

# 目 录

<b>第1章 引论</b>	1
1-1 本书的任务	1
1-2 数值计算中的一些基本概念	2
1. 误差概念	2
2. 算法与数值稳定性	5
3. 条件问题与病态概念	8
4. 实际应用中的注意事项	10
1-3 程序设计语言概述	13
1. 从电子计算机谈起	13
2. 程序与程序设计语言	15
3. 软件	16
4. 计算机系统的使用方式	17
5. 计算机中数的表示	17
1-4 数学补充材料	19
1. 大O记号	19
2. 内积·向量和矩阵的范数	20
习题一	23
<b>第2章 FORTRAN 语言的基本概念与基本语句</b>	27
2-1 FORTRAN 导引	27
2-2 FORTRAN 的一些基本概念	32
1. FORTRAN 字符集	32
2. 数据类型·常数与变量	32
3. 算术表达式与标准函数	34
4. 数组与下标变量	36
2-3 赋值语句·停语句与暂停语句	38
1. 算术赋值语句	38
2. 停语句 STOP 与暂停语句 PAUSE	40
2-4 输入/输出初步	41
1. 概述	41
2. 带格式输出·写语句与格式语句	42
3. 格式说明符·字段描述符	43
4. 带格式输入·读语句	46
2-5 控制转移语句	47
1. 无条件 GO TO 语句·算术 IF 语句	47
2. 关系表达式与逻辑表达式·逻辑 IF 语句	49
3. 计算 GO TO 语句·赋标号语句与赋标号 GO TO 语句	51

2-6 循环语句与继续语句 .....	53
1. 继续语句 .....	53
2. 循环语句 (DO 语句) 与循环程序设计例 .....	58
3. 隐 DO 循环的输入/输出与循环程序设计例 (续) .....	66
习题二 .....	71
附录 A FORTRAN IV 标准函数表 .....	78
<b>第 3 章 FORTRAN 语言的较复杂部分</b> .....	<b>81</b>
3-1 FORTRAN 的基本概念 (续) .....	81
1. 双精度型常数与变量 .....	81
2. 复型常数与变量 .....	82
3. 逻辑型常数与变量 · 逻辑赋值语句 .....	85
4. 字符型常数 .....	87
3-2 函数与子程序 .....	89
1. 语句函数及其引用 .....	89
2. 函数子程序及其调用 .....	90
3. 子例程子程序与 CALL 语句 .....	94
4. 可调数组 .....	97
5. 外部语句 EXTERNAL .....	99
3-3 程序块间的数据交换 .....	101
1. 等价语句 EQUIVALENCE .....	101
2. 公用语句 COMMON .....	103
3. 数据初值语句 DATA 和数据块子程序 .....	105
3-4 输入/输出综述与补充 .....	107
1. 文件与记录的概念 .....	107
2. 字段描述符 (补充) · 比例因子 P .....	108
3. 字段分隔符 · 走纸控制 .....	109
4. 格式数组 .....	111
5. 输入/输出表 · 带格式输入/输出语句综述 .....	112
6. 无格式输入/输出语句 · 辅助输入/输出语句 .....	113
3-5 程序实例与一些常用子程序 .....	115
1. 控制系统频率相关函数计算例 .....	115
2. 一组简单多项式计算子程序 .....	117
3. 打印曲线子程序 .....	122
习题三 .....	126
<b>第 4 章 从 FORTRAN IV 到 FORTRAN 77</b> .....	<b>132</b>
4-1 FORTRAN 77 的目标 .....	132
4-2 源程序格式与基本概念的扩充 .....	133
1. 源程序书写格式的新规定 .....	133
2. 字符集与字符数据类型 .....	134
3. 数组与数组元素 .....	136
4. 表达式 .....	137

4-3 新增加和修改的几个语句 .....	139
1. PARAMETER 语句(参数语句) .....	139
2. IMPLICIT 语句(隐含类型说明语句) .....	140
3. IF-THEN-ELSE 结构 .....	140
4. DO 语句的改动 .....	149
5. DATA 语句功能的增加 .....	150
4-4 输入/输出功能的扩充 .....	151
1. 记录、文件与部件 .....	152
2. 带格式输入/输出语句 .....	154
3. 表控输入/输出语句 .....	157
4. 新增加的格式说明符 .....	160
5. 辅助输入/输出语句 .....	161
4-5 过程·函数与子程序 .....	165
1. 内部函数 .....	166
2. 内部函数说明语句 (INTRINSIC) 与外部过程说明语句 (EXTERNAL) .....	166
3. SAVE 语句、ENTRY 语句和选择 RETURN 语句 .....	167
4-6 FORTRAN 77 程序例 .....	171
1. 含双精度、逻辑、字符和复数功能的程序例 .....	171
2. 大批数据处理与数据文件的使用 .....	175
习题四 .....	179
附录 A FORTRAN 77 内部函数表 .....	185
附录 B 程序单位中注解行与语句的次序 .....	190
附录 C FORTRAN 77 语句一览表 .....	190
附录 D FORTRAN 77 排序序列 .....	192
<b>第 5 章 插值与拟合</b> .....	194
5-1 引言 .....	194
5-2 多项式与分段多项式插值 .....	194
1. 拉格朗日插值 .....	195
2. 埃尔米特插值 .....	200
3. 插值过程的稳定性分析 .....	203
4. 几种分段多项式插值公式 .....	204
5. 一元三点不等距成组插值的 FORTRAN 程序 .....	207
5-3 样条函数与三次样条插值 .....	210
1. 样条函数概念 .....	210
2. 三次样条插值 .....	211
5-4 差分与均差·牛顿插值公式 .....	216
1. 差分概念 .....	216
2. 等距节点插值公式的差分形式 .....	218
3. 均差概念与牛顿基本插值多项式 .....	220
5-5 曲线拟合的最小二乘法 .....	223
1. 线性最小二乘拟合原理 .....	223

2. 多项式曲线拟合与指数曲线拟合 .....	225
3. 正交多项式曲线拟合及其 FORTRAN 程序 .....	227
习题五 .....	233
<b>第 6 章 数值积分与数值微分</b> .....	238
6-1 引言 .....	238
6-2 梯形求积公式与辛普生求积公式 .....	240
1. 梯形求积公式 .....	240
2. 辛普生求积公式 .....	241
3. 自动选步长梯形求积 .....	242
4. 自动选步长辛普生求积及其 FORTRAN 程序 .....	244
6-3 龙贝格积分法及其 FORTRAN 程序 .....	246
6-4 正交多项式与高斯型求积公式 .....	252
1. 正交多项式 .....	252
2. 高斯型求积公式 .....	256
3. 高斯-勒让德求积公式的 FORTRAN 程序 .....	260
6-5 多重积分与广义积分计算 .....	263
1. 求多重积分的高斯法及其 FORTRAN 程序 .....	263
2. 无穷区间上的广义积分计算 .....	267
3. 无界函数的广义积分计算 .....	269
6-6 数值微分 .....	270
1. 用插值多项式求数值导数 .....	270
2. 用三次样条插值函数求数值导数 .....	272
习题六 .....	273
<b>第 7 章 方程求根与非线性方程组数值解</b> .....	276
7-1 引言 .....	276
1. 方程根的存在性 .....	276
2. 关于根的隔离问题 .....	278
3. 迭代法的一般理论 .....	282
7-2 方程求根的几个常用方法 .....	285
1. 对分法及其 FORTRAN 程序 .....	285
2. 牛顿迭代法及其 FORTRAN 程序 .....	288
3. 劈因子法及其 FORTRAN 程序 .....	293
7-3 非线性方程组数值解法 .....	299
1. 牛顿-拉夫逊方法 .....	300
2. 布罗登方法(拟牛顿法)及其 FORTRAN 程序 .....	302
3. 最速下降法及其 FORTRAN 程序 .....	306
习题七 .....	311
<b>第 8 章 线性代数方程组数值解法</b> .....	314
8-1 引言 .....	314
8-2 解线性方程组(包括求逆矩阵及行列式值)的高斯消去法 .....	315
1. 高斯消去法的基本思想 .....	315

2. 列主元高斯消去法解线性方程组的 FORTRAN 程序 .....	318
3. 高斯-约当消去法 .....	321
4. 行主元高斯-约当消去法求逆矩阵与行列式值的 FORTRAN 程序 .....	323
8-3 解线性方程组的三角分解法 .....	328
1. 矩阵三角分解原理 .....	328
2. 解线性方程组的三角分解法及其 FORTRAN 程序 .....	331
3. 解对称正定矩阵方程组的平方根法 .....	335
4. 改进平方根法及其 FORTRAN 程序 .....	337
8-4 解三对角方程组的追赶法及其 FORTRAN 程序 .....	341
8-5 解线性方程组的迭代法 .....	344
1. 雅可比迭代法 .....	344
2. 高斯-赛德尔迭代法及其 FORTRAN 程序 .....	346
3. 迭代法收敛性讨论 .....	350
8-6 方程组的条件问题 .....	353
习题八 .....	356
附录 矩阵指数 $e^{At}$ 的数值计算方法及其 FORTRAN 程序 .....	359
<b>第 9 章 常微分方程数值解法</b> .....	364
9-1 引言: 基本概念 .....	364
9-2 尤拉法与预测-校正法 .....	366
1. 尤拉法及其精度分析 .....	366
2. 改进的尤拉法与预测-校正法 .....	368
9-3 龙格-库塔法 .....	372
1. 龙格-库塔法及其 FORTRAN 程序 .....	372
2. 自动选步长的龙格-库塔法 .....	382
9-4 阿当姆斯公式与预测-校正方法 .....	382
1. 阿当姆斯显式与隐式公式 .....	382
2. 预测-校正方法及其 FORTRAN 程序 .....	384
9-5 收敛性与稳定性 .....	387
1. 收敛性 .....	388
2. 稳定性 .....	389
9-6 边值问题的数值解法 .....	392
1. 线性边值问题的差分方法 .....	393
2. 打靶法 .....	395
习题九 .....	396
<b>参考资料</b> .....	400

# 第1章 引 论

## 1-1 本书的任务

大致说来,数学模型的求解有两条途径:第一,求解析解,即解是解析表达式的形式,这是准确解;第二,如果解析解不可能或不容易求得,则求数值解,即解只是在一些离散点上取值的形式,多数情况下,数值解是近似解<sup>①</sup>。求数值解主要利用电子计算机。

我们把求数值解的问题称为数值计算问题,求解数值计算问题的方法称为**数值计算方法**,简称**数值方法**或**计算方法**。

那么,假如我们遇到数值计算问题,又有电子计算机可供我们使用的话,是不是把问题直接交给计算机,计算机就立刻把问题计算出来呢?不是的。这里有两件事情要做。第一,必须先为该数值计算问题选择合适的数值计算方法,并把它设计成可以让计算机一步步执行的所谓算法;第二,采用计算机能认识的所谓程序设计语言,把算法编码成计算机语言程序。这时,把语言程序和初始数据送入计算机,即上机计算,计算机才能(如果没有其它错误的话)把问题计算出来。这个过程可用图1-1表示。

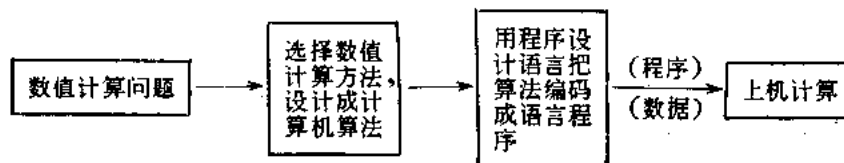


图 1-1

本书的任务是:

1) 介绍一些计算机上适用的数值计算方法,研究如何把它们设计成计算机算法,讨论算法的数值稳定性、收敛性和误差估计。

2) 介绍数值计算中广泛使用的 FORTRAN 程序设计语言,讨论如何把算法编码成 FORTRAN 语言程序,给出主要算法的通用 FORTRAN 程序。

必须说明的是,随着计算机的广泛应用,在数值计算方法不断发展的同时,人们已经为许多行之有效的数值计算方法设计好计算机算法,并编写了计算机语言程序。这些程序或者汇编成册(见 [c1] [c2] [c3] [c5]),供用户引用;或者直接库存在计算机系统中,供用户直接调用 (CALL)。因此,实际上我们并不需要对每一个数值计算方法都去研究它的算法及其程序设计。但是,另一方面,我们也不能因此而认为,连学习数值计算方法的原理,掌握程序设计的技术,也不必了。事实是,如果你没有为具体问题选择和使用数值计算方法的能力和知识,如果你不会动手编写自己需要的程序,或必要时读懂别人的程序,那么,你所能处理的问题在范围、深度和效率方面,将是极其有限的。

<sup>①</sup> 也有关于近似解析解的研究,例如见 [c9]。

本章先讨论数值计算中的一些基本概念,然后是程序设计语言的概述,最后提供一些有关的数学补充材料.

## 1-2 数值计算中的一些基本概念

### 1. 误差概念

首先讨论误差的来源.

采用数值计算方法求数值解的整个过程,随处都有可能引进误差. 所谓误差,就是一个量的真实值与其近似值之差.

为实际问题建立数学模型时,往往进行简化和抽象,因此,所建立的数学模型与实际现象之间便可能存在误差,这种误差称为**模型误差**.

数学模型中通常包含一些根据观测或实验得到的参量,这些参量的值与真实值之间也可能存在误差,这种误差称为**观测误差**.

当数学模型不能获得准确解时,通常便采用数值方法求数值解;但数值方法中通常又采用一些近似处理,例如用收敛无穷级数前有限项之和来代替无穷级数之和,即截去无穷级数的后段. 因此,采用数值方法求得的数值解与模型本来存在的准确解之间便存在着误差,这种误差称为**截断误差**(或**方法误差**).

由于计算机只能表示有限位数并对有限位数进行运算,因此,在计算机的计算过程中需要进行舍入,这种由舍入而产生的误差称为**舍入误差**.

此外,在有的计算过程中,初始数据的误差对计算结果的影响可能很大,这种由初值引起的误差称为**初值误差**.

在数值计算方法的应用中,主要考虑截断误差、舍入误差和初值误差,研究它们的产生、传播和对计算结果的影响,尽量控制它们,使得计算结果有足够的精确度.

下面,讨论数值运算中近似值的误差概念.

**定义1** 设  $x$  为某量的准确值,  $x^*$  为  $x$  的近似值,则  $x$  与  $x^*$  之差  $e^*$

$$e^* = x - x^*$$

称为近似值  $x^*$  的**绝对误差**,简称**误差**. 显然,绝对误差可能正也可能负,而并非误差绝对值之意.

一般情况下,我们不能算出准确值  $x$ ,因而也就不能算出绝对误差  $e^*$ . 但我们可以根据具体测量或计算情况估计出误差  $e^*$  的绝对值  $|e^*|$  的某个上界  $\varepsilon^*$ , 即

$$|e^*| = |x - x^*| \leq \varepsilon^*$$

$\varepsilon^*$  称为近似值  $x^*$  的**绝对误差限**,简称**误差限**.

可以通过近似值  $x^*$  及其误差限  $\varepsilon^*$  来表示准确值  $x$  的范围:

$$x^* - \varepsilon^* \leq x \leq x^* + \varepsilon^*$$

或

$$x = x^* \pm \varepsilon^*$$

例如,近似值为 7.5304、误差限为 0.0001 的准确值  $x$  可表示为

$$7.5304 - 0.0001 \leq x \leq 7.5304 + 0.0001$$

或

$$x = 7.5304 \pm 0.0001$$

绝对误差在一定意义下反映了近似值的准确程度,但还未能完全刻划近似值的好坏.例如,  $x_1 = 5 \pm 0.1$  与  $x_2 = 5000 \pm 1$ , 虽然近似值  $x_1^* = 5000$  的误差限 1 是近似值  $x_1^* = 5$  的误差限 0.1 的 10 倍,但若考虑所讨论的值本身的大小,在 5000 之内差 1,比在 5 之内差 0.1,显然前者的准确程度更高.为此,我们引入相对误差的概念.

**定义 2** 近似值的绝对误差  $e^*$  与准确值  $x$  之比

$$e_r^* = \frac{e^*}{x} = \frac{x - x^*}{x}$$

称为近似值  $x^*$  的**相对误差**.

同样,准确值  $x$  一般是不知道的,因此,实际计算中常取相对误差为

$$e_r^* = \frac{e^*}{x^*} = \frac{x - x^*}{x^*}$$

与绝对误差类似,我们也可估计出相对误差绝对值的一个上界

$$|e_r^*| \leq \varepsilon_r^*$$

并称  $\varepsilon_r^*$  为近似值  $x^*$  的**相对误差限**.

由上述定义我们可知:

1) 绝对误差和误差限与所讨论量的单位有关,而相对误差和相对误差限则与所讨论量的单位无关,因此,后者常用百分比表示.

2) 同一个近似值  $x^*$  的绝对误差限与相对误差限有关系

$$\varepsilon_r^* = \frac{\varepsilon^*}{|x^*|}$$

例如,值  $x_1 = 5 \pm 0.1$  的近似值 5 的相对误差限为

$$\varepsilon_r^*(x_1) = \frac{0.1}{5} = 0.02 = 2\%$$

值  $x_2 = 5000 \pm 1$  的近似值 5000 的相对误差限为

$$\varepsilon_r^*(x_2) = \frac{1}{5000} = 0.0002 = 0.02\%$$

可见  $x_2$  的近似值比  $x_1$  的近似值相对地准确.

下面,再引入准确数字与有效数字的概念.

当数  $x$  有很多位时,我们常按“四舍五入”的规则取前面若干位数  $x^*$  作为  $x$  的近似值.如  $x = 3.14159265 \dots$ ,常取 3.14 或 3.1416 作近似值.这种取法的特点是近似值的误差不超过其末位的半个单位①.

**定义 3** 若近似值  $x^*$  的误差不超过某一个位数的半个单位,便称该位数为近似值  $x^*$  的**准确数字**;同时,若该位数到  $x^*$  的第一位非零数字共有  $n$  位,便称这  $n$  位数字为**有效数字**,或称近似值  $x^*$  有  $n$  位有效数字.

例如,  $x = 0.7136 \dots$ . 若取  $x^* = 0.714$ , 由于  $|x - x^*| \leq 0.0005$ , 故  $x^*$  准确到小数后第三位;若取  $x^* = 0.715$ , 由于  $|x - x^*| \leq 0.005$ , 则  $x^*$  只准确到小数后第二位.

又例如,  $x = 30.4500364 \dots$ . 若取  $x^* = 30.45004$ , 则它有 7 位有效数字;若取

① 这里均按绝对值而言,下面的定义也同.



$x^* = 30.4500$ , 则它有 6 位有效数字; 若取  $x^* = 30.45$ , 则它有 4 位有效数字:

根据定义, 任意一个具有  $n$  位有效数字的近似值  $x^*$  总可以表示成标准形式

$$x^* = \pm 0.a_1a_2\cdots a_n \times 10^m \quad (1.1)$$

其中  $m$  是一个整数  $a_1, a_2, \cdots, a_n$  是 0 到 9 中的一个数字而  $a_1 \neq 0$ .

必须指出的是, “四舍五入”的规则有时改为: 四以下舍, 五以上入, 刚好五时, 若前面是偶数, 则将五舍去, 若前面是奇数, 则将五进一. 实践表明, 这种舍入规则有可能减少误差积累.

有效数字与绝对误差和相对误差都有密切的关系:

1) 对任意一个具有  $n$  位有效数字的近似值  $x^*$ , 若把它写成标准形式 (1.1), 则得  $x^*$  的绝对误差估计式

$$|e^*| = |x - x^*| \leq \frac{1}{2} \times 10^{-n} \times 10^m = \frac{1}{2} \times 10^{m-n} \quad (1.2)$$

可见, 近似值的有效数字的位数越多, 绝对误差越小; 而且只要知道了有效数字的位数, 就可写出它的绝对误差限  $e^* = \frac{1}{2} \times 10^{m-n}$ .

2) 对于写成标准形式 (1.1) 的近似值  $x^*$ , 若它具有  $n$  位有效数字, 则由于

$$a_1 \times 10^{m-1} \leq |x^*| \leq (a_1 + 1) \times 10^{m-1}$$

于是可得相对误差估计式

$$|e_r^*| = \frac{|x - x^*|}{|x^*|} \leq \frac{\frac{1}{2} \times 10^{m-n}}{a_1 \times 10^{m-1}} = \frac{1}{2a_1} \times 10^{-n+1} \quad (1.3)$$

即可取相对误差限  $e_r^* = \frac{1}{2a_1} \times 10^{-n+1}$ . 可见, 近似值的有效数字的位数越多, 相对误差也越小.

反之, 若  $x^*$  的相对误差限  $e_r^* = \frac{1}{2(a_1 + 1)} \times 10^{-n+1}$ , 则由于

$$|x - x^*| = |x^*| |e_r^*| \leq (a_1 + 1) \times 10^{m-1} \times \frac{1}{2(a_1 + 1)} \times 10^{-n+1} = \frac{1}{2} \times 10^{m-n}$$

故知  $x^*$  至少具有  $n$  位有效数字.

**例 1** 查表得  $\sqrt{3} = 1.7320581\cdots$ , 问需取几位有效数字, 才使近似值的相对误差限小于 0.1%.

**解** 把  $\sqrt{3}$  写成  $\sqrt{3} = 10 \times 0.17320581\cdots$ . 根据 (1.3) 式, 要求

$$e_r^* \leq \frac{1}{2a_1} \times 10^{-n+1} = \frac{1}{2 \times 1} \times 10^{-n+1} \leq 0.1\%$$

可知取  $n = 4$ , 即有

$$e_r^* \leq \frac{1}{2 \times 1} \times 10^{-4+1} = 0.5 \times 10^{-3} \leq 10^{-3} = 0.1\%$$

故需取四位有效数字  $\sqrt{3} \approx 1.732$ .

我们指出, 实践中通常约定: 原始数据都用有效数字写. 也就是说, 凡不标明绝对或相对误差界的近似数, 都将被认为是有效数字. 例如, 某产量五万八千四百二十万吨, 应

写成

$$58420 \times 10^4 \text{ 吨}$$

它表明有 5 位有效数字,绝对误差限为 0.5 万吨. 如果把它写成

$$584200000 \text{ 吨或 } 5842 \times 10^5 \text{ 吨}$$

则前者表示有 9 位有效数字,绝对误差限为半吨;后者表示有 4 位有效数字,绝对误差限为 5 万吨. 显然两者都是不符合实际情况的.

在科学计算中,常常采用科学记数法,即数的浮点表示法. 例如把

$$25.01760 \text{ 记成 } 0.2501760 \times 10^2$$

$$0.00013579 \text{ 记成 } 0.13579 \times 10^{-3}$$

这时, 0.2501760 是前者的有效数, 0.13579 是后者的有效数.

## 2. 算法与数值稳定性

算法 (Algorithm) 这个概念可以从不同角度加以描述. 一种最简单的说法是: 算法是解决问题的一个逻辑顺序. 稍为详细的说法是: 算法是由一套明确的规则组成的若干步骤, 它指定操作的顺序, 将问题按一定数目的步骤加以解决. 在数值计算领域中, 我们所指的算法是: 可供计算机执行的、对一些数据按某种规定的顺序进行运算的一个有穷序列.

以求一元二次方程

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

的两个实根为例. 我们知道, 当  $b^2 - 4ac \geq 0$  时, 两个实根存在, 计算公式为

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (1.4)$$

由此, 我们可以设计计算机算法如下:

### 算法 A

- 【步 1】输入  $a, b, c$ ;
- 【步 2】判断: 如果  $b^2 - 4ac \geq 0$ , 则转步 4, 否则执行下一步;
- 【步 3】输出无实根标志, 转步 6;
- 【步 4】按公式(1.4)计算  $x_1$  与  $x_2$ ;
- 【步 5】输出  $x_1$  与  $x_2$ ;
- 【步 6】结束.

一般地, 一个算法具有下列重要特性<sup>[c10]</sup>:

- 1) 输入, 具有 0 个或 0 个以上由外界提供的量;
- 2) 输出, 具有 1 个或多个输出;
- 3) 确定性, 即算法的每一步都有确切、无二义性的定义;
- 4) 有穷性, 即每个算法总是在执行有穷步之后结束;
- 5) 可行性, 即算法中所有有待实现的运算都是基本的, 原则上可由一个人仅用笔和纸就能完成的.

算法也可借助框图(或称流程图)来表示。上述算法 A 的框图如图 1-2。

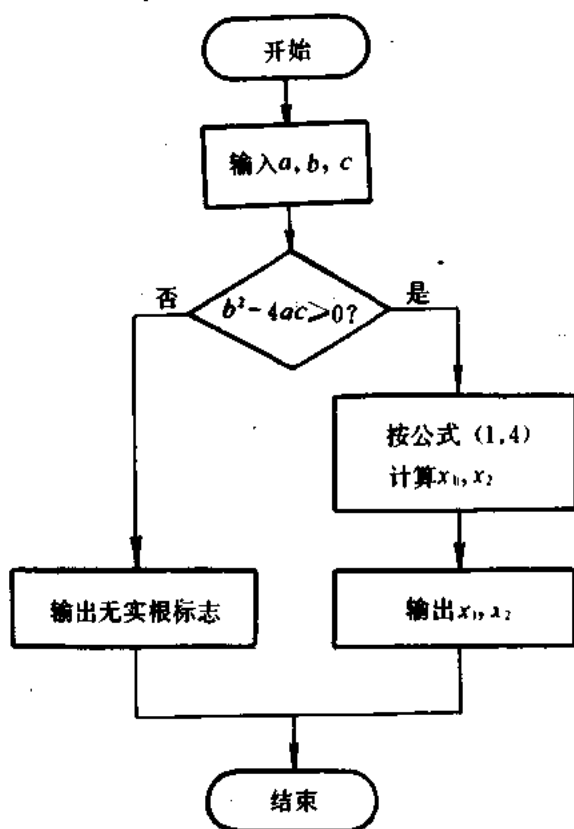
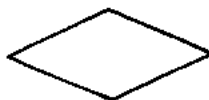


图 1-2

框图中各种符号的意义如下:



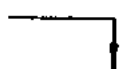
表示一般的计算处理过程, 称工作框。



表示判断或检查。

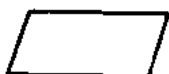


表示入口或出口, 包括开始、结束、暂停和返回。

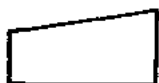


表示流程的路线及方向

在较详细的框图中,有时还采用如下符号:



表示输入或输出。



表示从键盘输入。



表示打印机输出。



表示过程的调用。



表示流程线连接点。

算法框图是直观地描述算法的一种工具。

与算法有关的一个重要问题是,初始数据的误差或计算中产生的舍入误差,在计算过程中的传播和积累,常因算法而异。因而,同一个数值计算问题,采用不同算法得到的结果,其精度可能差别很大。这就是算法的数值稳定性问题。一个算法,如果计算结果受上述的误差影响较小,就称这个算法是**数值稳定的**,否则,就称这个算法是**数值不稳定的**。

例如,对上述二次方程,考虑当  $a = 1, b = -1.000000001 \times 10^9, c = 10^9$  这种比较特殊的情形。由方程

$$\begin{aligned} ax^2 + bx + c &= x^2 - 1.000000001 \times 10^9 x + 10^9 \\ &= x^2 - (10^9 + 1)x + 10^9 \\ &= (x - 10^9)(x - 1) = 0 \end{aligned}$$

可知这时两个实根的准确值为  $x_1 = 10^9$  与  $x_2 = 1$ 。但如果按上述算法 A (即直接用求根公式计算) 编写程序并上机计算,比如在 APPLE II 微型计算机上计算,则得

$$x_1 = 10^9, \quad x_2 = 0.703125$$

可见  $x_1$  很好,  $x_2$  很差。这说明上述算法 A 是数值不稳定的。它受计算中产生的舍入误差影响。

在电子计算机上求解二次方程(其中令  $a > 0$ )的一种做法是,先用下列公式算出绝对值大的一个根,比如  $x_1$

$$x_1 = \begin{cases} -\frac{b}{2a} + \sqrt{\left(\frac{b}{2a}\right)^2 - \frac{c}{a}}, & \text{若 } -\frac{b}{2a} > 0 \\ -\frac{b}{2a} - \sqrt{\left(\frac{b}{2a}\right)^2 - \frac{c}{a}}, & \text{若 } -\frac{b}{2a} \leq 0 \end{cases} \quad (1.5)$$

$$(1.6)$$

然后根据根与系数的关系  $x_1 x_2 = \frac{c}{a} = 10^9$ , 计算另一根

$$x_2 = 10^9/x_1 \quad (1.7)$$

即重新设计算法如下:

### 算法 B

【步 1】输入  $a, b, c$ ;

【步 2】判断: 如果  $\left(\frac{b}{2a}\right)^2 - \frac{c}{a} \geq 0$ , 则转步 4, 否则执行下一步;

【步 3】输出无实根标志, 转步 9;

【步 4】如果  $-\frac{b}{2a} > 0$ , 则转步 6, 否则执行下一步;

【步 5】按公式(1.6)计算  $x_1$ , 转步 7;

【步 6】按公式(1.5)计算  $x_1$ ;

【步 7】按公式(1.7)计算  $x_2$ ;

【步 8】输出  $x_1, x_2$ ;

【步 9】结束.

按这个算法编写程序并上机计算, 同样在 APPLE II 上计算, 则得

$$x_1 = 10^9, \quad x_2 = 1$$

可见  $x_1$  与  $x_2$  都很好, 说明算法 B 有较好的数值稳定性.

也可以举出受初始数据的误差影响的不稳定算法(例如习题一的第 11 题).

### 3. 条件问题与病态概念

什么是数值计算中的条件问题? 我们以计算函数值  $f(x)$  ( $x \in [a, b]$ ) 为例来说明. 假定运算时  $x$  代以近似值  $x^* \in [a, b]$ , 相应地函数值  $f(x)$  代以  $f(x^*)$ ; 又假定  $x^*$  的绝对误差很小, 因而高阶量可以略去不计, 于是我们可得

$$e^*[f(x)] = f(x) - f(x^*) = f'(x^*)e^*(x)$$

这时, 若  $|f'(x)| \leq 1$  ( $x \in [a, b]$ ), 则有

$$|e^*[f(x)]| \leq |e^*(x)|$$

即自变量的微小变动引起的函数值的变动更微小. 如果对某一点  $\bar{x} \in [a, b]$ , 使  $|f'(\bar{x})|$  的值很大, 就说函数  $f(x)$  在  $\bar{x}$  这一点的计算在绝对误差意义下是坏条件的.

相应地, 若  $x \neq 0$ ,  $x^*$  充分接近  $x$ , 由  $e_r^*(x) = \frac{e^*(x)}{x} = \frac{x - x^*}{x}$ ,  $e_r^*[f(x)] = \frac{e^*[f(x)]}{f(x)}$   
 $= \frac{f(x) - f(x^*)}{f(x)}$ , 有

$$\begin{aligned} |e_r^*[f(x)]| &= \left| \frac{e^*[f(x)]}{f(x)} \cdot \frac{x}{e^*(x)} \cdot e_r^*(x) \right| \\ &\approx \left| \frac{xf'(x)}{f(x)} \right| |e_r^*(x)| \end{aligned}$$

因此,如果对某一点  $\bar{x} \in [a, b]$ , 使得  $\left| \frac{\bar{x}f'(\bar{x})}{f(\bar{x})} \right|$  的值很大, 就称函数  $f(x)$  在  $\bar{x}$  这一点的计算在相对误差意义下是坏条件的.

因子  $|f'(x)|$  和  $\left| \frac{xf'(x)}{f(x)} \right|$  分别反映了误差  $e^*(x)$  和  $e_r^*(x)$  对计算结果影响的程度, 其值愈大, 影响程度也愈大, 这些因子就称为所计算问题的**条件数**. 它是问题固有的一种属性.

**例 2** 考虑函数

$$f(x) = x^2 + x - 10100$$

在  $\bar{x} = 99$  处在绝对误差意义下和相对误差意义下的条件数:

$$|f'(\bar{x})| = |2 \times 99 + 1| = 199;$$

$$\left| \frac{\bar{x}f'(\bar{x})}{f(\bar{x})} \right| = \left| \frac{99 \times 199}{99^2 + 99 - 10100} \right| \approx 99$$

可见, 不论在什么意义下,  $f(x)$  在  $\bar{x} = 99$  处都是坏条件的. 事实上,

$$f(99) = -200, \quad f(100) = 0$$

这就是说, 计算  $f(100)$  时, 用  $f(99)$  作为其近似值是不适宜的.

条件数除了能反映出计算结果误差对初始数据误差的增长倍数外, 还能反映出一个计算问题的计算结果对初始数据微小变动(摄动)的敏感程度. 一个计算问题, 当初始数据有微小摄动时, 计算结果对之很敏感, 即变化很大, 就说这个问题是**病态的**, 否则, 当初始数据有微小摄动时, 计算结果对之不敏感, 即变化也很小, 就说这个问题是**良态的**. 一个计算问题的敏感程度就是它的**性态**.

用不准确的数据解病态问题, 不管用什么方法计算, 其解总是不准确的. 然而, 在另外一些问题中, 如上述求二次方程实根的算法 A 与算法 B, 算法 A 失败的原因是使用了不适当的计算; 当换另外的计算公式时, 计算结果就比较准确了. 这就是病态与算法的数值不稳定性两个概念不同之处.

举一个病态的例子.

**例 3** 考虑线性方程组

$$\begin{cases} 3.000\xi_1 + 4.127\xi_2 = 15.41 \\ 1.000\xi_1 + 1.374\xi_2 = 5.147 \end{cases}$$

可以验证, 它有解

$$\xi_1 = 13.6658, \quad \xi_2 = -6.2$$

但如果将第一个方程除以 3.000, 再从第二个方程中减去它, 则可得

$$\xi_2 = \frac{5.47 - (15.41/3)}{1.374 - (4.127/3)}$$

逐步计算, 并准确到小数点后 3 位, 即

- 1)  $15.41/3 = 5.137 \rightarrow \alpha$
- 2)  $5.147 - \alpha = 0.010 \rightarrow \beta$
- 3)  $4.127/3 = 1.376 \rightarrow \gamma$
- 4)  $1.374 - \gamma = -0.002 \rightarrow \delta$
- 5)  $\beta/\delta = -5.000 \rightarrow \xi_2$

可见, 计算值与准确解完全不一致. 我们有充分理由相信, 即使重新选择计算公式, 也不可能使方程组的解准确到小数后 4 位. 这是因为若将原方程组中的系数 4.127 改为 4.122, 它仅在小数后第 3 位上不同, 即原方程组改为

$$\begin{cases} 3.000\xi_1 + 4.122\xi_2 = 15.41 \\ 1.000\xi_1 + 1.374\xi_2 = 5.147 \end{cases}$$

可以验证, 新的方程组是不相容的, 也即无解. 因此, 可知由于第 4 位上的舍入误差, 无论用什么方法解这个方程组, 都会使计算所得的解变动很激烈.

这就是一个病态方程组的例子. 与上述求二次方程实根的算法 A 有本质的区别, 在那里, 可以说是用坏方法解合理问题; 在这里, 是问题本来就不合理.

#### 4. 实际应用中的注意事项

为了尽可能减少误差, 改善算法的数值稳定性, 这里我们讨论实际应用中的若干注意事项.

数值运算中的误差分析是重要而又复杂的问题. 六十年代以来发展了两种误差估计理论, 一种是威肯逊 (J. H. Wilkinson) 基于计算机浮点算法并针对矩阵计算提出的一套研究误差方法<sup>[61]</sup>; 另一种是穆尔 (R. E. Moore) 应用区间分析理论提出的估计误差的区间分析方法<sup>[61, 62]</sup>. 此外, 由于舍入误差的随机性, 也有人应用概率观点研究误差规律; 工程计算还常用几种不同算法 (包括实验方法) 进行比较, 以确定计算结果的可靠性. 在实际处理数值计算问题时, 如果能够注意下列讨论的若干事项, 必将有助于减少误差的危害, 改善算法的数值稳定性.

