数语句

```
int qhbsh(p, n, a, b, m)
```

四、形参说明

p ——字符串指针一维数组, 长度为 n。依次指向字符串有序序列。

n ——整型变量。字符串有序序列的长度。

a ——字符串指针。指向待查元素的下限值(字符串)。

b ——字符串指针。指向待查元素的上限值(字符串)。

m ——整型变量指针。在该指针指向的变量中返回值在区间 [a, b] 中的元素个数。若元素个数非零, 则本函数返回值在区间 [a, b] 中的第一个字符串指针在指针数组 p 中的下标; 若元素个数为零, 则说明在给定字符串有序序列中没有值在区间 [a, b] 中的字符串, 本函数返回的下标值表示值在 [a, b] 区间中的字符串指针在指针数组 p 中的应有位置(此信息可供插入元素之用)。

五、函数程序(文件名:qhbsh.c)

六、例

用希尔排序法对下列单词序列进行排序: "main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub", "china", "beijin", "liang", "juan", "yan", "teacher", "student", "qssort"。

然后在有序字符串序列中查找由小写字母 l~q 开头的所有字符串。

本主函数要调用字符串希尔排序函数 phshl, 参看 16.12 节。

主函数程序(文件名:qhbsh 0.c)如下:

```
# include "stdio.h"  
# include "phshl.c"  
# include "qhbsh.c"  
main()  
{ int i, j, m0, * m;  
char aa[8], bb[8], * a = aa, * b = bb;  
static char * p[18] = {"main", "gou", "zhao", "lin", "wang", "zhang", "li", "zhen", "ma", "sub", "chi-  
na", "beijin", "liang", "juan", "yan", "teacher", "student", "qssort"};  
m = &m0;  
phshl(p, 18, 0, 17);  
printf("\n");  
for (i = 0; i <= 1; i++)  
| for (j = 0; j <= 8; j++)  
| printf("%s, ", p[9 * i + j]);  
| printf("\n");  
|  
printf("\n");  
a = "l"; b = "r";  
i = qhbsh(p, 18, a, b, m);  
printf("m = %d\n", m0);  
printf("i = %d\n", i);  
for (j = i; j <= i + m0 - 1; j++)  
| printf("p(%d) = %s\n", j, p[j]);  
| printf("\n");  
|
```

运行结果为

```
beijin china gou juan li liang lin ma main  
qssort student sub teacher wang yan zhang zhao zhen
```

```
m = 6  
i = 4  
p(4) = li  
p(5) = liang  
p(6) = lin  
p(7) = ma  
p(8) = main  
p(9) = qssort
```

17.7 整型、实型及字符型关键字成员有序的结构体数组的对分查找

一、功能

用对分查找法在一个关于整型、或实型、或字符型关键字成员有序的结构体数组中查找

· 396 ·

关键字值在指定区间内的结构体类型元素。

二、方法说明

对分查找的方法同 17.4 节的方法说明。

根据结构体数组的特点,再作如下几点说明:

(1) 在主函数外部对结构体类型进行定义,并用常量 BISERCH 代替结构体类型定义中的

struct 结构体名

即结构体类型的定义格式为

```
#define BISERCH struct 结构体名
BISERCH
{ 结构体成员表列 }
```

(2) 在主函数外部用常量 SERCHTYPE 代替查找关键字成员的类型,用常量 KEY 代替查找关键字成员名。即在主函数外部还需作如下说明:

```
#define SERCHTYPE 查找关键字成员的类型
#define KEY 查找关键字成员名
```

(3) 在主函数中用一个结构体指针数组,其中每一个指针元素依次指向关于关键字有序的结构体类型的元素序列中的各元素。查找返回的下标是结构体指针数组元素的下标。

三、函数语句

```
int qbkey(p, n, a, b, m)
```

四、形参说明

p——结构体指针数组,长度为 n。依次指向关于关键字有序的结构体类型元素序列中的各元素。

n——整型变量。有序序列的长度,即结构体指针数组的长度。

a——关键字类型(整型、或实型、或字符型)的变量。待查元素的关键字成员的下限值。

b——关键字类型(整型、或实型、或字符型)的变量。待查元素的关键字成员的上限值。

m——整型变量指针。在该指针指向的变量中返回关键字成员值在区间 [a, b] 中的结构体类型元素个数。若元素个数非零,则本函数返回关键字成员值在区间 [a, b] 中的第一个指针元素(指向结构体类型元素)下标;若元素个数为零,则说明在给定有序序列中没有关键字成员值在区间 [a, b] 中的结构体类型元素,本函数返回的下标值表示 [a, b] 区间中的关键字成员值所对应的结构体类型元素的指针在指针数组 p 中的应有位置(此信息可供插入元素之用)。

五、函数程序(文件名:qbkey.c)

六、例

有一学生情况表如下:

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

用结构排序法按关键字成绩进行排序,然后利用对分查找法输出成绩在 95 分到 100 分之间的学生情况。

本主函数要调用结构体排序函数 prkey,参看 16.18 节。

主函数程序(文件名:qbkey 0.c)如下:

```
#define HEAPSORT struct student
#define KEY score
#define BISERCH struct student
#define SERCHTYPE double
struct student
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
}

#include "stdio.h"
#include "qbkey.c"
#include "prkey.c"
main()
{
    int i, m0, *m, j;
    BISERCH *p[10];
    static BISERCH stu[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
    m = &m0;
    for (i = 0; i <= 9; i++) p[i] = &stu[i];
    prkey(p, 10, 0, 9);
    printf("\n");
    for (i = 0; i <= 9; i++)
        printf("%-8d %s %c %d %-8.2f\n",
            p[i].num, p[i].name, p[i].sex, p[i].age, p[i].score);
```

```

(* p[i]).num, (* p[i]).name, (* p[i]).sex,
(* p[i]).age, (* p[i]).score);
printf("\n");
i = qbkey(p, 10, 95.0, 100.0, m);
printf("m = %d\n", m);
printf("i = %d\n", i);
for (j = i; j <= i + m0 - 1; j++)
    printf("p(%d) = % - 8d % - 9s % - 8c % - 8d % - 5.2f\n", j,
        (* p[j]).num, (* p[j]).name, (* p[j]).sex,
        (* p[j]).age, (* p[j]).score);
printf("\n");
}

```

运行结果为：

```

103 Zhao M 19 85.70
109 Xu M 19 89.80
108 Zhen M 21 90.10
105 Gou M 19 90.20
106 Lin M 18 91.50
102 Wang F 18 92.40
110 Mao F 18 94.90
101 Zhang M 19 95.60
104 Li M 20 96.30
107 Ma F 17 98.70

m = 3
i = 7
p(7) = 101 Zhang M 19 95.60
p(8) = 104 Li M 20 96.30
p(9) = 107 Ma F 17 98.70

```

17.8 字符串关键字成员有序的结构体数组的对分查找

一、功能

用对分查找法在一个关于字符串关键字成员有序的结构体数组中查找关键字值在指定区间内的结构体类型元素。

二、方法说明

对分查找的方法同 17.4 节的方法说明。

根据结构体数组的特点，再作如下几点说明：

(1) 在主函数外部对结构体类型进行定义，并用常量 BISERCH 代替结构体类型定义中的

struct 结构体名

即结构体类型的定义格式为

```
# define BISERCH struct 结构体名
```

BISERCH

{ 结构体成员表列 }

(2) 在主函数外部用常量 KEY 代替查找关键字成员名。即在主函数外部需作如下说明：

```
# define KEY 查找关键字成员名
```

(3) 在主函数中用一个结构体指针数组，其中每一个指针元素依次指向关于关键字有序的结构体类型的元素序列中的各元素。查找返回的下标是结构体指针数组元素的下标。

三、函数语句

```
int qhkey(p, n, a, b, m)
```

四、形参说明

p —— 结构体指针数组，长度为 n。依次指向关于关键字有序的结构体类型元素序列中的各元素。

n —— 整型变量。有序序列的长度，即结构体指针数组的长度。

a —— 字符串指针变量。指示待查元素的关键字成员的下限值。

b —— 字符串指针变量。指示待查元素的关键字成员的上限值。

m —— 整型变量指针。在该指针指向的变量中返回关键字成员值在区间 [a, b] 中的第一个指针元素（指向结构体类型元素）下标；若元素个数为零，则本函数返回关键字成员值在区间 [a, b] 中的第一个指针元素（指向结构体类型元素）下标；若元素个数为零，则说明在给定有序序列中没有关键字成员值在区间 [a, b] 中的结构体类型元素，本函数返回的下标值表示 [a, b] 区间中的关键字成员值所对应的结构体类型元素的指针在指针数组 p 中的应有位置（此信息可供插入元素之用）。

五、函数程序(文件名:qhkey.c)

六、例

有一学生情况表如下：

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

用结构排序法按关键字姓名进行排序，然后利用对分查找法输出姓名以 Z 开头的学生

情况。

本主函数要调用结构体排序函数 phkey, 参看 16.20 节。

主函数程序(文件名:qhkey 0.c)如下:

```
#define HEAPSORT struct student
#define KEY name
#define BISERCH struct student
struct student
{
    int num;
    char name[8];
    char sex;
    int age;
    double score;
}
#include "stdio.h"
#include "qhkey.c"
#include "phkey.c"
main()
{
    int i, m0, *m, j;
    BISERCH *p[10];
    static BISERCH stu[10] = {{101, "Zhang", 'M', 19, 95.6}, {102, "Wang", 'F', 18, 92.4}, {103, "Zhao", 'M', 19, 85.7}, {104, "Li", 'M', 20, 96.3}, {105, "Gou", 'M', 19, 90.2}, {106, "Lin", 'M', 18, 91.5}, {107, "Ma", 'F', 17, 98.7}, {108, "Zhen", 'M', 21, 90.1}, {109, "Xu", 'M', 19, 89.8}, {110, "Mao", 'F', 18, 94.9}};
    m = &m0;
    for (i = 0; i <= 9; i++) p[i] = &stu[i];
    phkey(p, 10, 0, 9);
    printf("\n");
    for (i = 0; i <= 9; i++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[i]).num, (*p[i]).name, (*p[i]).sex, (*p[i]).age, (*p[i]).score);
    printf("\n");
    i = qhkey(p, 10, "Z", "Zzzzzz", m);
    printf("m = %d\n", m0);
    printf("i = %d\n", i);
    for (j = i; j <= i + m0 - 1; j++)
        printf("%-8d%-9s%-8c%-8d%-5.2f\n",
               (*p[j]).num, (*p[j]).name, (*p[j]).sex, (*p[j]).age, (*p[j]).score);
    printf("\n");
}
```

运行结果为:

105	Gou	M	19	90.20
104	Li	M	20	96.30
106	Lin	M	18	91.50
107	Ma	F	17	98.70
110	Mao	F	18	94.90
102	Wang	F	18	92.40
109	Xu	M	19	89.80
101	Zhang	M	19	95.60

103	Zhao	M	19	85.70
108	Zhen	M	21	90.10

```
m = 3
i = 7
p(7) = 101 Zhang M 19 95.60
p(8) = 103 Zhao M 19 85.70
p(9) = 108 Zhen M 21 90.10
```

17.9 按关键字有序的磁盘随机文本文件的对分查找

一、功能

用对分查找法在关于关键字有序的磁盘随机文本文件中查找关键字值在某个指定区间内的记录。

二、方法说明

对分查找的方法同 17.4 节的方法说明。

本函数要求随机磁盘文本文件中的每一个记录为同一结构体类型, 并要在主函数外部作如下定义:

```
#define STU 结构体类型的变量名
#define KEY 关键字成员名
struct 结构体名
    | 结构体成员表列 | STU;
其中结构体成员均为字符数组。
```

三、函数语句

```
int qdbsh(fp, n, a, b, m)
```

本函数返回关键字值在区间 $[a, b]$ 内的第一个记录号。

四、形参说明

fp —— 文件指针变量。

n —— 整型变量。随机磁盘文本文件中记录个数。

a —— 字符串指针变量。指示查找关键字值的下限值。

b —— 字符串指针变量。指示查找关键字值的上限值。

m —— 整型变量指针。该指针指向的变量返回关键字值在 $[a, b]$ 内的记录个数。

五、函数程序(文件名:qdbsh.c)

六、例

设有一学生情况表如下:

• 402 •

学号	姓名	性别	年龄	成绩
101	Zhang	M	19	95.6
102	Wang	F	18	92.4
103	Zhao	M	19	85.7
104	Li	M	20	96.3
105	Gou	M	19	90.2
106	Lin	M	18	91.5
107	Ma	F	17	98.7
108	Zhen	M	21	90.1
109	Xu	M	19	89.8
110	Mao	F	18	94.9

将此表写入文件 student.dat 中, 然后对此文件按成绩进行排序。最后在排序后的文件中查找分数在 95.0 到 100 分之间的学生情况。

本主函数要调用磁盘文件排序函数 pdisk, 参看 16.21 节。

主函数程序(文件名: qdbsh 0.c)如下:

```
#define STU stu
#define KEY score
struct student
{
    char num[5];
    char name[8];
    char sex[3];
    char age[4];
    char score[7];
} STU;
#include "stdio.h"
#include "pdisk.c"
#include "qdbsh.c"
main()
{ FILE * fp;
long int n, k, m;
static struct student a[10] =
    {"101", "Zhang", "M", "19", "095.6",
     "102", "Wang", "F", "18", "092.4",
     "103", "Zhao", "M", "19", "085.7",
     "104", "Li", "M", "20", "096.3",
     "105", "Gou", "M", "19", "090.2",
     "106", "Lin", "M", "18", "091.5",
     "107", "Ma", "F", "17", "098.7",
     "108", "Zhen", "M", "21", "090.1",
     "109", "Xu", "M", "19", "089.8",
     "110", "Mao", "F", "18", "094.9"};
int i, j, mm0, * mm = &mm0;
char cc[7], dd[7], * c = cc, * d = dd;
fp = fopen("student.dat", "w+");
for (i = 0; i < = 9; i++)
    if (fwrite(&a[i], sizeof(stu), 1, fp) != 1)
        printf("file write error\n");
}
```

```
fclose(fp);
if ((fp = fopen("student.dat", "r+")) == NULL)
    printf("cannot open this file\n"); exit(0);
for (i = 0; i < = 9; i++)
    fread(&a[i], sizeof(stu), 1, fp);
printf("%-10s%-10s%-5s%-5s%-8s\n", a[i].num,
       a[i].name, a[i].sex, a[i].age, a[i].score);
}
printf("\n");
n = 10; k = 0; m = 9;
pdisk(fp, n, k, m);
fseek(fp, 0L, 0);
for (i = 0; i < = 9; i++)
    fread(&a[i], sizeof(stu), 1, fp);
printf("%-10s%-10s%-5s%-5s%-8s\n", a[i].num,
       a[i].name, a[i].sex, a[i].age, a[i].score);
}
printf("\n");
c = "095"; d = "100";
i = qdbsh(fp, 10, c, d, mm);
for (j = i; j < = i + mm0 - 1; j++)
    fseek(fp, j * sizeof(stu), 0);
    fread(&a[j], sizeof(stu), 1, fp);
    printf("%p(%d) = %-10s%-10s%-5s%-5s%-8s\n",
           j, a[j].num, a[j].name, a[j].sex, a[j].age,
           a[j].score);
}
printf("\n");
fclose(fp);
```

运行结果为:

101	Zhang	M	19	095.6
102	Wang	F	18	092.4
103	Zhao	M	19	085.7
104	Li	M	20	096.3
105	Gou	M	19	090.2
106	Lin	M	18	091.5
107	Ma	F	17	098.7
108	Zhen	M	21	090.1
109	Xu	M	19	089.8
110	Mao	F	18	094.9
103	Zhao	M	19	085.7
109	Xu	M	19	089.8
108	Zhen	M	21	090.1
105	Gou	M	19	090.2
106	Lin	M	18	091.5
102	Wang	F	18	092.4
110	Mao	F	18	094.9

```

101    Zhang   M   19   095.6
104    Li      M   20   096.3
107    Ma      F   17   098.7

p(7) = 101    Zhang   M   19   095.6
p(8) = 104    Li      M   20   096.3
p(9) = 107    Ma      F   17   098.7