(1) 避免相近两数相减. 这是因为相近两数相减会严重丢失有效数字, 影响计算的精度. 例如

$$\begin{aligned} \sqrt{8976} - \sqrt{8975} &\approx 94.742 - 94.736 \\ &= 0.006 \end{aligned}$$

由五位有效数字降为一位有效数字. 为此, 可采取变换表达式的方法. 例如:

当  $x$  充分大时可取

$$\sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1} + \sqrt{x}};$$

当  $x$  充分大时可取

$$\operatorname{arctg}(x+1) - \operatorname{arctg} x = \operatorname{arctg} \frac{1}{1+x(x+1)};$$

当  $x$  接近于 0 时可取

$$1 - \cos x = 2 \sin^2 \frac{x}{2};$$

当  $\varepsilon$  充分小时可取

$$\sin(x+\varepsilon) - \sin x = 2 \cos\left(x + \frac{\varepsilon}{2}\right) \sin \frac{\varepsilon}{2};$$

当  $x \approx y$  时可取

$$\lg x - \lg y = \lg \frac{x}{y};$$

当  $x \approx x^*$  时, 利用泰勒展开式

$$f(x) - f(x^*) = (x - x^*)f'(x^*) + \frac{(x - x^*)^2}{2!}f''(x^*) + \dots$$

计算左端可用右端的前若干项来代替。

(2) 防止大数“吃掉”小数。这是因为计算机的位数有限, 运算过程中要进行所谓对阶和规格化, 因此当参加运算的数的数量级相差很大时, 若不注意运算次序, 就有可能把数量级小的数“吃掉”。例如, 设在四位浮点数字计算机上作下列运算

$$0.7315 \times 10^3 + 0.4506 \times 10^{-3}$$

$$\text{对阶} \rightarrow 0.7315 \times 10^3 + 0.0000 \times 10^3$$

$$\text{规格化} \rightarrow 0.7315 \times 10^3$$

其结果大数“吃掉”了小数。又例如

$$0.8153 \times 10^0 + 0.6303 \times 10^3$$

$$\text{对阶} \rightarrow 0.0008 \times 10^3 + 0.6303 \times 10^3$$

$$\text{规格化} \rightarrow 0.6311 \times 10^3$$

其结果大数“吃掉”了部分小数。再如, 假设

$$A = 10^3, B = 10, C \approx -A$$

若按  $(A + B) + C$  次序计算, 则结果接近于零, 失真; 若按  $(A + C) + B$  次序计算, 则结果接近于 10, 正确。

防止大数“吃掉”小数特别要防止重要的物理参量被“吃掉”。

(3) 简化算法步骤, 减少运算次数。这不但可以节省计算机时间, 而且还能减少误差积累。以多项式求值为例, 设

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

若直接逐项求和运算, 每算一个  $a_k x^k$  都要进行  $k$  次乘法, 全式共要  $\frac{1}{2}n(n+1)$  次乘法

和  $n$  次加法。若采用我国的秦九韶算法:

$$\begin{cases} u_n = a_n \\ u_k = u_{k+1}x + a_k, & k = n-1, n-2, \dots, 1, 0 \\ P_n(x) = u_0 \end{cases}$$

则只需  $n$  次乘法和  $n$  次加法, 且简化了算法的逻辑结构。

(4) 可以利用泰勒展开式估计误差。以二元函数为例, 设  $y = f(x_1, x_2)$ , 由  $x_1, x_2$  的近似值  $x_1^*, x_2^*$  计算出  $y^* = f(x_1^*, x_2^*)$ , 可得  $y^* = f(x_1^*, x_2^*)$  的绝对误差估计式

$$\begin{aligned} e^*(y) &= f(x_1, x_2) - f(x_1^*, x_2^*) \\ &\approx \left(\frac{\partial f}{\partial x_1}\right)^* (x_1 - x_1^*) + \left(\frac{\partial f}{\partial x_2}\right)^* (x_2 - x_2^*) \\ &= \left(\frac{\partial f}{\partial x_1}\right)^* e^*(x_1) + \left(\frac{\partial f}{\partial x_2}\right)^* e^*(x_2) \end{aligned}$$

相对误差估计式



$$\begin{aligned} e_r^*(y) &= \frac{e^*(y)}{y^*} \approx \left(\frac{\partial f}{\partial x_1}\right)^* \frac{x_1^* e_r^*(x_1)}{y^*} \\ &\quad + \left(\frac{\partial f}{\partial x_2}\right)^* \frac{x_2^* e_r^*(x_2)}{y^*} \end{aligned}$$

其中  $\left(\frac{\partial f}{\partial x_1}\right)^*$ ,  $\left(\frac{\partial f}{\partial x_2}\right)^*$  分别为偏导数  $\left(\frac{\partial f}{\partial x_1}\right)$ ,  $\left(\frac{\partial f}{\partial x_2}\right)$  在点  $(x_1^*, x_2^*)$  处的值.

根据上述的估计式, 我们可以导出两个近似数  $a, b$  作四则运算时的误差限估计公式.

1) 绝对误差限公式:

$$\begin{aligned} \varepsilon^*(a+b) &= \varepsilon^*(a) + \varepsilon^*(b); \\ \varepsilon^*(a-b) &= \varepsilon^*(a) + \varepsilon^*(b); \\ \varepsilon^*(ab) &\approx |a|\varepsilon^*(b) + |b|\varepsilon^*(a); \\ \varepsilon^*\left(\frac{a}{b}\right) &\approx \frac{|a|\varepsilon^*(b) + |b|\varepsilon^*(a)}{b^2}. \end{aligned}$$

2) 相对误差限公式:

$$\begin{aligned} \varepsilon_r^*(a+b) &= \max \left\{ \frac{\varepsilon^*(a)}{|a|}, \frac{\varepsilon^*(b)}{|b|} \right\} \quad (a, b \text{ 同号}); \\ \varepsilon_r^*(a-b) &= \frac{\varepsilon^*(a) + \varepsilon^*(b)}{|a-b|} \quad (a, b \text{ 同号}); \\ \varepsilon_r^*(ab) &\approx \frac{\varepsilon^*(a)}{|a|} + \frac{\varepsilon^*(b)}{|b|}; \\ \varepsilon_r^*\left(\frac{a}{b}\right) &\approx \frac{\varepsilon^*(a)}{|a|} + \frac{\varepsilon^*(b)}{|b|}. \end{aligned}$$

其中, 第2)组公式可根据  $\varepsilon_r^*(x) = \frac{\varepsilon^*(x)}{|x|}$  由第1)组公式直接导出, 只是第一个公式需

要补充说明如下: 不妨设  $\frac{\varepsilon^*(a)}{|a|} \geq \frac{\varepsilon^*(b)}{|b|}$ , 从而有  $|b|\varepsilon^*(a) \geq |a|\varepsilon^*(b)$ , 两边加上  $|a|\varepsilon^*(a)$ , 并除以  $|a|(|a| + |b|)$ , 可得

$$\frac{\varepsilon^*(a)}{|a|} \geq \frac{\varepsilon^*(b) + \varepsilon^*(a)}{|a| + |b|} = \frac{\varepsilon^*(a) + \varepsilon^*(b)}{|a+b|}$$

这就导致了第一个公式.

**例4** 根据欧姆定律

$$R = \frac{V}{I}$$

其中  $V$  为电压,  $I$  为电流,  $R$  为电阻. 今测得  $V = 110 \pm 2$  (伏特),  $I = 20 \pm 0.5$  (安培). 则得  $R$  的近似值

$$R^* = \frac{110}{20} = 5.5 \text{ (欧姆)}$$

$R^*$  的绝对误差限为

$$\varepsilon^*(R^*) \approx \frac{110 \times 0.5 + 20 \times 2}{20^2} = 0.2375 \text{ (欧姆)}$$

$R^*$  的相对误差限为

$$\epsilon_1(R^*) \approx \frac{2}{110} + \frac{0.5}{20} = 0.0432 = 4.32\%$$

## 1-3 程序设计语言概述

### 1. 从电子计算机谈起

电子计算机常简称计算机 (Computer), 也称电脑。

说计算机是能进行计算的机器, 这容易束缚人们对计算机应用潜力的想象。比较好的说法是, 计算机是信息处理机。信息的一种载体就是我们经常遇到的, 也经常要处理的数据。数据可分为数值数据和非数值数据。数值数据我们比较熟悉, 如 1, 2, 3, 4, 3.1416,  $10^3$ ,  $10^5$ , 等等。计算机可用来主要处理数值数据, 如求方程的数值解, 求数值积分等等, 这就是人们通常说的, 计算机可用来作数值计算或称科学计算。然而, 一本书的书名或一台机器的型号, 如书名《数值计算方法与 FORTRAN 语言》, 计算机型号“IBM-PC”等等, 这些也是数据, 称为非数值数据。计算机也可用来主要处理非数值数据, 如作情报检索、自动订票、企业管理等等。这称为计算机的非数值应用。计算机的数值应用与非数值应用既有区别又没有截然分开, 它们组成了计算机应用的广阔领域。

计算机作为信息处理机, 也象其它处理过程一样, 通常包括如下三个主要部分(图 1-3):

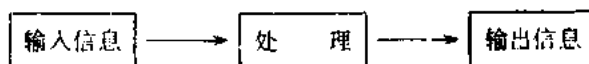


图 1-3

这个过程可用来说明计算机的组成。计算机的组成包括输入设备、存储器、控制器、运算器和输出设备等五大部件。分别说明如下:

**(1) 输入设备** 输入设备接收输入计算机的信息, 包括计算程序和已知的数据, 并把它们从用户可以识别的形式转换成计算机可以存储和处理的形式。通常, 输入的信息被记录在某种介质上, 如穿孔卡片(见第 2 章图 2-1)、磁盘和磁带等。因而, 相应的输入设备有卡片阅读机、磁盘机和磁带机等。此外, 另一种常用的输入设备是键盘。通过键盘, 可以直接把程序和数据送入计算机, 也可以直接向计算机发出各种命令。还有诸如光笔、光学字符机和磁性墨水等都是新型的输入设备。

**(2) 存储器** 存储器用来存放各种信息, 如从输入设备送来的各种程序和数据, 或从运算器中送来的计算结果等。存储器的功能可以比喻人脑的“记忆”功能, 所以存储器也称为记忆装置。存储器通常可分为内存储器和外存储器两种。这里先讲内存储器, 简称内存或主存。

存储器存储信息的最小单位是一个二进制位 (Bit), 即一个位可存储一位二进制数 0 或 1。通常由八个二进位组成一个存储单元, 称为字节 (Byte)。八位二进制数可以组成  $2^8=256$  种不同状态, 足以表示计算机所使用的各种符号。这些符号, 如数字 0, 1, 2, ..., 字母 A, B, C, ..., 运算符 +, -, \*, / 等等, 在计算机中称为“字符”。每个字符用一个二进制编码来表示, 不同的字符对应不同的编码, 称之为“代码”。各个计算机中所用的代码系统可以不同, 在现代的计算机中, 一般采用 7 位编码或 8 位编码, 字符的长度大多

定为 8 位。

由一个或若干个字节又组成一个处理和传送信息的单位,称为字 (Word)。一个字所包含的二进制位的个数,称为字长。各种计算机的字长不完全一样,有 32 位,16 位,8 位等多种。字长较大的机器处理能力较强,故字长常用作衡量计算机功能的指标之一。

存储器的容量由它可使用的字节个数来表示。字节的个数总是 1024 的倍数,每 1024 个字节称为 1K。不同的计算机的内存容量可能相差很大,有的 64K,有的 256K,有的多达 1 兆或几兆,1 兆等于 1024K。内存容量的大小直接关系到计算机处理信息的能力的大小,故内存容量也是衡量计算机功能的指标之一。

存储器中的特定单元是由它的地址来标识的。在信息处理过程中,要引用一个数据时,总是按其所在的存储单元的地址或按依次相连存放它的单元的单元的地址对其进行访问的。

存储单元的一个重要特性是,信息一旦存入,就一直保留在那里,它可以重复读出,直至把新的信息存入,才把原来的信息冲掉,换为新的信息。

**(3) 控制器** 控制器是用来对计算机各个组成部分的工作进行控制和管理的装置。如从内存储器取信息、按计算程序执行各种指令、控制内存储器与运算器或输入/输出设备之间的信息传递等等,都是在控制器的指挥下,有节奏而又协调地进行的。

控制器一般由指令寄存器、指令计数器、译码器、时序电路和控制台组成。

**(4) 运算器** 运算器是计算机中直接完成各种算术和逻辑运算、以及数码的传送、移位和一些中间结果的“寄存”的装置。在控制器的管理下,数据按照需要由存储器传送到运算器,在运算器里进行算术和逻辑运算,然后再返回存储器。数据要在这两个装置之间往返若干次,一直到计算任务完成,计算结果从存储器被传送到输出设备。

运算器一般由加法器和若干寄存器组成。在整个计算机系统中,控制器起中枢神经系统的作用。

**(5) 输出设备** 输出设备用来从计算机给用户输出信息。输出的信息可以用用户能认识的形式出现,也可仍保持原来面向机器的形式输入到磁盘、磁带或另一机器。常用的输出设备有荧光屏显示器 (CRT)、打印机、磁盘机、磁带机以及绘图仪等。

计算机各组成部分的联系如图 1-4 所示。其中细实线箭头表示控制信息流向,粗实线箭头表示数据信息流向。存储器、控制器和运算器构成计算机的主机部分;控制器和运算器合称为中央处理机,简称 CPU (Central Processing Unit)。

除了内存储器外,计算机通常还配置有外存储器(简称外存),用来存储计算机运行过程中暂时不用的程序和成批数据。外存储器一般使用磁存储介质。常用的外存储器有磁盘、磁带等。与内存储器相比,外存储器对于信息的存取速度要慢一些,但其容量却可以做得很大,大至几十到几百兆字节。外存储器与内存储器配合使用,可解决信息的高速与大量存取之间的矛盾。

通常,程序或数据由输入装置通过中央处理机经内存而送进外存。当要在计算机上运行某程序时,中央处理机便发出命令,把存在外存中的该程序调入内存某一存储区内,然后执行该程序。当运行结束或系统要中断该程序时,中央处理机又发出命令,把该程序及有关的数据送回外存,而把占用的内存存储区退出来供其它程序使用。在运行过程中,中央处理机实际上只与内存发生联系。

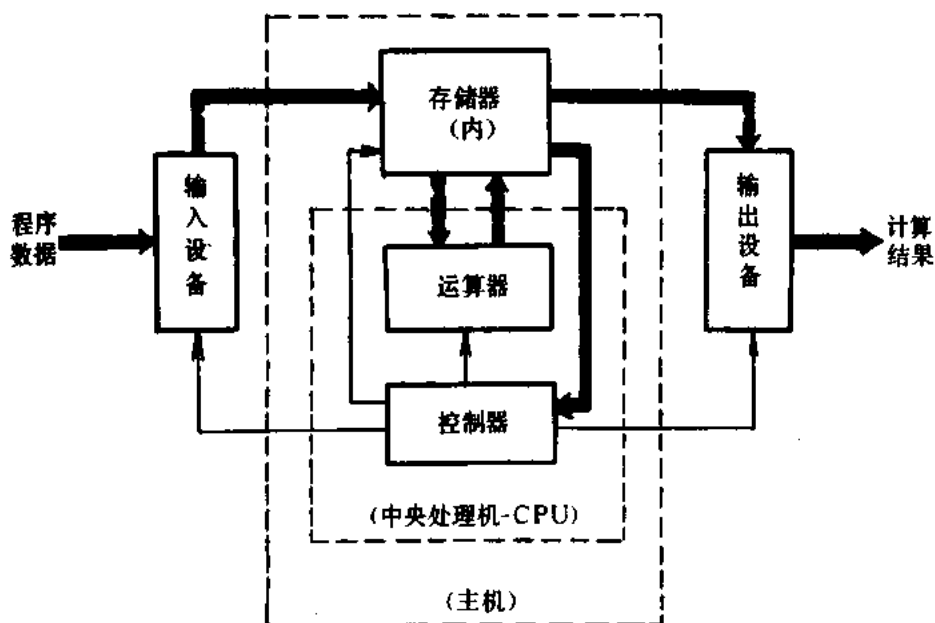


图 1-4

## 2. 程序与程序设计语言

为了使计算机能够按照我们的要求,对给定的数据进行预定的加工处理,我们必须给出一组特定的所谓**指令** (Command),即指示计算机执行操作的命令。这种指令序列就称为**程序** (Program)。从算法的角度说,编码了的算法就是程序。

初期的计算机,程序指令并不存放在内存储器里。计算机按照程序指令对存储器里的数据进行加工处理时,仍需一定的人工干预。这种处理方式自然速度有限。1945年著名数学家冯·诺依曼 (John Von Neumann) 和他的同事们首先提出了把程序指令编成与数据一样的二进制代码形式存入计算机的内存储器里,计算机对内存储器里的数据进行加工处理,将直接按照存放在内存储器里的程序指令自动地进行。这就形成了所谓**存储程序式计算机**。它的出现大大提高了计算机的计算能力,刺激了计算机的飞速发展。现代计算机都是建立在这种思想上的存储程序式计算机。程序是否存储在计算机里,这是通常所说的计算机与算盘、计算器的本质区别之一。

把预定的任务或设计好了的算法用程序实现的全过程称为**程序设计** (Programming),编写计算机程序时所采用的一种事先约定来传达信息的代码组合形式,就称为**程序设计语言** (Programming Language)。

程序设计语言可分为三大类,形成相应的三个级:

(1) **机器语言** 直接使用机器的指令作为程序设计的语言,这种语言称为机器语言。机器的指令是计算机设计者制定的,一部计算机的所有指令的集合组成这部计算机的指令系统。不同型号的计算机有不同的指令系统。由于计算机只能接受二进制信息,因此输入计算机的指令和需要的数据都必须是二进制代码的形式,而且还要给数据指定存储单元。一条机器指令至少有一个所谓操作码和一个或多个所谓地址码。因而用机器语言编写的程序尽是一串串0、1交错的代码,相当繁难,而且因机而异。

(2) **汇编语言** 为了改善机器语言程序设计的繁重劳动,人们开始发展一种面向机

器的、用英文字母及符号代表指令的语言,这种语言称为汇编语言或符号语言。用汇编语言编写的汇编语言程序,它必须转换成机器语言程序以后,才能被计算机所接受和执行。为此,计算机专业人员利用计算机本身,用机器语言写一个所谓汇编程序,就让汇编程序把汇编语言程序翻译成机器语言程序,并代替人工给数据分配存储单元。通常把原来写的汇编语言程序称为源程序,把由源程序翻译出来的机器语言程序称为目标程序。

(3) **高级语言** 或称算法语言。汇编语言比机器语言显然前进了一大步。但仍然是一种依赖于具体机器、与数学计算公式写法差别很大的程序语言。五十年代中期以来,人们相继创造了各种面向算法、面向处理过程并与具体机器指令系统无关的比较高级的语言,这种语言便是高级语言或算法语言。这种程序语言既保持人类自然语言(英语)的原义,又接近数学公式的表达形式。这种语言易学好懂,只要掌握这种语言,并不需要了解机器语言,便可进行程序设计。

与汇编语言程序类似,由高级语言编写的源程序也需要变换成对应的用机器语言表达的目标程序,才能为计算机所执行。这项工作是由一种称为编译程序或另一种称为解释程序的机器语言程序来做的。编译程序将整个源程序翻译成目标程序后,再让计算机去执行,即分编译和执行两个阶段来完成任务。

程序设计语言目前已成为计算机科学中的一个重要分支。就高级语言来说,至今已经设计出不下三、四百种,其中较著名的也有二、三十种。目前最常用的有 BASIC 语言、FORTRAN 语言、PASCAL 语言、COBOL 语言、ALGOL 语言、PL/1 语言、LISP 语言和 APL 语言等。这些语言各有主要的适用范围。例如, FORTRAN 和 ALGOL 就特别适用于数值计算。

### 3. 软件

通常把计算机的各个组成部分,包括输入/输出设备、中央处理机和内外存储器等称为**硬件**(Hardware)。相对于硬件来说,使计算机运转并完成交给它的任务的各种程序总称为**软件**(Software)。粗略来说,计算机软件可分系统软件、程序语言和各种应用程序三个方面。系统软件包括操作系统(Operating System)、各种翻译程序、诊断程序等。这些软件一般都由计算机厂家随机提供。应用程序由用户根据各种应用问题的需要编制,也称应用软件。

操作系统起控制计算机各项操作的作用,它使用户高效率地利用计算机的各项资源,保证人-机通信的顺利进行。操作系统主要包括监督程序、文件系统、装入程序、编辑程序和输入/输出控制等。其中监督程序是核心,它实现动态分配存储区、处理机调度、通信管理、资源共享的指挥和系统基本处理程序的控制等。

语言翻译程序分汇编程序、编译程序和解释程序等几种。通常,一种汇编语言或算法语言总有相应的翻译程序,如 FORTRAN 编译程序(或称编译系统)、BASIC 解释程序(或称解释系统),等等。一般情况总是把编译程序先存放在外存储器(磁盘或磁带)里。当源程序被送入计算机后,系统把编译程序调入内存,并用此编译程序对源程序进行编译,其中包括把用算法语言所写的各指令(这时称为语句)分解成一系列的机器指令,为在运算中要用到的各变量分配内存单元,以及检查源程序中是否有语法错误。如无错误,便最后形成目标程序。此后,就可以执行目标程序了。执行过程又包括读入需要的初始数据、

计算并输出计算结果等。

应用程序的范围很广泛。随着计算机应用的发展,人们把预先编制好的、正确可靠并且使用频繁的程序,存放在外存储器里,组成一个程序的仓库,称为程序库。程序库又可分为子程序库和应用程序包。子程序库是一些程序段,一般都包括范围广泛的数值计算或统计计算等方面的各种程序。使用计算机的人可以在自己的程序中插入这些程序或调用这些程序。应用程序包通常用来解决某些用户的一类问题;使用应用程序包时用户只须准备和填写数据。

计算机硬件和软件组成一个计算机系统。

#### 4. 计算机系统的使用方式

计算机系统的工作方式主要有分时处理和批处理两种<sup>①</sup>。

分时处理是每个用户占用一个分时终端,系统轮流为每个分时终端服务,一次一个时间片(比如 20ms)。当终端用户在思考或敲打键盘时,系统可以为其它用户服务,因此,每个终端用户并不感到有许多用户同时在使用机器,而是好象整个计算机由他自己支配一样。用户通过一系列分时命令与计算机系统会话,他可以在终端上输入、编辑、调试和修改程序,并把程序投入运行。运行结果可以在显示器或打印机上输出。

批处理是将作业输入到计算机系统,等机器有空时再逐个取出来进行处理。为处理每一个问题而编制的程序及其所控制的数据在一起,称为计算机中的一个作业。批处理适用于大型的,但并不要求立即看到运行结果的解题程序。

批处理还可分本地批处理和远程批处理。本地批处理是事先将程序和数据通过脱机的数据录入系统录制到软磁盘上,然后经主机房的磁盘驱动器将软盘上的信息输入到计算机里等候处理,处理结果可以在打印机上印出。远程批处理是通过所谓远程作业输入站(RJE)的键盘,把程序和数据打入到硬盘或软盘;或者事先通过脱机的数据录入系统录制到软盘,再由键盘命令把盘上的内容通过通讯线路传输到中心机等候处理。处理结果可以送回,在 RJE 的打印机上输出,也可以在中心的打印机上输出。

#### 5. 计算机中数的表示

计算机表示数有两种方法:定点表示和浮点表示,相应地称定点数和浮点数。

定点表示指约定小数点在某一固定位置上。例如,设用两个 8 位的字节来表示一个定点数,其中约定最前面一位表示数的符号,小数点在最后,那么它可以表示一个 15 位二进制数,其最大值与最小值如下所示:

$$\begin{array}{ccc} \pm 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline & & & & & & 8 & \text{位} \end{array} \qquad \begin{array}{ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline & & & & & & 8 & \text{位} \end{array}$$

化成十进制数是  $\pm(2^{15} - 1) = \pm 32767$ , 即在这种情况下所能表示的定点数范围是  $-32767$  至  $+32767$  之间的整数。

由于误差分析本质上和计算时所采用的数制无关,所以虽然计算机中是采用二进制的,但我们的讨论还是常常以十进制为例。

<sup>①</sup> 还有实时处理方式。

定点运算的加减法,只要运算结果不超出机器的允许范围,则机器不造成任何误差。当超出允许范围时,机器出现“溢出”而停机。对于两个  $t$  位数相乘,得到  $2t$  位乘积。这时,若小数点约定在最前面,则当机器把它舍入成单字长数时,引入至多  $\frac{1}{2} \times 10^{-t}$  的误差,即末位数的一半;若小数点约定在最后面,则由于乘积是  $2t$  位的整数,此时不能舍入,通常使用两个存储单元,一个存放乘积的前一半,另一个存放后一半,此乘积称双字长数。由上可见,定点表示的缺点是表示范围有限,还可能出现“溢出”的麻烦。

浮点表示与科学记数法类似。浮点表示是指把每一个数表示成形式

$$x^* = \sigma(0.a_1a_2\cdots a_t)_\beta \times \beta^r \quad (1.8)$$

其中  $\sigma = \pm 1$  称为数符,  $0.a_1a_2\cdots a_t$  称为尾数,  $a_i$  均为整数,  $t$  为自然数,称为浮点数的字长,  $\beta$  为浮点数的基底,取  $\beta = 10, a_i$  为  $0, 1, \dots, 9$  中的某个数时,所表示的数是十进制数,取  $\beta = 2, a_i$  取  $0$  或  $1$  中的某一个数时,所表示的数是二进制数,等等;  $\sigma$  称为浮点数的阶码,它有固定的上下限。

通常,浮点数均进行规格化,即要求(1.8)中的  $a_1$  不能为零。可见,浮点数包括尾数和阶码及各自的符号位两个部分。在计算机的存储单元中,用一对数来表示这两部分,用  $(0, 0)$  表示数零。对不同的计算机,尾数和阶码的位数是不同的。

一般地,假设计算机中浮点数的全体组成的集合记为  $F$ ,显然  $F$  是一个离散的有限集合。利用计算机进行计算时,初始数据和中间结果都可能不在  $F$  中。因此,便存在如何用  $F$  中的浮点数来近似地表示相应数值的问题。

这时,若要表示的数字的绝对值大于  $F$  中的最大正数或小于  $F$  中的最小正数,则机器出现“上溢”与“下溢”现象。

如果对非十进制数字,也仿照十进制数字引进舍入规则,即假定要表示的数字经舍入后为

$$fl(x) = \begin{cases} \sigma(0.a_1a_2\cdots a_t)_\beta \times \beta^r, & 0 \leq a_{t+1} < \frac{\beta}{2} \\ \sigma[(0.a_1a_2\cdots a_t)_\beta + \beta^{-t}] \times \beta^r, & \frac{\beta}{2} \leq a_{t+1} < \beta \end{cases}$$

其中  $a_1 \neq 0$ ,则可用上述方法确定  $fl(x)$  作为  $(x)$  的浮点近似值,绝对误差和相对误差满足关系

$$|fl(x) - x| \leq \frac{1}{2} \times \beta^{-t}$$

$$\left| \frac{fl(x) - x}{x} \right| \leq \frac{1}{2a_1} \times \beta^{-t+1} \leq \frac{1}{2} \times \beta^{-t+1}$$

我们称  $\epsilon_{ps} = \frac{1}{2} \times \beta^{-t+1}$  为计算机的精度,并称上述确定浮点数的方法为“舍入”。记  $\epsilon$

为  $fl(x)$  的相对误差,则由  $\frac{x - fl(x)}{x} = \epsilon$ , 得

$$fl(x) = x - x\epsilon = x(1 - \epsilon) \quad |\epsilon| \leq \frac{1}{2} \times \beta^{-t+1}$$

即  $x\epsilon$  是用  $fl(x)$  表示  $x$  的绝对误差。

考虑计算机中浮点数的运算. 设规格化浮点数  $x, y \in F$ , 它们的算术运算

$$x + y, \quad x - y, \quad x \times y, \quad x / y$$

的精确结果不一定是  $F$  中的浮点数. 在计算机进行浮点运算时, 机器自动把运算结果用  $F$  中的浮点数表示出来, 记算术运算的结果为

$$\begin{aligned} fl(x + y), \quad fl(x - y), \\ fl(x \times y), \quad fl(x / y). \end{aligned}$$

这些运算结果如何用机器数来表示, 随计算机的功能而不同. 假设计算机具有双精度累加寄存器, 即在运算时先保留  $2t$  位, 最后再把第  $t + 1$  位“舍入”. 这时可得浮点运算和精确运算之间的关系为

$$\begin{aligned} fl(x + y) &= (x + y)(1 + \varepsilon_1) \\ fl(x - y) &= (x - y)(1 + \varepsilon_2) \\ fl(x \times y) &= (x \times y)(1 + \varepsilon_3) \\ fl(x / y) &= (x / y)(1 + \varepsilon_4) \end{aligned}$$

其中  $|\varepsilon_i| \leq eps = \frac{1}{2} \times \beta^{-t+1}$ ,  $i = 1, 2, 3, 4$ , 右端表示算术运算的精确结果.

## 1-4 数学补充材料

### 1. 大 $O$ 记号

这是数学、计算机科学和科技文献中广泛使用的符号, 是为处理近似值而允许我们用  $=$  号代替  $\approx$  号的方便符号.

**定义** 设  $X, Y$  为两个变量(不必是无穷小或无穷大), 且  $X \neq 0$ , 如果从某一时刻以后, 便有

$$\left| \frac{Y}{X} \right| \leq M \quad (M \text{ 为大于零的常数})$$

就记成

$$Y = O(X)$$

这就是说, 记号  $O(X)$  表示这样一个数量, 我们并不明显地知道它, 也不必说出定义中的  $M$  是什么, 它的出现即意味着, 当  $X$  变化到某一时刻以后, 便有  $|O(X)| \leq M|X|$ .

下面用例子说明记号的用法.

**例 1** 我们知道

$$\begin{aligned} 1^2 + 2^2 + \cdots + n^2 &= \frac{1}{3} n \left( n + \frac{1}{2} \right) (n + 1) \\ &= \frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n \end{aligned}$$

由此可记

$$1^2 + 2^2 + \cdots + n^2 = O(n^3)$$

或

$$1^2 + 2^2 + \cdots + n^2 = \frac{1}{3} n^3 + O(n^2)$$



**例2** 设  $x, y$  为两个无穷小, 把大  $O$  记号用于阶的比较有:

若  $y$  不比  $x$  阶低, 可记  $y = O(x)$ ;

若  $y$  与  $x$  同阶, 可记  $y = O(x)$ .

另外, 若  $x$  是有界变量, 则  $|x| = \frac{|x|}{1} \leq M$ , 于是可记  $x = O(1)$ .

**例3** 当  $x \rightarrow 0$  时, 记号  $f(x) = O(x)$  表示  $f(x)$  至少象  $x$  那样快地趋于零; 记号  $g(x) = O(x^2)$  表示  $g(x)$  至少象  $x^2$  那样快地趋于零. 因此当  $x \rightarrow 0$  时,  $g(x)$  比  $f(x)$  更快地趋于零.

必须注意, 大  $O$  记号具有单向相等性. 可以写  $\frac{1}{2}n^2 + n = O(n^2)$ , 决不能写  $O(n^2)$

$= \frac{1}{2}n^2 + n$ . 否则由于  $\frac{1}{4}n^2 = O(n^2)$ , 岂不有  $\frac{1}{4}n^2 = \frac{1}{2}n^2 + n$ . 使用大  $O$  记号时我们

总是这样约定, 右端只是左端的“粗略化”, 一个等式的右端不能给出比左端更多的信息.

大  $O$  记号可以进行某些简单运算:

- 1)  $X = O(X)$ ;
- 2)  $C \cdot O(X) = O(X)$  ( $C$  为不等于零的常数);
- 3)  $O(X) + O(X) = O(X)$ ;
- 4)  $O(O(X)) = O(X)$ ;
- 5)  $O(X)O(Y) = O(X \cdot Y)$ ;
- 6)  $O(X \cdot Y) = X \cdot O(Y)$ .

## 2. 内积 · 向量和矩阵的范数

我们知道, 引入实数的绝对值和复数的模 (也称绝对值) 来表示实数和复数的“大小”, 从而带来了许多用处. 例如, 数列收敛的概念就是通过绝对值来表示的. 此外, 平面向量  $x$ , 当其在直角坐标中的分量为  $x_1, x_2$  时, 也用  $\sqrt{x_1^2 + x_2^2}$  给出其大小 (或长度) 的度量. 类似地, 空间向量也有相仿的结果. 范数这个概念就是这些表示“大小”的数值的普遍化. 它在研究数值计算方法的收敛性和稳定性中有重要的应用.

这里我们只考虑实  $n$  维向量空间  $R^n$  的情形, 但所得结果可以推广到复向量空间和无限维空间 (包括函数空间).

我们先给出  $R^n$  空间中内积的概念.

**定义1** 设  $x, y \in R^n, x = [x_1, x_2, \dots, x_n]^T, y = [y_1, y_2, \dots, y_n]^T$ , 则实数

$$(x, y) = x^T y = \sum_{i=1}^n x_i y_i$$

称为向量  $x, y$  的数量积或内积.

容易看出, 内积具有下列性质:

- 1)  $(x, x) \geq 0$ , 当且仅当  $x = 0$  时  $(x, x) = 0$ ;
- 2) 对任意  $\alpha \in R, (\alpha x, y) = \alpha(x, y)$ ;
- 3)  $(x, y) = (y, x)$ ;
- 4)  $(x_1 + x_2, y) = (x_1, y) + (x_2, y)$ .

现在讲向量的范数.

将平面向量  $x$  的大小  $\sqrt{x_1^2 + x_2^2}$  记为  $\|x\|$ , 则它具有与实数及复数的绝对值相似的性质:

- 1) 非负性, 即  $\|x\| \geq 0$ , 当且仅当  $x = 0$  时  $\|x\| = 0$ ;
- 2) 齐次性, 即对任一纯数量  $\alpha$ , 有  $\|\alpha x\| = |\alpha| \|x\|$ ;
- 3) 三角不等式, 即  $\|x + y\| \leq \|x\| + \|y\|$ .

$n$  维向量范数的一般概念就在这个基础上建立起来的.

**定义 2** 设向量  $x, y \in R^n$ , 满足下列三个性质的数量值函数  $\|x\|$  称为向量  $x$  的范数:

- 1)  $\|x\| \geq 0$ , 当且仅当  $x = 0$  时  $\|x\| = 0$ ;
- 2) 对任一纯数量  $\alpha$ ,  $\|\alpha x\| = |\alpha| \|x\|$ ;
- 3) 三角不等式  $\|x + y\| \leq \|x\| + \|y\|$ .

可以验证, 对  $n$  维向量  $x = [x_1, x_2, \dots, x_n]^T \in R^n$ , 可以定义范数

$$\|x\| = \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{(x, x)} = \sqrt{x^T x}.$$

这种范数称为欧几里德范数, 有时记为  $\|x\|_2$ , 或称 2-范数. 它是向量长度的直接推广. 还可以定义其它形式的范数, 只要所定义的范数满足定义 2 中的三个性质. 除了欧氏范数, 另外两种常用的向量范数是:

$$\|x\|_1 = \sum_{i=1}^n |x_i| \quad (1\text{-范数})$$

$$\|x\|_\infty = \max_i |x_i| \quad (\infty\text{-范数或最大范数}).$$

**例 1** 向量  $x = [1, -2, 0]^T$  的三种范数:

$$\|x\|_1 = |1| + |-2| + |0| = 3;$$

$$\|x\|_2 = \sqrt{1^2 + (-2)^2 + 0^2} = \sqrt{5};$$

$$\|x\|_\infty = \max\{|1|, |-2|, |0|\} = 2.$$

**例 2** 利用内积和范数的性质, 可以证明著名的柯西-许瓦兹 (Cauchy-Schwarz) 不等式:

$$|(x, y)| \leq \|x\|_2 \|y\|_2$$

事实上, 若  $x = 0$ , 此不等式显然成立. 今设  $x \neq 0$ , 对任意参量  $\lambda$ , 都有

$$\begin{aligned} \|\lambda x + y\|^2 &= (\lambda x + y, \lambda x + y) \\ &= \lambda^2 \|x\|^2 + 2\lambda(x, y) + \|y\|^2 \geq 0 \end{aligned}$$

由于  $x, y$  是给定的向量, 所以这个含  $\lambda$  的二次三项式的判别式  $(x, y)^2 - \|x\|^2 \|y\|^2 \leq 0$  故得

$$|(x, y)| \leq \|x\|_2 \|y\|_2$$

有了向量范数, 就可讨论  $R^n$  中向量序列的收敛概念. 我们定义  $R^n$  中的向量序列  $\{x^{(k)}\}$ , 其中

$$x^{(k)} = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]^T$$

$\{x^{(k)}\}$  收敛于向量  $x \in R^n$ , 是指对  $i = 1, 2, \dots, n$ , 都有

$$\lim_{k \rightarrow \infty} x_i^{(k)} = x_i \text{ 或 } \lim_{k \rightarrow \infty} |x_i^{(k)} - x_i| = 0$$

且记为

$$\lim_{k \rightarrow \infty} x^{(k)} = x$$

利用范数记号, 我们有  $\lim_{k \rightarrow \infty} x^{(k)} = x$  的充分必要条件是

$$\|x^{(k)} - x\| \rightarrow 0 \quad (k \rightarrow \infty)$$

其中  $\|\cdot\|$  是上述定义的任一种向量范数.

下面讲矩阵的范数.

**定义 3** 设矩阵  $A, B \in R^{n \times n}$ , 满足下列四个性质的数量值函数  $\|A\|$  称为矩阵  $A$  的范数:

- 1)  $\|A\| \geq 0$ , 当且仅当  $A = 0$  时  $\|A\| = 0$ ;
- 2) 对任一纯数量  $\beta$ ,  $\|\beta A\| = |\beta| \|A\|$ ;
- 3) 三角不等式  $\|A + B\| \leq \|A\| + \|B\|$ ;
- 4)  $\|AB\| \leq \|A\| \|B\|$ .

由于在大多数问题中, 矩阵和向量常常在一起参与讨论, 因此我们希望引进一种矩阵的范数, 它和向量范数相联系又和向量范数相容, 即对任何  $A \in R^{n \times n}$  和  $x \in R^n$ , 满足

$$\|Ax\| \leq \|A\| \|x\|$$

为此, 我们引进矩阵的所谓算子范数: 设  $x \in R^n, A \in R^{n \times n}$ , 对于每一种向量范数  $\|x\|_p$  (如  $p = 1, 2, \infty$ ); 相应地定义矩阵  $A$  的算子范数  $\|A\|_p$  为

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$$

可以验证,  $\|A\|_p$  满足矩阵范数的条件, 而且满足相容性条件

$$\|Ax\|_p \leq \|A\|_p \|x\|_p$$

对于  $x \in R^n, A \in R^{n \times n}$ , 常见的范数有:

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}| \quad (\text{称为 } A \text{ 的行范数})$$

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}| \quad (\text{称为 } A \text{ 的列范数})$$

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)} \quad (\text{称为 } A \text{ 的 2-范数})$$

其中  $\lambda_{\max}(A^T A)$  表示  $A^T A$  的最大特征值.  $\|A\|_2$  又称为  $A$  的谱范数.

**例 3** 设  $A = \begin{bmatrix} 1 & -2 \\ -3 & 4 \end{bmatrix}$ , 计算  $A$  的各种范数:

$$\|A\|_1 = 6;$$

$$\|A\|_2 = \sqrt{15 + \sqrt{221}} \approx 5.46;$$

$$\|A\|_\infty = 7.$$

与定义向量收敛的做法相仿, 我们定义:  $R^{n \times n}$  中矩阵序列  $\{A^{(k)}\}$  当各元素收敛时, 称  $\{A^{(k)}\}$  收敛, 且以各元素的极限为元素的矩阵  $A$  称为  $\{A^{(k)}\}$  的极限, 记为  $\lim_{k \rightarrow \infty} A^{(k)} = A$  或  $A^{(k)} \rightarrow A (k \rightarrow \infty)$ .

利用范数记号, 我们有  $\lim_{k \rightarrow \infty} A^{(k)} = A$  的充分必要条件是

$$\|A^{(k)} - A\| \rightarrow 0 \quad (k \rightarrow \infty)$$

其中  $\|\cdot\|$  是上述定义的任一种矩阵范数.

矩阵范数还有如下一些性质:

1) 当  $k \rightarrow \infty$  时  $A^k \rightarrow 0$  的充分必要条件是  $A$  的一切特征值的模都小于 1. (证略)

2) 欲使  $A^k \rightarrow 0$ , 只须有一种范数小于 1.

证 这是因为

$$\|A^k\| \leq \|A^{k-1}\| \|A\| \leq \|A^{k-2}\| \|A\|^2 \leq \cdots \leq \|A\|^k$$

所以, 如果  $\|A\| < 1$ , 则  $k \rightarrow \infty$  时,  $\|A^k\| \rightarrow 0$ , 即  $A^k \rightarrow 0$ .

3) 矩阵级数

$$I + A + A^2 + \cdots + A^k + \cdots$$

收敛的充分必要条件是当  $k \rightarrow \infty$  时,  $A^k \rightarrow 0$ , 这时  $(I - A)$  有逆, 级数和为

$$S = (I - A)^{-1}$$

证 必要性显然. 关于充分性, 先证  $|I - A| \neq 0$ . 如果  $|I - A| = 0$ , 则  $\lambda = 1$  为矩阵  $A$  的特征值, 这与假设有矛盾(根据性质 1)), 故  $|I - A| \neq 0$ , 从而  $(I - A)^{-1}$  存在.

由

$$(I + A + A^2 + \cdots + A^k)(I - A) = I - A^{k+1}$$

有

$$I + A + A^2 + \cdots + A^k = (I - A)^{-1} - A^{k+1}(I - A)^{-1}$$

当  $k \rightarrow \infty$  时,  $A^k \rightarrow 0$ , 故

$$I + A + A^2 + \cdots + A^k + \cdots$$

收敛, 和为  $(I - A)^{-1}$ .

4) 如果  $\|A\| < 1$ , 则有

$$\|(I - A)^{-1} - (I + A + A^2 + \cdots + A^k)\| \leq \frac{\|A\|^{k+1}}{1 - \|A\|}$$

证 根据 3) 我们有

$$(I - A)^{-1} - (I + A + A^2 + \cdots + A^k) = A^{k+1} + A^{k+2} + \cdots,$$

因此,

$$\begin{aligned} & \|(I - A)^{-1} - (I + A + A^2 + \cdots + A^k)\| \\ & \leq \|A\|^{k+1} + \|A\|^{k+2} + \cdots = \frac{\|A\|^{k+1}}{1 - \|A\|} \end{aligned}$$

## 习 题 一

1. 我国古代数学家祖冲之, 曾经以  $\frac{355}{113}$  作为  $\pi$  的近似值, 试求此近似值的有效数字的位数.

2. 给出下列三个数:

$$a = 123 \pm 1$$

$$b = 1.23 \pm 0.01$$

$$c = 0.00123 \pm 0.00001$$

试问这三个数哪个精度高?由此可得出什么结论?

3. 下列各数都是对准确值进行四舍五入得到的近似值,试分别指出它们的绝对误差限、相对误差限及有效数字的位数:

$$x_1^* = 685.3, \quad x_2^* = 0.0135,$$

$$x_3^* = 13.50, \quad x_4^* = 700000,$$

$$x_5^* = 7 \times 10^7.$$

4. 设下列运算中所有初始数据均准确到小数后两位. 试估计当计算

$$a = 1.21 \times 3.65 + 9.81$$

的绝对误差限和相对误差限.

5. 假定查表取平方根的值准确到小数两位,计算

$$x = \sqrt{1986} - \sqrt{1985}$$

并估计减法绝对误差限与相对误差限.

6. 已知数据  $a_i, b_i (i = 1, 2, \dots, n)$  有相同的误差限  $\varepsilon$ . 求数量积  $\sum_{i=1}^n a_i b_i$  的误差限.

7. 如何计算  $x^2 - 56x + 1 = 0$  的两个根,使它至少有四位有效数字(取  $\sqrt{783} \approx 27.982$ ).

8. 计算球体积时要使相对误差限为 1%,问度量半径  $R$  时允许的相对误差限是多少?

9. 已知递推公式

$$u_{n+1} = u_n - \frac{1}{100} \sqrt{783}, \quad n = 0, 1, 2, \dots$$

取  $u_0 = 28, \sqrt{783} \approx 27.982$  (五位有效数字),计算  $u_{100}$ . 试问误差将有多大?

10. 讨论下列函数在指定点附近是好条件还是坏条件:

$$(1) f(x) = \frac{1}{n} \sin(n^2 x) \quad (x = 0, n \rightarrow \infty)$$

$$(2) f(x) = \lg x \quad (x = 1)$$

11. 给出递推公式

$$x_k = ax_{k-1} + 1 \quad (a > 0 \text{ 的常数})$$

若取  $x_0 = \sqrt{2} \approx 1.41$ , 试估计当计算到  $x_{10}$  时的误差,并讨论这个计算过程是否稳定?

12. 设计计算机算法时,常需要对计算公式作适当的变形. 试处理下列问题:

(1) 计算两复数  $a + ib$  和  $c + id$  商的公式是:

$$\frac{a + ib}{c + id} = \frac{(ac + bd) + i(bc - ad)}{c^2 + d^2}$$

为了防止在直接计算  $c^2$  或  $d^2$  时出现溢出,试把公式加以适当的变形.

(2) 复数  $z = x + iy$  的模的计算公式为

$$|z| = \sqrt{x^2 + y^2}$$

仿照(1),也把上述公式作适当的变形.

13. 要计算  $f = (\sqrt{2} - 1)^4$ , 其中取  $\sqrt{2} \approx 1.4$ ,

利用下列等式进行计算:

$$f = f_1 = \frac{1}{(3 + 2\sqrt{2})^3}$$

$$f = f_2 = (3 - 2\sqrt{2})^3$$

问哪一个效果好?

14. 设  $x_1, x_2, x_3, \dots, x_n$  为观测值,公式

$$s^2 = \frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - n\bar{x}^2 \right)$$

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

在数学上是等价的,其中  $\bar{x} = \frac{1}{N} \sum_{i=1}^n x_i$ , 问这两个计算  $s^2$  的公式哪个好?

15. 已知  $p > 0, q > 0, p \gg q$ . 要计算

$$v = -p + \sqrt{p^2 + q}$$

今设计如下两个算法:

〔算法 1〕  $p^2 \rightarrow s$

$$s + q \rightarrow t$$

$$\sqrt{t} \rightarrow u$$

$$-p + u \rightarrow v$$

〔算法 2〕  $p^2 \rightarrow s$

$$s + q \rightarrow t$$

$$\sqrt{t} \rightarrow u$$

$$p + u \rightarrow t$$

$$q/t \rightarrow v$$

试分析两个算法的好坏.

16. 画一框图,可以接受一系列数据组  $(x, y)$ , 并计算下式

$$z = \begin{cases} 3.45x^2 - 0.45xy + 2.06y^2, & \text{当 } x \leq 0; \\ -0.11x^2 + 0.79xy + 1.40y^2, & \text{当 } x > 0. \end{cases}$$

且打印计算结果. 当  $x$  与  $y$  同时为零时, 停止计算.

17. 在字长为十进制二位的计算机上用浮点运算分别从左到右及从右到左计算

$$1 + 0.4 + 0.3 + 0.2 + 0.04 + 0.03 + 0.02 + 0.01$$

试比较所得结果.

18. 利用级数展开方法计算定积分

$$I = \int_0^1 e^{-x^2} dx$$

试分析截断误差, 舍入误差以及为使近似值的相对误差不超过 1%, 至少应取  $n$  位有效数字.

19. 研究如下问题:

建立计算积分

$$I_n = \int_0^1 \frac{x^n}{x+5} dx, \quad n = 0, 1, \dots, 20$$

的递推关系式, 在电子计算机上(假定你已经熟悉在微型计算机上使用 BASIC 语言)实现解题, 分析计算是否稳定, 必要时找出新的对策.

提示: 导出递推关系式

$$\begin{cases} I_n = -5I_{n-1} + \frac{1}{n}, & n = 1, 2, \dots, 20 \\ I_0 = \ln 6 \end{cases}$$

并上机计算. 根据  $I_n$  的特性, 可见到理论分析与计算结果严重不符. 原因是理论值  $I_n$  与实际计算出的值  $\hat{I}_n$  满足关系

$$I_n - \hat{I}_n = -5(I_{n-1} - \hat{I}_{n-1}) = \dots = (-1)^n 5^n (I_0 - \hat{I}_0)$$

采用新的计算方案, 粗略估出  $I_{10}$ , 比如  $I_{10} \approx 0.0087301587$ , 使用新的递推关系式

$$\begin{cases} l_{n-1} = -\frac{l_n}{5} + \frac{1}{5n}, & n = 20, 19, \dots, 1 \\ l_{20} \approx 0.0087301587 \end{cases}$$

再上机计算,可见到计算结果可靠。这时有

$$l_0 - l_n = -\frac{1}{5}(l_1 - l_n) = -\frac{1}{5^2}(l_2 - l_n) = \dots = \frac{(-1)^n}{5^n}(l_n - l_n)$$

20. 设  $x \rightarrow +0$ , 证明:

- 1)  $2x - x^2 = O(x)$ ;
- 2)  $x \sin \sqrt{x} = O(x^{\frac{3}{2}})$ ;
- 3)  $x \sin \frac{1}{x} = O(|x|)$ .

21. 设  $x \rightarrow +\infty$ , 证明:

- 1)  $2x^3 - 3x^2 + 1 = O(x^3)$ ;
- 2)  $\frac{x+1}{x^2+1} = O\left(\frac{1}{x}\right)$ ;
- 3)  $x + x^2 \sin x = O(x^2)$ ;
- 4)  $\frac{\arctg x}{1+x^2} = O\left(\frac{1}{x^2}\right)$

22. 设  $u = [x_1, x_2]^T$ ,  $v = [y_1, y_2]^T$ . 验证

$$(u, v) = x_1 y_1 - x_1 y_2 - x_2 y_1 + 3x_2 y_2$$

是  $R^2$  中的一种内积.

23. 计算向量  $x = [1, -2, 3]^T$  的三种常用范数.

24. 已知准确解  $x$  和近似解  $x^*$  分别为

$$\begin{aligned} x &= [1.0000, 1.0000, 1.0000]^T, \\ x^* &= [1.2001, 0.99991, 0.92538]^T \end{aligned}$$

试计算  $\|x - x^*\|_1$  和  $\|x - x^*\|_\infty$ .

25. 设有向量序列

$$x^{(1)} = \begin{bmatrix} 2.1 \\ 0.9 \end{bmatrix}, x^{(2)} = \begin{bmatrix} 2.01 \\ 0.99 \end{bmatrix}, \dots, x^{(k)} = \begin{bmatrix} 2 + 10^{-k} \\ 1 - 10^{-k} \end{bmatrix}, \dots$$

求  $\lim_{k \rightarrow \infty} x^{(k)}$ .

26. 分别计算下列矩阵  $A, B$  的范数  $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty$ :

$$A = \begin{bmatrix} 10 & 15 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0.6 & 0.5 \\ 0.1 & 0.3 \end{bmatrix}$$

27. 设  $G_1 = \{x | \|x\|_\infty = 1, x \in R^3\}$ , 问  $G_1$  的几何意义是什么? 设  $G_2 = \{x | \|x\|_1 = 1, x \in R^3\}$ , 问  $G_2$  的几何意义是什么?

28. 如果  $\lim_{k \rightarrow \infty} \|x_k\| = \|a\|$ , 是否就有  $\{x_k\}$  收敛于  $a$ ?

29. 若  $\lim_{k \rightarrow \infty} A_k = A$ ,  $\lim_{k \rightarrow \infty} x_k = x^*$ , 则  $\lim_{k \rightarrow \infty} A_k x_k = Ax^*$

30. 设  $n$  阶方阵  $A$  非奇异, 并在  $R^n$  中定义了范数  $\|\cdot\|$ , 则  $\|x\|_A = \|Ax\|$  也是  $R^n$  中的一种范数,  $x \in R^n$ .

## 第2章 FORTRAN 语言的基本概念与基本语句

### 2-1 FORTRAN 导引

FORTRAN 语言是国际上一直很流行的一种高级程序设计语言。FORTRAN 这个名字取自英语 FORMula TRANslator (公式翻译)的头几个字母。FORTRAN 语言特别适用于科学研究和工程设计中的数值计算或称科学计算。

用 FORTRAN 语言编写的 FORTRAN 源程序,由 FORTRAN 编译程序(或称编译系统)将它翻译成机器指令程序,即目标程序,然后由计算机执行。

FORTRAN 语言于 1954 年首先在美国提出,两年后开始使用。其后不断发展,形成了各种文本。大致情况如下:

- 1) 1958 年出现 FORTRAN II;
- 2) 1962 年出现 FORTRAN IV;
- 3) 1966 年公布两个美国国家标准:
  - 标准 FORTRAN, ANSI X3.9—1966 (大致相当于 FORTRAN IV, 有时称 FORTRAN 66);
  - 标准基本 FORTRAN, ANSI X3.10—1966(大致相当于 FORTRAN II);
- 4) 1972 年国际标准化组织 ISO 公布三级标准 FORTRAN 推荐文本:
  - ISO R1539 完全级 FORTRAN (一级,相当于 FORTRAN IV);
  - ISO R1539 中间级 FORTRAN(二级);
  - ISO R1539 基本级 FORTRAN(三级,相当于 FORTRAN II);
- 5) 1976—1978 年间,美国标准化协会又修订了 FORTRAN ANSI X3.9-1966,并重新公布新标准:  
FORTRAN ANSI X3.9—1978,且采用 FORTRAN 77 作为文本的名称。
- 6) 1980 年国际标准化组织公布了 ISO 1539—1980,宣布以 FORTRAN 77 作为国际标准文本。
- 7) 1982 年,中国国家标准总局颁布《程序设计语言 FORTRAN 77》国家标准,采用国际标准 1539—1980 文本的编写格式,并于 1983 年正式施行。

从上述情况可以看出,从 FORTRAN II 到 FORTRAN IV 又到 FORTRAN 77,实际上是一个不断扩充、修正、完善以及标准化的发展过程。其主要特征是,大体上后者与前者兼容,前者是后者的“子集”。只有少数地方,后者扬弃或改进了前者的不合理部分。

目前大多数计算机使用 FORTRAN 语言的情况是:一方面 FORTRAN IV 与 FORTRAN 77 两级文本均可使用;一方面许多现成的程序(如数值计算方面的程序)主要还是用 FORTRAN IV 写成的(这是一笔巨大的财富)。因此,本书关于 FORTRAN 语言的写法是:先用前两章介绍相当于 FORTRAN IV 一级的 FORTRAN 程序设计方法,然后再用一章介绍从 FORTRAN IV 到 FORTRAN 77 的发展。从学习程序设计语言的角度



度说,这种由浅入深的写法也是可取的。

为了区别和叙述方便,书中我们有时使用“标准 FORTRAN IV”或“在 FORTRAN IV 中”这样一些话,这时,我们基本上是指国际标准化组织公布的 ISO R1539 完全级 FORTRAN。

但必须指出,几乎没有一个实际使用的 FORTRAN 编译程序,是能够完完全全,不折不扣地的按照某一个标准文本来设计的。在特定的计算机上实现时,总是根据计算机的具体配置作一定的限制或扩充。因此,为具体计算机编写 FORTRAN 程序以及上机计算时,我们还要注意参考所用机器的“FORTRAN 语言文本”和“上机操作手册”。

下面我们介绍 FORTRAN 语言规则及其程序设计方法。与许多教科书的做法一样,我们从分析一个简单的 FORTRAN 程序开始。

仍以求二次方程

$$ax^2 + bx + c = 0 \quad (a > 0)$$

的两个实根为例。如果  $d = b^2 - 4ac \geq 0$ , 则两个实根为:

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = -\frac{1}{2} \left( \frac{b}{a} + \sqrt{\left(\frac{b}{a}\right)^2 - \frac{4c}{a}} \right);$$

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = -\frac{b}{a} - x_1,$$

其中公式的变形是为了省去一些重复的计算,在程序设计中就常常要考虑这一点。现在假定方程的三个系数  $a, b, c$  ( $a > 0$ ) 从卡片读入,计算结果  $x_1$  和  $x_2$  从打印机输出。我们可以编写 FORTRAN 程序如下:

```
C      FINDING THE REAL ROOTS OF QUADRATIC EQUATION
C      READ(5,100)A,B,C
      100  FORMAT(3F8.2)
      D = B*B - 4.0*A*C
      IF(D .LT. 0.0) STOP
      T = B/A
      X1 = -0.5*(T + SQRT(T*T - 4.0*C/A))
      X2 = -T - X1
      WRITE(6,200) X1, X2
      200  FORMAT(1X,F8.2,10X,F8.2)
      STOP
      END
```

如果我们要求解的二次方程不止一个,而是若干个。那么,我们可以写一个子程序,描述求解二次方程实根的主要计算过程;又写一个主程序,根据不同方程的三个不同系数,多次调用 (CALL) 子程序,求出各个方程的实根。

还是假定每个方程的三个系数  $a, b, c$  ( $a > 0$ ) 分别穿孔在每一张卡片上,即数据将从卡片读入。因为不知道将要处理的是多少个方程,我们在数据卡片的最后增加一张穿

有第一个数不为零，其它两个数任意的卡片作为结束标志。读入每一张数据卡片后，首先检查第一个数据是否为零。如果为零，则表示方程已求解完毕，否则又读入新的一组数据，继续求解。在子程序中，如果遇到判别式  $d = b^2 - 4ac < 0$ ，即无实根的情形，我们约定，这时将两个实根都赋予数值 99999.99 作标志，也就是说，当输出的两个实根都是 99999.99，即意味着此时并无实根存在。

现在把 FORTRAN 程序编写如下：

主程序：

```
C      MAIN PROGRAM
      10  READ(5,100)A1,B1,C1
      100 FORMAT(3F8.2)
      IF (A1. EQ. 0.0) GO TO 50
      CALL QERR(A1,B1,C1,R1,R2)
      WRITE(6,200) R1, R2
      200 FORMAT(1X,F8.2,10X,F8.2)
      GO TO 10
      50  STOP
      END
```

子程序

```
C      FINDING THE REAL ROOTS OF QUADRATIC EQUATION
      SUBROUTINE QERR(A,B,C,X1,X2)
      D = B*B - 4.0*A*C
      IF(D .LT. 0.0) GO TO 15
      T = B/A
      X1 = -0.5*(T + SQRT(T*T - 4.0*C/A))
      X2 = -T - X1
      RETURN
      15  X1 = 99999.99
      X2 = 99999.99
      RETURN
      END
```

把编写好的 FORTRAN 源程序，每行穿一张卡片，连同所需的初始数据，按程序要求，每个方程的三个系数也穿一张卡片（程序和数据用的卡片如图 2-1 所示）。这样，按照所使用计算机的操作方法，我们就可上机计算了。

通过这个例子，我们可以分析 FORTRAN 程序的一些基本特征：

1) FORTRAN 程序是块状结构。每一个 FORTRAN 源程序或者只由一个主程序块组成，或者由一个主程序块和一个或多个子程序块组成。程序块也称程序单位。子程

序也称副程序或辅程序。在一个 FORTRAN 程序中,程序块的前后次序并没有严格规定,也没有要求主程序块一定放在整个程序的最前面。

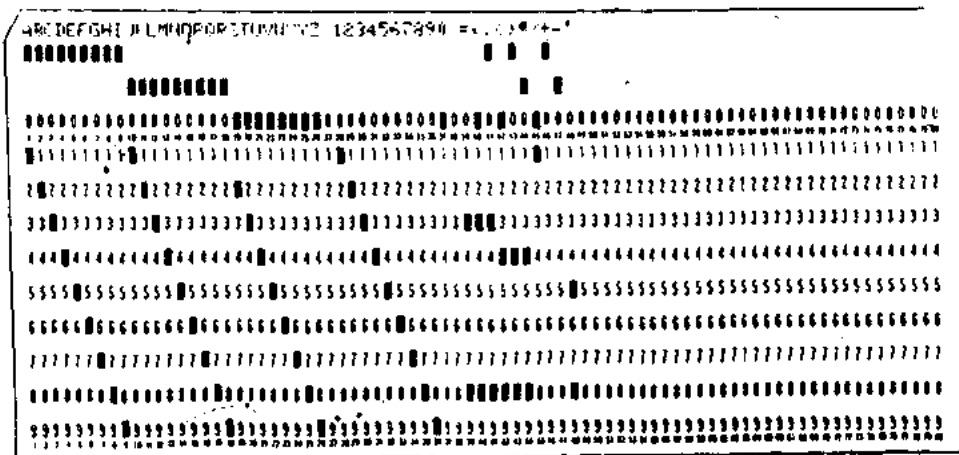


图 2-1 程序卡片与数据卡片

2) FORTRAN 原是基于卡片的语言,因此它有严格的书写格式。FORTRAN 源程序应按规定格式写在专门的 FORTRAN 程序纸上。程序纸的横方向格式与卡片的横方向格式一致,都有 80 列(即 80 格),如图 2-2。通常,并不是都有专门的程序纸可供使用,只要按程序纸规定的格式书写程序就行了。程序纸每一行的每一格书写一个字符;每一行又分成三个区,前 5 格为“标号区”,第 6 格为“续行标志区”,第 7 格至 72 格为“语句区”,第 73 格至第 80 格用于写程序序号或作其它附注。标号区书写语句标号,标号是从 1 到 99999 之间的任何不带符号的整数,其数值本身没有什么意义,也不代表先后次序;不到 5 位数的标号可以写在 1—5 格中的任何位置上;同一程序块中不同语句不能有相同标号,不同程序块中标号可以相同,但一个程序块不能引用另一个程序块中的标号,这就是说,标号只在本程序块中有效。

3) 一个 FORTRAN 源程序大致包括注解行,语句行和结束行三部分。如果程序纸的第一格写上字母 C,则这行就是注解行。它不是程序的实质性部分,而是程序员为方便阅读程序所加的标题注解或文字说明。注解内容可以从 C 后的任何一格开始写,一行纸写不完可分多行纸写,但每一行都必须在该行纸的第一格写上字母 C。语句行写在语句区中,一行最多只能写一个语句。若一个语句在一行中写不下,可以继续写到下一行的语句区,这个下一行相对于前一行来说称为续行,而原第一行称为起始行。起始行的第 6 格应留空或写上数字 0,而续行的第 6 格应写上一个非 0 或非空格符号;一个语句最多允许分成 20 行书写。结束行只需在语句区中写上 END,它的前 6 格不写任何字符。每一个程序块都必须有一结束行 END,用以通知编译程序,这一程序块至此结束。

这里,我们需要强调指出的是,随着计算机科学技术的发展,在现代计算机系统上使用 FORTRAN 语言时,除了可以通过用卡片输入源程序和数据外,更方便的方法是通过用户终端的键盘直接输入源程序和数据。其中数据可以通过人-机对话方式现场提供,也可以事先存储在磁盘上(作成数据文件),供运行 FORTRAN 程序过程中需要时再输入。就本质而言,卡片输入和键盘输入是一样的。因此,为了叙述方便,下面我们常常以卡片

# FORTRAN CODING SHEET

华南工学院计算机中心  
(C.C. of S.C.I.T.)

Program		Page		of		Note	
Programmer		Type		Date			
Statement	Label	FORTRAN STATEMENT					
1	5	10	15	20	25	30	35
		40	45	50	55	60	65
		70	75	80	85	90	95
		100	105	110	115	120	125
		130	135	140	145	150	155
		160	165	170	175	180	185
		190	195	200	205	210	215
		220	225	230	235	240	245
		250	255	260	265	270	275
		280	285	290	295	300	305
		310	315	320	325	330	335
		340	345	350	355	360	365
		370	375	380	385	390	395
		400	405	410	415	420	425
		430	435	440	445	450	455
		460	465	470	475	480	485
		490	495	500	505	510	515
		520	525	530	535	540	545
		550	555	560	565	570	575
		580	585	590	595	600	605
		610	615	620	625	630	635
		640	645	650	655	660	665
		670	675	680	685	690	695
		700	705	710	715	720	725
		730	735	740	745	750	755
		760	765	770	775	780	785
		790	795	800	805	810	815
		820	825	830	835	840	845
		850	855	860	865	870	875
		880	885	890	895	900	905
		910	915	920	925	930	935
		940	945	950	955	960	965
		970	975	980	985	990	995
		1000	1005	1010	1015	1020	1025
		1030	1035	1040	1045	1050	1055
		1060	1065	1070	1075	1080	1085
		1090	1095	1100	1105	1110	1115
		1120	1125	1130	1135	1140	1145
		1150	1155	1160	1165	1170	1175
		1180	1185	1190	1195	1200	1205
		1210	1215	1220	1225	1230	1235
		1240	1245	1250	1255	1260	1265
		1270	1275	1280	1285	1290	1295
		1300	1305	1310	1315	1320	1325
		1330	1335	1340	1345	1350	1355
		1360	1365	1370	1375	1380	1385
		1390	1395	1400	1405	1410	1415
		1420	1425	1430	1435	1440	1445
		1450	1455	1460	1465	1470	1475
		1480	1485	1490	1495	1500	1505
		1510	1515	1520	1525	1530	1535
		1540	1545	1550	1555	1560	1565
		1570	1575	1580	1585	1590	1595
		1600	1605	1610	1615	1620	1625
		1630	1635	1640	1645	1650	1655
		1660	1665	1670	1675	1680	1685
		1690	1695	1700	1705	1710	1715
		1720	1725	1730	1735	1740	1745
		1750	1755	1760	1765	1770	1775
		1780	1785	1790	1795	1800	1805
		1810	1815	1820	1825	1830	1835
		1840	1845	1850	1855	1860	1865
		1870	1875	1880	1885	1890	1895
		1900	1905	1910	1915	1920	1925
		1930	1935	1940	1945	1950	1955
		1960	1965	1970	1975	1980	1985
		1990	1995	2000	2005	2010	2015
		2020	2025	2030	2035	2040	2045
		2050	2055	2060	2065	2070	2075
		2080	2085	2090	2095	2100	2105
		2110	2115	2120	2125	2130	2135
		2140	2145	2150	2155	2160	2165
		2170	2175	2180	2185	2190	2195
		2200	2205	2210	2215	2220	2225
		2230	2235	2240	2245	2250	2255
		2260	2265	2270	2275	2280	2285
		2290	2295	2300	2305	2310	2315
		2320	2325	2330	2335	2340	2345
		2350	2355	2360	2365	2370	2375
		2380	2385	2390	2395	2400	2405
		2410	2415	2420	2425	2430	2435
		2440	2445	2450	2455	2460	2465
		2470	2475	2480	2485	2490	2495
		2500	2505	2510	2515	2520	2525
		2530	2535	2540	2545	2550	2555
		2560	2565	2570	2575	2580	2585
		2590	2595	2600	2605	2610	2615
		2620	2625	2630	2635	2640	2645
		2650	2655	2660	2665	2670	2675
		2680	2685	2690	2695	2700	2705
		2710	2715	2720	2725	2730	2735
		2740	2745	2750	2755	2760	2765
		2770	2775	2780	2785	2790	2795
		2800	2805	2810	2815	2820	2825
		2830	2835	2840	2845	2850	2855
		2860	2865	2870	2875	2880	2885
		2890	2895	2900	2905	2910	2915
		2920	2925	2930	2935	2940	2945
		2950	2955	2960	2965	2970	2975
		2980	2985	2990	2995	3000	3005
		3010	3015	3020	3025	3030	3035
		3040	3045	3050	3055	3060	3065
		3070	3075	3080	3085	3090	3095
		3100	3105	3110	3115	3120	3125
		3130	3135	3140	3145	3150	3155
		3160	3165	3170	3175	3180	3185
		3190	3195	3200	3205	3210	3215
		3220	3225	3230	3235	3240	3245
		3250	3255	3260	3265	3270	3275
		3280	3285	3290	3295	3300	3305
		3310	3315	3320	3325	3330	3335
		3340	3345	3350	3355	3360	3365
		3370	3375	3380	3385	3390	3395
		3400	3405	3410	3415	3420	3425
		3430	3435	3440	3445	3450	3455
		3460	3465	3470	3475	3480	3485
		3490	3495	3500	3505	3510	3515
		3520	3525	3530	3535	3540	3545
		3550	3555	3560	3565	3570	3575
		3580	3585	3590	3595	3600	3605
		3610	3615	3620	3625	3630	3635
		3640	3645	3650	3655	3660	3665
		3670	3675	3680	3685	3690	3695
		3700	3705	3710	3715	3720	3725
		3730	3735	3740	3745	3750	3755
		3760	3765	3770	3775	3780	3785
		3790	3795	3800	3805	3810	3815
		3820	3825	3830	3835	3840	3845
		3850	3855	3860	3865	3870	3875
		3880	3885	3890	3895	3900	3905
		3910	3915	3920	3925	3930	3935
		3940	3945	3950	3955	3960	3965
		3970	3975	3980	3985	3990	3995
		4000	4005	4010	4015	4020	4025
		4030	4035	4040	4045	4050	4055
		4060	4065	4070	4075	4080	4085
		4090	4095	4100	4105	4110	4115
		4120	4125	4130	4135	4140	4145
		4150	4155	4160	4165	4170	4175
		4180	4185	4190	4195	4200	4205
		4210	4215	4220	4225	4230	4235
		4240	4245	4250	4255	4260	4265
		4270	4275	4280	4285	4290	4295
		4300	4305	4310	4315	4320	4325
		4330	4335	4340	4345	4350	4355
		4360	4365	4370	4375	4380	4385
		4390	4395	4400	4405	4410	4415
		4420	4425	4430	4435	4440	4445
		4450	4455	4460	4465	4470	4475
		4480	4485	4490	4495	4500	4505
		4510	4515	4520	4525	4530	4535
		4540	4545	4550	4555	4560	4565
		4570	4575	4580	4585	4590	4595
		4600	4605	4610	4615	4620	4625
		4630	4635	4640	4645	4650	4655
		4660	4665	4670	4675	4680	4685
		4690	4695	4700	4705	4710	4715
		4720	4725	4730	4735	4740	4745
		4750	4755	4760	4765	4770	4775
		4780	4785	4790	4795	4800	4805

输入为例。

## 2.2 FORTRAN 的一些基本概念

### 1. FORTRAN 字符集

在 ISO 公布的完全级 FORTRAN 中,允许使用的字符有:

字母: A, B, C, ..., X, Y, Z (26 个大写字母);

数字: 0, 1, 2, ..., 9 (10 个数字);

符号: “ ” (空 白),

= (等 号),

+ (加 号),

- (减 号),

\* (星 号),

/ (斜 线),

( (左括号),

) (右括号),

, (逗 号),

. (小数点),

“币符”(货币符)。

其中,需要补充说明的是“ ”表示手写的空格,在计算机实际显示或打印时是真正的空白;“币符”在美国标准中通常采用美元符号\$。

### 2. 数据类型·常数与变量

计算机将数据划分为不同的类型来进行加工处理,每种类型都有不同的数学意义和内部表示。

FORTRAN 定义六种不同的数据类型,这就是:

- 整型;
- 实型;
- 双精度型;
- 复型;
- 逻辑型;
- 字符型 (Hollerith 型)。

下面讲常数。

由直接出现在 FORTRAN 程序中的数字所描述的数据就是常数。常数是在程序执行过程中不发生变化的量。这一节我们先讨论整型常数和实型常数,也称整常数和实常数。

整型常数表示一个整数值,可带正号或负号(正号还可省略),但不允许出现小数点、逗号或指数。

例如,下列各数

25    0    1    -3675    -1

都是整型常数。但是,下列各数

25.5    5,000    -1.0    49E4    0.0

都不是整型常数。

整型常数在计算机中用定点表示。不同的计算机系统对于整型数的数值范围有不同的规定。例如,有的机器用16位二进制位来表示整型常数,其中第一位用以表示符号,后15位表示数值。因此,所能表达的整型常数范围是 $-2^{15} \sim 2^{15}-1$ ,即 $-32768 \sim 32767$ 。又例如有的机器用32位二进制位来表示整型常数,这时的表示范围是 $-2^{31} \sim 2^{31}-1$ ,即 $-2147483648 \sim 2147483647$ 。整型常数总是整数值的确切表示,这里是没有误差的。

实型常数也称浮点数,是指带小数点和(或)带指数(用符号E表示取10的幂而且要求按一定的格式)的常数。

例如,下列各数

3.1416    5.    -7.34    0.0    -8.12E-5

都是实型常数。但是,下列各数

-405    5    1.5E    E+8    0

则不是实型常数。一般地,若记  $m, n$  为非空数字串,  $s$  是不带符号的整数,如  $m.nE-s$  表示  $m.n \times 10^{-s}$ , 则实型常数可以取下列形式的任何一种:

$\pm m.n$      $\pm m.$      $\pm .n$  基本实常数;

$\pm m.nE \pm s$      $\pm m.E \pm s$      $\pm .nE \pm s$  基本实常数后带十进制指数;

$\pm mE \pm s$  整数后带十进制指数,

其中+号也可省略不写。

实型常数如5.0和5.,它们与整型常数5的数值虽然是相同的,但在计算机中的存储方式却是不同的。实型常数在计算机中用浮点表示。不同的计算机系统对实型数的范围和有效位都有不同的规定。例如,有的规定范围为 $|R| \leq 10^7$ ,有效位数6位;有的规定范围为 $|R| \leq 10^8$ ,有效位数7位。

采用浮点表示可以扩大数的表示范围,还可以保证一定的有效位数。如数1234567000,用整型数表示不出来,可用实型数表示为0.1234567000 E10。实型常数是实数值在加工程序表示的意义下的近似值,所以有时会有一些误差。因此,只要在允许的场合,可尽量采用整型常数表示。