```

17.10 磁盘顺序文本文件的字符串匹配

一、功能

用 KMP 方法在磁盘顺序文本文件中寻找给定的字符串。

二、方法说明

首先,根据给定的字符串构造一个失败链接数组 *flink*,其长度应不小于给定字符串的长度。

然后根据失败链接数组 *flink*,在磁盘文件中寻找给定字符串。

三、函数语句

```
int qfkmp(fp, p)
```

当本函数返回的函数值为零时,表示在文件中没有找到给定的字符串 *p*,*fp* 指向文件末尾;返回的函数值为非零时,表示在文件中找到了给定的字符串 *p*,*fp* 指向找到匹配字符串的下一个字符(必要时,在主函数中用 *fseek()* 函数将 *fp* 移到匹配字符串的第一个字符处)。

四、形参说明

fp——文件指针。指向寻找的起始点。

p——字符串指针。指向给定的字符串。

五、函数程序(文件名:qfkmp.c)

六、例

设文本文件(名为“abc”的内容如下:

```

eriypoiyyuprabcderyoabceiutreabcdporeuyperojhabcdopreuyurt
eriouytojrtypyrbabcdoiptpoyj

```

统计该文件中字符串“abcd”的个数。

主函数程序(文件名:qfkmp 0.c)如下:

```

#include "stdio.h"
#include "qfkmp.c"
main()
{
    FILE * fp;
    int i, j, jt;
}

```

```

char pp[8], * p = pp;
printf("\n");
p = "abcd";
if ((fp = fopen("abc", "r")) == NULL)
    printf("cannot open this file\n");
    exit(0);
jt = 1; j = 0;
while (jt == 1)
    i = qfkmp(fp, p);
    if (i == 0) j = j + i;
    else jt = 0;
}
printf("sum = %d\n", j);
printf("\n");
fclose(fp);

```

运行结果为

```
sum = 4
```

第 18 章 图形模式下读写屏幕象点

18.1 设置显示模式

一、功能

设置各种显示卡的显示模式。

二、方法说明

显示模式按功能可以分为字符模式和图形模式两大类。

字符模式也称字母数字模式,即 A/N 模式(Alpha Number mode)。在这种模式下,显示缓冲区中存放的是显示字符的代码和属性,显示屏幕被划分为若干字符显示行和列。

图形模式(Graphics modes)也称 APA 模式(All Points Addressable mode)。在这种模式下,显示缓冲区中存放的是监视器屏幕上的每个象素点的颜色或灰度值,显示屏幕按分辨率被划分成象素行和列。

显示模式分为标准模式与非标准模式两种。标准模式在不同类型显示卡与不同厂家的显示卡之间是通用的,通常,显示模式号小于 14H 的是标准模式;非标准模式是专用的,一般不能通用。

三、函数语句

```
void rmode(m)
```

四、形参说明

m——整型变量。指定的显示模式号。

五、函数程序(文件名:rmode.c)

18.2 CGA 图形模式(04H, 320 × 200, 4 色)

一、功能

在 CGA 图形模式 04H 下读写屏幕象点。

二、方法说明

在 CGA 图形模式 04H 下,每个象点占一个字节的两位(可选 4 种颜色),因此,显示内存中的一个字节可存放 4 个象点信息。在这种模式下,屏幕上每行有 320 个象点,需占 80 个字节,整个屏幕有 200 行。显示内存又分为两个区,每个区的长度为 8K 字节。显示内存起始地址为 B8000H。

屏幕上一个象点(*row*, *col*)所对应的相对于显示内存起始地址的偏移量为

$$ix = (\text{row}/2) * 80 + (\text{row} \% 2) * 8192 + (\text{col}/4)$$

在字节中所占的位号为 $2(3 - \text{col} \% 4)$ 与 $2(3 - \text{col} \% 4) + 1$ 。

三、函数语句

```
int rcg04(row, col, color, xor)
```

本函数返回指定象点(*row*, *col*)处的颜色值。

四、形参说明

row, *col*——均为整型变量。指定象点在屏幕上的行、列号。

color——整型变量。指定的颜色值,可取 4 种颜色号 0、1、2、3 或一个负整数。当 *color* ≥ 0 时,表示在象点(*row*, *col*)处用颜色值 *color* 显示,该值也作为函数值返回;当 *color* < 0 时,表示读象点(*row*, *col*)处的颜色值,并作为函数值返回。

xor——整型变量。异或标志。当 *xor* = 0 时,表示直接写象点;当 *xor* $\neq 0$ 时,表示以异或方式写象点。当 *color* < 0 时,本参数可任意。

五、函数程序(文件名:rcg 04.c)

六、例

首先用各种颜色在屏幕上画出彩条。

然后逐点读出左上角各象点颜色,并用该读出的颜色以异或方式写到右下角,以异或方式写到左下角,以直接方式写到右上角。

主函数程序(文件名:rcg 040.c)如下:

```
# define POINT rcg04
# include "stdio.h"
# include "rmode.c"
# include "rcg04.c"
main()
{
    int k, row, col, color;
    rmode(0x04);
    for (row = 0; row <= 500; row++)
        for (col = 0; col <= 500; col++)
            POINT(row, col, color, 0);
            color = color + 1;
    }
    for (row = 0; row <= 100; row++)
        for (col = 0; col <= 160; col++)
            {
                k = POINT(row, col, -1, 0);
                POINT(row + 101, col + 161, k, 1);
                POINT(row + 101, col, k, 1);
                POINT(row, col + 161, k, 0);
            }
```

18.3 EGA 图形模式(10H, 640×350, 16 色)

一、功能

在 EGA 图形模式 10H 下读写屏幕象点。

二、方法说明

1. EGA 在图形方式下的内存组织

EGA 图形模式 10H 具有 16 种颜色, 因此, 对于屏幕上的每个象素点由 4 个位组成, 而这 4 个位中的每一位分别来自 4 个位平面中的相同地址字节中的相同位。4 个位平面中的每个位平面表示一种基本颜色, 分别为蓝、绿、红、灰, 由这 4 种基本颜色组合成一个象点的 16 种颜色。4 个位平面具有相同的地址空间, 其起始地址为 A0000H。

EGA 图形模式 10H 共有 350 个象素行, 每行有 640 个象素点。因此, 每行需占 80 个字节, 每个位平面占 28000 个字节。屏幕上的一个象素点(row, col)所对应的相对于显示内存起始地址(A0000H)的偏移量为

$$ix = row * 80 + (col / 8)$$

在字节中所占的位号为 $7 - (col \% 8)$ 。

2. 与读写象点有关的 EGA 寄存器

与读写象点有关的 EGA 寄存器有以下几个。

(1) EGA 时序控制寄存器(TS)

它有一个地址索引寄存器(I/O 端口值为 03C4H)和五个被索引寄存器(I/O 端口值为 03C5H), 如表 18.1 所示。

表 18.1 EGA 时序控制寄存器

名 称	端口值	索引号	读写方式
地址索引寄存器	03C4H	—	只写
复位寄存器	03C5H	00	只写
时钟方式寄存器	03C5H	01	只写
存储位平面屏蔽寄存器	03C5H	02	只写
字符映象选择寄存器	03C5H	03	只写
存储方式寄存器	03C5H	04	只写

下面只介绍存储位平面屏蔽寄存器。

这是一个只写寄存器, I/O 端口值为 03C5H。其中位 0~3 分别对应于位平面 0~3 的写允许或不允许。即:

位 0: 置 1 时表示允许写位平面 0, 置 0 时表示不允许写位平面 0;

位 1: 置 1 时表示允许写位平面 1, 置 0 时表示不允许写位平面 1;

位 2: 置 1 时表示允许写位平面 2, 置 0 时表示不允许写位平面 2;

位 3: 置 1 时表示允许写位平面 3, 置 0 时表示不允许写位平面 3。

位 4~7: 保留。

但必须注意, 在写此寄存器之前, 先要向地址索引寄存器(03C4H)写值 2。

(2) 图形控制寄存器(GDC)

它有一个数据索引寄存器(I/O 端口值为 03CEH)和 9 个被索引寄存器(I/O 端口值为 03CFH), 如表 18.2 所示。

表 18.2 EGA 图形控制寄存器

名 称	端口值	索引号	读写方式
数据索引寄存器	03CEH	—	只写
置位/复位寄存器	03CFH	00	只写
置位/复位允许寄存器	03CFH	01	只写
颜色比较寄存器	03CFH	02	只写
数据循环移位寄存器	03CFH	03	只写
读映象选择寄存器	03CFH	04	只写
方式寄存器	03CFH	05	只写
杂用寄存器	03CFH	06	只写
颜色比较无关寄存器	03CFH	07	只写
位屏蔽寄存器	03CFH	08	只写

① 置位/复位寄存器(端口值 03CFH, 索引号 0)

位 0~3 分别对应于位平面 0~3。

当某位置 1 时, 在写操作期间向所屏蔽的位写 1; 当某位置 0 时, 在写操作期间向所屏蔽的位写 0。

该寄存器用于向显示内存写某个绝对的颜色值而不受逻辑运算功能影响。

② 置位/复位允许寄存器(端口值 03CFH, 索引号 1)

位 0~3 分别对应位平面 0~3。

当某位置 1 时, 使对应的位平面置位/复位功能允许; 置 0 时, 使对应位平面的置位/复位功能不允许。

③ 颜色比较寄存器(端口值 03CFH, 索引号 2)

位 0~3 分别对应位平面 0~3, 表示要比较的颜色值。

当某位平面被允许比较时(参看颜色比较无关寄存器), 在采用读方式 1 的情况下, 如果某位平面中的值与本寄存器设定的值相一致时, 则读出值为 1。

④ 数据循环移位寄存器(端口值 03CFH, 索引号 3)

位 0~2 表示循环移位计数值 n(0~7), 由 CPU 向显示内存写入的数据需向右循环移位 n 位后再写入。

位 3~4 表示由 CPU 向显示内存写入的数据需与位平面锁存器中的数据作相应的逻辑

操作后再写入。位 4,3 的定义如下：

- 00 数据不变
- 10 “或”操作
- 01 “与”操作
- 11 “异或”操作

⑤ 读映象选择寄存器(端口值 03CFH, 索引号 4)

位 0~1 表示在读方式 0 下从哪个位平面读取数据(0~3)。

⑥ 方式寄存器(端口值 03CFH, 索引号 5)

位 0~1 组合成写方式 0~2。

位 2 保留, 必须为 1。

位 3 表示读方式 0 或 1。

BIOS 的缺省值读、写方式均为 0。

位 4 为选择奇/偶方式。

位 5 为选择移位寄存器方式。

位 4,5 主要用于模拟 CGA 图形方式, 在 EGA 图形模式, 10H 中不用。

⑦ 杂用寄存器(端口值 03CFH, 索引号 6)

位 0 置 0 时, 选择字符方式寻址; 置 1 时选择图形方式寻址。

位 1 与方式寄存器中的位 4 配合使用, 主要用于 CGA 图形模式。

位 2~3 表示显示内存的地址空间。位 3 位 2 的意义如下:

00 表示地址空间 A0000H~BFFFFH, 共 128K;

01 表示地址空间 A0000H~AFFFFH, 共 64K;

10 表示地址空间 B0000H~B8FFFH, 共 32K;

11 表示地址空间 B8000H~BFFFFH, 共 32K。

⑧ 颜色比较无关寄存器(端口值 03CFH, 索引号 7)

位 0~3 分别对应位平面 0~3。

当某位置 0 时, 表示该位平面内容被允许与颜色比较寄存器值相比较; 置 1 时表示该位平面内容不参加颜色比较, 也未读取。

当该寄存器设置为 0FH 时, 仅与颜色比较寄存器中的颜色匹配; 当该寄存器设置为 00H 时, 可匹配任何颜色。

⑨ 位屏蔽寄存器(端口值 03CFH, 索引号 8)

位 0~7 表示位平面中数据字节的位 0~7。

当某位置 1 时, 表示该位未屏蔽; 当某位置 0 时, 表示该位被屏蔽。

对于未屏蔽的位, 在写入时用当前的 CPU 数据的对应位写入。

该寄存器不影响写方式 1。

3. EGA 在图形模式下的读写方式

EGA 的图形模式下有三种写方式与两种读方式。

(1) 写方式 0

在位平面未屏蔽, 且位平面设置成不允许“置位/复位”的情况下, 将 CPU 送出的数据按循环移位寄存器的设置值移位或进行逻辑运算后写进各存储位平面中。如果某个位平面被

允许“置位/复位”, 则该位平面地址的 8 位值用该位平面对应的“置位/复位”寄存器的位值去写。

因此, 在写方式 0 下, 要涉及到置位/复位寄存器、置位/复位允许寄存器、数据循环移位寄存器及位屏蔽寄存器的设置。

(2) 写方式 1

用图形控制器锁存的内容写每个位平面, 而锁存中的内容由前次 CPU 通过读操作装入。

在这种方式下, 对于图形的复制特别方便。

(3) 写方式 2

CPU 传来的数据作为颜色数据(低 4 位有效), 被写入允许写的象素位中, 而存储位平面屏蔽寄存器为当前要写得颜色值。

在上述三种写方式中, 均需要将当前要写入位置的数据先读入锁存器中保留。即在每次写以前应先执行一次 CPU 的读操作。

(4) 读方式 0

这种读方式是按单个位平面进行的。

首先设置“读映象选择寄存器”的值, 使之挑选要读取的存储位平面, 然后读取所挑选的位平面中相应显示内存地址处的值。

要注意的是, 读方式 0 下, 每次只能读取一个位平面中的一个字节内容。当需要读取 4 个位平面的合成颜色时, 必须对 4 个位平面逐一读取。当要读取某个象素点时, 还需从中挑选出相应的象素位的颜色。

(5) 读方式 1

这种读方式是通过将屏幕上的象素点颜色值与所设置的“颜色比较寄存器”的值进行比较的方法实现读操作。当象素点的颜色与所设置的“颜色比较寄存器”中的颜色值相匹配时, 则读取的低 4 位被置 1。

这种读方式, 对于扫描某种特定颜色或者需要在屏幕上置某一种颜色的情况下是非常有效的。而且当需要确定某一块的特定颜色值时, 还可通过设置“颜色比较无关寄存器”来忽略掉某些或所有的位平面。

读方式 1 对于需要读取每一象素点颜色值的情况下, 最多需要重复 16 次(每次只能判别一种颜色), 因此不是很有效。而用读方式 0, 则只需要读 4 次(每次读一个位平面)。

本函数采用读方式 0 读取指定象点处的颜色值, 即分别读取 4 个存储位平面上的对应位的颜色值, 最后组合成指定点处的当前颜色值。写象点时采用写方式 2, 首先设置“位屏蔽寄存器”的值, 使指定象点所对应的位置 1, 其它各位均置 0, 然后将 FFH 写入“存储位平面屏蔽寄存器”, 最后执行一次读对应象点字节地址操作后, 将颜色值写入对应象点的字节地址。在完成一个象点的读写后, 恢复缺省值写方式 0, 并置“位屏蔽寄存器”的各位为 1(即所有的位均为未屏蔽状态)。

三、函数语句

```
int reg10(row, col, color, xor)
```

本函数返回指定象点(*row*, *col*)处的颜色值。

四、形参说明

row、col——均为整型变量。指定象点在屏幕上的行、列号。

color——整型变量。指定的颜色值，可取 16 种颜色号 0~15 或一个负整数。当 color ≥ 0 时，表示在点 (row, col) 处用颜色值 color 显示，该值也作为函数值返回；当 color < 0 时，表示读象点 (row, col) 处的颜色值，并作为函数值返回。

xor——整型变量。异或标志。当 xor = 0 时，表示直接写象点；当 xor ≠ 0 时，表示以异或方式写象点。当 color < 0 时，本参数可任意。

五、函数程序(文件名:reg 10.c)