通常, FORTRAN 规定这样一种“浮点数的规格化形式”,其基本实常数部分小于1,且小数点后第一位不为零,如  $0.453042E-8$ 。是 FORTRAN 可以接受任何一种指数形式表示的数,但在输出时,则一律以规格化形式输出。

现在讲变量。

在程序设计语言中,变量是指由符号名字标识的数据。变量的名字简称变量名。对程序中出现的一个变量,计算机将在内存中开辟对应的存储单元,并通过名字(即变量名)来使用该单元。所谓调用某变量,实际上就是调用该变量对应的存储单元中的数据;说变量的值在程序执行过程中是可变的,就是说对应该变量的存储单元中的数据是可变的。给变量赋一个新值时,这个新值便取代原先储存在该对应单元中的旧值。

在 FORTRAN 中,每一个变量的名字在变量所在的程序块中是唯一的。FORTRAN

还规定变量名必须以字母开头,后面可以没有或有一个或多个字母或数字。不同的 FORTRAN 编译程序对变量名的有效位数有不同的规定,常见的有五位和六位。

下面是变量名的例子:

SUM Y X1 ROOT1 ALFA TU I2

下列符号则不能作变量名:

3DS \$200  $\alpha$  C1.2 U.S.A. A-N

最好选择与变量本身意义相关的变量名,这能增加程序的可读性。

由于常数分整型和实型,与此对应,变量也有整型变量和实型变量(还有双精度型变量、复型变量、逻辑型变量和字符型变量,详见下两章)。它们分别用来表示整型和实型常数。

为了把程序中的变量的类型告诉编译程序,必须对变量类型作出说明。整型和实型变量的类型说明有两种方法:

1) 隐式说明,即凡没有专门说明,而以 I,J,K,L,M,N 六个字母开头的变量名,均是整型变量,否则,均为实型变量。这种方式又称为 I-N 规则。

2) 显式说明,即用专门的类型说明语句 INTEGER 说明整型变量,用 REAL 说明实型变量。例如

INTEGER Z, C1, TIME

REAL J, K, N, MIN

前者表示在本程序块中,变量 Z, C1, TIME 被说明为整型变量;后者表示在本程序块中,变量 J, K, N, MIN 被说明为实型变量。需要注意的是,第二个语句中仅说明所列出的几个变量为实型,而并非指其它 J, K, N, M 开头的变量也被定义为实型。

### 3. 算术表达式与标准函数

先讲算术表达式。

FORTRAN 使用五个算术运算符:

+(加) -(减) \*(乘) /(除) \*\* (乘方)

其中“-”号也用于取负,即变号。

FORTRAN 的算术表达式由算术运算符、括号和数值型运算元素(如整型和实型常数、变量、标准函数调用等)构成。例如:

X + Y

I - 4 \* J

MAX \* L + MIN

1.0 / (A \*\* 2 + Y \*\* 3)

- 0.5 \* (T + SQRT(T \* T - 4.0 \* C / A))

0.934587E - 9 / (CC + DD)

显然,单独一个常数、一个变量或一个函数调用也可作为一个表达式看。

算术表达式指定了一个数值计算的规则,经过运算后可以得到一个数值。算术运算

的优先次序是:

括号→函数→乘方→取负→乘和除→加和减

FORTRAN 强调算术表达式中运算元素的类型要匹配。标准的 FORTRAN IV 规定,一个算术表达式中的所有运算元素必须同一类型,只有一个例外,实型表达式的指数既可以是实型,也可以是整型。考察下列表达式

$$0.5 * A + I - K * * N$$

它包括实型和整型两种运算元素,因而是允许的。

然而,在许多实际运行的 FORTRAN 编译系统中,还是允许不同类型的运算元素进行某种混合运算。例如

$$(1.03 * X + I) + J - Y$$

还是允许的。这时编译程序会先把它们转换成同一类型,然后再进行运算。转换的规则是:

1) 同类型的运算元素运算后保持原类型。其中要注意的是,整数相除和整数的整次方,其结果是舍去小数取整,如  $13/4$  得 3,  $1/2$  得 0,  $2 * * (-1)$  得 0。因此在不愿意丢掉小数的场合,可改写成实数运算,如  $13.0/4.0$  得 3.25,  $1.0/2.0$  得 0.5。另外,还必须注意整数运算的顺序,如  $5 * 6/3$  得 10, 可是  $5/3 * 6$  得 6。最好不要轻易使用整数相除和整数的整次方。如  $X^{1/2}$  应写成  $X * * 0.5$ ,  $\frac{\sin x}{2}$  应写成  $0.5 * \sin(X)$  或  $\sin(X)/2.0$

2) 对不同类型的运算元素,先将低级的运算元素转换成高级的运算元素,然后再将相同类型的运算元素进行运算。就整型和实型来说,整型低级,实型较高级。如  $2.4 * 5$ , 先将 5 化为 5.0,再计算  $2.4 * 5.0$  得 12.0。

不同类型的运算元素其运算的结果的类型如表 2-1 和表 2-2 所示。

表 2-1

表 2-2

+-*/	整型	实型	**	指数整型	指数实型
整型	(整)	(实)	基数整型	(整)	(实)*
实型	(实)*	(实)	基数实型	(实)	(实)

表中带\*的栏表示标准 FORTRAN IV 不允许的混合运算。

我们指出,在写表达式时,还要注意由于有效位数有限而引起的误差以及数值是否可能产生溢出的问题。一般的原则是,避免两个相差很小的量直接相减(即第 1 章中所说的避免严重丢失有效数字);避免两个相差甚大的量直接相加减(即第 1 章中所说的防止大数吃掉小数);避免两个数值很大的量相乘,以防止运算结果超出机器所能表示的范围而溢出。

下面讲标准函数。

把计算实践中经常遇到的计算或操作,诸如三角函数计算、指数与对数计算、开方计算、取绝对值、取整等等,预先作出规定,写成例行式的程序,作为语言的组成部分,配备在



编译程序及其程序库中,供程序员在源程序中直接引用,这就是标准函数的概念。

标准 FORTRAN IV 规定了 55 个标准函数。这 55 个标准函数被分成内部函数和基本外部函数两类。这种区分更多地是为了规定编译程序实现这些函数的方式,对程序员几乎不必介意。

对内部函数,编译程序在加工源程序遇到它们时,每次都是就地编出几条机器指令来实现其功能,因而执行速度一般是较快的。对基本外部函数,凡相同者,均只列入目标程序一次,形成自己独立的一个子程序块。当要计算某基本外部函数时,目标程序便将控制转移给相应的子程序块,子程序块执行完毕,则自动返回主程序。当然,实际上程序员并不觉察这两类函数之间的区别。

标准 FORTRAN IV 的 55 个标准函数详见本章附录 A,其中较常用的如下表 2-3。

表 2-3 标准函数表(较常用部分)

函数定义	函数名字	自变量个数	自变量类型	函数类型	内部或外部函数
绝对值 $ x $	ABS IABS	1	实 整	实 整	内 内
取整 $[x]$	AIN INT	1	实 实	实 实	内 内
取最大值 $\max(x_1, x_2, \dots)$	AMAX0 AMAX1	$\geq 2$	整 实	实 实	内 内
取最小值 $\min(x_1, x_2, \dots)$	AMIN0 AMIN1	$\geq 2$	整 实	实 实	内 内
浮点化	FLOAT	1	整	实	内
定点化	IFIX	1	实	整	内
指数 $e^x$	EXP	1	实	实	外
自然对数 $\log_e x$	ALOG	1	实	实	外
普通对数 $\log_{10} x$	ALOG10	1	实	实	外
正弦 $\sin x$	SIN	1	实	实	外
余弦 $\cos x$	COS	1	实	实	外
反正切 $\arctan x$	ATAN	1	实	实	外
平方根 $\sqrt{x}$	SQRT	1	实	实	外

标准函数的调用(简称函数调用)是:写出函数名,接着是圆括号,自变量放在圆括号内,有多个自变量时用逗号隔开。要严格按照规定的类型调用。自变量可以是表达式,也可以包含其它函数,甚至包括这个标准函数本身,如

SQRT(ALOG(1. + SQRT(X\*\*2 + Y\*\*2)))

这时要特别注意表达式的类型应与规定相符。此外,调用标准函数一般还要遵守数学上的基本规定,如三角函数的自变量是弧度,负数不能开方,负数和零不能求对数等等。

#### 4. 数组与下标变量

上面所讲的变量都比较简单,通常称为简单变量。实际计算中还会遇到更复杂的情形。例如,三维空间中的一个向量

$$r = (x, y, z)$$

有三个分量;一个  $n$  次多项式

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

有  $n+1$  个系数; 一个  $m \times n$  矩阵  $A$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

有  $m \times n$  个元素, 等等。数组的概念就是这样引进来的。它用一个名字来统一代表一组数据, 再用顺序号来区分该组数据中的每一个具体的数据。数组反映了计算机内大批数据顺序存放的特点。因此, 在程序中使用数组之前, 应该先说明数组的名字、类型和大小。

在 FORTRAN 中, 数组名的规定与变量名的规定相同, 也服从隐式规则。数组说明有两种方式:

一是用类型说明语句直接指出该数组包含多少个数据。例如

```
REAL R(3), X(10)
```

```
INTEGER K(5)
```

说明两个实型数组  $R, X$ , 一个整型数组  $K$ , 其中数组  $R$  包含三个数据  $R(1), R(2), R(3)$ , 数组  $X$  包含 10 个数据, 从  $X(1)$  到  $X(10)$ , 数组  $K$  包含 5 个数据, 从  $K(1)$  到  $K(5)$ 。每一个数据称为该数组的元素或下标变量。下标就是指明顺序的那个数, 它对应于普通数学公式中的脚标。

二是用专门的数组维数说明语句, 即通过 DIMENSION 语句来规定。DIMENSION 语句的一般形式为

```
DIMENSION <数组说明符>, <数组说明符>, ...
```

其中每一个<数组说明符>包括数组名、紧接一对圆括号以及写在括号中的数组元素个数。例如

```
DIMENSION R(3), X(10), K(5)
```

这时, 如果没有其它语句说明  $R, X, K$  的类型的话, 则根据 I-N 规则, 这个语句就说明  $R$  和  $X$  分别是三个元素和 10 个元素的实型数组,  $K$  是 5 个元素的整型数组。需要注意的是, 除实型和整型之外, 其它类型的数组 (双精度型、复型、逻辑型等) 都必须在 DIMENSION 语句的基础上再加上相应的类型说明语句, 只是在类型说明语句中无须在数组名字之后再写上圆括号及其中的元素个数, 详见下一章。

无论是用类型说明语句直接说明数组, 还是通过 DIMENSION 语句加上类型说明语句说明数组, 其中总是认为数组下标的第一个值 (即下标下界) 自动为 1, 而下标的最大值 (即下标上界) 等于数组中元素的个数, 也即说明语句中放在括号中的那个数。运行程序时, 如果下标的数值小于下界或大于上界, 就会产生“数组下标越界”的动态错误。

只有一个下标的数组叫一维数组, 有两个下标的数组叫二维数组。标准 FORTRAN IV 允许最大有三维数组。例如, 为了表示一个  $3 \times 5$  阶实矩阵  $A$ , 可以说明一个二维数组  $A$  如下:

```
DIMENSION A(3,5)
```

或

REAL A(3,5)

其中,下标变量如 A(2,3) 对应于元素  $a_{23}$ , 如 A(3,4) 对应于元素  $a_{34}$ , 等等。

二维数组虽然可以理解成一个长方形,但 FORTRAN 规定,它在计算机内部的存储顺序是“按列存放”的。如上述数组 A, 其元素的存储顺序是

A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2)

A(1,3) A(2,3) A(3,3) A(1,4) A(2,4) A(3,4)

A(1,5) A(2,5) A(3,5)

对于三维数组,其存储顺序也依此类推。在程序设计中,当准备输入数据和阅读输出结果时,要注意数组元素的存放排列规定。

还要注意的,在 FORTRAN IV 中,数组说明符的下标上界值,在主程序中只允许是正整数,只有在子程序内才能使用变量,而且数组名和说明其上界的整变量名都必须是子程序的所谓哑元。这叫做可调数组,我们将在下一章详细讨论。

当引用下标变量时,下标除了要求是正整数外,可以是变量或表达式,只须这些变量或表达式的值是正整数。实际计算中把数组下标写成什么样的式子,标准 FORTRAN IV 规定是严格的,它只允许使用形如

$c * v \pm k$

$c * v$

$v \pm k$

$v$

$k$

的表达式,其中  $c$  和  $k$  是整常数,  $v$  是整变量。例如 A(K), A(3 \* I + 6), A(K - 4) 等都是形式上正确的下标变量,而 A(I - J), A(I \* N + 6) 等都是不允许的。这种限制对于程序设计颇为不便。因此,有些编译程序以及后来的 FORTRAN 文本(例如 FORTRAN 77),均对下标表达式的限制作了相当的放宽,允许使用任何整型表达式,甚至允许使用任何类型的算术表达式,取其最后结果的整数部分。

## 2-3 赋值语句·停语句与暂停语句

FORTRAN 语句分两大类:可执行语句和非执行语句。可执行语句直接用来实现计算、处理和输入/输出,通常包括赋值语句、控制语句和输入/输出语句等。非执行语句用来把有关变量类型、数据初值、印刷格式、程序结构和编译方式等信息通知编译程序,通常包括说明语句、格式语句、数据初值语句、语句函数定义语句和子程序语句等。

FORTRAN 的赋值语句又分算术赋值、逻辑赋值和标号赋值三种。这一节我们先介绍算术赋值语句,并且为了以后举例方便,还将介绍停语句(STOP)和暂停语句(PAUSE)。

### 1. 算术赋值语句

在数值计算程序中,数值计算的工作是交给算术赋值语句来执行的。算术赋值语句的一般形式为

〈变量〉=〈算术表达式〉

其中〈变量〉是指整型、实型(以后会看到, 还可包括双精度型、复型等)的简单变量或下标变量。执行这个语句时, 首先计算右端算术表达式, 然后把得到的数值赋给赋值号=左端的变量, 即送入该变量的存储单元中。

自然, 赋值语句中涉及的变量必须事先作必要的类型说明, 包括用 I-N 规则的隐式说明和下标变量的数组维数说明。此外, 表达式中出现的各个量, 在执行这个语句之前也都必须已经具有确定的数值。

下面给出几个 FORTRAN 算术赋值语句的例子:

```
X = Y * Z - 6.95E - 7
AREA = SQRT(S * (S - A) * (S - B) * (S - C))
SUM = SUM + A(I) * * 2
DC(K) = B(K, 1) + X * (B(K, 2) + X * (B(K, 3) + X * B(K, 4)))
```

需要强调的是, 赋值号并非等号, 如  $N = N + 1$ , 其意义是取出  $N$  所代表的存储单元内的数值, 加上整数 1, 再送回  $N$  中去。又如  $X = 1/X$ , 其意义是取  $X$  的当前数值, 求出倒数再送回  $X$  中去。

显然, 算术赋值语句中赋值号右端的表达式和左端的变量又存在类型是否一致的问题。FORTRAN IV 中允许混合类型赋值, 即在算出表达式之后, 按照一定的规则进行数值类型转换, 然后赋给左端的变量。这种规则对整型和实型情形如表 2-4 所示。

表 2-4

左端变量类型	右端表达式类型	赋值规则
整型	整型	照赋, 结果整型
整型	实型	取整再赋, 结果整型
实型	整型	化实再赋, 结果实型
实型	实型	照赋, 结果实型

下面再给出几个赋值语句的例子:

```
A = 3.0
B = 1.3
C = A * B
TIME = T(I)
HLMOD = 20.0 * ALOG10(HMOD)
PHI = PHI * 180.0 / 3.14159
Y1(1, 1) = Y2(1, 1)
Y1(1, 2) = -Y2(1, 2) - Y2(2, 1)
NO = IFIX(1 + (TSTOP - TSTART) / TINC)
```

然而, 把下列式子作为 FORTRAN 的赋值语句, 则是错误的:

```
SA1 + SA2 = TOTAL
-DIST = VELOC * (-TIME)
```

$DIST = VELOC * TIME$

$C ** 2 = A ** 2 + B ** 2$

$COS(X) = AL / SQRT(AL ** 2 + BE ** 2 + GA ** 2)$

## 2. 停语句 STOP 与暂停语句 PAUSE

停语句与暂停语句属于控制方面的语句。

完成计算之后,要控制计算过程停下来、可使用停语句,其一般形式为

`STOP XXXXX`

其中XXXXX是一至五位的八进制数,可写可不写。执行停语句之后,程序不能再继续往下执行。必要时只能从头开始,重算一遍。实际执行到一个停语句时,计算机的打印机或显示设备将把整个停语句连同 STOP 后面的数字一起印出来,使操作人员了解停止计算的地方。如果程序中有多处停语句,就可用加在 STOP 后面的不同数字来区别它们。在某些非标准的 FORTRAN IV 以及 FORTRAN 77 中,还允许 STOP 后面不仅可以写任意的数字,而且可以写字符串。

一般情况下,在程序正常结束的地方必须写上停语句。虽然在特殊情况下,例如在周而复始没完没了地执行的程序中,停语句并非必要部分,没有它在语法上也并非有错。但如果由于疏忽而漏写停语句,就有可能在运行程序中引起难以预料的失误。对于习惯写 BASIC 程序者尤需注意。

此外,还需注意 STOP 与 END 的区别。在 FORTRAN IV 中,END 不是语句而称为结束行,它仅仅是每一个程序块的结束标志,决无停机之意。每一个主程序块或子程序块都需要有结束行 END,以通知编译程序“本块结束”。

如果不是只希望计算过程停下了事,而是希望计算过程暂停以后,需要的话,能从停止的地方继续执行下去。这时,可用暂停语句,其一般形式为

`PAUSE XXXXX`

其中XXXXX是八进制数字,其功用与 STOP 中八进制数字的功用相同。同样,在某些非标准的 FORTRAN IV 以及 FORTRAN 77 中,也允许XXXXX写成任意的数字或字符串。

暂停语句常用于下列情况:

- 1) 借助暂停语句,可以给人工操作,如安装磁盘、磁带、开打印机等提供机会。
- 2) 利用暂停语句,可以把大型程序,特别是运行时间比较长的程序分成若干段落来处理。
- 3) 调试程序时插入一些暂停语句,可以了解程序执行到什么地方,孤立出有问题的区间。

必须指出,“停”和“暂停”的实现方式往往随计算机而异。一种情况是,停和暂停都是使计算机真正“停机”;停机之后,可以通过按“启动”键来继续计算。另一种情况是,现代计算机系统总是运行不止的,一个程序中的停语句只是停止它本身的执行,返回到操作系统,其中的调度程序立即为计算机分配新任务。“暂停”则往往是把程序“挂”起来,操作系统转去处理其它程序。等到需要继续执行程序时(通常是向操作系统发出请求指令),操

作系统又回头从挂起点继续执行这个程序。总之，停和暂停语句的执行涉及到整个计算机系统的管理和运行方式，必须查阅所用计算机的操作手册。

## 2-4 输入/输出初步

### 1. 概述

计算机的各种输入/输出，在 FORTRAN 中用读/写语句 (READ 和 WRITE) 来实现。

“读”和“写”是相对于计算机作为主体来说的。“读”可能是从卡片阅读器输入卡片上的信息，从控制台或终端机键盘输入数据或命令，从显示屏取回光笔给出的信息，从通信设备接收信号，从磁盘、磁带取入数据等等。“写”可能是控制台或终端机荧光屏显示，快速打印机打印，把数据送到磁盘、磁带，输出穿孔卡片，在绘图仪上绘图，向通信设备发信号，等等。可见读/写语句与计算机及其配套设备有关。一般来说，是很难全部标准化的。FORTRAN 语言的特色之一就是相当成功地实现了读/写语句的标准化。但就各种具体机器的 FORTRAN 文本而言，其输入/输出的非标准成分仍难以避免，因此，使用读/写语句时，需随时核对所用计算机的使用文本。

容易理解，读/写语句必须包括如下三方面的内容：

1) 指出使用哪一台外部设备进行输入或输出。FORTRAN 给每一种外部设备指定一个设备号，或称通道号。

2) 指出哪些变量的值要输出，或者输入的数据要赋给哪些变量。FORTRAN 中用输入/输出变量表来规定要读/写的变量名单，称为输入名表和输出名表，简称输入表和输出表。

3) 指出外部设备的数据形式和机器内部的数据形式按什么转换关系进行输入或输出。例如，在输出时是按二进制输出还是按十进制输出？如果按十进制输出，则还应指出采用什么格式，即一个数据占几格长度，取小数点后几位，是指数形式还是非指数形式，等等。

在 FORTRAN IV 中，以上内容就通过读语句 (READ)、写语句 (WRITE) 和格式语句 (FORMAT) 来描述；并把输入/输出语句分成下列几类：

1) 带格式输入/输出语句，它由格式语句 (FORMAT) 规定其输入/输出格式。

2) 无格式输入/输出语句，即在输出时，按数据在计算机中的存放形式 (二进制数表示) 直接送到外部设备；在输入时，外部设备上的二进制数表示的数据，不经转换直接送入计算机。

3) 辅助输入/输出语句，用于与文件有关的输入和输出，配合无格式输入/输出语句。

此外，对某些具体系统来说，往往还有一些非标准的输入/输出语句，如自由格式输入、固定格式输出、会话型输入/输出等。这些都可以从具体机器的使用说明书中查得。值得注意的是，这些非标准输入/输出语句经过加工修订，有的已被确认为 FORTRAN 77 标准了。参见第 4 章 4-4 节。

关于外部设备号 (通道号) 的设置，FORTRAN 没有统一的规定。常见的一种做法是，以奇数代表输入设备，以偶数代表输出设备，具体设备的具体号码，则由编译系统自行

规定。

为了下面叙述方便,我们这里约定:以5统一代表输入设备,以6统一代表输出设备。也就是说,当我们使用5号输入设备时,程序员可以想象,这个号就是你心目中想使用的输入设备号,你尽管按照所用具体机器的约定,换一个你需要的号码好了。同理,当我们使用6号输出设备时,程序员也尽管换一个你需要的输出设备号好了。

但是,为了具体起见,有时我们又说“从卡片机输入…”,“从打印机输出…”。这时自然认为卡片机的设备号是5,打印机的设备号是6。不过,正如我们前面已经说过的,以用户终端的键盘输入来代替卡片输入,而以用户终端的荧光屏显示来作输出(包括显示某些输入信息),正越来越广泛地被使用。这时,自然又可认为键盘输入的设备号是5,荧光屏显示器的设备号是6。

事实上,一个计算机系统的设备号(通道号)并不是绝对不变的,程序员可以改变或重新规定。磁盘、磁带等外存储器的读/写通常就要临时规定通道号。

## 2. 带格式输出·写语句与格式语句

带格式输出由写语句 WRITE 连同格式语句 FORMAT 构成。

写语句的一般形式为

WRITE (<通道号>,<格式语句标号>)<输出名表>

格式语句的一般形式为

<标号>    FORMAT (<格式说明符>)

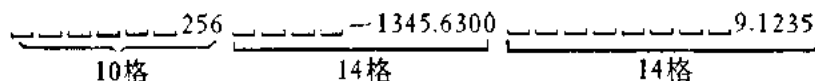
其中,<输出名表>是指写语句中所有需要输出的对象,包括简单变量、下标变量或数组,但不能是常数或表达式;两两之间用逗号隔开;<通道号>是指将变量值输出到对应于这个通道号的外部设备,或者说用所指通道号的外部设备进行输出(这里再重复说一下,我们已假定用6统一代表输出设备);<格式语句标号>是指按照这个标号的格式语句的格式进行输出;格式语句必须带标号,以便给读/写语句引用;<格式说明符>描述输入/输出的格式,下一段我们专门详细介绍。

现在看一个示例:

```
WRITE (6,100) N,A,B
100  FORMAT(1X,I10,F14.4,F14.4)
```

其意义是,以通道号为6的外部设备、按标号为100的格式语句所描述的格式、输出整型变量N和实型变量A与B之值。

FORMAT 描述的格式是,1X 表示换行后再打印(输出),它用来控制行与行的距离,我们在下一章再详讲,目前阶段暂且都在 FORMAT 语句中写上 1X 好了;I10 与整型变量N对应,I 是 Integer (整型)的第一个字母,表示按整型常数的格式,I 后面的10 表示输出的这个整型常数占10格;类似地,F14.4 与实型变量A对应,F 是 Float (浮点)的第一个字母,表示按浮点数(实型数)的格式,F 后面的14.4 表示输出的这个数共占14位,其中小数部分取4位(超过4位则四舍五入,不足4位则补足0)。例如,设  $I = 256$ ,  $A = -1345.63$ ,  $B = 9.12345$ , 则按所规定格式,输出结果为



要特别注意的是, WRITE 语句中各变量要与 FORMAT 语句中各字段描述符(见下一段)在类型、顺序方面一一对应,但是,也可以有下列情况:

```
WRITE(6,250) L,M,N
```

250 FORMAT (1X,I7)

这时,当输出第一个变量后,字段描述符已用完,而输出名表中还有未输出的变量。在这种情况下,则格式说明符从头开始,即 L,M,N 的值将分别打印在三行上。

输入/输出表中的元素是下标变量和数组时,执行情况如下例:

若要输出数组  $ARR(10)$  的下标变量  $ARR(5)$ ，还要输出整个数组  $T(4,10)$ ，则写语句为

```
WRITE(5,150) ARR(5),T
```

当执行此语句时,将按照标号为 150 的格式语句提供的格式,先输出数组元素 ARR(5),然后按列排列的顺序输出二维数组 T 的全部元素。

一般地,当数组名出现在输入/输出名表时,该数组将按下标值从小到大的顺序(包括按列存放的顺序)输入/输出其全部元素。

### 3. 格式说明符·字段描述符

在格式语句中，格式说明符连同两端的括号一起称为格式说明。格式说明符又分为字段描述符和字段分隔符。字段描述符由逗号、斜线或一串斜线隔开，逗号和斜线称为字段分隔符。字段分隔符我们一直留到下一章的 3-4 节再详细讨论，目前我们仿照例子照样使用逗号。

这一段我们先介绍字段描述符。

FORTRAN IV 中有九种字段描述符，这里先介绍常用的四种：

### (1) I 型字段描述符 `rlw`

即上一段已经见到的 I 型描述符, 其中  $r$  和  $w$  为非零整常数,  $rlw$  即  $r$  个  $lw$ ,  $r$  为重复次数,  $r=1$  时可以省略,  $w$  表示输入/输出的字符在外部介质上所占的位数, 称字段宽度。

用于输入时, Iw 表示把外部介质上的 w 个字符的整常数形式送到相应的整型变量中。被输入的整常数的正号可以省略, 第一个数字之后的空格被转换成数字 0, 全空白的字段也被转换成数值 0。若外部介质上的符号个数多于字段宽度 w, 则输入出错。

用于输出时, `Iw` 表示将整型变量的值按整常数格式输到外部介质的相应位置上, 共占 `w` 个字符, 输出值靠右排列对齐。当输出正的数值时, 是否印正号, 各编译程序可能做法不同。输出时所选用的字段宽度 `w` 至少应为需要输出的数的位数再加 1。

(2) F 型与 E 型字段描述符 rFw.d 与 rEw.d.

F 型已在上面用过。F 型和 E 型都是用来描述输入或输出实型数据的格式，同时也可以用来描述复数的实部和虚部。其中，r 为重复次数，w 为字段宽度，即总字符个数，d 为小数部分的位数。

用于输入时,F型和E型意义相同,都表示把外部介质上相应的w个字符转换成十进



制浮点数送到相应的实型变量中,例如

外部形式的数	字段描述符	输入后内存值
1986	F8.3(或 E8.3)	198.600
-134.5	F8.3(或 E8.3)	-134.5
-12345678	F8.3(或 E8.3)	-1234.567
123.4E+01	F10.4(或 E10.4)	1234.0
-156789E-01	F10.4(或 E10.4)	-1.56789

从例中可见,若数据有小数点,则小数位  $d$  不起作用;若总位数  $w$  不够,则输入错误;若数据为指数形式,又指数前面的部分有小数点,则将此数乘以指数部分,得机内数值;若数据为指数形式,又指数前面的部分不带小数点,则取该部分的后  $d$  位为小数位,然后乘以指数部分,即得机内数值。

用于输出时, F 型和 E 型的作用是不同的。F 型表示把要输出的实型变量的值按带小数点的十进制形式输出,该数据在外部介质上占  $w$  位,小数部分占最后  $d$  位,小数点占一位,符号占一位,即必须  $w \geq d + 2$ 。E 型表示把要输出的实型变量的值按带指数的十进制规格化形式输出,即在外部介质上共占  $w$  个字符,最后两个字符为指数部分,指数部分前一位为指数的符号,再前一位是字母 E,接着的前  $d$  个字符为小数部分,小数部分前面为小数点,整数部分一定是 0,还有一位符号位。可知  $d$  是输出数的有效数字位数,还必须有  $w \geq d + 7$ 。例如:

内存中的数	字段描述符	输出后外部形式的数
-123.45	F7.2	-123.45
~	F8.3	-123.450
~	F6.1	-123.4
123.45	F7.0	123
~	F6.4	(W位数不移)
-0.0375	E11.4	-0.3750E-01
1234567	E12.5	0.123456E_07
-12.345	E12.5	-0.12345E_02

从例中可见,当被输出的值的小数部分超过  $d$  位时将四舍五入输出。另外,输出字符将一律在右端对齐,因而我们有可能获得齐整的输出结果。

### (3) X 型字段描述符 $nX$ .

它用来跳过一些符号或输出一些空格,其中  $n$  为无符号整数,  $n \geq 1$ , 且  $n = 1$  时不能省略不写。与 I、F、E 描述符不同, X 描述符用于输入/输出时,在输入/输出表中设有与之对应的项目。

X 型字段描述符主要用于输出。在输出时,用来输出  $n$  个空格,以便编辑打印结果的形式。如把上一段的例改为

```
WRITE(6,100) N,A,B
100 FORMAT(1X,I10,5X,F14.4,5X,F14.4)
```

则在 N 与 A 和 A 与 B 的打印结果之间,我们就插入了 5 个空格。

(4) H型字段描述符  $nHh_1h_2\cdots h_n$ .

它用来输出一些文字符号,例如打印标题、表头、变量符号等,其中  $n$  为大于零的整数,它指明H后跟的字符个数,  $h_1h_2\cdots h_n$  为 FORTRAN 中的基本字符,包括空格.

和X描述符一样,在输入/输出表中没有与之对应的项目.例如,要在打印机上输出如下信息:

1986 - 05 - 10

则相应的写语句和格式语句可写为

```
WRITE(6,15)
```

```
15 FORMAT(1X,10H1986 - 05 - 10)
```

又如我们写

```
WRITE(6,222) X1, X2
```

```
222 FORMAT(1X,3HX1 = ,F8.2,5X,3HX2 = ,F8.2)
```

设  $X1 = -725.04$ ,  $X2 = 1133.125$ ,则输出结果为

$X1 = \_ -725.04 \_ \_ \_ \_ X2 = \_ 1133.13$

有两种特殊情况.如

```
WRITE(6,100)
```

```
100 FORMAT(1X,5HAAAAAA,1X,5HBBBBBB)
```

即 WRITE 中无输出表,这时输出结果是

AAAAAA\\_BBBBBB

又如

```
WRITE(6,100)
```

```
100 FORMAT(1X,I10,F14.4)
```

即不仅 WRITE 中无输出表,而且 FORMAT 中无字符型(即H型)描述符,这时执行 WRITE 语句的结果是输出一空行.

在某些非标准的 FORTRAN IV 以及 FORTRAN 77 中,我们将看到,还允许在字符串两端加上单引号来代替  $nH$  符.这显然比较方便.如上述例中的

```
222 FORMAT(1X,3HX1 = ,F8.2,3HX2 = ,F8.2)
```

可以写成

```
222 FORMAT(1X,'X1 = ',F8.2,'X2 = ',F8.2)
```

最后,还要指出的是,重复的字段描述符可以简写,如

```
300 FORMAT(1X,I10,I10,F14.4,F14.4,F14.4)
```

可以写成

```
300 FORMAT(1X,2I10,3F14.4)
```

又如

```
400 FORMAT(1X,I10,F15.5,1X,I10,F15.5)
```

可以写成

400 FORMAT(2(1X,I10,F15.5))

在 FORTRAN 中,把这称为字段描述符组。

#### 4. 带格式输入·读语句

与带格式输出相仿,带格式输入由读语句 READ 连同格式语句 FORMAT 构成。

读语句的一般形式为

READ (<通道号>,<格式语句标号>) <输入名表>

其中<通道号>和<格式语句标号>与写读句相应的意义相同,<输入名表>是指通过输入要被赋值的那些简单变量、下标变量或数组,两两之间用逗号隔开。

仍以示例来说明:

READ(5,100) X, Y, M

100 FORMAT(2F8.3,I6)

其意义是以通道号为 5 的外部设备、按标号为 100 的格式语句所描述的格式,输入并加工三个数据,依次赋给变量 X,Y,M。这里,为叙述确定起见,我们假定数据将从卡片上输入。在卡片上数据排列的形式按标号为 100 的格式语句中的规定,对应 X,Y 的是字段描述符 F8.3,对应 M 的是字段描述符 I6。假定卡片上提供的数据形式如下:

——24500——135040——20  
8 格 8 格 6 格

程序运行后遇到 READ 语句时,便自动将数据卡片中的数据送入计算机。这时,根据 FORMAT 语句中字段描述符 F8.3 的要求,把 1 至 8 列的数据赋给 X,其中最后 3 位是小数部分,故实际是将 24.500 赋给 X,同理将 135.040 赋给 Y,将整数 20 赋给 M。

从示例我们可以看到,在输入格式语句中,格式说明符的最前面并不需要写 1X。数据从卡片的第 1 列开始穿孔,有效的范围是 1~80 列,这与源程序语句部分规定从第 7 列到第 72 列有效是不同的。

事实上,从上一段介绍 F 型与 E 型字段描述符的例子中可知,通过 READ 语句输入数据时,计算机采用两种处理方法。一种是如上例所述,自动将 1~8 列、9~16 列和 17~22 列分开为三个数据,不需要在这三个数据间加逗号,在实型数据中也不必加小数点,编译程序将按 F8.3 规定把 8 位中最后 3 位定为小数部分。另一种是,如果输入的数据本身带小数点,则这时按所谓“自带小数点优先”的原则(即不再按 F 格式)去加工该数值,把该数据直接输入赋给相应的变量。如上例,若卡片准备的数据是

——24.5——135.04——20

则按“自带小数点优先”原则,把前 8 个字符作为一个数据,从而直接把 24.5 赋给 X;同理,直接把 135.04 赋给 Y,把 20 赋给 M。

要特别指出的是,在使用键盘输入的系统,数据的输入可以采用如下的人-机对话方式,例如

READ(5,777)A,B,C

## 777 FORMAT(3F10.4)

当执行此读语句时,荧光屏显示出符号“=”,然后等待使用者提供数据。这时,使用者只需由键盘输入:

13.57,-0.012,246.8)

即一个数据不必按 F10.4 的格式占满 10 格,而可以少于 10 格,并用逗号将本数据与下一个数据隔开,又当送完三个数据后,打入回车键(↵),计算机即将 13.57,-0.012,246.8 依次赋给 A,B,C。在这种情况下,“自带小数点优先”的规则不起作用。

最后要指出的是,READ 语句所需的数据,并不写在源程序中。与 BASIC 不同,FORTRAN 中的 READ 语句并不和 DATA 语句配合使用。FORTRAN 的 READ 语句所需要的数据,是在程序开始运行以后输入的,程序执行到 READ 时,自动地读入所指外部设备上提供的数据。这一点,请习惯写 BASIC 程序者留意。

## 2-5 控制转移语句

一般情形下,FORTRAN 源程序的执行顺序是,由程序块最前面的第一个可执行语句开始,从前到后,依次执行各可执行语句。如果要改变这种顺序,可使用控制转移语句。FORTRAN IV 中的控制转移语句包括:

- 无条件 GO TO 语句;
- 计算 GO TO 语句;
- 赋标号 GO TO 语句;
- 算术 IF 语句(算术条件语句);
- 逻辑 IF 语句(逻辑条件语句)。

现在分别介绍如下,其中还要附带介绍关系表达式、逻辑表达式和赋标号语句。

### 1. 无条件 GO TO 语句·算术 IF 语句

最简单的控制转移语句是无条件 GO TO 语句,其一般形式为

GO TO (语句标号)

其中(语句标号)就是指要转去的那个可执行语句的标号。

**例 1** 设有许多张数据卡片,每张卡片上都有三个数据。要求计算机从每张卡片上读入三个数据,求它们的和及平均值,然后打印出来,试写其 FORTRAN 程序。

设变量 A,B,C 存放每次读入的三个数据,SUM 存其和,AVER 存其平均值,FORTRAN 程序如下:

```
10 READ(5,100) A,B,C
100 FORMAT(3F15.4)
    SUM = A + B + C
    AVER = SUM/3.0
    WRITE(6,200)A,B,C,SUM,AVER
200 FORMAT(1X,5F15.4)
```

```

GO TO 10
STOP
END

```

运行这个程序时,首先由标号为 10 的读语句读入第一张卡片上的三个数据,作计算、打印,然后由 GO TO 10 语句把控制再转移到标号为 10 的读语句,再读入第二张卡片上的三个数据,再作计算、打印,又再转回读语句等等,直到把所有卡片处理完为止。显然,这是一个实际上没有执行到 STOP 语句的程序,最后只好用人工干预的方法强迫停机。当然,可以象示例所采取的方法那样,用最后一组特殊数据来做结束标志,如下面的例 8。这种方法在实际中是很常用的。

现在讲算术 IF 语句。

算术 IF 语句是有条件的转移语句,其一般形式为

IF (<算术表达式>) L1,L2,L3

其中 L1,L2,L3 是本程序块内三个可执行语句的标号,<算术表达式>在简单情况是一个数值变量。执行这个语句时,首先计算括号中的算术表达式,若所得的结果  $<0$ ,则转去执行标号为 L1 的语句;若所得结果  $=0$ ,则转去执行标号为 L2 的语句;若所得结果  $>0$ ,则转去执行标号为 L3 的语句。

**例 2** 仍以处理数据卡片的简单问题为例。从每张卡片上读入一个  $x$  值,然后按下式计算并打印  $y$  值:

$$y = \begin{cases} -2x^3 + 8x^2 - 3x + 1 & x < 1, \\ 0 & x = 1, \\ e^x + \ln(x-1) & x > 1. \end{cases}$$

这里假定共有 100 张卡片。

注意到改写

$$y = -2x^3 + 8x^2 - 3x + 1 = ((-2x + 8)x - 3)x + 1$$

FORTRAN 程序:

```

      N = 1
50  READ(5,100)X
100  FORMAT(F8.2)
      IF(X - 1.0) 10,20,30
10  Y = ((-2.0 * X + 8.0) * X - 3.0) * X + 1.0
      GO TO 40
20  Y = 0.0
      GO TO 40
30  Y = EXP(X) + ALOG(X - 1.0)
40  WRITE(6,200) X,Y
200  FORMAT(1X,2HX = ,F8.2,10X,2HY = ,F8.2)
      N = N + 1

```

```

      IF(N - 100)50,50,60
60  STOP
      END

```

## 2. 关系表达式与逻辑表达式 · 逻辑 IF 语句

两个量大小的比较,如 $<$ , $\leq$ , $>$ , $\geq$ , $=$ , $\neq$ ,在 FORTRAN 中通过六种关系运算符来实现。这六种关系运算符是:

```

.LT.   (表示<)
.LE.   (表示≤)
.GT.   (表示>)
.GE.   (表示≥)
.EQ.   (表示=)
.NE.   (表示≠)

```

由两个算术表达式中间夹一个关系运算符就构成一个关系表达式。一个关系表达式依关系成立与否,分别取逻辑值“真”(True)或“假”(False)。有关逻辑型常数和变量的概念,我们在下一章 3-1 节中还要详讲。现在看关系表达式的例子:

```

1 .EQ. 1    结果为真;
5 .LE. 0    结果为假;
X .GT. Y    结果的真假随 X,Y 而定;
M .NE. M/2*2 若M为奇数则为真,M为偶数则为假。

```

关系运算符的左右两个算术表达式可能分属于不同类型。标准 FORTRAN IV 规定只允许相同类型的算术表达式作比较。有的 FORTRAN 编译程序放宽这个限制,通常是先把不同类型转换成同类型,然后再作比较。

当使用 .EQ. 或 .NE. 比较非整数算术表达式时要小心。例如, A 和 B 两个变量的计算结果理论上同样是 3,但在计算机内可能表示为  $A = 3.000001$ ,  $B = 2.999998$ ,这时  $A .EQ. B$  就是“假”。遇到这种情形,可写成

```

      ABS(A - B) .LE. 0.00001
或      (A - B) ** 2 .LE. 0.1E - 7

```

关系表达式的运算顺序是:先完成算术运算,再作关系比较。

在简单的关系表达式的基础上,运用逻辑运算符还可建立一般的逻辑表达式。FORTRAN IV 引用三种逻辑运算符:

```

.NOT.   (表示逻辑非,记号¬)
.AND.   (表示逻辑与,记号∧)
.OR.    (表示逻辑或,记号∨)

```

.NOT. 是一元运算符,它与紧接在它后面的关系表达式一起构成逻辑表达式。当关系表达式为“真”,则整个逻辑表达式为“假”;反之,当关系表达式为“假”,则整个逻辑表达式为“真”。例如

若  $M .EQ. N$  为“真”,则  $.NOT. M .EQ. N$  为“假”;

若  $U .LT. V$  为“假”, 则  $.NOT. U .LT. V$  为“真”。

$.AND.$  和  $.OR.$  是二元运算。即两个关系表达式可用  $.AND.$  或  $.OR.$  联结起来构成逻辑表达式。用  $.AND.$  联结的逻辑表达式, 当且仅当两个关系表达式全为“真”时, 结果才为“真”, 否则就是“假”。例如,  $2.5 \leq Y < 10$ , 写成

$$2.5 .LQ. Y .AND. Y .LT. 10.$$

当  $Y$  为 5 时, 上式结果为真。

用  $.OR.$  联结的逻辑表达式, 当所联结的两个关系表达式中任一个为真或两个都为真时, 结果就是真; 当且仅当两个关系表达式都为假时, 结果才是假。例如

$$ER .LT. 0.5E - 8 .OR. KOU .GE. 100$$

如果  $ER < 0.5E - 8$  与  $KOU \geq 100$  两式中有一个为真, 或两式都为真, 则结果为真; 如果两式都为假, 则结果为假。

逻辑运算符的运算规则可以归纳为真值表 2-5(a) 与表 2-5(b)。

表 2-5(a)  $.NOT.$  的真值表

$c$	$.NOT. c$
真	假
假	真

表 2-5(b)  $.AND.$  和  $.OR.$  的真值表

$c_1$	$c_2$	$c_1 .AND. c_2$	$c_1 .OR. c_2$
真	真	真	真
真	假	假	真
假	真	假	真
假	假	假	假

逻辑运算符运算时相对的优先顺序是:

$$.NOT. \rightarrow .AND. \rightarrow .OR.$$

根据真值表, 可知

$$(3 .GT. 4 .AND. 3 .LT. 2) .OR. 2 .LT. 4 \text{ 为真;}$$

$$3 .GT. 4 .AND. (3 .LT. 2 .OR. 2 .LT. 4) \text{ 为假.}$$

根据优先规则, 可知

$$(X + Y) .LT. SIN(Z * * 3 - 1.) .AND. (W .EQ. 8.)$$

与  $X + Y .LT. SIN(Z * * 3 - 1.) .AND. W .EQ. 8.$  等价。

现在讲逻辑 IF 语句。

逻辑 IF 语句也是有条件的转移语句, 其一般形式为

$$IF (\langle \text{逻辑表达式} \rangle) S$$

其中  $S$  可以是除 DO 语句 (见下一节) 或另一逻辑 IF 语句外的任一可执行语句。执行这个语句时, 先计算  $\langle \text{逻辑表达式} \rangle$  的值, 若其值为真, 则执行语句  $S$ ; 若其值为假, 则跳过  $S$  语句, 顺序执行这个逻辑条件语句后面的其它语句。  $S$  语句也称为逻辑 IF 语句的内嵌语句。

**例 3** 重新考虑例 2 中的程序。在倒数第三行我们采用算术 IF 语句

IF(N - 100) 50, 50, 60

事实上,我们也可以改用逻辑 IF 语句

IF (N .LE. 100) GO TO 50

**例 4** 考虑如图 4-3 的线性渐增函数和阶跃函数。在区间(0,A)上从 0 渐增到 1 的函数

$$F = \begin{cases} 0 & T < 0 \\ \frac{T}{A} & 0 \leq T < A \\ 1 & T \geq A \end{cases}$$

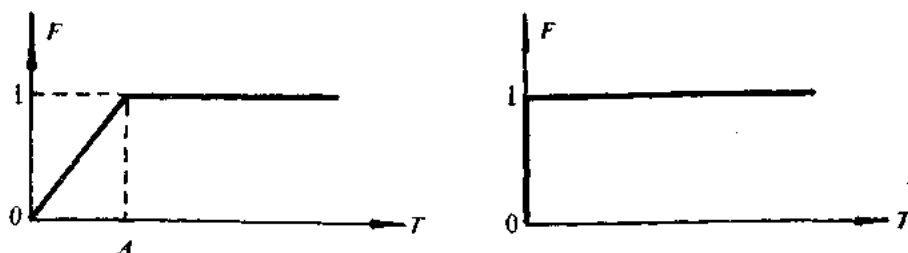


图 2-3 线性渐增函数和阶跃函数

当  $A = 0$  时成为阶跃函数。这个函数可用逻辑条件语句描述为:

F = 0.0

IF(T .GE. 0 .AND. T .LT. A) F = T/A

IF(T .GE. A) F = 1.0

其中,先假定当  $T < 0$  时  $F = 0.0$ , 然后判断是否是其它两种情况,并相应地重新赋 F 的值。

### 3. 计算 GO TO 语句·赋标号语句与赋标号 GO TO 语句

逻辑 IF 语句可使程序实现两个分枝,算术 IF 语句可使程序实现三个分枝。更多分枝的程序则可借助计算 GO TO 语句来实现。

**计算 GO TO 语句的一般形式为**

GO TO( $L_1, L_2, \dots, L_n$ ), <整形变量>

其中  $L_1, L_2, \dots, L_n$  是一串整常数,都是本程序块内的可执行语句的标号。执行这个语句时,根据<整形变量>的值是几,程序便转到括号内第几个标号所在的语句上去。如<整形变量>的值是 2,程序便转到标号为  $L_2$  的语句上去。

需注意的是,括号中语句标号的个数必须与整形变量从 1 到其最大可能值相对应,那怕是整型变量的某些值可能在程序中永不出现。如果整型变量的值超出了一定的范围(如负数或大于括号中标号的个数),则计算机执行此语句后往哪儿转移将无法控制,因此最好是事先加以避免。另外,计算 GO TO 语句括号中的标号不能再是本语句的标号,如

90 GO TO(80,90,100,110),K



当K取2时将产生一个无穷的循环。

**例5** 还是以处理数据为例。这次假定有若干组数据,每组数据有5个。根据实际情况可能有四种不同的预处理方式,然后再作统一处理。为此,我们在每组数据的末尾增加一个整数(1~4)来指明本组数据所要求的预处理方式。这个程序的大致结构如下:

```
      READ (5,100) A1,A2,A3,A4,A5,N
100  FORMAT (5F8.3,I1)
      GO TO(10,20,30,40),N
10  ..... (第一种预处理)
      GO TO 200
20  ..... (第二种预处理)
      GO TO 200
30  ..... (第三种预处理)
      GO TO 200
40  ..... (第四种预处理)
200  ..... (统一处理)
```

现在讲**赋标号语句**和**赋标号 GO TO 语句**,它们是配合使用的。

**赋标号语句**或称**标号赋值语句**,其一般形式为

ASSIGN <语句标号> TO K

**赋标号 GO TO 语句**的一般形式为

GO TO K,(L<sub>1</sub>,L<sub>2</sub>,...,L<sub>n</sub>)

其中K是整型变量名, L<sub>1</sub>,L<sub>2</sub>,...,L<sub>n</sub> 是一串本程序块内的可执行语句标号。**赋标号语句**的作用是把指定的<语句标号>赋给整变量 K。执行**赋标号 GO TO 语句**时,其中K的值已由**赋标号语句**确定,因此,本语句将使控制转移到标号为K的语句上去。

应注意计算 **GO TO 语句**与**赋标号 GO TO 语句**的差别:前者中的K值是标号的顺序,后者的K是标号本身,且是 L<sub>1</sub>,L<sub>2</sub>,...,L<sub>n</sub> 中的某一个。此处,程序中往往使用多个**赋标号语句**来和一个**赋标号 GO TO 语句**配合。根据不同的 ASSIGN 语句中给K赋的标号值,使在执行**赋标号 GO TO 语句**时转移到不同的分支上去。

**例6** 设函数F定义如下:

$$F = \begin{cases} 0 & \text{当 } x < 0, \\ x^2 & \text{当 } 0 \leq x < 10, \\ x^2 - 10 & \text{当 } 10 \leq x < 20, \\ x^2 - 5x + 1 & \text{当 } 20 \leq x < 30, \\ 0 & \text{当 } x \geq 30. \end{cases}$$

若自变量x是程序执行的一个中间结果,则计算函数F的程序段如下:

```
.....
IF(X .GE. 0 .AND. X .LT. 10) ASSIGN 100 TO KX
```

```

        IF(X .GE. 10 .AND. X .LT. 20) ASSIGN 200 TO KX
        IF(X .GE. 20 .AND. X .LT. 30) ASSIGN 300 TO KX
        IF(X .LT. 0. 0. OR. X .GE. 30) ASSIGN 400 TO KX
        GO TO KX,(100,200,300,400)
100  F = X * X
        GO TO 500
200  F = X * X - 10.
        GO TO 500
300  F = X * X - 5. * X + 1.
        GO TO 500
400  F = 0.
500  .....

```

为了熟练掌握控制转移语句,以下我们继续编写几个应用控制转移语句的完整FOR-TRAN 程序.

**例7** 欧几里得 (Euclid) 算法.

读入两个正整数  $m$  和  $n$ , 求它们的最大公因子, 即能够同时整除  $m$  和  $n$  的最大正整数. 解这个问题的欧几里得算法如下:

E1. [求余数]以  $m$  除以  $n$  并令  $r$  为所得余数(有  $0 \leq r < n$ ).

E2. [判断余数为 0 否]若  $r = 0$ , 则  $n$  为答案, 算法结束, 否则作下一步.

E3. [互换]置  $m \leftarrow n, n \leftarrow r$ , 并返回步 E1.

算法框图如图 2-4. 求余数可用标准函数 MOD(见本章附录 A). 设余数  $r$  为 REM, 所求最大公因子为 HCF.

程序如下:

```

C      EUCLID'S ALGORITHM
      READ(5,100)M,N
100  FORMAT (2I6)
20  REM = MOD (M,N)
      IF(REM .EQ. 0.0) GO TO 10
      M = N
      N = REM
      GO TO 20
10  HCF = N
      WRITE(6,200)M,N,HCF
200  FORMAT (1X,20H GIVEN TWO INTEGER , I6,
1  5H AND , I6, 15H THEIR HCF IS , I6)
      STOP
      END

```

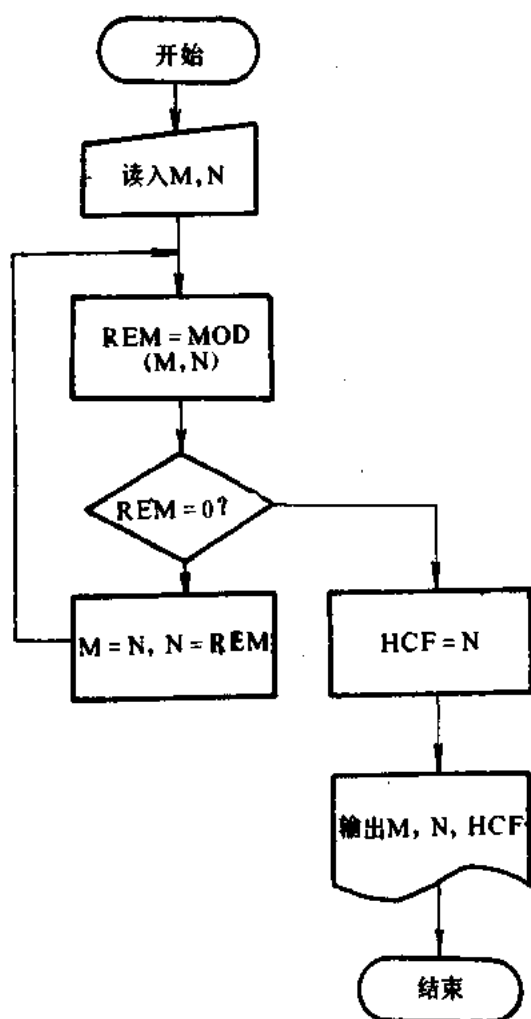


图 2-4 欧几里德算法框图

**例 8** 已知交流电路中电流计算公式为:

$$I = \frac{E}{\sqrt{R^2 + \left(2\pi fL - \frac{1}{2\pi fC}\right)^2}}$$

其中:  $I$  = 电流(安培),  $E$  = 电压(伏特),  
 $R$  = 电阻(欧姆),  $L$  = 电感(亨利),  
 $C$  = 电容(法拉),  $f$  = 频率(周/秒).

要求对一组固定的  $E, R, L, C$  值计算和打印出一系列  $f$  值所对应的  $I$  值, 以便提供绘制电路中电流与电源频率之关系图. 为了使写出的 FORTRAN 程序能够处理任意个频率数据, 我们在所考虑的一系列  $f$  值的后面, 增加一个负的  $f$  值, 用以作这一系列  $f$  值的结束标志. FORTRAN 程序如下:

```

C      CURRENT IN AN A. C. CIRCUIT MASTER ACCIRCUIT
      READ(5,100)E,R,L,C
  
```

```

100  FORMAT(4F10.2)
      PI2 = 6.2832
C      READ A FREQUENCY FROM DATA CARD
200  READ (5,150) CYCLE
150  FORMAT (F10.0)
      IF(CYCLE)300,200,200
      AMPERE = E/SQRT(R*R + (PI2 * CYCLE * L - 1.0/
1(PI2 * CYCLE * C)) * * 2)
      WRITE(6,80)CYCLE,AMPERE
80   FORMAT(1X,2F15.5)
      GO TO 200
300  STOP
      END

```

**例 9** 一工程设计中，需要一些有关圆柱细长比率  $R$  与圆柱承受的安全负荷  $S$  的资料，现从工程设计手册中查得计算公式

$$S = \begin{cases} 18,000 - 0.485R^2 & R \leq 110 \\ \frac{18,000}{1 + (R^2/18,000)} & R > 110 \end{cases}$$

试写一完整的 FORTRAN 程序，求出  $R$  从 40 到 200，每次增加 10 时  $S$  之值，并打印出  $R$  与  $S$ 。

FORTRAN 程序：

```

      R = 40
15  IF(R - 110.)20,20,25
20  S = 18000.0 - 0.485 * R * * 2
      GO TO 30
25  S = 18000.0 / (1.0 + R * * 2 / 18000.0)
30  WRITE(6,35)R,S
35  FORMAT(1X,2HR = ,F6.1,5X,2HS = ,E13.6)
      R = R + 10.
      IF(R - 200.)15,15,40
40  STOP
      END

```

计算输出结果如下：

R =	40.0	S =	0.172240E + 05
R =	50.0	S =	0.167875E + 05
R =	60.0	S =	0.162540E + 05

R =	70.0	S =	0.156235E + 05
R =	80.0	S =	0.148960E + 05
R =	90.0	S =	0.140715E + 05
R =	100.0	S =	0.131500E + 05
R =	110.0	S =	0.121315E + 05
R =	120.0	S =	0.100000E + 05
R =	130.0	S =	0.928366E + 04
R =	140.0	S =	0.861702E + 04
R =	150.0	S =	0.000000E + 04
R =	160.0	S =	0.743119E + 04
R =	170.0	S =	0.690831E + 04
R =	180.0	S =	0.642857E + 04
R =	190.0	S =	0.598891E + 04
R =	200.0	S =	0.558620E + 04

#### 例 10 求平方根程序.

怎样求得一个数  $A$  的平方根  $X = \sqrt{A}$  呢? 一种简单的算法是这样导出来的: 先估计平方根的一个近似值  $x_0$ . 如果  $x_0$  小于实际应有的值, 即  $x_0^2 < A$ , 则  $\frac{A}{x_0}$  就会大于  $x_0$ ; 相反, 如果  $x_0$  估得偏大, 即  $x_0^2 > A$ , 则  $\frac{A}{x_0}$  将小于  $x_0$ . 两种情形下, 取  $x_0$  和  $\frac{A}{x_0}$  的平均值, 都可能得到比  $x_0$  好一些的近似值, 记作  $x_1$ :

$$x_1 = \frac{1}{2} \left( x_0 + \frac{A}{x_0} \right)$$

如果仍不满意, 可以如法炮制, 求得

$$x_2 = \frac{1}{2} \left( x_1 + \frac{A}{x_1} \right)$$

以至于多次重复地做下去:

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{A}{x_n} \right), \quad n = 0, 1, 2, \dots$$

其中  $x_0$  是最初估计的初值. 这种求平方根的迭代法, 由一千多年前的希腊数学家 Heronius 所提出. 迭代到什么时候满意? 过程重复若干次后,  $x_{n+1}$  和  $x_n$  的差别就会很小. 通常, 只要

$$|x_{n+1} - x_n| < \epsilon$$

$\epsilon$  是适当选择的一个正小实数, 这时  $x_n$  和  $x_{n+1}$  的前若干位就相同, 故可以结束计算. 这叫作“绝对误差控制”.

例如  $A = 4.245$ , 同时取  $A$  本身作初值  $x_0 = 4.245$ , 则按上面的办法迭代的结果是 (保留八位有效值):

$$x_1 = 2.6225000$$

$$x_2 = 2.1205922$$

$$x_3 = 2.0611957$$

$$x_4 = 2.0603399$$

$$x_5 = 2.0603397$$

如果工程计算中需保留四位有效值,则取  $\epsilon = 0.0001$  就足够了,算到  $x_5$  结束。

一般来说,如何为迭代计算选取初值是颇有讲究的,我们将在第 7 章中讨论。就求平方根这个特殊例子来说,初值可以随便取。这里我们简单地取  $x_0 = A$ 。

我们把待求平方根的每一个数(正数)分别穿孔在每张卡片上。最后增加一张穿上 0 或负数的卡片作结束标志。先作程序框图如图 2-5。

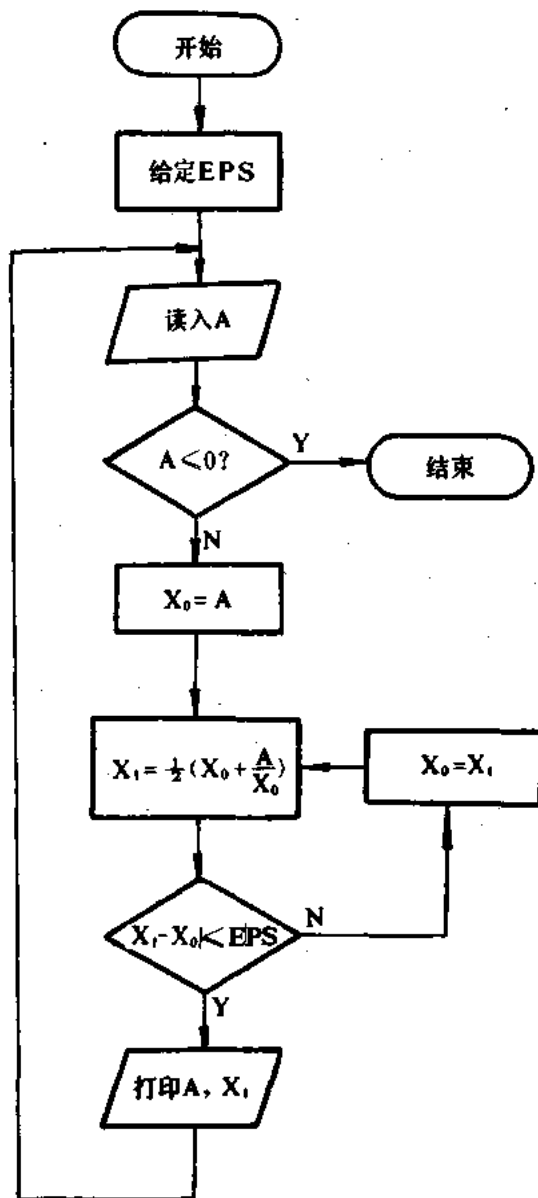


图 2-5

另外,假定所欲求的平方根的数均取至小数后 3 位,取迭代的结果也准确到小数后 3

位,则可取  $\varepsilon = 0.0001$ .

FORTTRAN 程序如下:

```
      EPS = .1E - 4
1  READ(5,11)A
11  FORMAT(F6.3)
44  IF(A .LE. 0.0) STOP 77777
      X0 = A
2  X1 = 0.5*(X0 + A/X0)
      IF(ABS(X0 - X1) .LT. EPS) GO TO 3
      X0 = X1
      GO TO 2
3  WRITE(6,22)A,X1
22  FORMAT(1X,2HA = ,F6.3,10X,3HX1 = ,F6.3)
      GO TO 1.
      END
```

例如,取  $A = 3.142$ , 上机计算结果可得

$A = 3.142$

$X1 = 1.773$

## 2-6 循环语句与继续语句

循环语句和继续语句也属于控制语句。继续语句比较简单,我们先讲继续语句再讲循环语句。

### 1. 继续语句

继续语句又叫空语句,它唯一的功能就是用来在程序中适当的地方被挂上一个标号。不带标号的继续语句几乎毫无用处。

继续语句的一般形式为

〈标号〉 CONTINUE

执行这个语句时,控制就转移到程序中的下一个紧接着的语句。因此,CONTINUE 语句显然可以在程序中的任何地方出现。

继续语句可以在两种场合使用:一种是用它来把程序分成比较清楚的段落,以便于阅读。当然,这是可有可无的用法。另一种是在没有其它出路时,用它来挂上一个标号。这种情况在循环语句中经常遇到。

### 2. 循环语句 (DO 语句)与循环程序设计例

循环是程序设计中运用少量语句实现大量计算的主要手段,每种程序设计语言都规定了专门的循环语句。FORTTRAN 语言的循环语句也称 DO 语句,其一般形式为

```
DO L I = M1, M2, M3
```

```
.....
```

```
L CONTINUE
```

其中: L 整型常数,循环终端语句的标号;

I 整型变量,循环控制变量;

M<sub>1</sub> 整型常数或整型变量,循环变量初值;

M<sub>2</sub> 同上,循环变量终值;

M<sub>3</sub> 同上,循环变量增量(步长),如果 M<sub>3</sub> = 1, 则 M<sub>3</sub> (连同前面的逗号)可以省略。

从 DO 语句后面一句开始,包括标号 L 所在的最后一句称为循环体,它是循环中真正重复执行的那些语句。终端语句不一定是 CONTINUE 语句,只要不违反 DO 语句的规定(下面详细说明),可在循环体内最后一个允许的可执行语句上加标号 L。一般说,用空语句 CONTINUE 来结束每一个循环,这是比较方便而又稳妥的做法。

循环语句的执行过程是: 将循环变量初值 M<sub>1</sub> 赋给循环变量 I,依次执行循环体中各语句,直至执行完标号为 L 的循环终端语句,循环变量 I 增加步长值 M<sub>3</sub>, 即 I + M<sub>3</sub> 赋给 I, 比较 I 是否大于 M<sub>2</sub>? 如果 I 不大于 M<sub>2</sub>, 则返回循环体,重复上述做法; 如果 I 大于 M<sub>2</sub>, 则结束循环过程,执行终端语句的下一语句。

必须注意,在 FORTRAN IV 中, M<sub>1</sub>, M<sub>2</sub>, M<sub>3</sub> 规定应是整型常数或已赋值的整型变量,不能是下标变量或算术表达式,而且要求 M<sub>2</sub> ≥ M<sub>1</sub> > 0, M<sub>3</sub> > 0。相对于其它算法语言来说,这些规定比较苛刻,对程序设计是不太方便的。因此,不少 FORTRAN 编译程序对此往往有适当的放宽。

如果坚持 FORTRAN IV 标准,想打印出从 0 到 0.5 步长为 0.1 的平方根,不能把 DO 语句写成

```
DO 10 I = 0,0.5,0.1
```

而应该写成

```
X = 0.0
```

```
DO 10 I = 1,6
```

```
Y = SQRT(X)
```

```
WRITE(6,20)I,X,Y
```

```
10 X = X + 0.1
```

```
20 FORMAT(1X,I5,2F10.4)
```

```
STOP
```

```
END
```

又如想打印从 -10 至 -1 步长为 1 的立方根,同样不能写

```
DO 10 I = -10,-1,1
```

可写成



```

DO 10 I = 1,10
N = -11 + I
M = N * *(1/3)
10 WRITE(6,200)N,M
200 FORMAT(1X,2I10)
STOP
END

```

下面我们编写一些循环程序设计例。

**例 1** 编写计算  $N$  个不同三角形面积的程序,其中数  $N$  从第一张数据卡片读入,每个三角形三条边长  $A, B, C$  则从随后的各张数据卡片读入。

采用如下的三角形面积公式:

$$AREA = \sqrt{S(S-A)(S-B)(S-C)}$$

其中

$$S = (A + B + C)/2$$

FORTRAN 程序:

```

READ(5,100)N
100 FORMAT(I3)
DO 50 I = 1,N
READ(5,200)A,B,C
200 FORMAT(3F6.2)
S = 0.5*(A + B + C)
X = S*(S - A)*(S - B)*(S - C)
IF(X .LT. 0.0)GO TO 20
AREA = SQRT(X)
WRITE(6,300) A,B,C,AREA
300 FORMAT (1X, 7HSIDES_ = ,3F6.2,8H__AREA__ = ,F6.2)
GO TO 50
20 WRITE(6,400) A,B,C
400 FORMAT(1X,3F6.2,10H__ANNOT__BE,
1 24H__THE__SIDES__OF__A__TRIANGLE)
50 CONTINUE
STOP
END

```

**例 2** 样品算术平均值、方差与标准偏差统计程序。

设  $N$  次观测所得的样品为  $X_i (i = 1, 2, \dots, N)$ , 则

$$\text{算术平均值 } \bar{X} = \sum_{i=1}^N X_i / N;$$

$$\text{方差 VAR} = \frac{\sum_{i=1}^N (X_i - \bar{X})^2}{N-1} = \frac{N \sum_{i=1}^N X_i^2 - \left(\sum_{i=1}^N X_i\right)^2}{N(N-1)};$$

$$\text{标准偏差 STDEV} = \sqrt{\text{VAR}}.$$

因为采样次数  $N$  不一定能事先确定,我们在所得样品  $X_i$  的末尾补上一个数,比如 999,作为结束标志。当把样品送入计算机时,即可首先统计出  $N$  值,设  $N$  最大可达 1000,置数组  $X(1000)$  存放样品,每个  $X(I)$  值的格式允许有 7 位宽度,其中包括二位小数。

程序中采用符号如下:

$N$ 与 $XN$	采样的次数;
$X(I)$	样品 $X_i$ ;
$SUMX$	$X_i$ 之总和;
$SUMSQ$	$X_i$ 之平方和;
$AMEAN$	样品的算术平均值;
$VAR$	样品的方差;
$STDEV$	样品的标准偏差。

FORTRAN 程序:

```

C      * * * * *
C      SAMPLE MEAN, VARIANCE AND STANDARD DEVIATION
C      * * * * *
      DIMENSION X(1000)
      N = 0
      SUMX = 0.0
      SUMSQ = 0.0
      DO 10 I = 1,1000
      READ (5,100) X(I)
100  FORMAT (F7.2)
      IF (X(I) .EQ. 999.0) GO TO 20
      N = N + 1
      SUMX = SUMX + X(I)
      SUMSQ = SUMSQ + X(I) * X(I)
10  CONTINUE
20  XN = N
      AMEAN = SUMX/XN
      VAR = (XN*SUMSQ - SUMX**2)/(XN*(XN - 1.0))
      STDEV = SQRT(VAR)
      WRITE(6,200)N,AMEAN,VAR,STDEV
200  FORMAT(1X,2HN = ,I3,6HAMEAN = ,F10.2,
      1 5X,4HVAR = ,F10.2,5X,6HSTDEV = ,F10.2)

```

STOP  
END

使用 DO 语句时还要注意它的一些其它规定:

1) DO 语句的终端语句既不能是 FORMAT 语句、DATA 语句(后面讲)等非执行语句,也不能是 GO TO 语句、算术 IF 语句、STOP 或 PAUSE 语句、RETURN 语句(后面讲)和逻辑 IF 语句(指它的内嵌语句又为上述控制语句的情况)等控制语句。正是这诸多限制,所以导致我们常用 CONTINUE 语句来作终端语句。

2) 在执行循环体的过程中,可以不必完成 DO 语句规定的循环次数,而用 GO TO 语句或 IF 语句中途转出循环体外。这种情况称为非正常出口,而把执行完全部循环次数后离开循环体的情况称为正常出口。非正常出口带出来的循环变量的值仍保持不变,可以继续引用;而正常出口后的循环变量的值则不确定,不能继续引用。

**例 3** 考察下列程序:

```
DIMENSION X(100)
5  DO 10 I = 1, 100
    READ (5,111) X(I)
111 FORMAT (F10.4)
    IF (X(I)) 5, 5, 10
10  WRITE (6,222) X(I)
222 FORMAT (1X,F10.4)
    STOP
    END
```

这个程序企图给数组 X 依次输入 100 个值,并打印出其中的正数,这是做不到的。当 X 的某一个元素小于等于零时,转回标号为 5 的语句重新执行 DO 语句,这时,并不能实现取下一个 I,而是使 I 又重新取初值 1,循环又从 I = 1 开始。利用 CONTINUE 语句,程序可以改写如下:

```
DIMENSION X(100)
DO 10 I = 1, 100
READ (5,111) X(I)
111 FORMAT (F10.4)
IF (X(I)) 5, 5, 10
10  WRITE (6,222) X(I)
222 FORMAT (1X,F10.4)
5  CONTINUE
    STOP
    END
```

3) DO 语句可以有嵌套结构,即可以有多种循环。常被举来说明二重循环的简单例子是打印九九表程序。

**例4** 打印九九表程序(这里不去考虑打印格式)。

```
      DO 10 I = 1, 9
      DO 20 J = 1, 9
      K = I * J
      WRITE (6,100) I, J, K
100  FORMAT (1X,I2,1H*,I2,1H=,I3)
      20  CONTINUE
      10  CONTINUE
      STOP
      END
```

编写多重 DO 语句时还要注意的事情是:内循环必须完全嵌套在外循环体内,不得交叉;内循环可以并列,并列循环的循环变量可以同名;内外循环的终端语句连接在一起时,可以使用共同的一个终端语句;如果几重循环共用一个终端语句,则只有最内层循环体的转移语句可以转向这个终端语句。

标准的 FORTRAN IV 还允许从最内层循环体转出整个循环体之外,去执行一段程序后再转移回最内层循环体内。转去执行的这段程序称为循环的**扩充域**。它的位置虽不在循环体内,但可以看做是循环的一部分。扩充域中可以包括循环语句,但这个循环语句中再不能有扩充域。碰巧时,几个不同的循环也可用同一个扩充域。

**例5** 输入 10 个整数,挑出其中的正数,求各正数的阶乘的累加。FORTRAN 程序如下:

```
      SUM = 0.0
      DO 10 K = 1, 10
      T = 1.0
      READ (5,500) N
500  FORMAT (I3)
      IF (N .GT. 0) GO TO 800
550  SUM = SUM + T
      10  CONTINUE
      WRITE (6,600) SUM
600  FORMAT (1X,F10.0)
      STOP
800  DO 5 J = 1, N
      5  T = T * FLOAT(J)
      GO TO 550
```

END

程序用了扩充域。其中 FLOAT 是整数化实数的标准函数。

与循环程序设计密切相关的程序技巧是所谓**累加、累乘和计数**的概念。累加、累乘和计数,都是程序设计中经常遇到的运算,它们由几步标准手续组成。

对于形如  $S = \sum_{i=1}^N A_i$  的累加,选变量 S 作“累加器”,先将它置初值(往往是 0),

然后进行累加。用 FORTRAN 语句描述如下:

```
S = 0.0
DO 10 I = 1, N
10 S = S + A(I)
```

对于形如  $P = \prod_{i=1}^M B_i$  的连乘积用累乘,也是选一个变量作“累乘器”,先将它置初值

1,然后进行累积。用 FORTRAN 语句描述如下:

```
P = 1.0
DO 20 J = 1, M
20 P = P * B(J)
```

计数是累加的特例,即每次累加的是 1。

**例 6** 设有一个  $N \times N$  大方阵  $A(N,N)$  和一个有  $N$  个元素的指示数组  $IND(N)$ 。当  $IND(J) = -1$  时,就把大方阵  $A$  的第  $J$  行第  $J$  列去掉。这样可压缩出一个小一些的方阵  $B(M,M)$ 。设置两个计数器  $K$  和  $L$ ,我们可把这段矩阵压缩程序编写如下:

```
K = 0
DO 10 I = 1, N
IF (IND(I) .EQ. -1) GO TO 10
K = K + 1
L = 0
DO 20 J = 1, N
IF (IND(J) .EQ. -1) GO TO 20
L = L + 1
B(L,K) = A(J,I)
20 CONTINUE
10 CONTINUE
```