六、例

同 18.2 节的例。

主函数程序(文件名:reg 100.c)如下：

```
#define POINT reg10
#include "stdio.h"
#include "rmode.c"
#include "reg10.c"
main()
{ int k, row, col, color;
  rmode(0x10);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
    { POINT(row, col, color, 0);
      color=color+1;
    }
  }
  for (row=0; row<=250; row++)
  for (col=0; col<=250; col++)
  { k=POINT (row, col, -1, 0);
    POINT (row+251, col+251, k, 1);
    POINT (row+251, col, k, 1);
    POINT (row, col+251, k, 0);
  }
}
```

18.4 VGA 图形模式(12H, 640×480, 16 色)

一、功能

在 VGA 图形模式 12H 下读写屏幕象点。

二、方法说明

VGA 图形模式 12H 共有 480 个象素行，每行有 640 个象素点。为了表示 16 种颜色，显

示内存被组织成 4 个可独立寻址的存储位平面，它们具有相同的地址空间，其起始地址为 A0000H。在这种模式下，每个象素行占 80 个字节，每个位平面占 38400 个字节。屏幕上的一个象素点 (row, col) 所对应的相对于显示内存起始地址的偏移量为

$$ix = row * 80 + (col / 8)$$

在对应字节中所占的位号为 $7 - (col \% 8)$ 。

有关存储位平面的概念以及读写方式、与读写象点有关的寄存器同 EGA 图形模式 10H，参看 18.3 节的方法说明。

三、函数语句

```
int rvgl2(row, col, color, xor)
```

本函数返回指定象点 (row, col) 处的颜色值。

四、形参说明

row, col——均为整型变量。指定象点在屏幕上的行、列号。

color——整型变量。指定的颜色值，可取 16 种颜色号 0~15 或一个负整数。当 color ≥ 0 时，表示在象点 (row, col) 处用颜色值 color 显示，该值也作为函数值返回；当 color < 0 时，表示读象点 (row, col) 处的颜色值，并作为函数值返回。

xor——整型变量。异或标志。当 xor = 0 时，表示直接写象点；当 xor ≠ 0 时，表示以异或方式写象点。当 color < 0 时，本参数可任意。

五、函数程序(文件名:rvgl2.c)

六、例

同 18.2 节的例。

主函数程序(文件名:rvgl20.c)如下：

```
#define POINT rvgl2
#include "stdio.h"
#include "rmode.c"
#include "rvgl2.c"
main()
{ int k, row, col, color;
  rmode(0x12);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
    { POINT(row, col, color, 0);
      color=color+1;
    }
  }
  for (row=0; row<=250; row++)
  for (col=0; col<=250; col++)
  { k=POINT (row, col, -1, 0);
    POINT (row+251, col+251, k, 1);
    POINT (row+251, col, k, 1);
    POINT (row, col+251, k, 0);
  }
}
```

```

POINT(row+251,col+251,k,1);
POINT(row+251,col,k,1);
POINT(row,col+251,k,0);
}

```

18.5 VGA 图形模式(13H, 320×200, 256 色)

一、功能

在 VGA 图形模式 13H 下读写屏幕象点。

二、方法说明

VGA 图形模式 13H 支持 256 种颜色, 屏幕上的一个象素点占显示内存的一个字节。

在本模式下, 共有 200 个象素行, 每行有 320 个象素点, 因此, 每个象素行占 320 个字节, 整个显示屏幕需占 64000 个字节。显示内存的起始地址为 A0000H, 并组织成线性内存数组的形式。

屏幕上一个象素点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$ix = row * 320 + col$$

三、函数语句

```
int rvg13(row, col, color, xor)
```

本函数返回在指定象点(row, col)处的颜色值。

四、形参说明

row, col ——均为整型变量。指定象点在屏幕上的行、列号。

$color$ ——整型变量。指定的颜色值, 可取 256 种颜色号 0~255 或一个负整数。当 $color \geq 0$ 时, 表示在象点(row, col)处用颜色值 $color$ 显示, 该值也作为函数值返回; 当 $color < 0$ 时, 表示读象点(row, col)处的颜色值, 并作为函数值返回。

xor ——整型变量。异或标志。当 $xor = 0$ 时, 表示直接写象点; 当 $xor \neq 0$ 时, 表示以异或方式写象点。当 $color < 0$ 时, 本参数可任意。

五、函数程序(文件名:rvg13.c)

六、例

同 18.2 节的例。

主函数程序(文件名:rvg130.c)如下:

```

#define POINT rvg13
#include "stdio.h"
#include "rmode.c"
#include "rvg13.c"

```

```

main()
{
    int k, row, col, color;
    rmode(0x13);
    for (row = 0; row <= 500; row++)
    {
        color = 0;
        for (col = 0; col <= 500; col++)
        {
            POINT(row, col, color, 0);
            color = color + 1;
        }
    }
    for (row = 0; row <= 100; row++)
    {
        for (col = 0; col <= 160; col++)
        {
            k = POINT(row, col, -1, 0);
            POINT(row+101, col+161, k, 1);
            POINT(row+101, col, k, 1);
            POINT(row, col+161, k, 0);
        }
    }
}

```

18.6 TVGA 图形模式(5BH, 800×600, 16 色)

一、功能

在 TVGA 图形模式 5BH 下读写屏幕象点。

二、方法说明

本模式支持 16 种颜色, 显示内存被组织成 4 个可独立寻址的存储位平面, 它们具有相同的地址空间, 其起始地址为 A0000H。有关存储位平面的概念、与读写象点有关的寄存器以及读写象点的方式同 EGA 图形模式 10H, 参看 18.3 节的方法说明。

本模式共有 600 个象素行, 每个象素行有 800 个象素点。每行需占 4 个位平面中地址相同的各 100 个字节, 整个屏幕需占地址相同的 4 个位平面中各 60000 个字节。

屏幕上象点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$ix = row * 100 + (col / 8)$$

在对应字节中的位号为 $7 - (col \% 8)$ 。

三、函数语句

```
int rtv5b(row, col, color, xor)
```

本函数返回象点(row, col)处的颜色值。

四、形参说明

row, col ——均为整型变量。指定象点在屏幕上的行、列号。

$color$ ——整型变量。指定的颜色值, 可取 16 种颜色号 0~15 或一个负整数。当 $color \geq 0$ 时, 表示在象点(row, col)处用颜色值 $color$ 显示, 该值也作为函数值返回; 当 $color < 0$

时,表示读象点(row, col)处的颜色值,并作为函数值返回。

xor ——整型变量。异或标志。当 $xor = 0$ 时,表示直接写象点;当 $xor \neq 0$ 时,表示以异或方式写象点。当 $color < 0$ 时,本参数可任意。

五、函数程序(文件名:rtv 5b.c)

六、例

同 18.2 节的例。

主函数程序(文件名:rtv 5b0.c)如下:

```
# define POINT rtv5b
# include "stdio.h"
# include "rmode.c"
# include "rtv5b.c"
main()
{ int k, row, col, color;
  rmode(0x5b);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
      { POINT(row, col, color, 0);
        color=color+1;
      }
    for (row=0; row<=250; row++)
      for (col=0; col<=250; col++)
        { k=POINT(row, col, -1, 0);
          POINT(row+251, col+251, k, 1);
          POINT(row+251, col, k, 1);
          POINT(row, col+251, k, 0);
        }
  }
}
```

18.7 TVGA 图形模式(5DH, 640×480, 256 色)

一、功能

在 TVGA 图形模式 5DH 下读写屏幕象点。

二、方法说明

本模式支持 256 种颜色,屏幕上一个象素点占显示内存的一个字节。

在本模式下,共有 480 个象素行,每个象素行有 640 个象素点,因此,每个象素行占 640 个字节,整个显示屏幕需占 307200 个字节。显示内存的起始地址为 A0000H,并组织成线性内存数组的形式。但由于整个显示内存超过了 64K,因此,在寻址时需要作跨段处理。

屏幕上一个象素点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$u = row * 640 + col$$

段地址增量为

$$s = u / 65536$$

在 64K 内的偏移量为

$$ix = u \% 65536$$

其中段地址增量 s 的二进制第 1 位取反后写入寄存器 03C5.0E 中。

三、函数语句

```
int rtv5d(row, col, color, xor)
```

本函数返回象点(row, col)处的颜色值。

四、形参说明

row, col ——均为整型变量。指定象点在屏幕上的行、列号。

$color$ ——整型变量。指定的颜色值,可取 256 种颜色号 0~255 或一个负数。当 $color \geq 0$ 时,表示在象点(row, col)处用颜色值 $color$ 显示,该值也作为函数值返回;当 $color < 0$ 时,表示读象点(row, col)处的颜色值,并作为函数值返回。

xor ——整型变量。异或标志。当 $xor = 0$ 时,表示直接写象点;当 $xor \neq 0$ 时,表示以异或方式写象点。当 $color < 0$ 时,本参数可任意。

五、函数程序(文件名:rtv 5d.c)

六、例

同 18.2 节的例。

主函数程序(文件名:rtv 5d0.c)如下:

```
# define POINT rtv5d
# include "stdio.h"
# include "rmode.c"
# include "rtv5d.c"
main()
{ int k, row, col, color;
  rmode(0x5d);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
      { POINT(row, col, color, 0);
        color=color+1;
      }
    for (row=0; row<=250; row++)
      for (col=0; col<=250; col++)
        { k=POINT(row, col, -1, 0);
          POINT(row+251, col+251, k, 1);
          POINT(row+251, col, k, 1);
          POINT(row, col+251, k, 0);
        }
  }
}
```

```

POINT(row,col+251,k,0);
}

```

18.8 TVGA 图形模式(5EH, 800×600, 256 色)

一、功能

在 TVGA 图形模式 5EH 下读写屏幕象点。

二、方法说明

本模式支持 256 种颜色, 屏幕上的一个象素点占显示内存的一个字节。

在本模式下, 共有 600 个象素行, 每行有 800 个象素点, 因此, 每个象素行占 800 个字节, 整个显示屏幕占 480000 个字节。显示内存的起始地址为 A0000H, 并组织成线性内存数组的形式。由于整个显示内存超过了 64K, 因此, 在寻址时需要作跨段处理。

屏幕上一个象点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$u = row * 800 + col$$

段地址增量为

$$s = u / 65536$$

在 64K 内的偏移量为

$$ix = u \% 65536$$

其中段地址增量 s 的二进制第 1 位取反后写入寄存器 03C5.0E 中。

三、函数语句

```
int rtv5e(row, col, color, xor)
```

本函数返回象点(row, col)处的颜色值。

四、形参说明

row, col ——均为整型变量。指定象点在屏幕上的行、列号。

$color$ ——整型变量。指定的颜色值, 可取 256 种颜色号 0~255 或一个负整数, 当 $color \geq 0$ 时, 表示在象点(row, col)处用颜色值 $color$ 显示, 该值也作为函数值返回; 当 $color < 0$ 时, 表示读象点(row, col)处的颜色值, 并作为函数值返回。

xor ——整型变量。异或标志。当 $xor = 0$ 时, 表示直接写象点; 当 $xor \neq 0$ 时, 表示以异或方式写象点。当 $color < 0$ 时, 本参数可任意。

五、函数程序(文件名:rtv 5e.c)

六、例

同 18.2 节的例。

主函数程序(文件名:rtv 5e0.c)如下:

```

#define POINT rtv5e
#include "stdio.h"
#include "rmode.h"
#include "rtv5e.c"
main()
{ int k, row, col, color;
rmode(0x5e);
for (row=0; row<=500; row++)
{ color=0;
for (col=0; col<=500; col++)
| POINT(row,col,color,0);
color=color+1;
|
for (row=0; row<=250; row++)
for (col=0; col<=250; col++)
| k=POINT(row,col,-1,0);
POINT(row+251,col+251,k,1);
POINT(row+251,col,k,1);
POINT(row,col+251,k,0);
|
}
}

```

18.9 TVGA 图形模式(5FH, 1024×768, 16 色)

一、功能

在 TVGA 图形模式 5FH 下读写屏幕象点。

二、方法说明

本模式支持 16 种颜色, 显示内存被组织成 4 个可独立寻址的存储位平面, 它们具有相同的地址空间, 其起始地址为 A0000H。有关存储位平面的概念、与读写象点有关的寄存器以及读写象点的方式同 EGA 图形模式 10H, 参看 18.3 节的方法说明。

本模式共有 768 个象素行, 每个象素行有 1024 个象素点。每行需占 4 个位平面中地址相同的各 128 个字节, 整个屏幕需占地址相同的 4 个位平面中各 98304 个字节。由于显示内存超过了 64K, 因此, 在寻址时需作跨段处理。

屏幕上一个象点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$u = row * 128 + (col / 8)$$

在对应字节中的位号为 $7 - (col \% 8)$ 。段地址增量为

$$s = u / 65536$$

在 64K 内的偏移量为

$$ix = u \% 65536$$

其中段地址增量 s 的二进制第 1 位取反后写入寄存器 03C5.0E 中。

三、函数语句

```
int rtv5f(row, col, color, xor)
本函数返回象点(row, col)处的颜色值。
```

四、形参说明

row、col——均为整型变量。指定象点在屏幕上的行、列号。

color——整型变量。指定的颜色值,可取16种颜色号0~15或一个负整数。当color ≥ 0 时,表示在象点(row, col)处用颜色值color显示,该值也作为函数值返回;当color<0时,表示读象点(row, col)处的颜色值,并作为函数值返回。

xor——整型变量。异或标志。当xor=0时,表示直接写象点;当xor $\neq 0$ 时,表示以异或方式写象点。当color<0时,本参数可任意。

五、函数程序(文件名:rtv 5f.c)

六、例

同18.2节的例。

主函数程序(文件名:rtv 5f0.c)如下:

```
# define POINT rtv5f
# include "stdio.h"
# include "rmode.c"
# include "rtv5f.c"
main()
{ int k, row, col, color;
  rmode(0x5f);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
    { POINT(row, col, color, 0);
      color=color+1;
    }
  }
  for (row=0; row<=250; row++)
  for (col=0; col<=250; col++)
  { k=POINT(row, col, -1, 0);
    POINT(row+251, col+251, k, 1);
    POINT(row+251, col, k, 1);
    POINT(row, col+251, k, 0);
  }
}
```

18.10 TVGA 图形模式(62H, 1024×768, 256 色)

一、功能

在TVGA图形模式62H下读写屏幕象点。

二、方法说明

本模式支持256种颜色,屏幕上的一象素点占显示内存的一个字节。

在本模式下,共有768个象素行,每行有1024个象素点,因此,每个象素行占1024个字节,整个显示屏幕需占786432个字节。显示内存的起始地址为A0000H,并组织成线性内存数组的形式。由于显示内存已超过64K,因此,在寻址时需要作跨段处理。

屏幕上一个象素点(row, col)所对应的相对于显示内存起始地址的偏移量为

$$u = row * 1024 + col$$

段地址增量为

$$s = u / 65536$$

在64K内的偏移量为

$$ix = u \% 65536$$

其中段地址增量s的二进制第1位取反后写入寄存器03C5.0E中。

三、函数语句

```
int rtv62(row, col, color, xor)
```

本函数返回象点(row, col)处的颜色值。

四、形参说明

row、col——均为整型变量。指定象点在屏幕上的行、列号。

color——整型变量。指定的颜色值,可取256种颜色号0~255或一个负整数。当color ≥ 0 时,表示在象点(row, col)处用颜色值color显示,该值也作为函数值返回;当color<0时,表示读象点(row, col)处的颜色值,并作为函数值返回。

xor——整型变量。异或标志。当xor=0时,表示直接写象点;当xor $\neq 0$ 时,表示以异或方式写象点。当color<0时,本参数可任意。

五、函数程序(文件名:rtv 62.c)

六、例

同18.2节的例

主函数程序(文件名:rtv 620.c)如下:

```
# define POINT rtv62
# include "stdio.h"
# include "rmode.c"
# include "rtv62.c"
main()
{ int k, row, col, color;
  rmode(0x62);
  for (row=0; row<=500; row++)
  { color=0;
    for (col=0; col<=500; col++)
  }
```

```

    | POINT(row,col,color,0);
    color=color+1;
    |
for (row=0; row<=250; row++)
for (col=0; col<=250; col++)
    k=POINT(row,col,-1,0);
    POINT(row+251,col+251,k,1);
    POINT(row+251,col,k,1);
    POINT(row,col+251,k,0);
    |

```

第 19 章 基本图形操作

19.1 直 线

一、功能

根据给定的起点(row_1, col_1)和终点(row_2, col_2)在屏幕上画一条指定颜色 $color$ 的直线。

二、方法说明

设

$$dx = |row_2 - row_1|, dy = |col_2 - col_1|$$

$$dm = \begin{cases} 1, & row_2 > row_1 \\ -1, & row_2 < row_1 \end{cases}$$

$$dn = \begin{cases} 1, & col_2 > col_1 \\ -1, & col_2 < col_1 \end{cases}$$

$$m = \max\{dx, dy\}, xl = yl = 0$$

则直线插补的过程如下：

从起点(row_1, col_1)开始, 即令 $(x, y) = (row_1, col_1)$, 直至终点(row_2, col_2)为止, 作如下操作:

$$xl = xl + dx, \text{ 若 } xl \geq m, \text{ 则 } xl = xl - m, x = x + dm,$$

$$yl = yl + dy, \text{ 若 } yl \geq m, \text{ 则 } yl = yl - m, y = y + dn,$$

然后显示象点 (x, y) 。

三、函数语句

void sline(row1, col1, row2, col2, color, xor)

本函数需要调用屏幕象点的读写函数 POINT。在调用本函数之前, 应对 POINT 用第 18 章中的某个屏幕象点读写函数进行宏定义。

四、形参说明

row_1, col_1 ——均为整型变量。直线起点在屏幕上的行、列号。

row_2, col_2 ——均为整型变量。直线终点在屏幕上的行、列号。

$color$ ——整型变量。颜色值。

xor ——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式画直线; 当 $xor \neq 0$ 时, 表示以异或方式画直线。

五、函数程序(文件名: sline.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下,画一个行宽为 200、列宽为 320 的矩形以及两条对角线。颜色值为 4,采用异或方式。

主函数程序(文件名: sline 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
#include "sline.c"
main()
{
    rmode(0x62);
    sline(0, 0, 0, 319, 4, 1);
    sline(0, 319, 199, 319, 4, 1);
    sline(199, 319, 199, 0, 4, 1);
    sline(199, 0, 0, 0, 4, 1);
    sline(199, 319, 0, 0, 4, 1);
    sline(0, 319, 199, 0, 4, 1);
}
```

19.2 线段构成的图形

一、功能

根据给定 n 个线段的起终点信息,在屏幕上同时画出这 n 个线段。

二、方法说明

n 个线段的起终点信息存放在一个 n 行 4 列的数组 g 中,其中: ($i = 0, 1, \dots, n - 1$)
 $g(i, 0)$ 与 $g(i, 1)$ 为第 $i + 1$ 个线段起点相对于参考点的行、列号;
 $g(i, 2)$ 与 $g(i, 3)$ 为第 $i + 1$ 个线段终点相对于参考点的行、列号。

三、函数语句

```
void sline(row, col, g, n, color, xor)
```

本函数要调用画直线的函数 sline,参看 19.1 节。在函数 sline 中又要调用读写象点函数 POINT,在调用前可用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。参考点在屏幕上的行列号。

g ——整型二维数组,体积为 $n \times 4$ 。其中 $g(i, 0)$ 与 $g(i, 1)$ 存放第 $i + 1$ 个线段起点相对于参考点(row, col)的行、列号; $g(i, 2)$ 与 $g(i, 3)$ 存放第 $i + 1$ 个线段终点相对于参考点(row, col)的行、列号($i = 0, 1, \dots, n - 1$)。

n ——整型变量。线段个数。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时,表示以直接方式画线段;当 $xor \neq 0$ 时,表示以异或方式画线段。

五、函数程序(文件名: smull.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

首先将边长为 10、颜色值为 4 的正方形边框从屏幕的左上角逐步移到 300 行、300 列的位置处;

然后用颜色值 4 画行宽为 350、列宽为 640 的长方形边框以及两条对角线。

在主函数程序中,对于 j 与 k 的双重循环是为了等待,以便能看清小正方形的移动过程。

主函数程序(文件名: smull 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "smull.c"
#include "rmode.c"
main()
{
    int i, j, k;
    static int g[4][4] = { {0, 0, 0, 10}, {0, 10, 10, 10},
                          {10, 10, 10, 0}, {10, 0, 0, 0} };
    rmode(0x62);
    smull(0, 0, g, 4, 1);
    for (i = 0; i <= 300; i = i + 10)
        {
            smull(i, i, g, 4, 1);
            for (j = 0; j <= 30; j++)
                for (k = 0; k <= 10000; k++)
                    smull(i + 10, i + 10, g, 4, 1);
            for (j = 0; j <= 30; j++)
                for (k = 0; k <= 10000; k++)
                    {
                        sline(0, 0, 0, 639, 4, 0);
                        sline(0, 639, 349, 639, 4, 0);
                        sline(349, 639, 349, 0, 4, 0);
                        sline(349, 0, 0, 0, 4, 0);
                        sline(0, 0, 349, 639, 4, 0);
                        sline(0, 639, 349, 0, 4, 0);
                    }
        }
}
```

19.3 虚 线

一、功能

根据给定的起点($row1, col1$)和终点($row2, col2$),实线长 rl 和虚部长 il ,用颜色值画

虚线。

二、方法说明

基本方法同直线插补(参看 19.1 节)。

实线部分用指定颜色的象点显示,而虚的部分(即空白部分)不显示象点。

三、函数语句

```
void simln(row1, col1, row2, col2, rl, il, color, xor)
```

本函数要调用象点读写函数 POINT,在调用之前要用第 18 章中某个象点读写函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。整个虚线起点在屏幕上的行、列号。

row2, col2——均为整型变量。整个虚线终点在屏幕上的行、列号。

rl——整型变量。实线部分的长度。

il——整型变量。空白部分的长度。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时,表示以直接方式画虚线;当 xor ≠ 0 时,表示以异或方式画虚线。

五、函数程序(文件名: simln.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

首先将边长为 10、颜色值为 4 的正方形边框从屏幕的左上角逐步移到 300 行、300 列的位置处;

然后用颜色值为 4 的虚线(实线长为 15、空白部分为 8)画一个行宽为 350、列宽为 640 的长方形边框以及两条对角线。

本主函数要调用画线段构成图形的函数 smull(参看 19.2 节)。

在主函数程序中,对于 j 与 k 的双重循环是为了等待,以便能看清小正方形的移动过程。

主函数程序(文件名: simln 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "smull.c"
#include "simln.c"
#include "rmode.c"
main()
{
    int i, j, k;
    static int g[4][4] = { {0, 0, 0, 10}, {0, 10, 10, 10},
```

```
    {10, 10, 10, 0}, {10, 0, 0, 0} };
rmode(0x62);
smull(0, 0, g, 4, 1);
for (i = 0; i <= 300; i = i + 10)
    { smull(i, i, g, 4, 1);
        for (j = 0; j <= 30; j++)
            for (k = 0; k <= 10000; k++)
                smull(i + 10, i + 10, g, 4, 1);
                for (j = 0; j <= 30; j++)
                    for (k = 0; k <= 10000; k++)
                        ;
    }
    simln(0, 0, 0, 639, 15, 8, 4, 0);
    simln(0, 639, 349, 639, 15, 8, 4, 0);
    simln(349, 639, 349, 0, 15, 8, 4, 0);
    simln(349, 0, 0, 15, 8, 4, 0);
    simln(0, 0, 349, 639, 15, 8, 4, 0);
    simln(0, 639, 349, 0, 15, 8, 4, 0);
}
```

19.4 单点划线

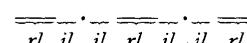
一、功能

根据给定的起点(row1, col1)与终点(row2, col2),实线长度 rl 和空白长度 il,用指定颜色值 color 画单点划线。

二、方法说明

基本方法同直线插补,参看 19.1 节。

设空白长度为 il,则两段实线之间的间隔为 $2 * il + 1$,中间有一个点,此点到两段相邻实线的间隔均为 il。即单点划线的形状如下:



三、函数语句

```
void sdlin(row1, col1, row2, col2, rl, il, color, xor)
```

本函数要调用读写象点函数 POINT,在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。整个单点划线起点在屏幕上的行、列号。

row2, col2——均为整型变量。整个单点划线终点在屏幕上的行、列号。

rl——整型变量。单点划线中实线部分长度。

il——整型变量。单点划线中空白部分长度。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画单点划线; 当 xor ≠ 0 时, 表示以异或方式画单点划线。

五、函数程序(文件名: sdlin.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

首先将边长为 10、颜色值为 4 的正方形边框从屏幕的左上角逐步移到 300 行、300 列的位置;

然后用单点划线(实线长为 15, 空白长为 4)以颜色值 4 画一个行宽为 350、列宽为 640 的长方形边框以及两条对角线。

本主函数要调用画线段构成图形的函数 smull, 参看 19.2 节。

在主函数程序中, 对于 j 与 k 的双重循环是为了等待, 以便能看清小正方形的移动过程。

主函数程序(文件名: sdlin0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "smull.c"
#include "sdlin.c"
#include "rmode.c"
main()
{
    int i, j, k;
    static int g[4][4] = { {0, 0, 0, 10}, {0, 10, 10, 10},
                          {10, 10, 10, 0}, {10, 0, 0, 0} };
    rmode(0x62);
    smull(0, 0, g, 4, 4, 1);
    for (i = 0; i < 300; i = i + 10)
        smull(i, i, g, 4, 4, 1);
    for (j = 0; j < = 30; j++)
        for (k = 0; k < = 10000; k++)
            smull(i + 10, i + 10, g, 4, 4, 1);
    for (j = 0; j < = 30; j++)
        for (k = 0; k < = 10000; k++)
            smull(0, 0, 639, 15, 4, 4, 0);
    sdlin(0, 0, 639, 15, 4, 4, 0);
    sdlin(0, 639, 349, 15, 4, 4, 0);
    sdlin(349, 639, 349, 0, 15, 4, 4, 0);
    sdlin(349, 0, 0, 0, 15, 4, 4, 0);
    sdlin(0, 0, 349, 15, 4, 4, 0);
    sdlin(0, 639, 349, 0, 15, 4, 4, 0);
}
```

19.5 双点划线

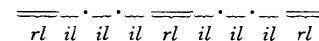
一、功能

根据给定的起点($row1, col1$)和终点($row2, col2$), 实线部分长度 rl 和空白长度 il , 用指定的颜色值 $color$ 画双点划线。

二、方法说明

基本方法同直线插补, 参看 19.1 节。

设实线长度为 rl , 空白长度为 il , 则两段实线之间的间隔为 $3 * il + 2$, 中间有两个点。即双点划线的形状如下:



三、函数语句

```
void sddln(row1, col1, row2, col2, rl, il, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。整个双点划线起点在屏幕上的行、列号。

row2, col2——均为整型变量。整个双点划线终点在屏幕上的行、列号。

rl——整型变量。双点划线中实线部分长度。

il——整型变量。双点划线中空白部分长度。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画双点划线; 当 xor ≠ 0 时, 表示以异或方式画双点划线。

五、函数程序(文件名: sddln.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

首先将边长为 10、颜色值为 4 的正方形边框从屏幕的左上角逐步移到屏幕的 300 行、300 列位置处;

然后用双点划线(实线长为 15、空白长为 4)以颜色值 4 画一个行宽为 350、列宽为 640 的长方形边框以及两条对角线。

本主函数中要调用由线段构成图形的函数 smull, 参看 19.2 节。

在主函数程序中, 对于 j 与 k 的双重循环是为了等待, 以便能看清小正方形的移动过程。

主函数程序(文件名: sddln 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "smull.c"
#include "sddln.c"
#include "rmode.c"
main()
{
    int i, j, k;
    static int g[4][4] = { {0, 0, 0, 10}, {0, 10, 10, 10},
                          {10, 10, 10, 0}, {10, 0, 0, 0} };
    rmode(0x62);
    smull(0, 0, g, 4, 4, 1);
    for (i = 0; i <= 300; i = i + 10)
        smull(i, i, g, 4, 4, 1);
    for (j = 0; j <= 30; j++)
        for (k = 0; k <= 10000; k++)
            smull(i + 10, i + 10, g, 4, 4, 1);
    for (j = 0; j <= 30; j++)
        for (k = 0; k <= 10000; k++);
    {
        sddln(0, 0, 639, 15, 4, 4, 0);
        sddln(0, 639, 349, 639, 15, 4, 4, 0);
        sddln(349, 639, 349, 0, 15, 4, 4, 0);
        sddln(349, 0, 0, 0, 15, 4, 4, 0);
        sddln(0, 0, 349, 639, 15, 4, 4, 0);
        sddln(0, 639, 349, 0, 15, 4, 4, 0);
    }
}
```

19.6 坐 标 轴

一、功能

根据给定的参数画坐标轴。

二、方法说明

略。

三、函数语句

```
void saxis(row, col, k, w, n, l, color, xor)
```

本函数要调用画直线的函数 sline, 参看 19.1 节。

四、形参说明

row, col——均为整型变量。坐标轴起始点在屏幕上的行、列号。

k——整型变量。当 $k > 0$ 时, 表示从起始点(row, col)向右画横坐标轴; 当 $k < 0$ 时,

表示从起始点(row, col)向下画纵坐标轴; 当 $k = 0$ 时, 不画坐标轴。

w——整型变量。坐标轴上一个刻度的宽度。

n——整型变量。坐标轴上刻度的总数。

l——整型变量。当 $l > 0$ 时, 刻度在坐标轴的上方(对于横坐标轴)或右侧(对于纵坐标轴); 当 $l < 0$ 时, 刻度在横坐标轴的下方或纵坐标轴的左侧; 当 $l = 0$ 时, 刻度线中心在坐标轴上。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式画坐标轴; 当 $xor \neq 0$ 时, 表示以异或方式画坐标轴。

五、函数程序(文件名: saxis.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 分别以(10, 10)、(30, 10)、(50, 10)为起始点, 用颜色值 4, 画三个横坐标轴, 刻度宽均为 15, 共 20 个刻度, 刻度分别在中心、上方与下方; 再分别以(60, 50)、(60, 100)、(60, 150)为起始点, 用颜色值 4, 画三个纵坐标轴, 刻度宽均为 15, 共 15 个刻度, 刻度分别在中心、右侧与左侧。

主函数程序(文件名: saxis 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "saxis.c"
#include "rmode.c"
main()
{
    rmode(0x62);
    saxis(10, 10, 1, 15, 20, 0, 4, 1);
    saxis(30, 10, 1, 15, 20, 1, 4, 0);
    saxis(50, 10, 1, 15, 20, -1, 4, 1);
    saxis(60, 50, -1, 15, 15, 0, 2, 1);
    saxis(60, 100, -1, 15, 15, 1, 2, 0);
    saxis(60, 150, -1, 15, 15, -1, 2, 1);
}
```

19.7 矩 形 及 其 填 充

一、功能

在屏幕的指定位置画矩形或矩形填充。

二、方法说明

(略)

三、函数语句