**例 7** 排序 (Sorting, 也称分类)程序。

排序是指把一批数据按一定的顺序,比如把  $N$  个数从大到小或从小到大,进行重新排列。排序程序很有用,排序的算法也很多。例如,有名的“气泡漂起”算法,说的就是:设有一个由  $N$  个实数构成的数组,要求从小到大重新排列这个数组。考察第一对数,如果前

一个大于后一个, 就让它们的位置交换过来。下一步是对第二与第三个数构成的数对作同样的处理, 直至第  $N-1$  与第  $N$  个数为止。结果, 最大的数就象气泡从水中“漂起”那样, 一直漂到最后(最上面)的位置。其后, 对前  $N-1$  个数构成的数组重复执行同样的过程, 直至按顺序排列整个数组为止。

“气泡漂起”算法可表述如下:

```

 $k \leftarrow N$ 
Next scan:  $i \leftarrow 1$ 
Next pair: 若  $a_i > a_{i+1}$ , 则  $a_i \leftrightarrow a_{i+1}$ 
 $i \leftarrow i+1$ 
若  $i < k$ , 则转向 Next pair
 $k \leftarrow k-1$ 
若  $k > 1$ , 则转向 Next scan

```

这里, 我们再介绍另一种排序算法, 即逐个元素比较法。设要排序的数组是  $A$ , 要求从大到小排列。将第一个元素与之后的元素逐一比较, 哪一个元素比第一个元素大, 就与第一个元素交换位置, 经过这样一轮扫描, 最大元素便被排到  $A(1)$  的位置上。

再将第二个元素与之后的元素逐一比较, 哪个元素比第二个元素大, 就与第二个元素交换位置, 经过这样第二轮扫描, 第二大元素便被排到  $A(2)$  的位置。

如此继续, 直至将倒数第二个元素与最后一个元素比较和必要时作交换, 便把整个数组的元素从大到小排列出来了。其中两个元素交换位置, 采用如图 2-6 所示的“三角交换法”。

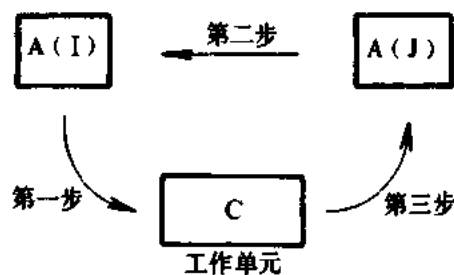


图 2-6

为了程序编写简单, 这里我们假定待排序的数组  $A$  共有 10 个元素。

FORTRAN 程序:

```

C      *** SORTING PROGRAM ***
C      -----
C      DIMENSION A(10)
C      DO 50 I = 1, 10
C      50 READ (5,100) A(I)
C      100 FORMAT (F10.2)

```

```

DO 60 I = 1, 9.
  I = I + 1
  DO 60 J = L, 10
    IF (A(J) - A(I)) 60, 60, 70
70  C = A(I)
    A(I) = A(J)
    A(J) = C
60  CONTINUE
DO 300 I = 1, 10
300  WRITE (6,400) A(I)
400  FORMAT (F10.2)
STOP
END

```

对各种排序算法,我们还可作时间复杂性和空间复杂性分析,即作程序执行时间和占用存贮空间量的分析,但这已超出本书的范围。

### 3. 隐 DO 循环的输入/输出与循环程序设计例(续)

在 READ 语句和 WRITE 语句中,可用隐 DO 循环方式来输入/输出数组元素的值。隐 DO 循环即隐含 DO 循环的意思。

假设数组 M 有 20 个元素, M(1) 至 M(20) 中的值分别为 101, 102, ..., 120。用隐 DO 循环方式写 WRITE 语句:

```

WRITE (6,100) (M(I), I = 1, 20)
100 FORMAT (1X,10 I5)

```

输出结果为

```

101 102 103 104 105 106 107 108 109 110
111 112 113 114 115 116 117 118 119 120

```

如果仅要求输出 M 中的奇下标变量值,则写

```

WRITE (6,100) (M(I), I = 1, 20, 2)
100 FORMAT (1X, 5 I5)

```

输出结果为

```

101 103 105 107 109
111 113 115 117 119

```

如果还要求同时输出下标值,则写

```

WRITE (6,100) (I,M(I),I = 1,20,2)

```

100 FORMAT (1X, 10I5)

输出结果为

```
1 101 3 103 5 105 7 107 9 109
11 111 13 113 15 115 17 117 19 119
```

隐 DO 循环中的初值、终值和步长也可以用已赋值的整型变量来表示,如

```
WRITE (6,100) (M(I), I = N1, N2, N3)
```

相仿, READ 语句中隐 DO 循环方式的用法如下例:

```
READ (5,200) (M(I), I = 1, 20)
```

读入的数据依次赋给数组 M 的 20 个下标变量。

二维以至更高维数组的输入/输出也可使用隐 DO 循环方式。设二维数组

$$MA = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

如果要求按原来排列形式把数组打印出来,可写

```
DO 5 I = 1, 3
  WRITE (6,200) (MA(I,J), J = 1, 3)
200 FORMAT (1X,3I5)
```

或写成

```
WRITE (6,200) ((MA(I,J), J = 1,3), I = 1,3)
200 FORMAT (1X, 3I5)
```

如果希望同时输出下标值,还可写成

```
WRITE (6,200) ((I,J,MA(I,J), J = 1,3), I = 1,3)
200 FORMAT (1X,2I2,I5,2I2,I5,2I2,I5)
```

隐 DO 循环也可用于普通变量的情形,如

```
WRITE(6,300) (X,Y,I = 1,4)
```

它相当于

```
WRITE (6,300) X,Y,X,Y,X,Y,X,Y
```

请注意的是,隐 DO 循环用于输入语句时,隐循环中的循环参数不能出现在括号内,如

```
READ (5,100) (J,A(I), I = 1,J,2)
```

这是错误的。应写成

```
READ (5,100) J,(A(I), I = 1,J,2)
```

总之,隐 DO 循环方式大大简化了输入/输出的程序设计。

下面继续给出一些循环程序设计举例。

**例 8** 矩阵相乘程序(简单情形)。

试编写 FORTRAN 程序: 首先,读入  $7 \times 5$  矩阵  $A$  和  $5 \times 4$  矩阵  $B$ , 其中每一行的



元素均按 F10.0 的格式穿孔在一张卡片上,并按行顺序读入;然后,计算  $A$  与  $B$  的乘积  $C$ ,即  $C = A \times B$ ;最后,按行顺序输出矩阵  $C$ 。

我们知道,  $C$  应该是  $7 \times 4$  矩阵,且其元素

$$c_{ij} = \sum_{k=1}^5 a_{ik} \times b_{kj}, \quad i = 1, 2, \dots, 7; \quad j = 1, 2, 3, 4.$$

显然,我们首先必须让计算机预留三个数组的存储位置,即  $A$  预留  $7 \times 5 = 35$  个,  $B$  预留  $5 \times 4 = 20$  个,  $C$  预留  $7 \times 4 = 28$  个。因此,程序中应有数组维数说明语句

```
DIMENSION A(7,5), B(5,4), C(7,4)
```

矩阵  $A$  与  $B$ , 按行顺序读入各行元素,且每行元素均穿孔在一张卡片上,故程序中的读语句应为

```
READ (5,60) ((A(I,J), J = 1,5), I = 1, 7)
READ (5,70) ((B(I,J), J = 1, 4), I = 1, 5)
60  FORMAT (5F10.0)
70  FORMAT (4F10.0)
```

矩阵  $C$  是  $A$  与  $B$  的乘积,对其中每一元素  $C(I,J)$  可按上述乘积计算公式计算,即有语句

```
C(I,J) = 0.
DO 80 K = 1, 5
80  C(I,J) = C(I,J) + A(I,K) * B(K,J)
```

或者,为了避免在循环运算中,多次计算数组  $C$  的下标变量而多费时间,我们也可引入新变量  $S$ , 而把上面语句改为

```
S = 0.
DO 80 K = 1, 5
80  S = S + A(I,K) * B(K,J)
90  C(I,J) = S
```

若我们再在  $S = 0.$  之前加上 DO 语句

```
DO 90 J = 1, 4
```

则我们可以求出  $C(I,1)$ ,  $C(I,2)$ ,  $C(I,3)$ ,  $C(I,4)$  各行元素之值。

又若我们再加上另一层 DO 语句

```
DO 90 I = 1, 7
```

即

```
DO 90 I = 1, 7
DO 90 J = 1, 4
S = 0.
DO 80 K = 1, 5
```

```

80  S = S + A(I,K) * B(K,J)
90  C(I,J) = S

```

则我们可以依次求出  $C(1,1)$ ,  $C(1,2)$ ,  $\dots$ ,  $C(1,4)$ ,  $C(2,1)$ ,  $C(2,2)$ ,  $\dots$ ,  $C(2,4)$ ,  $\dots$ ,  $C(7,4)$  各元素之值。

最后,我们将程序加上写语句

```

      WRITE (6,100) ((C(I,J), J = 1, 4), I = 1, 7)
100  FORMAT (1X,4E16.6)

```

则可将乘积矩阵  $C$  的元素按行输出。

综上各步,可装配矩阵乘积程序如下:

```

      DIMENSION A(7,5), B(5,4), C(7,4)
      READ (5,60) ((A(I,J), J = 1, 5), I = 1, 7)
      READ (5,70) ((B(I,J), J = 1, 4), I = 1, 5)
60  FORMAT (5F10.0)
70  FORMAT (4F10.0)
      DO 90 I = 1, 7
      DO 90 J = 1, 4
      S = 0.
      DO 80 K = 1, 5
80  S = S + A(I,K) * B(K,J)
90  C(I,J) = S
      WRITE (6,100) ((C(I,J), J = 1, 4), I = 1, 7)
100  FORMAT(1X,4E16.6)
      STOP
      END

```

**例9** 试编写一个完整的 FORTRAN 程序,按行读入一个  $6 \times 6$  方阵  $X$ ,  $X$  每行 6 个元素穿孔在一张卡片上;然后计算矩阵多项式

$$Z = 5X^4 + 9X^3 - 18X + 3I$$

其中  $I$  为  $6 \times 6$  单位矩阵。最后按行输出矩阵  $Z$ 。

FORTRAN 程序:

```

      DIMENSION X(6,6), Y(6,6), Z(6,6)
      READ(5,100) ((X(I,J), J = 1, 6), I = 1, 6)
100  FORMAT(6F10.0)
      DO 110 I = 1, 6
      DO 110 J = 1, 6

```

```

      S = 0.
      DO 105 K = 1, 6
105  S = S + X(I,K)*X(K,J)
110  Y(I,J) = S
      DO 120 I = 1, 6
      DO 120 J = 1, 6
      S = 0.
      DO 115 K = 1, 6
115  S = S + Y(I,K)*X(K,J)
120  Z(I,J) = S
      DO 130 I = 1, 6
      DO 130 J = 1, 6
130  Z(I,J) = 5.*Z(I,J) + 9.*Y(I,J) - 18*X(I,J)
      DO 140 I = 1, 6
140  Z(I,I) = Z(I,I) + 3.
      WRITE (6,150) ((Z(I,J), J = 1, 6), I = 1, 6)
150  FORMAT (1X,6E16.6)
      STOP
      END

```

**例 10** 为了打印 0~99 的整数平方表如图 2-7 所示的紧凑形式,试编 FORTRAN 程序。

	*	0	1	2	3	4	5	6	7	8	9
0	*	0	1	4	9	16	25	36	49	64	81
10	*	100	121	144	169	196	225	256	289	324	361
20	*	400	441	484	529	576	625	676	729	784	841
30	*	900	961	1024	1089	1156	1225	1296	1369	1444	1521
40	*	1600	1681	1764	1849	1936	2025	2116	2209	2304	2401
50	*	2500	2601	2704	2809	2916	3025	3136	3249	3364	3481
60	*	3600	3721	3844	3969	4096	4225	4356	4489	4624	4761
70	*	4900	5041	5184	5329	5476	5625	5776	5929	6084	6241
80	*	6400	6561	6724	6889	7056	7225	7396	7569	7744	7921
90	*	8100	8281	8464	8649	8836	9025	9216	9409	9604	9801

FORTRAN 程序:

```

      DIMENSION MA(10)
      DO 10 I = 1, 10
10  MA (I) = I - 1
      WRITE (6,100) MA
      WRITE (6,101)
      DO 11 I = 1, 10

```

```

      K = (I - 1) * 10
      DO 12 J = 1, 10
      L = J - 1
12  MA(J) = (K + L) * * 2
11  WRITE (6,102)K, MA
      WRITE (6,101)
100  FORMAT (6X,1H*,2X,10I5)
101  FORMAT (1X,29(2H--))
102  FORMAT (1X,I3,2X,1H*,2X,10I5)
      STOP
      END

```

## 习 题 二

1. 请问: 可供你使用的计算机系统是哪一厂家或什么名字的机器? 它使用的是哪一级的 FORTRAN 语言? 你索取到它的使用操作手册了吗?

2. FORTRAN 源程序纸上每行包括多少列? 分几个区? 各区的用途? 源程序中的语句部分可以从第几列写到第几列?

3. 指出下列符号哪些可以作为 FORTRAN 程序中的语句标号:

a) 10    b) 999    c) 00053A    d) \_\_\_\_123    e) L100    f) 001

4. 指出下列常数哪些是合乎 FORTRAN 规定的常数, 并指出它是整型还是实型:

a) 123.456    b) 12,345.2    c) 123,456    d) 1247    e) \$1234    f) \$12,345.00

5. 把下列各数表示成 FORTRAN 的整型常数:

4096.0    -195.5    +314     $0.11 \times 10^4$      $-2^5$      $0.03 \times 10^4$      $-10^3$     100,000

6. 把下列各数表示成 FORTRAN 的 F 型和 E 型实数:

25	3.14159	-98.01	45600
$-0.369$	$10^{12}$	$9.9 \times 10^{11}$	$-\frac{1}{100}$
$2^{22}$	8970	100,000,000	$2.718 \times 10^{-2}$

7. 指出下列各数中哪些是合乎 FORTRAN 规定的整型数或实型数:

a) +0.05    b) 32769    c) -E7    d) 10E2    e) .E-3    f) 3E-5    g) .1E2

h) .01    i) 6E1.2    j) 35.    k) .E-3    l) 13.000

8. 指出下列哪些符号是合乎 FORTRAN 规定的变量名, 并指出其类型:

a) THINK2    b) VEL\*V    c) INDICATOR3    d) INDI23    e) 6SIX    f) SAL\$

9. 指出下列哪些符号可以用来标识数组元素, 并指出其类型

a) X1    b) N(8)    c) N( $\rho$ )    d) A(M,N)    e) ROOT2    f) C(1,2,3)

10. 下列类型说明哪一个是对的:

a) INTEGER, VOL, X, Y

b) REAL I, J, K

c) REAL X, Y, I

d) INTEGER ROOT1, ROOT2

11. 利用隐式规则, 简写下列类型语句:

REAL VOL, X, Y, MASS, LIMIT

INTEGER KAPPA, LEVEL, P, QUANT

12. 对下列数组写出相应的数组说明语句:

(1) 8 行 8 列的整型数组 UMM.

(2) 100 个元素的实型数组 IKK.

(3)  $20 \times 5$  的实型数组 ARR.

13. 根据维数说明语句

DIMENSION L(7,12), A(4,3,7)

试求下列各数组元素(指定序号)的各下标值:

a) 8    b) 39    c) 61    d) 54    e) 19    f) 69

14. 根据维数说明语句:

DIMENSION X(8,4), A(4,6,5)

试求下列各数组元素的序号:

a) X(2,1)    b) X(8,3)    c) X(7,3)    d) A(2,4,5)    e) A(3,1,1)    f) A(3,3,3)

15. 将下列各式写成 FORTRAN 算术表达式:

(1)  $-(x^2 + y^2)$

(2)  $\frac{ax + by}{ax - by}$

(3)  $\left(\frac{x}{y}\right)'$

(4)  $\frac{1}{8} \cos(\alpha + \beta)$

(5)  $1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720}$

(6)  $(x + \sqrt{x^2 - 1})^n$

(7)  $R e^{-\frac{0.693}{T} \tau}$

(8)  $\ln |\sec x + \tan x|$

(9)  $x_{ij}^2 - \frac{\sin(y_{ij}^1 - x_{ij}^1)}{7.2 \times 10^{-3} - x_{ij}}$

(10)  $\frac{5.4(a_{ij} + b_{ij}) - B_4 \sin x}{0.05d_{ij} - 5\sqrt{b_{ij} - 1}}$

16. 下面右边的 FORTRAN 表达式是根据左边的数学表达式写成的, 请指出其中哪些是不正确的写法, 并把它改正:

(1)  $\frac{2x}{yx}$

$2X/Y * Z$

(2)  $3^x$

$3 ** X$

(3)  $\frac{1}{2} c^2 \sin x$

$1/2EXPY * SINX$

(4)  $(m + n)^{\frac{1}{2}}$

$SQRT(M + N)$

(5)  $\frac{1}{a^2} \left(\frac{i}{10}\right)^3$

$1/A ** 2 * I ** 3 / 1000$

17. 指出下列式子中哪些不能作为 FORTRAN 算术表达式, 并说明其原因:

(1) X

(2)  $2X + A(2,1 + 1) * Y$

(3)  $(-0.7E + 4 - Y ** 2 * ABS(Y1(2)))$

(4)  $2 ** (U + 3.1416V) + (A2 + B1 * U) ** 5$

(5)  $1/3$

18. 数学式子  $\frac{Im}{a}$  可写成 FORTRAN 表达式

$L * M / N$  或  $I./N * M$  或  $M/N * L$

如果  $I = 5, m = 13, n = 4$ , 按上面三个表达式计算, 结果是否相同? 为什么?

19. 下面两组表达式左右是否相同:

a)  $(1 + j)/N$  与  $1/N + j/N$

b)  $(X + Y)/Z$  与  $X/Z + Y/Z$

20. 把下面计算式写成 FORTRAN 的算术赋值语句:

$$(1) R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

$$(2) \tau = 2\pi\sqrt{\frac{l\cos\theta}{g}}$$

$$(3) v_i = \sqrt{gR}$$

$$(4) dx = \frac{a}{10} \ln \left| \operatorname{tg} \left( \frac{\pi}{4} + \frac{\varphi}{2} \right) \right|$$

$$(5) r_i = \frac{\alpha}{x_{i+1} - x_i} \left( \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \right)$$

$$(6) c_{ij} = \sum_{k=1}^3 a_{ik} \cdot b_{kj}$$

21. 指出下列算术赋值语句的错误:

$$(1) y = 2x + A$$

$$(2) A = B = 3.14159$$

$$(3) D = B * * 2 - 4AC$$

$$(4) 5 = K$$

$$(5) V - A = X * * 1.6$$

$$(6) 3X = 4Y$$

$$(7) W = 1./ - 2. * C * * 4$$

$$(8) S = \pi R12$$

$$(9) K(\rho) = 1 * * A$$

$$(10) \operatorname{TAN}(A) = V * R/G$$

$$(11) \operatorname{MAX} = \operatorname{SQRT}(1 + 5)$$

$$(12) -V = A + B$$

22. 指出下列每组语句的输出形式:

(1) DIMENSION A(4,4)

WRITE(6,10) A

10 FORMAT(1X,3F8.2)

(2) WRITE(6,20)A,B,C

20 FORMAT(1X,F8.2)

(3) WRITE(6,11)M,A

11 FORMAT(1X,I5,F10.5)

(4) DIMENSION A(10),M(10)

WRITE(6,15)A,M

15 FORMAT(1X,10F10.2,10I5)

(5) WRITE(6,30)I,A,J,B,K,C

30 FORMAT(1X,3(I5,F10.2))

23. 下面各题中的输出语句,能否用所给定的格式语句:

(1) DIMENSION A(3),I(3)

WRITE(6,10)A,I

10 FORMAT(1X,3(F10.2,I5))

(2) DIMENSION C(10)

```

WRITE(6,11) C
11 FORMAT(1X,E10.4)

(3) WRITE(6,20)W,I,A
20 FORMAT(1X,F5.3)

(4) DIMENSION M(10,10)
WRITE(6,100)M
100 FORMAT(1X,5I5)

(5) DIMENSION A(2,4)
WRITE(6,110)A
110 FORMAT(1X,2F5.2)

```

24. 已知  $A = 1.235$ ,  $B = -10.0$ ,  $C = 3.26$ ,  $D = -2.1785$ , 按下面的输入格式, 应如何送入数据:

```

(1) READ(5,10)A,B,C,D
10 FORMAT(F8.4)

(2) READ(5,20)A,C,B,D
20 FORMAT(2F8.4)

(3) READ(5,30)A,B,C,D
30 FORMAT(4F8.4)

```

25. 已知  $X = -50.993$ ,  $Y = 10.101$ ,  $Z = 357.23569$ , 执行完下列程序后, 输出结果是什么;

```

READ(5,10)X,Y,Z
10 FORMAT(3F10.4)
S = X + Y + Z
S = S/3.
S1 = ABS(X) + ABS(Y) + ABS(Z)
S1 = S1/3.
WRITE(6,15)X,Y,Z,S,S1
15 FORMAT(1X,3F10.4)
STOP
END

```

26. 已知  $X = 3.5^\circ$ ,  $Y = 6.8^\circ$ , 编程序计算

$$Z = \frac{\sin|X + Y|}{\sqrt{\cos|X + Y|}}$$

并输出  $X, Y, Z$ . (注意, 要把角度化为弧度.)

27. 已知三角形的两个角  $\angle A$ ,  $\angle B$  及夹边  $c$ , 试编写求三角形面积的程序, 面积公式为

$$S = \frac{c^2 \sin A \sin B}{2 \sin(A + B)}$$

28. 指出在执行下列条件语句后控制转到哪个语句:

```

(1) IF ((A + B)/R - C)30,40,50
    其中  $A = 3.4$ ,  $B = 2.09$ ,  $C = -6.3$ ,  $R = 2.1$ 

```

(2) IF ((A\*B - C)\*A + B/(C - B))82,140,200

其中 A = -14.2, B = 3.71, C = -6.1

(3) IF (IM - IM/IN\*IN)10,20,10

其中 IM = 48, IN = 13

29. 指出执行下列语句的结果所应转向的语句标号:

(1) GO TO 205

(2) ASSIGN 29 TO M

GO TO M,(6,14,79,29,104,31)

(3) ASSIGN 15 TO K

ASSIGN 18 TO K

GO TO K,(3,15,31,18,40,39,22)

(4) I = 4

GO TO (1,704,39,56,14,85),I

(5) I = 5

L = I - 3

GO TO (105,2,103,99,4,17),L

30. 假定给出数列  $n_1, n_2, \dots, n_k$ , 编写统计该数列中非负数个数及负数个数的程序.

31. 试编写计算下列函数的 FORTRAN 程序:

$$y = \begin{cases} 1 & x < 0 \\ x & 0 \leq x < 1 \\ \frac{3}{2}x^2 - \frac{1}{2} & 1 \leq x < 2 \\ \frac{5}{2}x^3 - \frac{3}{2}x & 2 \leq x < 3 \\ \frac{35}{8}x^4 - \frac{15}{4}x^3 + \frac{3}{8} & 3 \leq x < 4 \end{cases}$$

32. 指出下列哪些式子不能作为逻辑表达式,并说明其原因:

(1) A + B .OR. X .AND. (A - B) \* A .LT. 0

(2) .NOT. (A/(A+B) + B/(A-B)) \* C .GE. A \* B \* C .OR. A + B .LT. A - C

(3) X .OR. Y .OR. .NOT. (X .OR. .NOT. Y) .AND. X .NOT. Y

(4) .NOT. ((A - C \* B + 0.023) \* A .LT. (B/(C - A) - 102.3) .AND. A/(B + C - A) .OR. X .OR. Y

(5) .NOT. X(.AND. Y .OR. X) .AND. .NOT. Y

33. 确定下列逻辑表达式的值:

(1) A \* B - C .GE. 0.

其中, A = 0.63, B = 1.27, C = -2.3

(2) .TRUE. .OR. X

其中 X = .FALSE.

(3) (((Y .OR. .NOT. X) .AND. X .OR. .NOT. Y) .AND. Y .OR. X) .AND. .NOT. Z

其中 X = .TRUE., Y = .TRUE., Z = .FALSE.

(4) P .GE. Q .OR. S .NE. P

其中 P = 0.1E4, Q = 60.27E3, S = 1070.2

(5) (A + B) \* C - 0.6 \* A .GT. A \* \* 2 - 16.3 \* C

其中 A = -6.4, B = 0.3E4, C = 6001.56E-3

34. 已知



$$y = \begin{cases} (x-1)/x & x < 0 \\ 1 & x = 0 \\ x & x > 0 \end{cases}$$

$$z = \frac{3}{4}y \sin x + \frac{1}{x} \csc y + \sqrt{x^2 + y^2}$$

试编程序由键盘输入  $x$  值, 计算  $y, z$ , 并打印  $x, y, z$ .

35. 斐波那奇 (Fibonacci) 序列

0, 1, 1, 2, 3, 5, 8, 13, 21, ……

是这样形成的: 序列中第三个元素起, 每一个元素均由它前面两个元素之和构成, 例如  $3+5=8$ . 试写一个 FORTRAN 程序, 来产生这个序列的前 20 项.

36. 确定下列循环执行的次数:

(1) DO 235 K=1, 50, 2

(2) DO 6 L=3, 100, 4

(3) DO 4 M1=4, 5, 1

(4) DO 8 J1=2, 35

(5) DO 39 M=6, 80, 3

(6) DO 2 M2=2, 2, 2

(7) DO 12 L1=1, 202, 4

37. 指出下列循环语句在写法上存在的错误:

(1) DO 24 K=1, 10

A(K)=B1 \* K - B2 \* K \* \* 2 + B3 \* K \* \* 3

24 IF (X .EQ. Y) S=S+1

(2) DO 4 M=2, 16, 3

1 S=S+A(M)

IF(S) 1, 4, 4

4 GO TO 12

5 CONTINUE

(3) DO 101 I=1, 20

R=G(I)+0.75

IF(K) 101, 100, 100

100 DO101 J=1, 20

101 B=B+R+A(I, J)

(4) DO 31 I=1, 20

DO 32 K=1, 20

32 S=S+A(I, K)

31 B(I)=S

(5) DO I 71=1, 70

1 S=S+A(71)

38. 把下面的循环语句改用其它转向语句来完成:

(1) DO 125 I=1, 30

125 A(I)=B(I)

(2) DO 4 I=5, 50, 2

4 S=S+I

(3) DO 3 L=1, 10

DO 3 K=1, 10

3 A=B(L,K)+A

(4) DO 17 L=1, 25, 4

17 A(L)=L

(5) DO 2 I=1, 10

DO 2 J=1, 10

DO 2 K=1, 10

2 A=B(I,J,K)+A

39. 试用 FORTRAN 生成矩阵

$$A = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 2 & 1 & 1 & \cdots & 1 \\ 3 & 2 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 9 & 8 & 7 & \cdots & 1 \\ 10 & 9 & 8 & \cdots & 1 \end{bmatrix}$$

并打印输出。

40. 编写含有循环的程序, 计算下面多项式之值:

$$P = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

其中  $n, a_0, \dots, a_n, x$  均由键盘输入。

[提示] 将多项式写成  $((\cdots(a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0$

41. 已知数列  $n_1, n_2, \dots, n_k$ , 请编一程序重排此数列的顺序, 使到前面正数部分从大到小排列, 后面负数部分从小到大排列。

42. 试用计算机打印杨辉三角形

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

```

43. 写一个 FORTRAN 程序, 将英寸转换为厘米(1 英寸 = 2.54 厘米), 给出从 1 英寸到 1 英尺(步长 1 英寸)的各转换值。

44. 编写一程序, 求出 100 之内的所有各组勾股数(即满足  $A^2 + B^2 = C^2$  的正整数  $A, B, C$ ) 并输出所有结果。

45. 写一个 FORTRAN 程序, 计算

$$Y = \sum_{N=0}^M A_N X^N$$

其中  $N = 2M + 1$ , 即

$$Y = \sum_{N=0}^M (2N+1)X^N \\ = 1 + 3X + 5X^2 + 7X^3 + \cdots + (2M+1)X^M$$

$M$  和  $X$  之值由程序读入.

46. Wallis 公式计算  $\frac{\pi}{2}$  由下面无穷乘积给出:

$$\frac{\pi}{2} = \frac{2}{1 \cdot 3} \cdot \frac{4}{3 \cdot 5} \cdot \frac{6}{5 \cdot 7} \cdot \frac{8}{7 \cdot 9} \cdots$$

写一个程序,以这个无穷乘积的前 10 项来计算  $\frac{\pi}{2}$ , 并打印出计算过程的结果,以便你观察每一项对乘积所产生的影响. 注意乘法因子的通项为

$$\frac{(2n)^2}{(2n-1)(2n+1)}$$

然而你不必使用此通项. 为简单起见,可用一个 DO 语句如

DO 10 N = 2, 20, 2

另外,  $\pi$  的 16 位有效数值为

$$\pi \approx 3.141592653589793$$

47. 编 FORTRAN 程序,求下列方程组

$$\begin{cases} 2x^2 - xy - 5x + 1 = 0 \\ x - y^2 + 1.6 = 0 \end{cases}$$

的根  $\bar{x}, \bar{y}$ . 已知点  $(\bar{x}, \bar{y}) \in (3.5 \leq x < 3.6, 2.2 \leq y < 2.3)$  采用下列的迭代法来求根

$$x_{n+1} = \sqrt{\frac{x_n(y_n+5)-1}{2}}, y_{n+1} = \sqrt{x_n + 1.6}$$

其中初值为  $x_0 = 3.5, y_0 = 2.2$  其误差要求为  $|x_{n+1} - x_n| \leq \epsilon, |y_{n+1} - y_n| \leq \epsilon, \epsilon < 10^{-4}$ .

## 附录 A FORTRAN IV 标准函数表

表 A-1 内部函数

内 部 函 数	定 义	符号名字	变元数	类 型	
				变元	函数
绝对值	$ x $	ABS IABS DABS	1	实型 整型 双精度型	实型 整型 双精度型
截 断	$x$ 的符号乘 $\leq  x $ 的最大 整数	AINT INT IDINT	1	实型 实型 双精度型	实型 整型 整型
余 数 (见下面注)	$x_1(\text{mod } x_2)$	AMOD MOD	2	实型 整型	实型 整型
选最大值	$\max(x_1, x_2, \dots)$	AMAX0 AMAX1 MAX0 MAX1 DAMX	$\geq 2$	整型 实型 整型 实型 双精度型	实型 实型 整型 整型 双精度型

续表 A-1

内 部 函 数	定 义	符号名字	变元数	类 型	
				变元	函数
选最小值	$\min(x_1, x_2, \dots)$	AMIN0 AMINI MIN0 MINI DMINI	$\geq 2$	整型 实型 整型 实型 双精度型	实型 实型 整型 整型 双精度型
浮 点	从整型转为实型	FLOAT	1	整型	实型
定 点	从实型转为整型	FIX	1	实型	整型
符号传送	$x_2$ 的符号 乘 $ x_1 $	SIGN ISIGN DSIGN	2	实型 整型 双精度型	实型 整型 双精度型
正 差	$x_2 - \min(x_1, x_2)$	DIM IDIM		实型 整型	实型 整型
取双精度变元的最大有效部分		SNGL	1	双精度型	实型
取复变元的实部		REAL	1	复型	实型
取复变元的虚部		AIMAG	1	复型	实型
用双精度形式表示单精度变元		DBLE	1	实型	双精度型
用复数形式表示两个实型变元	$x_1 + x_2\sqrt{-1}$	CMPLX	2	实型	复型
取复变元的共轭		CONJG	1	复型	复型

注: 函数 MOD 或 AMOD( $x_1, x_2$ ) 定义为  $x_1 - [x_1/x_2]x_2$ , 其中  $[x]$  表示不超过  $x$  的最大整数, 且其符号与  $x$  一致。

表 A-2 基本外部函数

基本外部函数	定 义	符号名字	变元数	类 型	
				变元	函数
指 数	$e^x$	EXP	1	实型	实型
		DEXP	1	双精度型	双精度型
		CEXP	1	复型	复型
自然对数	$\log_e(x)$	ALOG	1	实型	实型
		DLOG	1	双精度型	双精度型
		CLOG	1	复型	复型
普通对数	$\log_{10}(x)$	ALOG10	1	实型	实型
		DLOG10	1	双精度型	双精度型
三角正弦	$\sin(x)$	SIN	1	实型	实型
		DSIN	1	双精度型	双精度型
		CSIN	1	复型	复型
三角余弦	$\cos(x)$	COS	1	实型	实型
		DCOS	1	双精度型	双精度型
		CCOS	1	复型	复型

续表 A-2

基本外部函数	定 义	符号名字	变元数	类 型	
				变 元	函 数
双曲正切	$\tanh(x)$	TANH	1	实型	实型
平方根	$(x)^{1/2}$	SQRT	1	实型	实型
		DSQRT	1	双精度型	双精度型
		CSQRT	1	复型	复型
反正切	$\arctan(x)$ $\arctan(x_1/x_2)$	ATAN	1	实型	实型
		DATAN	1	双精度型	双精度型
		ATAN2	2	实型	实型
		DATAN2	2	双精度型	双精度型
余数(见下面注)	$x_1(\text{mod } x_2)$	DMOD	2	双精度型	双精度型
模 数		CABS	1	复型	实型

注: 函数 DMOD( $x_1, x_2$ ) 定义为  $x_1 - [x_1/x_2]x_2$ , 其中  $[x]$  表示不超过  $x$  的最大整数, 且其符号与  $x$  一致。

## 第3章 FORTRAN 语言的较复杂部分

### 3-1 FORTRAN 的基本概念(续)

在 FORTRAN IV 中,除了上一章已经讨论过的整型和实型两种数值量外,还有双精度型、复型、逻辑型和字符型四种量。我们进一步讨论这些量的概念及其用法。

#### 1. 双精度型常数与变量

有时,普通实常数的精确度并不能满足实际计算的需要。为此, FORTRAN IV 定义了具有双倍精确度的双精度型常数和变量。把精度“翻了一翻”。

通常,一个普通实常数在计算机内占四个字节(16位机的两个单元),而一个双精度实常数则占八个字节。一个普通实常数的有效位是6~7位,一个双精度实常数的有效位则可多达16~17位。

双精度实常数一律以指数形式表示。它除了比普通实常数多一倍以上的有效位数之外,只需把指数符号E换成D,如

0.1357924680716552D3

-0.5004113005D-5

975.346D-8

0.0D0

如果写 1.234567890123456,编译时则把它截尾取7位有效数字作普通实数处理。请注意,并没有双精度整数。

用来存放双精度实数的变量就是双精度实变量。双精度实变量必须用显式说明,其形式为

DOUBLE PRECISION (变量表)

例如,说明语句

DOUBLE PRECISION M1, M2, DR, DS, XPU

这就在本程序块中引入了五个双精度变量 M1, M2, DR, DS, XPU。这时, I-N 规则对这些量就不起作用了。

与普通实型数的情形类似,双精度数的输入/输出可以用 rDw.d 字段描述符,输出时也可用 F 型输出。例如

DOUBLE PRECISION A, KK

A = 3.0D4

KK = 1/A

WRITE (6,100) A, KK

100 FORMAT (1X,2D20.12)

STOP

END

输出结果是

0.3000000000000D    3 0.333333333333D 0  
(20 格)                      (20 格)

如果把标号为 100 的格式语句改为

100 FORMAT (1X,2F20.12)

则输出结果是

30000.0000000000000    0.33333333333333  
(20 格)                      (20 格)

双精度数的运算速度比较慢,尤其是那些指令系统中没有双精度操作的计算机,要用专门的子程序来实现双精度运算,计算速度往往成十倍地降低,因此,非迫不得已,不轻易使用双精度运算。

## 2. 复型常数与变量

我们知道,一个复数包括一个实数作实部和另一个实数作虚部(系数).在 FORTRAN IV 中,用包括在一个括号内的两个实常数来表示一个复型常数,其中前一个表示实部,后一个表示虚部,用逗号隔开,例如

(-5.,7.2i)            表示  $-5 + 7.2i$   
(0.,2.i)            表示  $2i$   
(4.E-2,-5.E-1i) 表示  $0.04 - 0.5i$

标准 FORTRAN IV 中只用普通实型数来构成复型数,有的非标准 FORTRAN IV 还允许用双精度型实数来构成复型数,即所谓双精度复型数.允许在程序中直接编写和运算复数,这是 FORTRAN 的特点之一.复数计算是科技计算中经常遇到的计算.例如,交流电路中的电流、电压、阻抗等就采用复数来表示.如果电压  $U$  和电流  $I$  都是复型量,则通过赋值语句

$$Z = U/I$$

就可求出阻抗  $Z$  的复数值.

用来存放复型常数(相当于存放两个实型常数)的变量就是复型变量.复型变量也要用说明语句引入,一般形式是

COMPLEX (变量表)

例如,在本程序块中写

COMPLEX U, I, Z

这就引入复型变量  $U, I, Z$ . 这时 I-N 规则对这些量也不再起作用了.

复型量运算也使用实型量运算的五种算术运算符,只有一个重要的例外,当  $B$  为复型量时,不能作  $A ** B$  运算.我们知道,若  $A = a + bi$ ,  $B = c + di$ , 则

$$A \pm B = (a + bi) \pm (c + di)$$

$$\begin{aligned}
&= (a \pm c) + (b \pm d)i \\
A * B &= (a + bi) * (c + di) \\
&= (ac - bd) + (ad + bc)i \\
A / B &= (a + bi) / (c + di) \\
&= \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2} i \\
A ** N &= (a + bi) ** N = \rho^N (\cos N\theta + i \sin N\theta) \\
(N \text{ 为整数, } \rho &= \sqrt{a^2 + b^2}, \theta = \tan^{-1}(b/a))
\end{aligned}$$

一般说,复型数作加、减、乘、除和乘方运算后结果仍为复型数。在 FORTRAN 程序中,两个复数的运算,比如相乘,可以写成

$$(2.0, 3.0) * (3.0, 4.0)$$

如果还要把它赋给一个复型变量 A, 可以写成

$$A = (2.0, 3.0) * (3.0, 4.0)$$

但注意,当复数括号内是表达式时,不能直接赋给一个复变量。这时可用基本外部函数 CMPLX (见第 2 章附录 A 的标准函数表) 来把括号内代表实部和虚部的表达式组合成复数。如

$$\begin{aligned}
A &= \text{CMPLX}(X * 2.5, 1.0 + Y) \\
B &= \text{CMPLX}(R, S * T) + \text{CMPLX}(5.0 * R, T)
\end{aligned}$$

因为每个复型量实质上包含二个实数,所以复型数的输入/输出相当于对二个实数的输入/输出。例如:

```

COMPLEX, U, I, Z
.....
READ (5, 10) I, Z
10  FORMAT (4F10.4)
.....
WRITE (6, 20) U
20  FORMAT (1X, 2F10.4)
.....

```

**例 1** 设在某电工计算中, 已知

$$Z = \frac{Z_1 Z_2}{Z_1 + Z_2}, \quad I = \frac{U}{Z_0 + Z}$$

要求电流  $I_1$

$$I_1 = I \times \frac{Z}{Z_1} = \frac{U}{Z_0 + Z} \times \frac{Z}{Z_1}$$

的实部、虚部、模和相位  $\varphi_1$ , 其中各参数的值为:

$$\begin{aligned}
U &= 220 \text{ 伏}, & Z_0 &= r_0 + j\omega L_0, \\
Z_1 &= r_1 + 1/j\omega C_1, & Z_2 &= r_2 + j\omega L_2, \\
r_0 &= 10 \text{ 欧}, & L_0 &= 10^{-3} \text{ 亨}, \\
r_1 &= 100 \text{ 欧}, & C_1 &= 100 \text{ 微法},
\end{aligned}$$



$r_1 = 50$  欧,  $L_1 = 10^{-2}$  亨,  
 $C_1 = 200$  微法.

程序中设置 ABSI1 代表电流  $I_1$  的模, PHASE 代表  $I_1$  的相位  $\varphi_1$ . 注意到, 参数中的  $\omega = 50 \times 2\pi \approx 314$ , PHASE 化成角度时乘以 57.2958. 计算程序可设计如下:

```

      REAL L0, L2
      COMPLEX U, I1, Z0, Z1, Z2, Z
      U = (220., 0.)
      READ(5,100) R0, L0, R1, C1, R2, L2, C2
100  FORMAT (7F10.2)
      Z0 = CMPLX(R0, 314.0 * L0)
      Z1 = CMPLX(R1, -1.0 / (314.0 * C1))
      Z2 = CMPLX(R2, 314.0 * L2 - 1.0 / (314.0 * C2))
      Z = Z1 * Z2 / (Z1 + Z2)
      I1 = U / (Z0 + Z) * Z / Z1
      ABSI1 = CABS(I1)
      PHASE = 57.2958 * ATAN2(AIMAG(I1), REAL(I1))
      WRITE (6,200) I1
      WRITE (6,200) ABSI1, PHASE
200  FORMAT(1X, 3F15.6, F15.2)
      STOP
      END

```

如果我们从卡片或键盘(按具体机器规定的方式)依次输入数据:

10.0, 1E-3, 100.0, 1E-4, 50.0, 1E-2, 2E-4

则可得如下计算结果:

0.002815    0.068965    (I1 的实部与虚部)  
 0.069023    87.66        (I1 的模与相位)

综上所述, 我们可进一步总结不同类型数据混合运算的类型规则 (参见第 2 章 2-2 节) 如下表 3-1 和表 3-2.

表 3-1

+ - * /	整 型	实 型	双精度型	复 型
整 型	(整)	(实)。	(双)。	(复)。
实 型	(实)。	(实)	(双)	(复)
双精度型	(双)。	(双)	(双)	(复)。
复 型	(复)。	(复)	(复)。	(复)

表 3-2

	指数整	指数实	指数双	指数复
基数整	(整)	(实)*	(双)*	(不允许)
基数实	(实)	(实)	(双)	复*
基数双	(双)	(双)	(双)	复*
基数复	(复)	(复)*	(双复)*	复*

表中带\*的运算是标准 FORTRAN IV 没有允许的运算。

对于不同类型数据的赋值 ( $V = e$ ) 规则, 也可总结 (参见第 2 章 2-3 节) 如下表 3-3。

表 3-3

右端表达式 类型 左端变量类型	整 型	实 型	双精度型	复 型
整 型	直接赋	取整再赋	取整再赋	实部取整。
实 型	化实再赋	直接赋	化为普通实型再赋	实部照赋。
双精度型	先化成双精度再赋	换成双精度再赋	直接赋	实部扩成双精度再赋。
复 型	化实赋实部虚部赋 0。	赋实部虚部赋 0。	化实赋实部虚部赋 0。	直接赋

表中带\*的项也是标准 FORTRAN IV 没有允许而某些非标准 FORTRAN IV 所允许的规则。

### 3. 逻辑型常数与变量·逻辑赋值语句

在第 2 章 2-5 节中, 我们讲了关系表达式与逻辑表达式, 它们都与逻辑值有关。这里我们再系统讲逻辑型常数与变量, 还要讲到逻辑赋值语句。

到这里为止, 我们主要讨论的是整型数据、实型数据、双精度型数据和复型数据, 它们统称为数值型数据。象其它算法语言一样, 在 FORTRAN 中也使用逻辑型数据。逻辑型数据仅取两个逻辑值“真”或“假”, “真”与“假”这两个值就是逻辑常数。在 FORTRAN 中, “真”表示为 .TRUE., “假”表示为 .FALSE.. 取逻辑值的量称为逻辑量, 而用来存放逻辑常数的变量称为逻辑变量。

逻辑变量也要用类型说明语句来引入, 其一般形式为

LOGICAL 〈变量表〉

例如:

LOGICAL B1, B2, N, M

这就在本程序块中引入四个逻辑变量 B1, B2, N, M。这时 I-N 规则对这些量也就不起作用了。

有了逻辑常数和逻辑变量的一般概念,这将有助于我们加深对关系运算符、逻辑运算符、关系表达式和逻辑表达式的理解(见第2章2-5节)。

逻辑型数据也有输入/输出问题。逻辑型变量的输入和输出使用L型字段描述符 $rLw$ ,其中 $r$ 为重复数, $w$ 为字段宽度。例如,设已被说明为逻辑变量的B1, B2, N, M分别取值. TRUE., .FALSE., .FALSE., .TRUE., 那么

```
WRITE (6,111) B1, B2, N, M
111 FORMAT (1X,L4,L4,L3,L3)
```

或将111语句写成

```
111 FORMAT (1X,2L4,2L3)
```

则输出的结果为

```
      T      F      F      T
  _____
 (4格)  (4格) (3格) (3格)
```

在FORTRAN中,真值(.TRUE.)打印出字母T,假值(.FALSE.)打印出字母F。

如果要将逻辑常数,比如. TRUE. 与 .FALSE. 读入赋给已被说明的逻辑变量BB和CC,则可写成

```
READ(5,222) BB, CC
222 FORMAT (2L3)
```

这时准备输入的数据的格式应是

```
__T__F
```

而实际给BB和CC赋值的是. TRUE. 和 .FALSE.。一般地,只要在 $w$ 的宽度范围内输入的第一个字母是T或F,则不论其后跟的是什么符号,都分别作为逻辑常数. TRUE. 或. FALSE. 输入。

上一章我们已经介绍了算术赋值语句和标号赋值语句,现在介绍另一个赋值语句——逻辑赋值语句。逻辑赋值语句的一般形式为

〈逻辑变量〉=〈逻辑表达式〉

其意义就是先计算出右端〈逻辑表达式〉之值(逻辑值),然后赋给左端的〈逻辑变量〉。例如,假设B1, B2, N, M已被说明为逻辑变量,则

```
B1 = (1. .EQ. 0.)
B2 = .FALSE.
N = B1 .AND. B2
M = (X .GT. Y) .OR. (Z .GE. 0.)
```

都是逻辑赋值语句。

**例2** 阅读下列程序:

```
DIMENSION B(10), C(10)
```

```

LOGICAL A(10), LA(10)
TRU = .TRUE.
2  READ (5,77) (B(J), C(J), J = 1, 10)
   DO 50 I = 1, 10
4   A(I) = B(I) .GE. C(I)
6   IF (A(I)) LA(I) = .FALSE.
8   IF (A(I) .NOT. TRU) LA(I) = .TRUE.
50  CONTINUE
   WRITE (6,88) (LA(I), I = 1, 10)
77  FORMAT (2F6.1)
88  FORMAT (1X,6L7)
STOP
END

```

我们看到,在语句4中,根据 B(I) 与 C(I) 的大小,将给逻辑下标变量 A(I) 赋与真值或假值;在语句6中,如果 A(I) 是 .TRUE., 则数组元素 LA(I) 被赋与 .FALSE.; 而在语句8中,那些对应于 A(I) 不为 .TRUE. 的 LA(I) 均取 .TRUE.; 最后,在语句88中使用 L 型字段描述符,对应 LA(I) 的真、假值,实际打印输出 T 或 F.

#### 4. 字符型常数

在 FORTRAN IV 中,把一个或多个字符组成的一串符号,前面冠以 nH, n 是这串符号的个数,称为字符型常数。例如

```

8HCOMPUTER
17HNUMERICAL METHODS
9H (MM/DD/YY)
2HX =

```

都是字符型常数。要注意的是,空格也是字符型常数中的有效符号。

另外,如我们在上一章已经提到的,在某些非标准 FORTRAN IV 和(或)FORTRAN 77 中,字符型常数还可采用或改用单引号夹起来的等价形式。如

4HBOOK 等价于 'BOOK'

还要注意的,是 FORTRAN IV 语言中没有字符型变量的规定;字符常数存放在其它类型的变量或数组中。FORTRAN IV 规定每个整型变量或实型变量可以存放 4 个字符,双精度型变量可以存放 8 个字符,数组元素也有相应的存法。

给变量赋字符常数的方法有标准和非标准的几种。一种非标准但却经常使用的方法是,用赋值语句直接赋字符常数。例如,要把杂志名称 JOURNAL OF COMPUTER (计算机学报)作为字符常数存入计算机,可写

```

DIMENSION C(5)
C(1) = 4H JOUR

```

```

C(2) = 4H NAL_
C(3) = 4H OF_C
C(4) = 4H OMPU
C(5) = 3H TER

```

或容许的话,写成等价形式

```

DIMENSION C(6)
C(1) = 'JOUR'
C(2) = 'NAL_'
C(3) = 'OF_C'
C(4) = 'OMPU'
C(5) = 'TER'

```

执行上述语句,数组C的值就为 JOURNAL OF COMPUTER.

标准的赋字符常数的方法有三种:

- 1) 通过哑实结合给哑元赋字符常数,这种方法下一节我们再讲.
- 2) 用数据初值语句 DATA 给变量赋字符初值,也留在再下一节讲.
- 3) 用 READ 语句给变量输入字符常数.这时要用A型字段描述符  $rAw$ ,  $r$  是重复次数,  $w$  是输入/输出字符串的字段宽度(字符个数). 如

```

READ (5,90) L, K
90 FORMAT (2A3)

```

假定已准备的字符常数

```

ABCDEF

```

则执行上述语句结果是L的机内表示为 \_ABC, K的机内表示为 \_DEF.

字符串的输出也用A型字段描述符  $rAw$ . 例如

```

DIMENSION P(7)
P(1) = 4H THIS
P(2) = 4H _ _ IS
P(3) = 4H _ _ A _
P(4) = 4H FORT
P(5) = 4H RAN _
P(6) = 4H _ PRO
P(7) = 4H GRAM
WRITE (6, 900) P
900 FORMAT (1X,7A4)
STOP
END

```

执行上述程序输出结果是

```

THIS IS A FORTRAN PROGRAM

```

## 3-2 函数与子程序

在算法语言中,通常引入称为“过程”的概念。过程由一系列操作组成,执行预定的计算和处理任务。在 FORTRAN IV 中,过程包括标准函数、语句函数、函数子程序和子例程子程序四种。标准函数又称库函数,分成内部函数和基本外部函数;内部函数在编译时,由 FORTRAN 编译程序就地编出几条完成该功能的机器指令,需用几次就编译几次;基本外部函数是由机器指令编成的一些子程序,它们放在 FORTRAN 编译程序的程序库中,用户只需按指定的格式写出函数名就可调用这些子程序。标准函数已在上一章介绍过,标准函数一览表详见第 2 章附录 A。

在 FORTRAN IV 中,基本外部函数、函数子程序和子例程子程序合称为外部过程;而内部函数,基本外部函数、语句函数和函数子程序合称为函数过程。

下面我们分别介绍语句函数、函数子程序、子例程子程序以及哑元、实元、调用和返回等概念及其应用。

### 1. 语句函数及其引用

在需要多次计算某个表达式的地方,把这项计算工作定义成一个语句函数,这就使得在同一程序块中,可用一个较简单的函数来代替一个较复杂的表达式,从而简化了程序。

语句函数的一般形式为

$$\langle \text{函数名} \rangle (x_1, x_2, \dots, x_n) = \langle \text{表达式} \rangle$$

其中,〈函数名〉由程序员选取,最多由六个字母和数字构成,且以字母开头。由于函数名实际上代表一个值,所以函数名存在类型问题。如果没有对语句函数名加以专门的类型说明,则语句函数名遵循 I-N 隐含类型规则。语句函数名不应与本程序块中的其它变量名、数组名相同。定义式括号中的  $x_1, x_2, \dots, x_n$  为函数的自变量,称为哑元或形式参数,它们在形式上与普通变量名相同,而且只在形式上表示右边表达式有几个自变量、是什么类型、在表达式中起什么作用。因此,形式参数可以与本程序块的其它变量同名,两者并不会混淆。其次,定义中的表达式,除了包含自变量以外,还可包含常数、变量、标准函数和已经定义过的其它语句函数,但不允许包括数组元素和本语句函数名。

下面是几个语句函数例:

$$1) \text{ROOT1}(A, B, C) = (-B + \text{SQRT}(B * B - 4.0 * A * C)) / (2.0 * A)$$

$$2) \text{DEF}(X) = ((X - 3.5) * X + 5.18) * X - 33.1$$

$$3) \text{LOGICAL BOX}, L1, L2$$

$$\text{BOX}(L1, L2) = .NOT. L1 .AND. L2 .OR. L1 .AND. .NOT. L2$$

定义好了的语句函数,就可以在程序块内被引用,用法如同我们引用标准函数一样。代替哑元(即形式参数)的是实元,或称实在参数。实元可以是常数、变量、数组元素或算术表达式,还可以是语句函数本身(即语句函数的引用可以嵌套)。实元的个数、顺序和类型必须与哑元一一对应。

下面是几个引用语句函数例,其中被引用的函数假定是如上已定义好的三个函数:

- 1) X1 = ROOT1(3.,4.,-2.)
- 2) PN = DEF((SQRT(X) - 1.)/5.) - SIN(X)
- 3) IF (BOX(B(1),L2)) GO TO 777

注意 3) 中第一个实元 B(1) 是已说明的逻辑型数组的第一个下标元素, 第二个实元 L2 虽与原来的哑元 L2 同名, 但这是两个不同的概念。

**例 1** 计算下列四个表达式的总和:

$$f_1 = x + \ln x + \ln\left(\frac{1}{x} + x + x^2\right)$$

$$f_2 = \cos x + \ln\left(\frac{1}{1+x} + (1+x) + (1+x)^2\right)$$

$$g_1 = \ln\left(\frac{1}{(x-y)^3} + (x-y)^3 + (x-y)^6\right)$$

$$g_2 = \ln\left(h_i + \frac{1}{h_i} + \frac{1}{h_i^2}\right)$$

容易看出, 这些式子都有一个共同格式

$$\ln\left(\frac{1}{z} + z + z^2\right)$$

其中  $z$  分别代表  $x, 1+x, (x-y)^3, \frac{1}{h_i}$ . 为此定义一个语句函数 F2G2(Z) 来描述这个

公式, 然后再以  $X, 1+X, (X-Y)**3, 1/H(I)$  作为实元来引用这个语句函数。

程序的主要部分如下:

```
.....
F2G2(Z) = ALOG(1.0/Z + Z + Z**2)
.....
F1 = X + ALOG(X) + F2G2(X)
F2 = COS(X) + F2G2(1.0 + X)
G1 = F2G2((X - Y)**3)
G2 = F2G2(1.0/H(I))
SUM = F1 + F2 + G1 + G2
.....
```

## 2. 函数子程序及其调用

语句函数有局限性: 它必须在一个语句内完成; 它只能得到一个函数值; 它只能在本程序块内被引用。

比语句函数更灵活的是函数子程序: 它可以由多个语句构成; 它不仅可以得到一个函数值, 而且可以得到其它附带的值; 它是一个单独进行编译的独立程序块, 除了不能直接或间接地自己调用自己外, 可以被要引用它的其它程序块(主程序块或子程序块)调用。

函数子程序的写法至少有四行: 一个函数语句说明它的名字、类型和哑元; 一个赋值语句为函数名赋值; 一个返回语句 RETURN; 一个结束行 END. 事实上, 一个函数子程序的结构通常是比较复杂的。可能有多个说明语句和多个可执行语句, 也可能有多处

返回语句。结束行 END 用于通知 FORTRAN 编译程序,函数子程序块至此结束。

函数子程序的函数语句的一般形式为

〈类型说明符〉 FUNCTION 〈函数名〉 ( $x_1, x_2, \dots, x_n$ )

其中,函数名的取法与语句函数名的取法规定相同,习惯上把函数名取得长一些,以免与其装配在一起的其它函数和子程序名冲突。FUNCTION 与函数名之间至少留出一个空格。如果不加类型说明符,则函数名遵从 I-N 隐含类型规则。括号中的  $x_1, x_2, \dots, x_n$  ( $n \geq 1$ ) 是形式参数,即哑元,它们是在编写子程序时虚拟出来的名字,并不占用实际存储单元。这里,这些哑元形式上可以是简单变量和数组名,还可以是其它外部过程名,但不能是常数,表达式和数组元素。哑元的类型如果是整型或实型,可用 I-N 规则作隐式说明,其它情形都必须在子程序中作类型说明。同样,数组名也必须在子程序中用维数语句或类型语句说明其维数。只有哑元是函数或子程序名时,才不需要作专门的说明(但在主调程序中对于当作实元用的函数或子程序名,必须用外部语句 EXTERNAL 说明,下面我们再讨论)。

我们给出几个简单的函数子程序例:

(1) 计算

$$f(x) = \begin{cases} e^{-x^2} & x < 0 \\ \frac{\cos x}{1+x} & x \geq 0 \end{cases}$$

```
FUNCTION  FUNC(X)
FUNC = EXP(-X*X)
IF(X) 10, 20, 20
20 FUNC = COS(X)/(1+X)
10 RETURN
END
```

(2) 求  $N!$  的函数子程序:

```
INTEGER FUNCTION FAC(N)
FAC = 1
IF (N.LE.1) GOTO 99
DO 100 I = 2, N
100 FAC = FAC*I
99 RETURN
END
```

(3) 计算方阵  $A$  的主对角线元素(或其中前  $K$  个)的乘积。

```
FUNCTION TRAC (A,K)
DIMENSION A(10,10)
```



```

        TRAC = A(1,1)
        DO 25 I = 2, K
25     TRAC = TRAC * A(I,1)
        RETURN
    END

```

因为 A 数组定义成  $10 \times 10$  大小，故本函数子程序可以计算最大为  $10 \times 10$  方阵的主对角线元素的乘积。

函数子程序的调用与语句函数的引用类似。程序中凡是可以出现表达式的地方都可以出现函数子程序调用。调用时写出函数子程序名，并把哑元换上实元。这里，实元允许是常数、变量、数组元素、数组名、任意算术表达式或逻辑表达式和其它外部过程名。要注意的是，实元的个数、顺序和类型要与哑元一一对应。调用的过程是，把实元的值传送给对应的哑元，然后按规定的算法运算，把结果赋给函数子程序名，而在遇到 RETURN 时，由函数名把所得结果带回到主调程序中调用它的表达式中，并继续进行需要的运算。哑元和实元之间的这种数据传递通常称为“哑实结合”。

还要注意的，如果哑元是数组名，则相应的实元也必须是数组名，而且子程序中的该数组的大小不得大于所对应的实数组。主程序中的数组各元素是按其在计算机存储器中排列的顺序——传送给函数子程序中相对应的数组各元素的。所以，无论是建立函数子程序，还是调用函数子程序，都要细心考虑哑实数组元素间的对应关系。

下面写出几个调用函数子程序的例，假定所调用的函数子程序就是如上定义的两个子程序。

(1) 利用函数子程序 FUNC 计算

$$F = \alpha f(x_1) + \beta f(x_2) + \omega$$

主程序(主要部分):

```

        READ(5,300) ALFA, BETA, OMEGA, X1, X2
300    FORMAT(5F10.4)
        F = ALFA * FUNC(X1) + BETA * FUNC(X2) + OMEGA
        .....

```

(2) 利用函数子程序计算  $P = \frac{N!}{(N-R)!}$

主程序:

```

        INTEGER FAC, P, R
        READ (5,10) N, R
        P = FAC(N)/FAC(N - R)
        WRITE(6,20) N, R, P
10    FORMAT (2I5)
20    FORMAT (1X,3I5)

```

```

STOP
END

```

(3) 计算  $10 \times 10$  方阵  $G$  的主对角元素的乘积。

主程序(主要部分):

```

DIMENSION G(10,10)
READ(5,400) G
DIG = TRAC (G,10)
.....

```

再写一个有实用意义的例子。

**例 2** 切比雪夫多项式计算程序。

数值计算方法中常用到切比雪夫多项式。这是一种定义在  $(-1, 1)$  区间上的正交多项式,可由递归关系式

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

给出,其中已知  $T_0(x) = 1$ ,  $T_1(x) = x$ 。由此,我们可得切比雪夫多项式的前几个是:

$$\begin{aligned}
 T_0(x) &= 1 \\
 T_1(x) &= x \\
 T_2(x) &= 2x^2 - 1 \\
 T_3(x) &= 4x^3 - 3x \\
 T_4(x) &= 8x^4 - 8x^2 + 1 \\
 T_5(x) &= 16x^5 - 20x^3 + 5x
 \end{aligned}$$

切比雪夫多项式的应用我们在第 6 章中还要讲到。现在编写计算切比雪夫多项式的通用子程序。

```

FUNCTION TCHB (N,X)
TCHB = 1.
IF(N .EQ. 0) RETURN
TCHB = X
IF(N .EQ. 1) RETURN
T0 = 1.
T1 = X
DO 10 I = 2, N
TCHB = 2. * X * T1 - T0
T0 = T1
10 T1 = TCHB
RETURN
END

```

如果需要打印切比雪夫多项式某一项,比如  $T_n(y)$  的函数表( $y$  的变化范围是 $(-1, 1)$ ,步长 $0.1$ ),则可编写主程序如下:

```
Y = -1.0
DO 10 I = 1, 21
  T = TCHB(10,Y)
  WRITE(6,100) Y, T
100 FORMAT(1X,2HY=,F10.3,5X,2HT=,F10.3)
10 Y = Y + 0.1
STOP
END
```

### 3. 子例程子程序与 CALL 语句

应用更为广泛的是子例程子程序,它与函数子程序在外形上颇为相似。对比两者的差别,我们便很容易掌握子例程子程序这个概念。

函数子程序和子例程子程序都是独立编译的程序块,且都可供其它程序块调用。但前者常常只是代表一个函数而作为表达式的一部分,后者不仅可以通过程序块之间的啞实结合,从子例程子程序带回多个数据供调用程序使用,也可以用这个子程序块完成多种特殊的功能。

子例程子程序用 SUBROUTINE 一词代替函数子程序中的 FUNCTION 一词;同样也要有一个子程序名;子例程子程序名不代表一个函数值,因而子程序中无需给这个子程序名赋值,从而这个子程序名也不存在类型说明问题。

子例程子程序可以有形式参数(即有啞元,包括啞元两边的括号),也可以无形式参数(即无啞元,也就无括号)。有形式参数时,有关的规定与函数子程序的规定相同。无形式参数时,称无参子程序。事实上,子例程子程序的名字仅仅作名字而已,它不能作为变量名在子程序中出现。子例程子程序主要用来定义一套要多次执行的“例行手续”,它比函数子程序还简单。

看两个示例。

(1) 为了能够对具有 100 个元素的任意数组  $X(100)$ ,可以任意求出其中第  $i$  个元素到第  $j$  个元素 ( $1 \leq i < j \leq 100$ ) 的和及算术平均值,可写出子例程子程序如下:

```
SUBROUTINE SUMAME (X,I,J,SU,AM)
  DIMENSION X(100)
  SU = 0.0
  DO 35 K = I,J
35  SU = SU + X(K)
  AM = SU/FLOAT(J - I + 1)
  RETURN
END
```

(2) 无参子程序, 用来专门打印以 10 个“—”号组成的一条虚线的子程序:

```
SUBROUTINE PRILI
WRITE(6,40)
40 FORMAT(1X,10H-----,
1 15H-----,
2 15H-----)
RETURN
END
```

调用子例程序与调用函数子程序不同。调用子例程序用 CALL 语句, 其一般形式为

CALL 〈子程序名〉( $a_1, a_2, \dots, a_n$ )

或对于无参子程序则为

CALL 〈子程序名〉

其中, 〈子程序名〉就是所要调用的子例程序的名字,  $a_1, a_2, \dots, a_n$  是对应于哑元  $x_1, x_2, \dots, x_n$  的实元, 有关规定与调用函数子程序时的规定相同。

以调用上述定义好的子程序 SUMAME 与 PRILI 为例。已知数组 A(100), 要求第 51 个元素至第 80 个元素之和及其算术平均值, 并且要求输出的结果上下打印一条横线。

主程序:

```
DIMENSION A(100)
READ (5,100) A
100 FORMAT (100 F10.4)
CALL SUMAME (A,51,80,SU,AM)
CALL PRILI
WRITE (6, 200) SU, AM
200 FORMAT (1X,6H——SUM=,F11.4,6X,
1 6HAMEAN=,F10.4)
CALL PRILI
STOP
END
```

注意, 其中实元 SU, AM 与哑元同名, 这自然是允许的。把主程序和子例程序装配在一起, 便可上机计算。假设计算结果是  $SU = 61437.5622$ ,  $AM = 2047.91874$ , 则打印输出格式为

```
-----
SUN = 61437.5622           AMEAN = 2047.9187
-----
```

下面我们给出一个说明性例子,它既可使用函数子程序来编写程序,也可使用子例程序来编写程序。

**例 3** 设两个向量

$$\mathbf{a} = [a_1, a_2, \dots, a_{10}]^T$$

$$\mathbf{b} = [b_1, b_2, \dots, b_{10}]^T$$

求这两个向量的欧几里得范数之和  $\|\mathbf{a}\|_2 + \|\mathbf{b}\|_2$ 。

我们知道,欧几里得范数

$$\|\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^{10} a_i^2}, \quad \|\mathbf{b}\|_2 = \sqrt{\sum_{i=1}^{10} b_i^2}$$

**FORTRAN 程序 I(使用函数子程序):**

```

C      MAIN PROGRAM
      DIMENSION A(10), B(10)
      READ (5, 100) A, B
      TSUM = ENORM1(A) + ENORM1(B)
      WRITE (6, 200) TSUM
100  FORMAT (10F8.2)
200  FORMAT (1X,F10.3)
      STOP
      END

C      *****
C      FUNCTION SUBPROGRAM
      FUNCTION ENORM1(X)
      DIMENSION X(10)
      ENORM1 = 0.0
      DO 50 I = 1, 10
50  ENORM1 = ENORM1 + X(I) * X(I)
      ENORM1 = SQRT(ENORM1)
      RETURN
      END
  
```

**FORTRAN 程序 II (使用子例程序)**

```

C      MAIN PROGRAM
      DIMENSION A(10), B(10)
      READ(5, 100) A, B
      CALL ENORM2(A,ENA)
      CALL ENORM2(B,ENB)
      TSUM = ENA + ENB
      WRITE(6, 200)TSUM
  
```

```

100 FORMAT (10F8.2)
200 FORMAT (1X,F10.3)
      STOP
      END
C      * * * * *
C      SUBROUTINE PROGRAM
      SUBROUTINE ENORM2 (X,ENX)
      DIMENSION X(10)
      ENX = 0.0
      DO 50 I = 1, 10
50    ENX = ENX + X(I) * * 2
      ENX = SQRT (ENX)
      RETURN
      END

```

#### 4. 可调数组

在讲数组说明语句时,我们已经说过(上一章 2-2 节),数组说明符中的下标上界值,在主程序中只允许是正整数,只有在子程序内才能使用变量(整变量),而且数组名和说明其上界的整变量名都必须是在子程序的哑元。这就是可调数组。

可调数组的引入,使得子程序具有相当的弹性,即数组的大小可随调用程序而变,而不必预先估计适用于调用程序的最大数组大小。

由于可调数组的数组名和作为其数组上界的整变量名都列为子程序的哑元,因此,当调用子程序进行哑实结合时,可调数组的大小(体积)就通过实在数组而确定了。

有了可调数组,这使我们有可能编写出更具通用性的子程序。

先看示例。

(1) 改写本节第 2 小节第 (3) 示例的函数子程序 FRAC。在那里因为数组 A 被定义成  $10 \times 10$  大小,故该子程序最多只能处理  $10 \times 10$  方阵的主对角线元素的乘积。现在改写成

```

      FUNCTION TRAC(A,N,K)
      DIMENSION A(N,N)
      TRAC = A(1,1)
      DO 25 I = 2, K
25    TRAC = TRAC * A(I,1)
      RETURN
      END

```

这样,这个函数子程序就可以被任意其它需要方阵对角线元素乘积的程序块调用。例如,可在下列程序中调用这个函数子程序:

```

      DIMENSION F(5,5), G(40,40)

```

```

.....
SUFG = TRAC(F,5,5) + TRAC(G,40,15)
.....

```

(2) 改写本节第 3 小节第(1)示例的子程序 SUMAME 为

```

SUBROUTINE SUMAME (X,N,I,J,SU,AM)
  DIMENSION X(N)
  SU = 0.0
  DO 35 K = I,J
35  SU = SU + X(K)
  AM = SU/FLOAT(J - I + 1)
  RETURN
END

```

这样,对不同长度的一维数组,我们都可以调用本子程序,如

```

DIMENSION A(100), B(25)
.....
CALL SUMAME (A,100,51,80,SU,AM)
.....
CALL SUMAME(B,25,11,11 + 10,SU1,AM1)
.....

```

其中,第二个调用之前假设 11 已赋了值.

下面看几个矩阵运算通用子程序.

**例 4** 矩阵加减子程序 MTADD.

设  $A, B, C \in R^{n \times m}$  矩阵, 取  $KK$  为控制常数,  $KK = 1$  时,  $C = A + B$ ; 当  $KK = -1$  时,  $C = A - B$ .

```

SUBROUTINE MTADD (A,B,C,N,M,KK)
  DIMENSION A(N,M), B(N,M), C(N,M)
  DO 15 J = 1, N
15  C(I,J) = A(I,J) + FLOAT(KK) * B(I,J)
  RETURN
END

```

**例 5** 矩阵相乘子程序 MTMUL.

设  $A \in R^{n \times m}$  矩阵,  $B \in R^{m \times l}$  矩阵, 则  $C = AB$ ,  $C \in R^{n \times l}$  矩阵, 且

$$C_{ij} = \sum_{k=1}^m a_{ik} b_{kj} \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, l)$$

```

SUBROUTINE MTMUL (A,B,C,N,M,L)
DIMENSION A(N,M), B(M,L), C(N,L)
DO 25 J = 1, L
DO 25 I = 1, N
S = 0.0
DO 35 K = 1, M
35 S = S + A(I,K) * B(K,J)
25 C(I,J) = S
RETURN
END

```

#### 例6 矩阵转置子程序 MTRA.

将矩阵  $A \in R^{n \times m}$  转置为  $A^T$  后送入矩阵  $T \in R^{m \times n}$ ,  $t_{ij} = a_{ji}, i = 1, 2, \dots, n; j = 1, 2, \dots, m$ .