```
void ssbox(row1, col1, row2, col2, color, k, xor)
```

• 431 •

本函数要调用读写象点函数 POINT，在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。矩形左上角在屏幕上的行、列号。

row2, col2——均为整型变量。矩形右下角在屏幕上的行、列号。

color——整型变量。颜色值。

k——整型变量。当 k=0 时，表示只画矩形边框而不填充；当 k≠0 时，表示要对矩形进行填充。

xor——整型变量。异或标志。当 xor=0 时，表示以直接方式画矩形或填充；当 xor≠0 时，表示以异或方式画矩形或填充。

五、函数程序(文件名: ssbox.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下，用颜色值 2 画一个左上角坐标为(30, 50)、右下角坐标为(80, 90)的矩形边框；再用颜色值 4 画一个左上角坐标为(100, 100)、右下角坐标为(200, 150)的矩形填充。

主函数程序(文件名: ssbox 0.c)如下：

```
#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
#include "ssbox.c"
main()
{
    rmode(0x62);
    ssbox(30, 50, 80, 90, 2, 0, 1);
    ssbox(100, 100, 200, 150, 4, 1, 1);
}
```

19.8 矩形域图形的清除

一、功能

将屏幕上指定矩形区域内的图形按点进行清除。

二、方法说明

在指定的矩形区域内逐点读象点值，再将该值以异或方式写回。

三、函数语句

```
void sscls(row1, col1, row2, col2)
```

本函数要调用读写象点函数 POINT，在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。矩形区域左上角在屏幕上的行、列号。

row2, col2——均为整型变量。矩形区域右下角在屏幕上的行、列号。

五、函数程序(文件名: sscls.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下：

首先以屏幕上的位置(170, 320)为顶点，起始点相对于顶点的坐标为(100, 100)画抛物线，旋转角分别为 0°, 30°, 60°, 90°, 120°, 150°, 180°, 210°, 240°, 270°, 300°, 330°，颜色值分别为 1~12；

然后清除矩形区域(70:270, 220:420)内的部分抛物线。

本主函数要调用画抛物线的函数 spara，参看 19.18 节。

主函数程序(文件名: sscls 0.c)如下：

```
#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
#include "spara.c"
#include "sscls.c"
main()
{
    int i, color;
    rmode(0x62);
    color = 1;
    for (i = 0; i < 11; i++)
        spara(170, 320, 100, 100, i * 30, color, 1);
    color = color + 1;
}
for (i = 0; i < = 10000; i++)
    for (color = 0; color < = 100; color++)
        sscls(70, 220, 270, 420);
```

19.9 矩形域图形的复制

一、功能

将屏幕上一个矩形域(称为源区)内的图形逐点复制到另一个矩形域(称为目的区)。

二、方法说明

当源区位于目的区的右下方时，从源区的左上角开始、以行为顺序逐点将象点信息复制到目的区；

当源区位于目的区的左上方时，从源区的右下角开始、以行为顺序逐点将象点信息复制

到目的区。

三、函数语句

```
void scopy (row1, col1, row2, col2, row, col, xor)
```

本函数要调用读写象点函数 POINT，在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。源区左上角在屏幕上的行、列号。

row2, col2——均为整型变量。源区右下角在屏幕上的行、列号。

row, col——均为整型变量。目的区左上角在屏幕上的行、列号。

xor——整型变量。异或标志。当 xor = 0 时，表示以直接方式对目的区写象点；当 xor ≠ 0 时，表示以异或方式对目的区写象点。

五、函数程序(文件名: scopy.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下：

首先以屏幕上的位置(170,320)为顶点，起始点相对于顶点的坐标为(100,100)，旋转角分别为 $t_i = i * 30^\circ$ ($i = 0, 1, \dots, 11$)的抛物线，颜色值分别为 1~12。

然后将矩形区域(50:300, 150:450)中的图形直接复制到左上角为(0,0)的另一矩形区域中。

再将矩形区域(0:200, 0:300)中的图形直接复制到左上角为(150,300)的矩形区域中。

本主函数要调用画抛物线的函数 spara, 参看 19.18 节。

主函数程序(文件名: scopy 0.c)如下：

```
# define POINT rtv62
# include "rtv62.c"
# include "rmode.c"
# include "spara.c"
# include "scopy.c"
main()
{
    int i, color;
    rmode(0x62);
    color = 1;
    for (i = 0; i <= 11; i++)
        spara(170, 320, 100, 100, i * 30, color, 1);
    color = color + 1;
}
scopy(50, 150, 300, 450, 0, 0, 0);
scopy(0, 0, 200, 300, 150, 300, 0);
```

19.10 矩形域图形的平移

一、功能

将屏幕上一个矩形区域(称为源区)上的图形逐点平移到另一矩形区域(称为目的区)中。

二、方法说明

当源区位于目的区的右下方时，从源区的左上角开始、以行为顺序逐点将象点平移到目的区中；

当源区位于目的区的左上方时，从源区的右下角开始、以行为顺序逐点将象点平移到目的区中。

三、函数语句

```
void smove(row1, col1, row2, col2, row, col, xor)
```

本函数要调用读写象点函数 POINT，在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。源区左上角在屏幕上的行、列号。

row2, col2——均为整型变量。源区右下角在屏幕上的行、列号。

row, col——均为整型变量。目的区左上角在屏幕上的行、列号。

xor——整型变量。异或标志。当 xor = 0 时，表示以直接方式对目的区写象点；当 xor ≠ 0 时，表示以异或方式对目的区写象点。

五、函数程序(文件名: smove.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下：

首先以屏幕上的位置(170,320)为顶点，相对于顶点的坐标(100,100)为起始点画抛物线，旋转角度分别为 $t_i = 30^\circ * i$ ($i = 0, 1, \dots, 11$)，颜色值分别为 1~12。

然后将矩形区域(50:300, 150:450)中的图形以直接方式平移到左上角为(0,0)的另一矩形区域中。

再将矩形区域(0:200, 0:300)中的图形以直接方式平移到左上角为(150,300)的矩形区域中。

本主函数要调用画抛物线的函数 spara, 参看 19.18 节。

主函数程序(文件名: smove 0.c)如下：

```
# define POINT rtv62
# include "rtv62.c"
# include "rmode.c"
```

```

#include "spara.c"
#include "smove.c"
main()
{
    int i, color;
    rmode(0x62);
    color = 1;
    for (i = 0; i <= 11; i++)
        spara(170, 320, 100, 100, i * 30, color, 1);
    color = color + 1;
}
smove (50, 150, 300, 450, 0, 0, 0);
smove (0, 0, 200, 300, 150, 300, 0);
}

```

19.11 圆形域图形的复制

一、功能

将屏幕上一圆形区域(称为源区)中的图形逐点复制到另一圆形区域(称为目的区)中。

二、方法说明

当源区位于目的区右下方时,从源区的上方开始、以行为顺序(同一行从左边界到右边界)逐点将象点信息复制到目的区;

当源区位于目的区左上方时,从源区的下方开始、以行为顺序(同一行从右边界到左边界)逐点将象点信息复制到目的区。

三、函数语句

```
void sccpy(row1, col1, a, b, row, col, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。源区中心在屏幕上的行、列号。

a——整型变量。圆形域横半轴长度(即横半轴在屏幕上所占的列数)。

b——整型变量。圆形域纵半轴长度(即纵半轴在屏幕上所占的行数)。

row, col——均为整型变量。目的区中心在屏幕上的行、列号。

xor——整型变量。异或标志。当 xor=0 时,表示以直接方式对目的区写象点;当 xor ≠ 0 时,表示以异或方式对目的区写象点。

五、函数程序(文件名: sccpy.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

对矩形区域(60:280, 150:450)用颜色值以异或方式进行填充;

将以(170, 300)为中心, 半轴 a=b=50 的圆形域内容用异或方式复制到以(60, 150)为中心以及以(280, 450)为中心的圆形域中。

本主函数要调用矩形填充的函数 ssbox, 参看 19.7 节。

主函数程序(文件名: sccpy 0.c)如下:

```

#define POINT rtv62
#include "rmode.c"
#include "rtv62.c"
#include "sccpy.c"
#include "ssbox.c"
main()
{
    rmode(0x62);
    ssbox(60, 150, 280, 450, 4, 1, 1);
    sccpy(170, 300, 50, 50, 60, 150, 1);
    sccpy(170, 300, 50, 50, 280, 450, 1);
}

```

19.12 圆形域图形的平移

一、功能

将屏幕上一圆形区域(称为源区)中的图形逐点平移到另一圆形区域(称为目的区)中。

二、方法说明

当源区位于目的区右下方时,从源区的上方开始、以行为顺序(同一行从左边界到右边界)逐点将象点信息平移到目的区;

当源区位于目的区左上方时,从源区的下方开始、以行为顺序(同一行从右边界到左边界)逐点将象点信息平移到目的区。

三、函数语句

```
void scmve(row1, col1, a, b, row, col, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row1, col1——均为整型变量。源区中心在屏幕上的行、列号。

a——整型变量。圆形域横半轴长度(即横半轴在屏幕上所占的列数)。

b——整型变量。圆形域纵半轴长度(即纵半轴在屏幕上所占的行数)。

row, col——均为整型变量。目的区中心在屏幕上的行、列号。

xor——整型变量。异或标志。当 xor=0 时,表示以直接方式对目的区写象点;当 xor ≠ 0 时,表示以异或方式对目的区写象点。

五、函数程序(文件名: scmve.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下:

对矩形区域(60:280,150:450)用颜色值进行填充;

将以(170,300)为中心、半轴 $a = b = 50$ 的圆形域内容平移到以(60,150)为中心的圆形域中(用异或方式);

再将以(60,150)为中心、半轴 $a = b = 50$ 的圆形域内容平移到以(280,450)为中心的圆形域中(用异或方式)。

本主函数要调用矩形填充函数 ssbox, 参看 19.7 节。

主函数程序(文件名: scmve 0.c)如下:

```
#define POINT rtv62
#include "rmode.c"
#include "rtv62.c"
#include "scmve.c"
#include "ssbox.c"
main()
{
    rmode(0x62);
    ssbox(60,150,280,450,4,1,1);
    scmve(170,300,50,50,60,150,1);
    scmve(60,150,50,50,280,450,1);
}
```

19.13 由中心、半轴(半径)以及起终点夹角画椭圆(圆)弧

一、功能

根据给定的中心坐标、横半轴与纵半轴的长度以及起终点相对于水平轴的中心角画椭圆或圆弧。

二、方法说明

设横半轴与纵半轴的长度分别为 a 与 b 。起终点相对于水平轴的中心角分别为 t_1 与 t_2 , 则圆弧所对的中心角为

$$t = t_2 - t_1$$

若令长半轴为

$$r = \max\{a, b\}$$

则以 r 为半径的圆弧起点相对于中心的坐标为

$$\begin{cases} x_1 = r \cos(t_1) \\ y_1 = r \sin(t_1) \end{cases}$$

圆弧插补的基本方法如下。

(1) 首先设置插补的初始信息:

若 $t > 0$, 则按逆时针画弧。此时按如下规则设置水平与垂直方向的移动增量 dx 与 dy :

$$dx = \begin{cases} 1, & \text{当 } y_1 = 0 \text{ 且 } x_1 < 0, \text{ 或 } y_1 < 0 \\ -1, & \text{当 } y_1 = 0 \text{ 且 } x_1 > 0, \text{ 或 } y_1 > 0 \end{cases}$$

$$dy = \begin{cases} 1, & \text{当 } x_1 = 0 \text{ 且 } y_1 < 0, \text{ 或 } x_1 > 0 \\ -1, & \text{当 } x_1 = 0 \text{ 且 } y_1 > 0, \text{ 或 } x_1 < 0 \end{cases}$$

若 $t < 0$, 则按顺时针画弧。此时按如下规则设置水平与垂直方向的移动增量 dx 与 dy :

$$dx = \begin{cases} 1, & \text{当 } y_1 = 0 \text{ 且 } x_1 < 0, \text{ 或 } y_1 > 0 \\ -1, & \text{当 } y_1 = 0 \text{ 且 } x_1 > 0, \text{ 或 } y_1 < 0 \end{cases}$$

$$dy = \begin{cases} 1, & \text{当 } x_1 = 0 \text{ 且 } y_1 < 0, \text{ 或 } x_1 < 0 \\ -1, & \text{当 } x_1 = 0 \text{ 且 } y_1 > 0, \text{ 或 } x_1 > 0 \end{cases}$$

然后置如下初值:

$$fx = 2 * x_1 * dx + 1, fy = 2 * y_1 * dy + 1, f = 0$$

$$lx = ly = mx = my = 0$$

(2) 进行弧插补

首先显示起始点 $(a \cos(t_1), b \sin(t_1)) \overset{\leftrightarrow}{=} (x, y)$ 。

然后按如下规则进行插补:

若 $f \geq 0$, 首先 $lx + a \Rightarrow lx$, 若 $lx \geq r$ 则 $x + dx \Rightarrow x$, $lx - r \Rightarrow lx$, 显示象点 (x, y) 。然后

$$f - |fx| \Rightarrow f, fx + 2 \Rightarrow fx$$

此时, 如果 $fx * (fx - 2) \leq 0$, 则 $-dy \Rightarrow dy, -fy + 2 \Rightarrow fy, -f \Rightarrow f$ 。

若 $f < 0$, 首先 $ly + b \Rightarrow ly$, 若 $ly \geq r$, 则 $y + dy \Rightarrow y, ly - r \Rightarrow ly$, 显示象点 (x, y) 。然后

$$f + |fy| \Rightarrow f, fy + 2 \Rightarrow fy$$

此时, 如果 $fy * (fy - 2) \leq 0$, 则 $-dx \Rightarrow dx, -fx + 2 \Rightarrow fx, -f \Rightarrow f$ 。

上述过程一直作到终点为止。

在使用本函数时特别要注意, 由于显示屏幕上各象点之间的行距与列距并不相同, 而且在不同的分辨率下的行距与列距比例关系也不相同。因此, 画圆与椭圆没有明确的界限。此时需要用户根据使用的分辨率, 调整好参数 a 与 b , 才能获得理想的圆或椭圆。

三、函数语句

void scir 1(row, col, a, b, t1, t2, color, xor)

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。椭圆或圆的中心在屏幕上的行、列号。

a——整型变量。横半轴长度(即横半轴在屏幕上所占的列数)。

b——整型变量。纵半轴长度(即纵半轴在屏幕上所占的行数)。

t1——整型变量。弧起始点相对于水平轴的中心角(单位为度)。

t2——整型变量。弧终点相对于水平轴的中心角(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画弧; 当 xor ≠ 0 时, 表示以异或方式画弧。

五、函数程序(文件名: scir 1.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 4 以直接方式画 4 个象限内的圆弧(拼成一个整圆)。其中 a = 140, b = 100, 中心坐标为(170, 320)。

主函数程序(文件名: scir 10.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "scir1.c"
#include "rmode.c"
main()
| rmode(0x62);
scir1(170, 320, 140, 100, 0, 90, 4, 0);
scir1(170, 320, 140, 100, 90, 180, 4, 0);
scir1(170, 320, 140, 100, 180, 270, 4, 0);
scir1(170, 320, 140, 100, 270, 360, 4, 0);
```

19.14 由中心、半轴(半径)以及起终点夹角画扇形

一、功能

根据给定的中心坐标、横半轴与纵半轴的长度以及起终点与水平轴的中心角画扇形。

二、方法说明

画弧的方法同 19.13 节。

从弧的起点与终点各画一条到中心的线段就得扇形。

三、函数语句

```
void scir2(row, col, a, b, t1, t2, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

本函数还要调用画直线的函数 sline, 参看 19.1 节。

四、形参说明

row, col——均为整型变量。中心在屏幕上的行、列号。

a——整型变量。横半轴长度(即横半轴在屏幕上所占的列数)。

b——整型变量。纵半轴长度(即纵半轴在屏幕上所占的行数)。

t1——整型变量。起点相对于水平轴的中心角(单位为度)。

t2——整型变量。终点相对于水平轴的中心角(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画扇形; 当 xor ≠ 0 时, 表示以异或方式画扇形。

五、函数程序(文件名: scir 2.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 4 以直接方式画整圆中等分的 6 个扇形。其中 a = 140, b = 100, 中心坐标为(170, 320)。

主函数程序(文件名: scir 20.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "scir2.c"
#include "rmode.c"
main()
| rmode(0x62);
scir2(170, 320, 140, 100, 0, 60, 4, 0);
scir2(170, 320, 140, 100, 60, 120, 4, 0);
scir2(170, 320, 140, 100, 120, 180, 4, 0);
scir2(170, 320, 140, 100, 180, 240, 4, 0);
scir2(170, 320, 140, 100, 240, 300, 4, 0);
scir2(170, 320, 140, 100, 300, 360, 4, 0);
```

19.15 由中心与半轴(半径)画椭圆(圆)

一、功能

根据给定的中心以及半轴长度, 用指定的颜色画椭圆或圆。

二、方法说明

画圆或椭圆实际上是画 0° 到 360° 的弧。圆弧插补的方法参看 19.13 节。

三、函数语句

```
void scir3(row, col, a, b, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。中心在屏幕上的行、列号。
a——整型变量。横半轴的长度(即横半轴在屏幕上所占的列数)。
b——整型变量。纵半轴的长度(即纵半轴在屏幕上所占的行数)。
color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画圆或椭圆; 当 xor ≠ 0 时, 表示以异或方式画圆或椭圆。

五、函数程序(文件名: scir3.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 4, 以异或方式画一个中心在(170, 320), 横半轴 a = 140、纵半轴 b = 100 的圆。

主函数程序(文件名: scir30.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "scir3.c"
#include "rmode.c"
main()
{
    rmode(0x62);
    scir3(170, 320, 140, 100, 4, 1);
}
```

19.16 由中心、半轴(半径)以及起 终点夹角画扇形填充

一、功能

根据给定的中心位置、横半轴与纵半轴长度以及起终点与水平轴的中心角画扇形填充。

二、方法说明

用弧插补的方法(参看 19.13 节)确定弧上的点, 然后画弧上的点到中心的线段, 即得到扇形填充。特别要注意, 本函数以直接方式显示象点。

三、函数语句

```
void sful1(row, col, a, b, t1, t2, color)
```

本函数要调用画直线函数 sline, 参看 19.1 节。

四、形参说明

row, col——均为整型变量。中心在屏幕上的行、列号。
a——整型变量。横半轴长度(即横半轴在屏幕上所占的列数)。
b——整型变量。纵半轴长度(即纵半轴在屏幕上所占的行数)。
t1——整型变量。起点相对于水平轴的中心角(单位为度)。
t2——整型变量。终点相对于水平轴的中心角(单位为度)。
color——整型变量。颜色值。

五、函数程序(文件名: sful1.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 4, 在中心(170, 320)处, 以横半轴 a = 140、纵半轴 b = 100 分别画 60° 到 120°、180° 到 240°、300° 到 360° 的三个扇形填充。

主函数程序(文件名: sful10.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "sful1.c"
#include "rmode.c"
main()
{
    rmode(0x62);
    sful1(170, 320, 140, 100, 60, 120, 4);
    sful1(170, 320, 140, 100, 180, 240, 4);
    sful1(170, 320, 140, 100, 300, 360, 4);
}
```

19.17 由中心与半轴(半径)画 椭圆(圆)填充

一、功能

根据给定的中心位置, 用指定的颜色值画圆或椭圆填充。

二、方法说明

利用弧插补的方法(参看 19.13 节), 对第一象限内的弧进行插补, 并对于该象限内弧上的每一个点, 在其它三个象限内找到与之对称的点, 然后由一、四两象限内的点画直线, 由二、三两象限内的点画直线。

三、函数语句

```
void sful2(row, col, a, b, color, xor)
```

本函数要调用画直线的函数 sline, 参看 19.1 节。

四、形参说明

row, col——均为整型变量。中心在屏幕上的行、列号。

a——整型变量。横半轴长度(即横半轴在屏幕上所占的列数)。

b——整型变量。纵半轴长度(即纵半轴在屏幕上所占的行数)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画圆填充; 当 xor ≠ 0 时, 表示以异或方式画圆填充。

五、函数程序(文件名: sfu12.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 4, 以异或方式在中心(170, 320)处画横半轴 a = 140、纵半轴 b = 100 的圆填充。

主函数程序(文件名: sfu12.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "sfu12.c"
#include "rmode.c"
main()
{ rmode(0x62);
  sfu12(170, 320, 140, 100, 4, 1);
}
```

19.18 抛物线

一、功能

根据给定的顶点位置、起点以及旋转角度, 用指定的颜色画抛物线。

二、方法说明

设抛物线为

$$y = ax^2$$

且从起点(x₁, y₁)到终点(-x₁, y₁)。其中(x₁, y₁)为相对于抛物线顶点的坐标。

抛物线插补的过程如下。

(1) 设置插补的初始信息。

水平方向和垂直方向的移动增量 dx 与 dy 为

$$dx = \begin{cases} 1, & \text{当 } x_1 < 0 \\ -1, & \text{当 } x_1 > 0 \end{cases}$$

$$dy = \begin{cases} 1, & \text{当 } y_1 < 0 \\ -1, & \text{当 } y_1 > 0 \end{cases}$$

其它插补信息的初值为

$$fx = -(2 * x_1 * dx + 1.0) * y_1, rx = -2 * y_1$$

$$fy = x_1^2 * dy, f = 0$$

(2) 进行抛物线插补。

首先显示起点(x₁, y₁) $\xrightarrow{\uparrow}$ (x, y)。

然后按如下规则进行插补:

若 f ≥ 0, 则 x + dx ⇒ x, 显示象点(x, y), 且

$$f - |fx| \Rightarrow f, fx + rx \Rightarrow fx$$

此时如果 fx * (fx - rx) ≤ 0, 则 -dy ⇒ dy, -fy ⇒ fy, -f ⇒ f。

若 f < 0, 则 y + dy ⇒ y, 显示象点(x, y), 且

$$f + |fy| \Rightarrow f$$

以上过程一直作到终点为止。

考虑到旋转的问题, 在插补过程中实际显示的象点为

$$x_2 = x \cos t - y \sin t, y_2 = x \sin t + y \cos t$$

其中 t 为绕抛物线顶点旋转的角度。

象点在屏幕上的实际行、列号为

$$(row - y_2, col + x_2)$$

其中 (row, col) 为抛物线顶点在屏幕上的行、列号。

三、函数语句

void spara(row, col, x1, y1, t, color, xor)

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。抛物线顶点在屏幕上的行、列号。

x1, y1——均为整型变量。抛物线起点相对于顶点的坐标。

t——整型变量。抛物线绕顶点旋转的角度(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor = 0 时, 表示以直接方式画抛物线; 当 xor ≠ 0 时, 表示以异或方式画抛物线。

五、函数程序(文件名: spara.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 分别用颜色值 1~12, 画顶点在(170, 320)

处、起点相对于顶点的坐标为(100, 100), 分别旋转 $i * 30^\circ$ ($i = 0, 1, \dots, 11$) 的抛物线(用异或方式)。

主函数程序(文件名: spara 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
#include "spara.c"
main()
{
    int i, color;
    rmode(0x62);
    color = 1;
    for (i = 0; i <= 11; i++)
        spara(170, 320, 100, 100, i * 30, color, 1);
    color = color + 1;
}
```

19.19 双 曲 线

一、功能

根据给定的中心在屏幕上的行、列号以及实半轴长度、起点相对于中心的坐标, 用指定的颜色画旋转后的双曲线。

二、方法说明

设双曲线方程为

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

起点在 (x_1, y_1) , 画从起点 (x_1, y_1) 到终点 $(x_1, -y_1)$ 的一段双曲线。

双曲线插补的过程如下。

(1) 设置初始的插补信息。

水平方向与垂直方向的移动增量 dx 与 dy 为

$$dx = \begin{cases} 1, & \text{当 } x_1 < 0 \text{ 时} \\ -1, & \text{当 } x_1 > 0 \text{ 时} \end{cases}$$

$$dy = \begin{cases} 1, & \text{当 } y_1 < 0 \text{ 时} \\ -1, & \text{当 } y_1 > 0 \text{ 时} \end{cases}$$

其它插补信息初值为

$$rx = \frac{2 * a^2 * y_1^2}{x_1^2 - a^2}, fx = \frac{a^2 * y_1^2}{x_1^2 - a^2} (2 * x_1 * dx + 1)$$

$$ry = -2a^2, fy = -a^2 (2 * y_1 * dy + 1)$$

$$f = 0$$

(2) 进行双曲线插补。

首先显示起点 $(x_1, y_1) \xrightarrow{\oplus} (x, y)$ 。

然后按如下规则进行插补:

若 $f \geq 0$, 则 $x + dx \Rightarrow x$, 显示象点 (x, y) , 且

$$f \leftarrow f - rx \Rightarrow f, fx + rx \Rightarrow fx$$

此时如果 $fx * (fx - rx) \leq 0$, 则 $-dy \Rightarrow dy, -fy + ry \Rightarrow fy, -f \Rightarrow f$ 。

若 $f < 0$, 则 $y + dy \Rightarrow y$, 显示象点 (x, y) , 且

$$f \leftarrow f + fy \Rightarrow f, fy + ry \Rightarrow fy$$

此时如果 $fy * (fy - ry) \leq 0$, 则 $-dx \Rightarrow dx, -fx + rx \Rightarrow fx, -f \Rightarrow f$ 。

以上过程一直作到终点为止。

考虑到旋转的问题, 在插补过程中, 象点 (x, y) 相对于中心的实际坐标为

$$x_2 = x \cos t - y \sin t, y_2 = x \sin t + y \cos t$$

其中 t 为绕中心旋转的角度。象点 (x_2, y_2) 在屏幕上的行、列号为

$$(row - y_2, col + x_2)$$

其中 (row, col) 为双曲线中心在屏幕上的行、列号。

三、函数语句

```
void shype(row, col, a, x1, y1, t, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。双曲线中心在屏幕上的行、列号。

a——整型变量。双曲线实半轴长度(即半轴在屏幕上所占的列数)。

x1, y1——均为整型变量。双曲线起点相对于中心的横坐标与纵坐标。

t——整型变量。双曲线绕中心旋转的角度(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式画双曲线; 当 $xor \neq 0$ 时, 表示以异或方式画双曲线。

五、函数程序(文件名: shype.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 分别用颜色值 1~12, 画中心在(170, 320)处, 起点相对于中心坐标为(100, 100), 实半轴长 $a = 40$, 分别旋转 $i * 30^\circ$ ($i = 0, 1, \dots, 11$) 的双曲线(用异或方式)。

主函数程序(文件名: shype 0.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
```

```

#include "shype.c"
main()
{
    int i, color;
    rmode(0x62);
    color = 1;
    for (i = 0; i <= 11; i++)
        shype(170, 320, 40, 100, 100, i * 30, color, 1);
    color = color + 1;
}

```

19.20 三次多项式曲线

一、功能

用指定的颜色画三次多项式曲线

$$y = bx + cx^2 + dx^3$$

二、方法说明

设三次多项式曲线方程为

$$y = bx + cx^2 + dx^3$$

x 从 0 到 x_t 画一段三次多项式曲线。

三次多项式曲线插补的过程如下。

(1) 设置初始的插补信息。

水平方向与垂直方向的移动增量 dx 与 dy 为

$$dx = \begin{cases} 1, & \text{当 } x_t > 0 \text{ 时} \\ -1, & \text{当 } x_t < 0 \text{ 时} \end{cases}$$

$$dy = \begin{cases} dx, & \text{当 } b + d + c * dx \geq 0 \text{ 时} \\ -dx, & \text{当 } b + d + c * dx < 0 \text{ 时} \end{cases}$$

其它插补信息初值为

$$fx = b + d + c * dx$$

$$rx = 2 * c * dx, rx1 = 6 * d, f = 0$$

(2) 进行三次多项式曲线插补。

首先显示原点 $(0, 0) \xrightarrow{\uparrow} (x, y)$ 。

然后按如下规则进行插补：

若 $f \geq 0$, 则 $x + dx \Rightarrow x$, 显示象点 (x, y) , 且

$$f \leftarrow fx \Rightarrow f, rx + rx1 \Rightarrow rx, fx + rx \Rightarrow fx$$

此时如果 $fx * (fx - rx) \leq 0$, 则 $-dy \Rightarrow dy, -f \Rightarrow f$ 。

若 $f < 0$, 则 $y + dy \Rightarrow y$, 显示象点 (x, y) , 且

$$f + 1 \Rightarrow f$$

以上过程一直作到终点为止。

考虑到旋转的问题, 在插补过程中, 象点 (x, y) 相对于中心的实际坐标为

$$x_2 = x \cos t - y \sin t, y_2 = x \sin t + y \cos t$$

其中 t 为绕原点旋转的角度。象点 (x_2, y_2) 在屏幕上的行、列号为

$$(row - y_2, col + x_2)$$

其中 (row, col) 为原点在屏幕上的行、列号。

三、函数语句

void scubc(row, col, b, c, d, xt, t, color, xor)

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

row, col——均为整型变量。原点在屏幕上的行、列号。

b, c, d——均为双精度实型变量。三次多项式 $y = bx + cx^2 + dx^3$ 中的系数。

xt——整型变量。三次多项式曲线终点相对于原点的横坐标。

t——整型变量。三次多项式曲线绕原点旋转的角度(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式画曲线; 当 $xor \neq 0$ 时, 表示以异或方式画曲线。

五、函数程序(文件名: scubc.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 分别用颜色值 1~4, 画三次多项式函数

$$y = 0.5x - 0.1x^2 + 0.00x^3$$

绕原点旋转 $0^\circ, 90^\circ, 180^\circ, 270^\circ$ 的曲线(用异或方式)。其中原点在屏幕上的行、列号为(170, 320)。

主函数程序(文件名: scubc 0.c)如下:

```

#define POINT rtv62
#include "rtv62.c"
#include "rmode.c"
#include "scubc.c"
main()
{
    int i;
    double b, c, d;
    rmode(0x62);
    d = 0.001; c = -0.1; b = 0.5;
    for (i = 0; i <= 3; i++)
        scubc(170, 320, b, c, d, 100, i * 90, i + 1, 1);
}

```

19.21 一般函数曲线

一、功能

画指定颜色的一般曲线。

二、方法说明

设给定函数

$$y = f(x)$$

本函数用逐点显示的方法绘制该函数从起点到终点的曲线。要注意，本函数只适用于画单值函数曲线。

三、函数语句

```
void sfunc(row, col, x0, xn, t, color, xor)
```

本函数要调用读写象点函数 POINT，在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

本函数还要调用画直线函数 sline，参看 19.1 节。

另外，本函数还要调用计算函数 $f(x)$ 值的函数 fun，由用户自编，其形式为

```
double fun(x)
double x;
{ double y;
  y = f(x) 的表达式;
  return(y);
}
```

四、形参说明

row, col——均为整型变量。原点在屏幕上的行、列号。

x0, xn——均为整型变量。曲线起点与终点相对于原点的横坐标。

t——整型变量。曲线绕原点旋转的角度(单位为度)。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor=0 时，表示以直接方式画曲线；当 xor≠0 时，表示以异或方式画曲线。

五、函数程序(文件名：sfunc.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下，分别用颜色值 1~4 画 $y = 50\sin(x/8)$ 绕原点旋转 $0^\circ, 90^\circ, 180^\circ, 270^\circ$ 的曲线(用异或方式)。其中原点在屏幕上的行、列坐标为(170, 320)，起点与终点相对于原点的横坐标分别为 0 与 200。

主函数程序以及计算 $f(x)$ 的函数程序(文件名：sfunc0.c)如下：

```
# define POINT rtv62
# include "rtv62.c"
# include "sline.c"
# include "rmode.c"
# include "sfunc.c"

main()
{ int i;
  rmode(0x62);
  for (i=0; i<=3; i++)
    sfunc(170, 320, 0, 200, i*90, i+1, 1);

# include "math.h"
double fun(x)
double x;
{ double y;
  y = 50.0 * sin(0.125 * x);
  return(y);
}
```

19.22 矩形域三维图形透视图

一、功能

根据指定的视点画定义在给定矩形区域上的三维图形透视图。

二、方法说明

设二元函数(即曲面方程) $z = f(x, y)$ 为单值函数，且定义在矩形区域 $[x_0, x_n; y_0, y_n]$ 上，视点为 (x_p, y_p, z_p) 。定义投影平面方向为

$$\vec{B} = b_1 \vec{i} + b_2 \vec{j} + b_3 \vec{k}$$

其中

$$b_1 = x_p - \frac{x_0 + x_n}{2}, b_2 = y_p - \frac{y_0 + y_n}{2}, b_3 = z_p$$

首先将曲面上的点 (x, y, z) 投影到 YOZ 平面上，有

$$u = \frac{yx_p - xy_p}{x_p - x}, v = \frac{zx_p - xz_p}{x_p - x}$$

然后再将 YOZ 平面上的点 (u, v) 投影到平面 B 上，有

$$u_1 = u(\cos^2 \theta + \sin^2 \theta + \cos \alpha) + v \sin \theta \cos \theta (1 - |\cos \alpha|) \\ v_1 = u \sin \theta \cos \theta (1 - |\cos \alpha|) + v (\sin^2 \theta + \cos^2 \theta + \cos \alpha)$$

其中

$$\cos \alpha = \frac{b_1}{\sqrt{b_1^2 + b_2^2 + b_3^2}}, \cos \theta = \frac{b_3}{\sqrt{b_3^2 + b_2^2}}, \sin \theta = \frac{-b_2}{\sqrt{b_3^2 + b_2^2}}$$

三、函数语句

```
void spspl(row, col, x0, xn, y0, yn, xp, yp, zp, dx, dy, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

本函数还要调用画直线的函数 sline, 参看 19.1 节。

另外, 本函数还要调用计算二元函数 $z = f(x, y)$ 值的函数 persplf, 由用户自编, 其形式为

```
double persplf(x, y)
double x, y;
| double z;
z = f(x, y) 的表达式;
return(z);
|
```

四、形参说明

row, col——均为整型变量。投影平面原点在屏幕上的行、列号。

x0, xn, y0, yn——均为整型变量。定义了一个矩形区域 $[x0: xn, y0: yn]$ (相对于原点)。

xp, yp, zp——均为整型变量。视点相对于原点的三维坐标。要求 $xp > xn$ 。

dx, dy——均为整型变量。 X, Y 方向上分割的增量。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式画透视图; 当 $xor \neq 0$ 时, 表示以异或方式画透视图。

五、函数程序(文件名: spsp 1.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下, 用颜色值 2, 以异或方式画双曲抛物面

$$z = -\frac{x^2}{10^2} + \frac{y^2}{15^2}$$

的透视图。其中原点在屏幕上的行、列坐标为(100, 150), 矩形区域相对于原点为 $[-70: 70, -100: 100]$, 视点的三维坐标为(850, 650, 300)(相对于原点), X, Y 方向上的分割增量分别为 $dx = 5$ 与 $dy = 5$ 。

主函数程序以及计算 $z = f(x, y)$ 值的函数程序(文件名: spsp 10.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "spspl.c"
#include "rmode.c"
main()
```

```
| rmode(0x62);
spspl(100, 150, -70, 70, -100, 100, 850,
650, 300, 5, 5, 2, 1);

double persplf(x, y)
double x, y;
| double z, a, b;
a = 10.0; b = 15.0;
z = (-x * x / (a * a)) + (y * y) / (b * b);
return(z);
|
```

19.23 矩形域三维图形平行投影

一、功能

根据给定的视线方向画定义在给定矩形区域上的三维图形的平行投影图。

二、方法说明

设二元函数(即曲面方程) $z = f(x, y)$ 为单值函数, 且定义在矩形区域 $[x_0, x_n; y_0, y_n]$ 上, 视线方向为

$$\vec{p} = x_p \vec{i} + y_p \vec{j} + z_p \vec{k}$$

投影平面方向为

$$\vec{B} = b_1 \vec{i} + b_2 \vec{j} + b_3 \vec{k}$$

其中 $b_1 = -x_p, b_2 = -y_p, b_3 = -z_p$

首先将曲面上的点 (x, y, z) 投影到 YOZ 平面上, 有

$$u = y - \frac{xy_p}{x_p}, v = z - \frac{xz_p}{x_p}$$

然后再将 YOZ 平面上的点 (u, v) 投影到平面 B 上, 有

$$u_1 = u(\cos^2 \theta + \sin^2 \theta | \cos \alpha |) + v \sin \theta \cos \theta (1 - |\cos \alpha|)$$

$$v_1 = u \sin \theta \cos \theta (1 - |\cos \alpha|) + v (\sin^2 \theta + \cos^2 \theta | \cos \alpha |)$$

其中

$$\cos \alpha = \frac{b_1}{\sqrt{b_1^2 + b_2^2 + b_3^2}}, \cos \theta = \frac{b_3}{\sqrt{b_3^2 + b_2^2}}, \sin \theta = \frac{-b_2}{\sqrt{b_3^2 + b_2^2}}$$

三、函数语句

```
void spsp2(row, col, x0, xn, y0, yn, xp, yp, zp, dx, dy, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

本函数还要调用画直线的函数 sline, 参看 19.1 节。

另外,本函数还要调用计算二元函数 $z = f(x, y)$ 值的函数 persp2f,由用户自编,其形式为

```
double persp2f(x, y)
double x, y;
| double z;
z = f(x, y)的表达式;
return(z);
{
```

四、形参说明

row, col——均为整型变量。投影平面原点在屏幕上的行、列号。

x0, xn, y0, yn——均为整型变量。表示矩形区域 $[x0: xn, y0: yn]$ (相对于原点)。

xp, yp, zp——均为整型变量。视线方向的三个分量。

dx, dy——均为整型变量。 X 方向与 Y 方向上分割的增量。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时,表示以直接方式画平行投影图;当 $xor \neq 0$ 时,表示以异或方式画平行投影图。

五、函数语句(文件名: spsp 2.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下,用颜色值 2、以异或方式画双曲抛物面

$$z = -\frac{x^2}{10^2} + \frac{y^2}{15^2}$$

的平行投影。其中原点在屏幕上的行、列坐标为 (100, 150), 矩形区域为 $[-70: 70, -100: 100]$ (相对于原点), 视线方向为

$$\vec{p} = -10\vec{i} - 8\vec{j} + 4\vec{k}$$

X 与 Y 方向上分割的增量分别为 $dx = 5$ 与 $dy = 5$ 。

主函数程序以及计算 $z = f(x, y)$ 值的函数程序(文件名: spsp 20.c)如下:

```
#define POINT rtv62
#include "rtv62.c"
#include "sline.c"
#include "spsp2.c"
#include "rmode.c"
main()
| rmode(0x62);
    spsp2(100, 150, -70, 70, -100, 100, -10, -8, -4, 5, 5, 2, 1);
}

double persp2f(x, y)
double x, y;
| double z, a, b;
a = 10.0; b = 15.0;
```

```
z = (-x * x / (a * a)) + (y * y) / (b * b);
return(z);
{
```

19.24 圆形域三维图形平行投影

一、功能

根据给定的视线方向画定义在给定圆形区域上的三维图形的平行投影图。

二、方法说明

同 19.23 节。

三、函数语句

```
void spsp3(row, col, x0, y0, r, xp, yp, zp, dx, dy, color, xor)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

本函数还要调用画直线的函数 sline, 参看 19.1 节。

另外,本函数还要调用计算二元函数 $z = f(x, y)$ 值的函数 persp3f, 由用户自编,其形式为

```
double persp3f(x, y)
double x, y;
| double z;
z = f(x, y)的表达式;
return(z);
{
```

四、形参说明

row, col——均为整型变量。投影平面原点在屏幕上的行、列号。

x0, y0——均为整型变量。圆心区域中心关于原点的坐标。

r——整型变量。圆心区域的半径。

xp, yp, zp——均为整型变量。视线方向的三个分量。

dx, dy——均为整型变量。 X 、 Y 方向上分割的增量。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时,表示以直接方式画平行投影;当 $xor \neq 0$ 时,表示以异或方式画平行投影。

五、函数程序(文件名: spsp 3.c)

六、例

在 TVGA 图形模式 62H(参看 18.10 节)下,用颜色值 2、以异或方式画曲面

$$z = \sqrt{72^2 - (x - 5)^2 - (y - 5)^2}$$

的平行投影。其中投影平面原点在屏幕上的行、列坐标为(100, 150), 圆形区域圆心相对于原点的坐标为(5, 5), 半径为 70, 视线方向为

$$\vec{p} = -10\vec{i} - 8\vec{j} - 4\vec{k}$$

X, Y 方向上分割的增量分别为 $dx = 5$ 与 $dy = 5$ 。

主函数程序以及计算 $z = f(x, y)$ 值的函数程序(文件名: spsp 30.c)如下:

```
# define POINT rtv62
# include "rtv62.c"
# include "sline.c"
# include "spsp3.c"
# include "rmode.c"

main()
{
    rmode(0x62);
    spsp3(100, 150, 5, 70, -10, -8, -4, 5, 5, 2, 1);
}

# include "math.h"
double persp3f(x, y)
double x, y;
double z;
z = sqrt(72.0 * 72.0 - (x - 5.0) * (x - 5.0)
        - (y - 5.0) * (y - 5.0));
return(z);
}
```

第 20 章 汉字操作

20.1 小汉字库的建立

一、功能

新建用户需要的小汉字库。

二、方法说明

在有些具有汉字显示功能的实际系统中,往往只用到很少一部分汉字。在这种情况下,为了节省存储空间,提高汉字的显示速度,没有必要将全部汉字库装入系统,而只需将有用的汉字新建成一个小汉字库。

目前,常用的汉字库有 16×16 点阵、 24×24 点阵、 32×32 点阵等。

根据汉字库中各汉字点阵的信息结构划分,可以分为以行为主及以列为主两种形式。

所谓以行为主,是指在汉字库中,每个汉字的点阵信息以行为顺序连续存放(即一行接一行地顺序存放),一行上连续的 8 个点阵信息放在一个字节内,且左边的点对应字节中的高位。在这种结构下,对于 16×16 点阵的一个汉字,一行有 16 个点,分别存放在 2 个字节内,共有 16 行,因此共占 32 个字节;对于 24×24 点阵的一个汉字,一行有 24 个点,分别存放在 3 个字节内,共有 24 行,因此共占 72 字节;对于 32×32 点阵的一个汉字,一行中的 32 个点分别用 4 个字节存放,32 行点阵信息共占 128 个字节。对于其它规格的汉字点阵信息的存放,可以此类推。

所谓以列为主,是指在汉字库中,每个汉字的点阵信息是以列为顺序连续存放的,即一列接一列地顺序存放,在一列上连续的 8 个点阵信息放在一个字节内,且上边的点对应字节中的高位。在这种结构下,对于 16×16 点阵的一个汉字,一列有 16 个点,分别存放在 2 个字节内,共有 16 列,因此共占 32 个字节;对于 24×24 点阵的一个汉字,一列中的 24 个点阵信息分别用 3 个字节存放,24 列点阵信息共占 72 个字节。对于 32×32 点阵及其它规格的汉字点阵信息的存放,可以此类推。

在本函数中,不管原来汉字库中的汉字点阵存放的顺序如何,新建成的小汉字库中的汉字点阵信息总是以行为主连续存放的。并且,在新建成的小汉字库中,每个汉字的代码即为该汉字点阵信息在小汉字库中的位置序号。

另外,在汉字库中,区位码从第 7 行到第 12 行之间的点阵信息可以由用户定义,因此,在一般的汉字库中,这个区域为空白。目前,有的汉字库中不留空白区的位置,这就导致在这个空白区中的区位码发生跳跃,在根据区位码取汉字库中各汉字点阵信息时要考虑到这个问题。在本函数中也对此作了处理。

因此,在使用本函数建立小汉字库时,用户应该要先弄清楚你所具有的原汉字库的点阵规格以及汉字库的具体结构。

本函数采用的方法是:根据给定的汉字区位码,从原汉字库中取出汉字点阵信息,然后

依次以行为主存放在新建的小汉字库中。

其中区位码为 $code$ 的汉字点阵信息在原汉字库中的序号 m 的计算分以下两种情况：

(1) 原汉字库中保留了第 7 行到第 12 行之间的空白区

$$m = ((code/100) - 1) * 94 + (code \% 100) - 1$$

(2) 原汉字库中不保留第 7 行到第 12 行之间的空白区

当 $(code/100) < 7$ 时

$$m = ((code/100) - 1) * 94 + (code \% 100) - 1$$

当 $(code/100) \geq 7$ 时

$$m = ((code/100) - 7) * 94 + (code \% 100) - 1$$

汉字点阵信息的第一个字节在原汉字库中的偏移量为

$$k = m * kk$$

其中 kk 为一个汉字点阵信息所占的字节数，它由汉字库中汉字点阵规格决定。

三、函数语句

```
void thzlb(num, flag, n, fn1, fn2, fn3)
```

四、形参说明

num ——整型变量。原汉字库中汉字点阵的规格。当 $num = 1$ 时，表示 16×16 点阵；

当 $num = 2$ 时，表示 24×24 点阵；当 $num = 3$ 时，表示 32×32 点阵。其它情况以此类推。

$flag$ ——整型变量。原汉字库的结构标志，其中

$flag =$	<p>1, 以行为主存放, 空白区不保留 2, 以行为主存放, 空白区保留 3, 以列为主存放, 空白区不保留 4, 以列为主存放, 空白区保留</p>
----------	--

n ——整型变量。放大标志。当 $n = 0$ 时，表示新建小汉字库中的点阵规格与原汉字库相同(即不放大)；当 $n \neq 0$ 时，表示新建小汉字库中的汉字点阵比原汉字库放大一倍，如 16×16 点阵放大成 32×32 点阵， 24×24 点阵放大成 48×48 点阵，即：如果原汉字库的点阵规格为 num ，则经放大一倍后，新建的小汉字库的点阵规格变为 $2 * num + 1$ 。

$fn1$ ——字符串指针。指向原汉字库的文件名。

$fn2$ ——字符串指针。指向新建小汉字库的文件名。

$fn3$ ——字符串指针。指向由各汉字区位码组成的文本文件名。在此文件中，存放用户所需要的部分汉字的区位码，各区位码之间用空格分隔(参看本节例)。汉字区位码在此文件中的序号(第一个的序号为 0)即为该汉字在新建小汉字库中的代码。

五、函数程序(文件名: thzlb.c)

六、例

设需要新建小汉字库中的 42 个汉字如下：

数 据 结 构 入 门 菜 单 选

择 操 作 说 明 键 空 白 继

续 光 1. 2. 3. 4. 5. 6. 7.

8. 9. 10. 11. 12. 13. 14. 一 二

三 四 五 六 七 八

以上 42 个汉字区位码组成的文本文件名为 code.txt，文件内容如下：

4293	3061	2965	2525	4075	3537	1843	2105	4901	5281
1857	5587	4321	3587	2892	3153	1655	2844	4888	2566
217	218	219	220	221	222	223	224	225	226
227	228	229	230	5027	2294	4093	4336	4669	3389
3863	1643								

(1) 使用 16×16 点阵(即 $num = 1$)的汉字库 cclib16 新建 32×32 点阵的小汉字库 cclib(即 $n \neq 0$)。其中原汉字库 cclib16 中的点阵信息是以行为主存放的，且没有保留第 7 行到第 12 行之间的空白区(即 $flag = 1$)。然后在 TVGA 图形模式 5FH(参看 18.9 节)下显示小汉字库中的所有汉字(五个为一行)。

本主函数要调用单个汉字显示的函数 thzpl，参看 20.3 节。

主函数程序(文件名: thzlb 0.c)如下：

```
#define POINT rtv5f
#include "stdio.h"
#include "rtv5f.c"
#include "thzlb.c"
#include "rmode.c"
#include "thzpl.c"

main()
{
    FILE *fp;
    int i, row, col;
    thzlb(1, 1, 1, "cclib16", "cclib", "code.txt");
    if ((fp = fopen("cclib", "r+b")) == NULL)
        printf("cannot open cclib! \n");
    exit(0);
}

row = 0;
rmode(0x5f);
for (i = 0; i < 41; i++)
    if ((i % 5) == 0)
        row = row + 50; col = 80;
    thzpl(3, row, col, i, 4, 1, fp);
    col = col + 40;
}

fclose(fp);
```

(2) 使用 24×24 点阵(即 $num = 2$)的汉字库 cclib24 新建 48×48 点阵的小汉字库 cclib(即 $n \neq 0$)。其中原汉字库 cclib24 中的点阵信息是以列为主存放的，且保留了第 7 行到第 12 行之间的空白区(即 $flag = 4$)。然后在 TVGA 图形模式 5FH(参看 18.9 节)下显示小汉字库中的所有汉字(五个为一行)。

本主函数要调用单个汉字显示的函数 thzpl, 参看 20.3 节。

主函数程序(文件名:thzlb 1.c)如下:

```
#define POINT rtv5f
#include "stdio.h"
#include "rtv5f.c"
#include "thzlb.c"
#include "thzpl.c"
#include "rmode.c"
main()
{
FILE *fp;
int i, row, col;
thzlb(2, 4, 1, "cclib24", "cclibb", "code.txt");
if ((fp = fopen("cclibb", "r+b")) == NULL)
printf("cannot open ccilib! \n");
exit(0);
}
row = 0;
rmode(0x5f);
for (i = 0; i <= 41; i++)
{
if ((i % 5) == 0)
row = row + 50; col = 80;
thzpl(5, row, col, i, 4, 1, fp);
col = col + 40;
}
fclose(fp);
}
```

20.2 ASCII 码字符图形库的建立

一、功能

建立 ASCII 码字符 8×16 点阵(即 16 行 8 列)的图形库。

二、方法说明

在一般汉字库中都有 ASCII 码字符的点阵信息, 但这种字符按汉字显示时, 其大小与汉字相同。在有些应用中, 希望夹在汉字中间的 ASCII 码字符规格与字符模式下显示的相同。例如, 如果汉字为 16×16 点阵, 则 ASCII 码字符用 8×16 点阵(其中上面 8 行也为空白)。本函数就是为这种需要而设置。

本函数采用的方法如下:

在 EGA 图形模式 10H(即 16)下, 首先以字符形式在屏幕上显示所有 ASCII 码字符(代码从 0 到 127, 在函数中实际是从 0 到 130), 其中代码小于 32 的字符均用空格(代码为 32)代替, 代码大于 126 的字符也均用空格代替; 然后从显示内存中依次提取这些字符的 8×16 点阵信息。

三、函数语句

```
void tasci(fn)
```

四、形参说明

fn——字符串指针。指向需要建立的 ASCII 码字符图形库的文件名。

五、函数程序(文件名:tasci.c)

六、例

建立 ASCII 码字符图形库 asci。

主函数程序(文件名:tasci 0.c)如下:

```
#include "tasci.c"
main()
{
tasci("asci");
}
```

20.3 一个汉字的显示

一、功能

利用新建的小汉字库在屏幕上指定位置显示指定代码的一个汉字。

二、方法说明

根据指定的汉字代码从新建的汉字库中取出汉字点阵信息, 然后将这些点阵信息逐点显示在屏幕的指定位置。

三、函数语句

```
void thzpl(num, row, col, code, color, xor, fp)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某个读写象点函数进行宏定义。

四、形参说明

num——整型变量。新建成的小汉字库的点阵规格。当 num = 1 时, 为 16×16 点阵; num = 2 时为 24×24 点阵; num = 3 时为 32×32 点阵; 其它以此类推。

row, col——均为整型变量。汉字左上角在屏幕上的行、列号。

code——整型变量。汉字代码(即汉字点阵信息在新建成的小汉字库中的序号)。

color——整型变量。颜色值。

xor——整型变量。异或标志。若 xor = 0, 则表示以直接方式显示汉字; 若 xor ≠ 0, 则表示以异或方式显示汉字。

fp——文件指针。指向新建的小汉字库文件。必须注意，小汉字库应是由 20.1 节中的函数所建立。

五、函数程序(文件名:thzpl.c)

六、例

见 20.1 节中的例。

20.4 一行汉字的显示

一、功能

在屏幕的指定位置显示指定小汉字库中的一行汉字。

二、方法说明

一行汉字的代码(在小汉字库中的序号)依次存放在一个一维数组中。

三、函数语句

```
void thzrw(num, row, col, color, xor, q, n, fp)
```

本函数要调用显示一个汉字的函数 thzpl, 参看 20.3 节。

四、形参说明

num——整型变量。汉字点阵规格。当 $num = 1$ 时为 16×16 点阵; $num = 2$ 时为 24×24 点阵; $num = 3$ 时为 32×32 点阵; 其它情况以此类推。

row, col——均为整型变量。行首汉字左上角在屏幕上的行、列号。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 $xor = 0$ 时, 表示以直接方式显示汉字; 当 $xor \neq 0$ 时, 表示以异或方式显示汉字。

q——整型一维数组, 长度为 n 。存放一行上各汉字在小汉字库中的代码(即序号)。当发现某个代码为负值时, 显示结束, 或者到最后一个汉字代码为止。

n——整型变量。一行汉字的个数。

fp——文件指针。指向指定的小汉字库文件。该小汉字库必须是用 20.1 节中的方法新建的。

五、函数程序(文件名:thzrw.c)

六、例

在 TVGA 图形模式 5FH(参看 18.9 节)下, 分别用颜色值 1~10 重复五次显示两行汉字, 这两行汉字分别取自于 20.1 节例(1)与(2)中新建的小汉字库 cclib 与 cclibb。其中每一个代码所代表的汉字参看 20.1 节中的例。

主函数程序(文件名:thzrw 0.c)如下:

```
# define POINT rtv5f
# include "rtv5f.c"
# include "rmode.c"
# include "thzpl.c"
# include "thzrw.c"
# include "stdio.h"
main()
{
    static int p1[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    static int p2[10] = {10, 11, 12, 13, 14, 15, 16, -1};
    int i, row, col, color;
    FILE * fp1, * fp2;
    if ((fp1 = fopen("cclib", "r+b")) == NULL)
        printf("cannot open cclib! \n");
    exit(0);
    if ((fp2 = fopen("cclibb", "r+b")) == NULL)
        printf("cannot open cclibb! \n");
    exit(0);
    rmode(0x5f);
    row = 80; col = 100; color = 1;
    for (i = 0; i <= 4; i++)
        thzrw(3, row, col, color, i, p1, 10, fp1);
    row = row + 60; color = color + 1;
    thzrw(5, row, col, color, i, p2, 10, fp2);
    row = row + 60; color = color + 1;
}
```

20.5 ASCII 码字符串按图形显示

一、功能

利用 ASCII 码字符图形库, 按图形显示 ASCII 码字符串。

二、方法说明

依次从给定的 ASCII 码字符串中取出各字符的 ASCII 码, 然后从 ASCII 码字符图形库中取出相应的点阵信息, 按给定的放大倍数放大后进行显示。

三、函数语句

```
void taspl(num, row, col, color, xor, q, n, fp)
```

本函数要调用读写象点函数 POINT, 在调用之前要用第 18 章中的某一个读写象点函数进行宏定义。

四、形参说明

num——整型变量。显示字符串的放大倍数。在由 20.2 节中的函数建立的 ASCII 码

· 464 ·

字符图形库中,其点阵规格为 8×16 ,经放大后,实际显示的每一个字符的点阵规格为 $8(n+1) \times 16(n+1)$ 。

row,col——均为整型变量。字符串中第一个字符左上角在屏幕上的行、列号。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor=0 时,表示以直接方式显示字符串;当 xor $\neq 0$ 时,表示以异或方式显示字符串。

q——字符型一维数组,长度为 n。依次存放 ASCII 码字符串的各字符(实际是字符的 ASCII 码)。

n——整型变量。给定字符串的长度。

fp——文件指针。指向 ASCII 码字符图形库文件。

五、函数程序(文件名:taspl.c)

六、例

在 TVGA 图形模式 5FH(参看 18.9 节)下,分别用颜色值 1~10 重复五次显示两行字符串,它们分别为“0123456789”与“abcdeABCDE”。其中点阵信息取自于由 20.2 节中的例所建立的 ASCII 码字符图形库 ascii。

主函数程序(文件名:taspl 0.c)如下:

```
#define POINT rtv5f
#include "rtv5f.c"
#include "rmode.c"
#include "taspl.c"
#include "stdio.h"
main()
{
    static char p1[10] = {'0', '1', '2', '3', '4',
                          '5', '6', '7', '8', '9'};
    static char p2[10] = {'a', 'b', 'c', 'd', 'e',
                          'A', 'B', 'C', 'D', 'E'};
    int i, row, col, color;
    FILE * fp;
    if ((fp = fopen("asci", "r+b")) == NULL)
        printf("cannot open ascii! \n");
        exit(0);
    rmode(0x5f);
    row = 80; col = 100; color = 1;
    for (i = 0; i <= 4; i++)
        taspl(i, row, col, color, 1, p1, 10, fp);
        row = row + 60; color = color + 1;
        taspl(i, row, col, color, 1, p2, 10, fp);
        row = row + 60; color = color + 1;
    }
fclose(fp);
```

20.6 汉字菜单的显示

一、功能

在屏幕上显示一个汉字菜单。

二、方法说明

设菜单中共有 m 个条目,每个条目中最多有 n 个汉字,则用一个二维整型数组 q(m 行 n 列)存放菜单中各汉字在小汉字库(由 20.1 节中的函数建成)中的代码。其中 q(i,j)(j=0,1,...,n-1)依次存放第 i 个条目中的汉字代码,当某个条目中的汉字个数不足 n 时,则用一个负值终止。

汉字菜单中的移动光标设置在每一条目的第一个汉字上。

三、函数语句

```
int tmenu(num, row, col, color, xor, q, m, n, item, fp)
```

本函数返回一个整型值,以指示菜单中的当前光标位置。若返回值为 i,则表示当前光标在菜单中的第 i 个条目上(即在第 i 个条目中的第一个汉字上)。

本函数要调用显示一行汉字的函数 thzrw,参看 20.4 节。

本函数还要调用矩形填充函数 ssbox(表示光标),参看 19.7 节。

四、形参说明

num——整型变量。汉字点阵规格。

num = $\begin{cases} 1, & \text{表示 } 16 \times 16 \text{ 点阵} \\ 2, & \text{表示 } 24 \times 24 \text{ 点阵} \\ 3, & \text{表示 } 32 \times 32 \text{ 点阵} \\ 4, & \text{表示 } 40 \times 40 \text{ 点阵} \\ \dots & \dots \end{cases}$

row, col——均为整型变量。汉字菜单左上角在屏幕上的行、列号。

color——整型变量。颜色值。

xor——整型变量。异或标志。当 xor=0 时,表示以直接方式显示菜单;当 xor $\neq 0$ 时,表示以异或方式显示菜单。

q——整型二维数组,体积为 m \times n。其中 q(i,j)(j=0,1,...,n-1)依次存放第 i 个菜单条目中的汉字在小汉字库中的代码。当某个条目不足 n 个汉字时,以负值终止。

m——整型变量。菜单条目的个数。

n——整型变量。每个菜单条目中最多的汉字个数。

item——整型变量。菜单中的光标初始位置,即光标在第 item 个条目中的第一个汉字上。

fp——文件指针。指向小汉字库文件。

五、函数程序(文件名:tmenu.c)

六、例

参看 20.7 节中的例。

20.7 汉字菜单的选择

一、功能

在汉字菜单中利用移动光标选择某菜单条目。

二、方法说明

利用键盘上的上、下移动键(即↑与↓)移动菜单中的光标。当选中某菜单条目时,按回车键,函数返回该菜单条目的序号。必须注意,如果按其它键,本函数将不予理睬。

三、函数语句

```
int tslect(num, row, col, color, m, item)
```

本函数返回一个整型值,指示被选中的菜单条目的序号。

本函数要调用矩形填充函数 ssbox(表示光标),参看 19.7 节。

四、形参说明

num——整型变量。汉字点阵规格。

num =	1, 16×16 点阵 2, 24×24 点阵 3, 32×32 点阵 4, 40×40 点阵 : ;
-------	--

row, col——均为整型变量。汉字菜单左上角在屏幕上的行、列号。

color——整型变量。颜色值。

m——整型变量。菜单条目个数。

item——整型变量。光标的初始位置。

五、函数程序(文件名:tslct.c)

六、例

在 TVGA 图形模式 5FH(参看 18.9 节)下,用颜色值 4 显示如下菜单:

1. 数据结构入门
2. 菜单选择
3. 操作说明

4. 键空白
5. 继续
6. 光

其中菜单中的汉字均取自 20.1 节例中建成的小汉字库 cclib。

然后进行菜单选择。当按回车键后,菜单将消失,再按回车键又将显示菜单,又可以进行选择。当选中第 6 项后按回车键,则将退出主函数。

主函数程序(文件名:tslct.c)如下:

```
#define POINT rtv5f
#include "dos.h"
#include "stdio.h"
#include "rmode.c"
#include "rtv5f.c"
#include "thzpl.c"
#include "thzrw.c"
#include "tmenu.c"
#include "ssbox.c"
#include "tslct.c"

main()
{
  int item, flag, color = 4;
  FILE * fp;
  static int a[6][10] =
  {
    {20, 0, 1, 2, 3, 4, 5, -1},
    {21, 6, 7, 8, 9, -1},
    {22, 10, 11, 12, 13, -1},
    {23, 14, 15, 16, -1},
    {24, 17, 18, -1},
    {25, 19, -1}
  };
  if ((fp = fopen("cclib", "r+b")) == NULL)
    printf("cannot open cclib! \n");
  exit(0);
}

rmode(0x5f);
item = 1; flag = 1;
item = tmenu(3, 100, 100, color, 1, a, 6, 10, item, fp);
while (flag == 1)
{
  item = tslct(3, 100, 100, color, 6, item);
  switch(item)
  {
    case 1: break;
    case 2: break;
    case 3: break;
    case 4: break;
    case 5: break;
    case 6: flag = 0; break;
  }
}
```

参 考 文 献



1. 徐士良.计算机常用算法.第二版.清华大学出版社,1995
2. 徐士良,朱明方.软件应用技术基础.清华大学出版社,1994
3. 徐士良.FORTRAN 常用算法程序集.第二版.清华大学出版社,1995
4. 王璞等译.数值方法大全——科学计算的艺术.兰州大学出版社,1991
5. 李庆扬等.数值分析.华中理工大学出版社,1986
6. 卢有杰等.C 语言高级程序设计.清华大学出版社,1991
7. 邹海.最优设计中的新算法.新时代出版社,1982
8. 王鹰翔等.C 高级程序设计.宇航出版社,1992
9. 尹彦芝.C 语言常用算法与子程序.清华大学出版社,1991
10. 徐士良.PC 机 C 图形编程手册.清华大学出版社,1994