```

SUBROUTINE MTRA (A, T, N, M)
DIMENSION A(N,M), T(M,N)
DO 45 I = 1, N
DO 45 J = 1, M
45 T(J,I) = A(I,J)
RETURN
END

```

重要的矩阵运算还有方阵求逆,我们在本书的第8章中讲述.

### 5. 外部语句 EXTERNAL

上面讲函数子程序时,我们曾指出,哑元是函数和子程序名时,在子程序中不需要作专门的说明,但在主调程序中,对于当作实元的函数或子程序名,则必须用外部语句 EXTERNAL 说明. EXTERNAL 语句用来通知 FORTRAN 编译程序,在本程序块中,在调用函数子程序或子例程子程序时实元中哪些名字是外部过程名. 外部过程名就是指基本外部函数名、函数子程序名和子例程子程序名.

外部语句 EXTERNAL 的一般形式为

EXTERNAL  $a_1, a_2, \dots, a_n$

其中  $a_1, a_2, \dots, a_n$  是外部过程名,即本程序块中作为调用子程序时的实元. 这些外部过程名如果是函数名时,还必须加以类型说明,包括用隐式类型说明.

EXTERNAL 语句属于说明语句,所以它必须位于本程序块第一个可执行语句之前.



出现在 EXTERNAL 语句中的名字,在本程序块中不能再作为数值变量名或数组名。如果出现在 EXTERNAL 中的名字是内部函数名,则此函数名在本程序块中再不能作为内部函数引用。

先看一个需要使用 EXTERNAL 语句的简单例子。

为了求出以角度为单位的任意三角函数的值,可另编函数子程序如下:

```
FUNCTION TRI(FUN,ANG)
  X = ANG*ATAN(1.0)/45.0
  TRI = FUN(X)
  RETURN
END
```

假定要求  $x = \sin 30^\circ$  和  $y = \cos 60^\circ$ , 则可写调用上述函数子程序的主程序如下:

```
EXTERNAL SIN, COS
.....
X = TRI(SIN,30.0)
Y = TRI(COS,60.0)
.....
```

我们注意到, SIN 和 COS 都是基本外部函数。

**例 7** 用简单的矩形公式计算定积分

$$S_1 = \int_0^1 \sin x \, dx, \quad S_2 = \int_0^1 \frac{1 - \cos x}{1 + \cos x} dx$$

计算定积分的矩形公式为

$$S = \Delta x \sum_{i=1}^n f[a + (i-1)\Delta x], \quad \Delta x = \frac{b-a}{n}.$$

FORTTRAN 程序如下:

```
EXTERNAL SIN, FUN
CALL INT (0.0,1.0,10,SIN,S1)
CALL INT (0.0,3.1416,30,FUN,S2)
WRITE (6,100) S1, S2
100 FORMAT (1X,3HS1=,F10.4,5X,3HS2=,F10.4)
STOP
END
```

C

```
SUBROUTINE INT (A,B,N,FN,S)
  DX = (B - A)/FLOAT(N)
  S = 0.0
```

```

      DO 20 I = 1, N
20  S = S + FN(A + FLOAT (I - 1)*DX)
      S = DX * S
      RETURN
      END

```

C

```

      FUNCTION FUN(X)
      A = COS(X)
      FUN = (1.0 - A)/(1.0 + A)
      RETURN
      END

```

### 3-3 程序块间的数据交换

在 FORTRAN 语言中, 算法模块可以通过函数子程序或子例程子程序来实现。但由于这些程序块在逻辑上是独立的, 即独立编译, 独立分配变量存储单元, 因此, 模块之间必须通过交换数据, 才有可能完成一定的计算任务并送回计算的结果。这就是程序块间数据交换的概念, 或称为程序块间的数据通信。

从上一节我们已经看到, 程序块可以通过变元的哑实结合以及在函数中通过函数名的传递来交换程序块间的数据。这一节我们讨论程序块间的另一种数据交换方式, 即关于公用块的概念。但必须指出, 公用块的应用又可能限制了程序的可移植性, 还有可能引起意想不到的副作用。

#### 1. 等价语句 EQUIVALENCE

我们从等价语句讲起。等价语句用来使同一程序块中的不同变量使用相同的存储单元。等价语句的一般形式为

```
EQUIVALENCE ((〈变量表〉), (〈变量表〉), ..., (〈变量表〉))
```

其中〈变量表〉可以是普通变量名、数组名或数组元素, 彼此间用逗号隔开, 且每个〈变量表〉至少有两个变量出现。通过 EQUIVALENCE 语句, 编译程序就把每个括号中的〈变量表〉的元素分配给同一个存储单元。

例如:

- (1) EQUIVALENCE (F, UU, W5)
- (2) DIMENSION X(5), Z(3), A(3,4), B(4)  
EQUIVALENCE(X(3), Z(2), W), (A(2,3), B(1))

在(1)中, 变量 F, UU, W5 三个变量在本语句所在的程序块中同占一个存储单元。通常就称这样的变量是等价的。在(2)中, 在本语句所在的程序块内, 数组元素 X(3)、数组

元素 Z(2)以及变量W同占一个存储单元;数组元素A(2,3)与数组元素 B(1)同占另一个存储单元。由于数组在内存单元中是连续存放的,因此当指定某个元素等价时,便连带引起整个数组的某些等价关系。如对于(2)中的情况,它们的存储位置是如表(a)和表(b)。

表(a)

X(1)
X(2) Z(1)
X(3) Z(2) W
X(4) Z(3)
X(5)

表(b)

A(1, 1)
A(2, 1)
A(3, 1)
A(1, 2)
A(2, 2)
A(3, 2)
A(1, 3)
A(2, 3) B(1)
A(3, 3) B(2)
A(1, 4) B(3)
A(2, 4) B(4)
A(3, 4)

又如,下列情况:

DIMENSION R(8), S(4,2), T(2,2,2)

EQUIVALENCE (R,S,T)

则它们在同一程序块中的存储位置是表(c)

表(c)

R(1)	S(1,1)	T(1, 1, 1)
R(2)	S(2,1)	T(2, 1, 1)
R(3)	S(3,1)	T(1, 2, 1)
R(4)	S(4,1)	T(2, 2, 1)
R(5)	S(1,2)	T(1, 1, 2)
R(6)	S(2,2)	T(2, 1, 2)
R(7)	S(3,2)	T(1, 2, 2)
R(8)	S(4,2)	T(2, 2, 2)

若把其中的等价语句改写成

EQUIVALENCE (R(1),S(1,1),T(1,1,1))

或

EQUIVALENCE (R(6),S(2,2),T(2,1,2))

效果是一样的。不仅如此,这个等价语句还可写成功能也一样的如下等价语句:

EQUIVALENCE (R(1),S(1),T(1))

在这个语句中, R, S, T 后面括号中的 1 并非下标值, 而是数组元素的序号, 必须特别指出, 只有在等价语句中数组元素的下标可用序号来代替。

此外, 还可间接建立等价关系, 如

```
DIMENSION KA(2,3), LB(4)
EQUIVALENCE (KA(1,2),M),(M,LB(1))
```

则它们在内存中的等价关系如表(d)。

标准 FORTRAN IV 规定, 整型和实型变量占一个存储单元, 双精度型和复型变量占两个存储单元, 因此, 对如下等价语句

```
DOUBLE PRECISION D(4)
DIMENSION E(5)
EQUIVALENCE (E(1), D(2))
```

则两数组共享存储单元的情况如表(e)。

表(d)

KA(1, 1)
KA(2, 1)
KA(1, 2) M LB(1)
KA(2, 2) LB(2)
KA(1, 3) LB(3)
KA(2, 3) LB(4)

表(e)

D(1)
E(1)
D(2) E(2)
E(3)
D(3) E(4)
E(5)
D(4)

等价语句属于说明语句, 它必须写在本程序块可执行语句之前。利用等价语句, 可以把存储单元中过时不用的值覆盖掉, 节省了存储空间。利用等价语句, 还可以把这样一些程序段连接起来, 这些程序段由于某种原因, 把相同的量取了不同的名字。

## 2. 公用语句 COMMON

如上所述, 一个程序块内的变量或数组仅在本程序块内有意义。要使程序块之间能够进行数据交换, 除了通过过程与过程调用间的哑实结合之外, 另一种形式是在内存中开辟公共存储区, 或称公用区, 使不同程序块的变量和数组元素共同占用公用区中相同的存储单元, 这样, 不同程序块的数据便联系起来了。

公用区的开辟和使用是通过公用语句来实现的。FORTRAN 规定, 可以开辟一个无名公用区, 也可以开辟多个有名公用区, 因此, 公用语句的一般形式分无名公用区中的形式

```
COMMON a1, a2, ..., an
```

和有名公用区中的形式

```
COMMON /n1/a1, a2, ..., an /n2/b1, b2, ..., bm .....
```

先讨论无名公用区 COMMON 语句。其中  $a_1, a_2, \dots, a_n$  是普通变量名、数组名或

数组说明符。

例如，我们在主程序和子程序各写上

```
COMMON A, B
```

则编译程序在存储区中开辟一个无名公用区，主程序和子程序的 COMMON 语句中的两个变量 A 和 B，依次共同占用公用区的第一、第二个存储单元。

又例如，在主程序和子程序中分别写上

```
COMMON L, CH, ARR(5)
```

```
COMMON I, PP, BSS(5)
```

则变量 L 与 I，CH 与 PP，将分别在无名公用区中分配同一个存储单元，数组 ARR 与 BSS 将依次同占用 5 个存储单元。

COMMON 语句是说明语句，它必须出现在程序块的第一个可执行语句之前。用公用区来作不同程序块间的数据交换，要比用哑实结合来作数据交换快得多，故在需要作大量数据传递的场合，通常采用开辟公用区的方式。当不同程序块中先后使用比较大的数组时，可把这些数组先后存入公用区的相同单元，从而可以节省计算机存储空间。

对于无名公用区 COMMON 语句的应用，还有下列几点补充说明：

1) COMMON 语句还可利用来说明数组的大小。例如

```
DIMENSION QQ(50), JJ(5,5), PQJ(2,3,4)
```

```
COMMON QQ, JJ, PQJ
```

可以简写成下列公用语句：

```
COMMON QQ(50), JJ(5,5), PQJ(2,3,4)
```

2) 如果准备放入公用区的变量比较多，可分几个 COMMON 语句来写。FORTRAN 编译程序将按 COMMON 语句先后出现的次序把其中的变量按顺序放入公用区的存储单元。不同程序块中公用区元素之间的联系，完全由它们在公用区的位置决定；相应的变量类型必须按位置一一对应。

3) COMMON 语句还可以和 EQUIVALENCE 语句联用。例如

```
DIMENSION X(3), Y(5)
```

```
COMMON X, A1
```

```
EQUIVALENCE (X(2), Y(1))
```

X(1)	
X(2)	Y(1)
X(3)	Y(2)
A1	Y(3)
	Y(4)
	Y(5)

先由 COMMON 语句开辟了四个存储单元的公用区：X(1)，X(2)，X(3)，A1；再由 EQUIVALENCE 语句把 Y 数组带入了公用区，即如左表所示。

这就把公用区扩大伸长了。通过公用语句与等价语句联用，可以使公用区向后延伸，但不允许向前延伸（这时称冒顶）。

现在讲有名公用区。有名公用区 COMMON 语句中的  $n_1, n_2, \dots$  是公用区名，其取法与变量名的取法相同，但无类型意义。它可以和本

公用区中的变量重名,但不允许与函数名和子程序名重名.  $a_1, a_2, \dots, a_n; b_1, b_2, \dots, b_m; \dots$  代表各区中的变量名、数组名或数组说明符.

例如,在主程序和子程序块中分别写上

```
COMMON /NAME/ I, J, UX, UY /HEL/ P1, P2
```

FORTTRAN 编译程序为此开辟两个名为 NAME 和 HEL 的有名公用区,分别存入变量 I, J, UX, UY 和 P1, P2 的值.

如果只有一个有名公用区,又把该区名连同两旁的斜线不写,则这时成为无名公用区. 如果不写区名,保存两斜线,这也相当于无名公用区. 无名公用区和有名公用区可以同时出现在一个 COMMON 语句中. 如上例

```
COMMON I, J, UX, UY/HEL/P1,P2
```

假设某主程序与三个子程序通过公用区传递某些信息,这些信息部分相同,部分不同. 例如,我们可以在各程序块中写出下列公用语句:

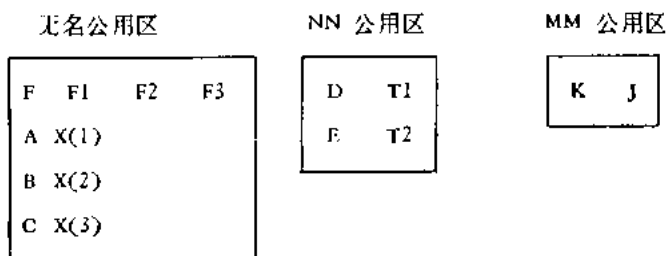
(在主程序) COMMON F, A, B, C/NN/D, E/MM/K

(第一子程序) COMMON F1, X(3)

(第二子程序) COMMON F2/NN/T1, T2

(第三子程序) COMMON F3/MM/J

这样,主程序中的 COMMON 语句开辟了三个公用区,而各子程序块的变量在各公用区的分布如下:



要注意的是,各程序块的同名公用区中,变量的类型要对应,个数要相同.

### 3. 数据初值语句 DATA 和数据块子程序

在 FORTRAN 语言中,除了可用赋值语句或 READ 语句给变量赋值外,还提供了两种给变量赋初值的手段,一种是用 DATA 语句给非公用区变量赋初值;另一种是用含有 DATA 语句的数据块子程序给有名公用区的变量赋初值.

先讲数据初值语句 DATA.

数据初值语句是说明语句,它用来通知 FORTRAN 编译程序给指定的变量或数组元素赋指定的值,并且这个赋值过程在编译源程序时就被办妥了.

数据初值语句的一般形式为

DATA  $x_1/v_1, x_2/v_2, \dots, x_n/v_n$

其中  $x_1, x_2, \dots, x_n$  可以是变量或数组元素;  $v_1, v_2, \dots, v_n$  是将分别赋给其前面的变量或数组元素的常数. 例如

DATA I/110/, J/220/, X/2.4/, Y/4.2/

经编译以后, 即有  $I = 110, J = 220, X = 2.4, Y = 4.2$ .

DATA 语句还可以有简化写法, 如上例可写成

DATA I, J, X, Y/110, 220, 2.4, 4.2/

如果赋给 I, J 的值都是 5, 赋给 X, Y 的值都是 -10, 则可写成

DATA I, J, X, Y/2\*5, 2\*-10/

DATA 语句还可用于给变量赋文字初值, 如

DATA U1/1H\$, U2/2H---/, U3/3H\*\*\* /

这使给变量 U1, U2, U3 分别赋文字初值 \$, ---, \*\*\*.

使用 DATA 语句还要注意的一点是, DATA 语句作为说明语句, 它却被规定必须写在其它说明语句之后. 此外, 在各种非标准 FORTRAN 以及 FORTRAN 77 中, 对 DATA 语句都有较多的扩充. 如允许在 DATA 中使用数组名等. 可随时参考所用计算机的使用手册.

现在讲数据块子程序.

它是用来给有名公用区中的变量赋初值的一种特殊子程序. 象其它子程序一样, 它是一个独立程序块, 单独进行编译. 但它没有名字, 也不需要 RETURN 语句, 其结构如下:

```
BLOCK DATA
.....
END
```

其中除了第一个语句 BLOCK DATA 和结束行 END 之外, 中间只允许是类型语句、数组说明语句、公用语句、等价语句和数据初值语句, 而且其中公用语句和数据初值语句是必不可少的. 数据块子程序正是通过 DATA 语句给放在 COMMON 语句中有名公用区的变量赋值, 从而达到给几个不同程序块的变量赋初值的目的.

一个完整的数据块子程序例子如下:

```
BLOCK DATA
REAL I, J, MAX
INTEGER SUM, ARRAY(5,5)
LOGICAL BOOL
COMMON /NA/SUM, MAX, I, J/CHE/ARRAY, BOOL
DATA SUM, I/-7, 9.25/, BOOL/.FALSE./
END
```

这个子程序的功能是给名为 NA 的公用区中变量 SUM 和 I 分别赋初值 -7 和 9.25, 又给名为 CHE 的公用区中的逻辑变量 BOOL 赋逻辑常数 .FALSE.

从这个例子可见, 虽然仅给某个有名公用区的部分元素赋初值, 但该公用区的全部元素还是必须都列在本块 COMMON 语句的项目中, 而且还必须作必要的说明.

请注意, 一个 FORTRAN 程序可以包含几个数据块子程序, 但每一个有名公用区的变量只能在其中一个数据块子程序中赋初值一次.

### 3-4 输入/输出综述与补充

在上一章的 2-4 节我们对输入/输出已作了初步介绍, 随后在 2-6 节以及这一章的 3-1 节又陆续讲到了输入/输出的一些内容. 这一节我们将对 FORTRAN IV 中的输入/输出问题作必要的综述和补充.

#### 1. 文件与记录的概念

计算机操作系统的一项重要任务, 是对信息进行有效的管理. 这里所指的信息, 包括操作系统本身在内的全部系统程序、应用程序、子程序以及程序中用到的各种类型的数据.

操作系统中的信息管理, 通常是在文件的基础上实现的, 因此, 常把信息管理称为文件管理, 用于信息管理的软件简称为文件系统.

计算机软件中所说的文件 (File), 通常指的是一组有标识的信息集合. 一个文件可以是一个或多个源程序或二进制目标程序, 也可以是一批有标识的数据.

文件总是建立在介质上的, 因此每一个文件总是与某个特定的外部设备相联系的. 建立在卡片上的文件称为卡片文件, 建立在磁盘或磁带上的文件分别称为磁盘文件和磁带文件. 打印机打印出来的信息可称为打印机文件. 穿孔在纸带上的信息称为光电输入机文件, 等等. 在一个介质上可建立多个文件.

根据文件的概念, 一切的输入/输出便归结为对文件的读/写. 一般地说, 当需要建立一个新文件时, 就要向操作系统提出申请, 以便分配必要的外部设备和存储空间; 当文件暂时不用时, 必须把文件关闭; 当文件不需要保留时, 必须撤消文件; 当关闭的文件需要重新使用时, 需要把文件重新打开. 对文件的存储, 需要先确定一个文件名, 文件系统就为这个文件名指定一定的物理位置, 信息就存放那里; 当需要取出这些信息时, 只需指出文件名, 文件系统就会自动将这些信息取出送到计算机内存中去.

通常, 一个文件由若干个记录 (Record) 组成. 数据的读/写就是以记录为单位来进行的. 对于不同的外部设备, 记录的含义可能是不同的.

卡片设备的一张卡片作为一个记录, 一叠卡片组成一个文件.

打印机和显示器以每行为一个记录, 一行内的字符数各不相同.

磁盘上以一定字节数作为一个记录, 一个磁盘文件由许多个这样的记录组成.

键盘与纸带穿孔机以回车符作为一个结束标志, 即每一个回车之前的字符构成一个记录.

一个 FORTRAN 程序的一行就是一个记录, 整个 FORTRAN 程序则组成一个文



件,等等。

文件中记录的信息可以是字符形式,也可以是二进制形式,前者称为格式记录,它可用格式读/写语句来描述;后者称无格式记录,可用无格式读/写语句来描述。

## 2. 字段描述符(补充)·比例因子 P

我们说过,格式说明符连同两端的括号一起称为格式说明,格式说明符分为字段描述符和字段分隔符两类。

在 FORTRAN IV 中,字段描述符共有九种,其它八种我们已经先后讲过,这九种字段描述符就是:

rIw	(见 2-4 节之 3)
srFw.d	(见 2-4 节之 3)
srEw.d	(见 2-4 节之 3)
srDw.d	(见 3-1 节之 1)
srGw.d	(上文未讲过)
rLw	(见 3-1 节之 3)
rAw	(见 3-1 节之 4)
nHh <sub>1</sub> h <sub>2</sub> ...h <sub>n</sub>	(见 2-4 节之 3)
nX	(见 2-4 节之 3)

这里 s 表示一个比例因子指示符,它可有可无,下面我们就要讲到,先补充介绍 G 型描述符。

G 型字段描述符 rGw.d。

它具备 F 型和 E 型两种描述符的优点。

用于输入时,外部数的形式及效果与 F 型的情形相同, d 表示外部数中小数部分的位数。

用于输出时, d 并不表示外部数中小数部分的位数,而是表示外部数中有效数字的位数。当输出的值  $x$  满足  $0.1 \leq |x| < 10^d$  时,则按精确到 d 位有效数字的十进制数小数形式输出,否则,按具有 d 位有效数字的带指数部分的十进制浮点数形式输出。当按十进制小数形式输出时,右边留出四个空白。例如:

内 部 值	描 述 符	外 部 数
-0.0105	G 15.5	____-0.10500E-01
0.786378	G 15.5	____0.78637_____
-15.03729	G 15.5	____-15.037_____
-251305.9	G 15.5	____-0.25130E_06

易见, G 型用于输出时,应保证  $w \geq d + 7$ 。

下面讲比例因子。

设机内值 123.45, 用字段描述符 E10.5 输出时, 数据形式为 0.12345E + 03。如果

需要把它表示成  $1.2345E+02$ , 可以改用格式语句

`FORMAT (1X, 1PE10.5)`

其中 `1P` 就是比例因子。当用指数形式表示数值时, `1P` 的作用是使小数点的位置向右移一位, 相应的指数减 1。看几个例子:

内 部 值	描 述 符	输 出 形 式
69.0301	E 12.6	0.690301 E + 02
69.0301	0P E12.6	0.690301 E + 02
69.0301	1P E12.6	6.90301 E + 01
69.0301	2P E12.6	69.0301 E + 00
69.0301	3P E12.6	690.301 E - 01
69.0301	-1P E12.6	0.069030 E + 03
69.0301	-2P E12.6	0.006903 E + 04

在 F 型描述符前加上比例因子 `nP` 时, 它使输入数据自动除以  $10^n$ , 而使输出数据自动乘以  $10^n$ 。例如, 当输出电流  $A1 = 0.00054$  安培时, 改用格式语句

`FORMAT (1X, 3PF10.6, 4H (MA))`

则输出结果为 `0.540000(MA)`(毫安)。

在 G 型描述符前加比例因子只对指数型输出有影响, 而对 F 型输出无影响。

要注意的是, 凡前面已用比例因子的描述符, 后跟的描述符也起作用。例如, 写

`FORMAT(1X, 1PE10.5, 3E14.4)`

这便相当于

`FORMAT (1X, 1PE10.5, 1P3E14.4)`

如果不希望后一个描述符受影响, 可改写成

`FORMAT (1X, 1PE10.5, 0P3E14.4)`

### 3. 字段分隔符·走纸控制

格式说明符中的逗号“,”和斜线“/”称为字段分隔符, 用于分隔字段描述符。

当分隔符为逗号时, 表示其前后两个字段描述符的相应字符位于同一个记录上。对于输入, 表示对应的字符处在外部介质的同一行上; 对于输出, 表示对应的字符印在同一行上。

当分隔符是斜线时, 表示结束当前所处理的记录, 且同时开始处理一个新的记录。对于输入, 表示已处理完前一个记录, 且开始读入下一个记录; 对于输出, 表示输出当前的记录, 且开始组织新的记录。

几个斜线连在一起形成分隔字符串, 它表示在结束当前所处理的记录后, 接着处理相应的几个空记录, 然后又开始处理下一个记录。

要更好地理解斜线/的用法, 最好从格式说明的整体来看。格式语句对格式的描述是按记录来分段的, 而这种分段又是以括号和斜线来给出的。例如, 语句

100 FORMAT( )

括号内的符号(假定没有斜线)描述一行的格式。又语句

200 FORMAT ( // )

以两个斜线分成三段,描述了三行的格式。如果两个//之间没有其它符号,则表示空一行。如语句

WRITE (6,300)

300 FORMAT (////)

则在打印输出时空五行。同样的格式 300 用于卡片的输入

READ (5,300)

则表示跳过五张卡片。

为了对打印机输出的走纸进行控制,FORTTRAN 提供四个控制符:

空格符	(表示前进一行后再打印)。
0	(表示前进二行后再打印)
1	(表示跳至下页的第一行)
+	(停止前进,回车到本行开始)

当需要计算机执行所希望的动作时,就把相应控制符作为每一个记录中第一个被送出的字符送到 FORMAT 语句中。FORTTRAN 规定,使用格式写语句时,每个记录中第一个被送出去的字符不会被印出来,而是当作走纸控制符用。例如,语句

WRITE (6,123)

123 FORMAT (1X,8HAPPLE\_ I 1/3X,6HIBM-PC/

1 9H\_ \_ \_ TRS-80//9H\_ \_ \_ \* \* \* \* \*)

注意 X 是“空格”,打印输出如下:

第 1 列  
↓  
空   一   行  
APPLE II  
IBM-PC  
TRS-80  
空   一   行  
\* \* \* \* \*

又如,使用下列一段循环

DO 10 I = 1, 10

```

        WRITE (6,500) I
500  FORMAT (1H1,8H---PAGE_, 12, 3H---)
        .....

```

```

10  CONTINUE

```

循环中其它格式语句再不使用换页控制符，则可把结果分 10 页印刷，每页最前面印上

```

        ---PAGE      1---

```

等等。

#### 4. 格式数组

在 FORTRAN 中，还可以把输入/输出的格式说明首先放在数组里，然后把这存放格式说明的数组名，放在 READ 和 WRITE 语句中原来写格式语句标号的位置上。读/写语句将按此数组所包含的格式说明进行读/写，这就是格式数组的概念。有了格式数组，在程序执行过程中，我们就能够根据实际需要，对格式说明加以改变，这显然给应用带来了很大的方便。例如

```

        DIMENSION B(10)
        I = 1111
        R = 999.9
        READ (5,100) B
100  FORMAT (10A2)
        WRITE (6,B) I, R
        .....

```

程序执行 READ 语句时，假设先给数组 B 送入字符串(1X,I8,F10.2)，则在执行 WRITE 语句时输出的结果是

```

      _ _ _ _ 1 1 1 1 _ _ _ _ 9 9 9 . 9 0
      ( 8 格)      (10 格)

```

在程序执行过程中给格式数组输入的格式称为自带格式，嵌入格式数组的 WRITE 语句称为自带格式输出语句。

给格式数组放入格式说明的方法有两种：

(1) 通过与 A 型描述符联系的格式读语句来实现。如上例，它是在程序运行时把格式说明放进数组中的。

(2) 用数据初值语句，例如

```

        DATA A(1), A(2), A(3), A(4), A(5), A(6), A(7), A(8),
1  A(9)/2H(1, 2H4/, 2H/(, 2H1X, 2H, 8, 2HF1, 2H4.,
2  2H4), 1H)/

```

这种方法是在编译时就把格式说明放在数组中。

我们指出,存放格式说明的数组可以是任何类型,因为任何类型的变量或数组元素都可以存放常数;格式说明不一定要填满整个数组,因此格式数组可以取得大一些,以便留有余地;存放在数组中的格式说明不能有H型描述符,因为H型描述符中空白符有意义,当数组中每个元素存放的字符个数少于所能存放的字符个数时,就要补上空白,这时,当这些空白落在H符之后,就可能被当作H符描述的一部分,从而出现含糊不清的情况;另外,存放格式说明的数组,也可列为哑元。例如

```
SUBROUTINE INPUT (N,B,K,FMA)
  DIMENSION B(N,N), FMA(25)
  WRITE (K,FMA) ((B(I,J),J = 1,N), I = 1,N)
  RETURN
END
```

本子程序中的 FMA 表示存放格式说明的数组。调用语句

```
CALL INPUT (12,A,6,FF)
```

表示把数组 A(12, 12) 按格式数组 FF 中存放的格式从 6 号设备上输出。

#### 5. 输入/输出表·带格式输入/输出语句综述

输入/输出表(以前我们也称为输入/输出名表)的一般形式为

$$d_1, d_2, \dots, d_n$$

其中每个  $d_i$  称为表元素。它可以是一个简单表,也可以是一个用括号括起来的简单表或一个隐 DO 表。一个变量名、一个数组元素名或一个数组名是一个简单表,用逗号隔开的两个简单表也是一个简单表。

隐 DO 表的一般形式为

$$(d_1, d_2, \dots, d_n, I = m_1, m_2, m_3)$$

其中  $d_i$  是表元素, I 为隐 DO 循环的控制变量,  $m_1, m_2, m_3$  分别表示循环初值,终值和步长,与循环语句中的规定相同,当  $m_3 = 1$  时可以省略不写,  $d_1, d_2, \dots, d_n$  就是隐 DO 的循环体。易见,隐 DO 表可以层层嵌套,但实际应用中最多可嵌套三层。

FORTRAN IV 中的 DO 循环有较强的限制,隐 DO 循环也一样。如控制变量只能是整型变量;  $m_1, m_2, m_3$  不仅要求是整型常数或整型变量,而且要求  $m_1, m_2, m_3 > 0$  和  $m_1 \leq m_2$ 。

我们试来组织一个这样的输入表:先输入整变量 K 和实变量 U,然后依次输入二维数组 AA 和 BB。已知 AA 的外部数按列排列, BB 的外部数按行排列。注意到数组的输入或输出是按其元素的下标值从小到大的顺序进行的,对二维数组即按列排列进行的,因此,输入表应为

```
K,U,AA,((B(I,J),J = 1,M),I = 1,N)
```

再看一个例子。按行排列输出矩阵(数组) A(10,10) 的下三角形

$$\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \dots & \dots & \dots & \\ a_{10,1} & a_{10,2} & \dots & a_{10,10} \end{bmatrix}$$

输出表应为

$$((A(I,J), J=1,10), I=1,10)$$

现在我们对带格式输入/输出语句作一个综述。

带格式输入/输出(读/写)语句的一般形式依次为

```
READ (u,f) k
WRITE (u, f) k
```

其中,  $u$  是所指定的外部设备号(通道号),  $f$  是格式语句标号或格式数组名,  $k$  是一个输入或输出表,也可以没有这个表。

带格式读语句被执行时,首先从指定通道号的文件上读入一个记录,暂时存放在计算机内的某个缓冲区中,然后,按已给定标号的格式语句或格式数组中所规定的格式,对记录中的每一个字段进行加工,每加工完一个字段就按格式要求赋给输入表中相应的变量、数组元素或数组。每一个读语句可以按格式说明中所指示的格式读入若干个记录。

当读语句无输入表时,则跳过一个或多个记录,或输入文字信息。

相仿地,格式写语句被执行时,输出表中各项的值,按其在表中出现的顺序输出,并按格式语句或格式数组所规定的格式,在指定的外部设备上产生一个或多个有格式记录。

当写语句无输出表时,则可以输出空记录或文字信息。

## 6. 无格式输入/输出语句·辅助输入/输出语句

我们说过(第2章2-4节),在标准的 FORTRAN IV 中,输入/输出语句可分为:

- 带格式输入/输出语句;
- 无格式输入/输出语句;
- 辅助输入/输出语句。

当然,在某些非标准的 FORTRAN IV 中,还往往有诸如自由格式输入、固定格式输出、会话型输入/输出等。

上面我们已经详细介绍了带格式输入/输出语句,下面我们简略地介绍其它两类语句,先讲无格式输入/输出语句。

无格式输入/输出语句用来直接处理二进制数据。因为是按二进制数据的本来形式,不作任何格式转换,所以谓之无格式。无格式输入/输出语句也称二进制读/写语句。其一般形式依次为

```
READ (u) k
WRITE (u) k
```

其中  $u$  是设备号,  $k$  是一个表。

执行无格式读语句时,就引起从  $u$  所标识的外部设备上输入一个无格式记录,并且,

如果有表的话, 则这些二进制数值将依次赋给表  $k$  中所指明的各元素, 表中所需要的值序列, 不能多于这个无格式记录中的值序列。如果没有表  $k$  的话, 它的作用是跳过一个无格式记录, 或者说, 记录仍被读入, 但不赋值。

执行无格式写语句时, 就是依照表  $k$  所指明的值的顺序, 在  $n$  所标识的外部设备上产生一个无格式记录。

总之, 每个无格式读/写语句在指定的设备(文件)上只能读入或产生一个无格式记录。这与每个带格式读/写语句能读入或产生一个或多个有格式记录是不同的。无格式记录长度不受限制, 而有格式记录由于受外部设备的限制, 只能允许一定的长度。

由于无格式读/写语句所传输的是二进制信息, 无需进行十进制转换, 所以输入/输出速度比较快。

现在讲辅助输入/输出语句。

辅助输入/输出语句用来控制外存储设备的操作, 与文件输入/输出有关。标准 FORTRAN IV 规定有三种辅助输入/输出语句:

(1) **重绕语句** 一般形式为

REWIND 〈通道号〉

它使得与〈通道号〉对应的设备(磁盘或磁带, 也即文件)返转回到起始位置, 或者说, 它使该文件的第一个记录的起始位置置于磁头之下。例如, 要多次使用设备号为 5 的文件上的数据, 可写程序:

```
.....  
REWIND 5  
.....  
READ (5,100) A, B, C  
REWIND 5  
READ(5,100) R, S, T, U, V  
.....
```

(2) **回退语句** 一般形式为

BACKSPACE 〈通道号〉

它使〈通道号〉指明的设备(文件)从当前所处的记录位置回退到前一记录的开始位置。例如, 要重复两次处理某记录, 可写程序:

```
.....  
READ (5,100) M, N, F, G  
.....  
BACKSPACE 5  
READ (5,100) I1, I2, TA, TB  
.....
```

### (3) 文件结束语句 一般形式为

ENDFILE (通道号)

它用于在所指(通道号)的设备(文件)上写一个文件结束标志,使得以后读该文件时,如遇此文件结束标志,读过程即自动停止。

关于非标准 FORTRAN IV 中增加的各种非标准输入/输出语句,最好参阅具体计算机的使用说明书。值得指出的是,由于用户终端的广泛使用,有关键盘输入和荧光屏显示的输入/输出规定,是很常用的。但我们将注意到,许多非标准扩充功能,在后来的 FORTRAN 77 中又成为标准了。请见下一章。

## 3-5 程序实例与一些常用子程序

### 1. 控制系统频率相关函数计算例

设某频率相关函数  $G(j\omega)$  的计算公式为

$$G(j\omega) = \frac{200j(2 + j\omega)}{\omega^3(1 + 0.1j\omega)(1 + 0.3j\omega)(1 + 0.6j\omega)(1 + 0.8j\omega)}$$

其中  $\omega$  是角频率(弧度/秒),  $j = \sqrt{-1}$ 。要求对

$$\omega = \omega_0, h\omega_0, h^2\omega_0, h^3\omega_0, \dots, \omega_1$$

( $\omega_0, h, \omega_1$  已知)制表输出  $G$  的实部、虚部、模和振幅。

可以规划输入/输出和计算的步骤如下:

- 1) 打印标题;
- 2) 读入  $\omega_0, \omega_1, h$ ;
- 3) 打印  $\omega_0, \omega_1, h$ ;
- 4) 打印各列表头;
- 5) 将  $\omega_0$  赋给  $\omega$ ;
- 6) 在  $\omega \leq \omega_1$  的情况下,循环做:
  - 计算  $G, G$  的模,  $G$  的振幅;
  - 打印  $\omega, G$  的实部、虚部、模和振幅;
  - 将  $h\omega$  赋给  $\omega$ 。

设置变量名如下:

W0, W1, H 分别表示  $\omega_0, \omega_1, h$ ;

ABSG, ARGG 分别表示  $G$  的模与振幅。

程序:

```
COMPLEX J, G
J = (0.0, 1.0)
READ (5, 200) W0, W1, H
WRITE (6, 100)
100 FORMAT (10X, 25HEVALUATION OF A FREQUENCY,
```



```

&18HRESPONSE FUNCTION (//)
200 FORMAT (3F10.4)
    WRITE (6,50) W0, W1, H
50  FORMAT (10X, 4HW0=, F10.4, 5X, 4HW1 = F10.4, 5X, 3HH=,
    &F10.4/)
    WRITE (6,300)
300 FORMAT (12X, 5HOMEGA, 15X, 5HGREAL, 12X, 10HGIMAGINARY,
    &11X, 9HMAGNITUDE, 12X, 9HAMPLITUDE/)
    W = W0
999 IF (W .GT. W1) GO TO 500
    G = 200.*J*(2. + J*W)/(W**3*(1. + 0.1*J*W)*
    &(1. + 0.3*J*W)*(1. + 0.6*J*W)*(1. + 0.8*J*W))
    ABSG = ABS (G)
    ARGG = ATAN2(AIMAG(G), REAL(G))*180./3.14159265
    WRITE(6,400) W, G, ABSG, ARGG
400 FORMAT (1X,5E20.8)
    W = H*W
    GO TO 999
500 STOP
    END

```

假设输入数据  $\omega_0 = 0.02, \omega_1 = 190, h = 1.2$ , 则输出结果如下图所示, 其中仅列出最前面 20 组数据。

W0 = 0.0200      W1 = 190.0000      H = 1.2000				
OMEGA	GREAL	GIMAGINARY	MAGNITUDE	AMPLITUDE
0.20000000E+01	0.12995490E+07	0.49974640E+08	0.49991504E+08	0.88510361E+02
0.23999993E+01	0.90232700E+06	0.28914032E+08	0.28928080E+08	0.88212494E+02
0.28799985E+01	0.62647900E+06	0.16727312E+08	0.16739022E+08	0.87855087E+02
0.34559973E+01	0.43491719E+06	0.96756790E+07	0.96854350E+07	0.87426270E+02
0.41471958E+01	0.30188831E+06	0.55956240E+07	0.56037550E+07	0.86911743E+02
0.49766339E+01	0.20950681E+06	0.32350980E+07	0.32418740E+07	0.86294632E+02
0.59719596E+01	0.14535337E+06	0.18695740E+07	0.18752150E+07	0.85554337E+02
0.71663499E+01	0.10080300E+06	0.10797810E+07	0.10844750E+07	0.84666550E+02
0.85996151E+01	0.69865375E+05	0.62308650E+06	0.62699094E+06	0.83602203E+02
0.10319531E+02	0.48381473E+05	0.35909937E+06	0.36234381E+06	0.82326706E+02
0.12383431E+02	0.33462770E+05	0.20658319E+06	0.20927575E+06	0.80799011E+02
0.14860111E+02	0.23103672E+05	0.11853500E+06	0.12076556E+06	0.78970703E+02
0.17832130E+02	0.15911418E+05	0.67760937E+05	0.69603937E+05	0.76785370E+02
0.21398550E+02	0.10918918E+05	0.38529125E+05	0.40046406E+05	0.74177551E+02
0.25678253E+02	0.74549883E+04	0.21740879E+05	0.22983523E+05	0.71073044E+02
0.30813897E+02	0.50536875E+04	0.12134441E+05	0.13144746E+05	0.67389496E+02
0.36976665E+02	0.33918459E+04	0.66681484E+04	0.74812266E+04	0.63039200E+02
0.44371986E+02	0.22453936E+04	0.35842117E+04	0.42294609E+04	0.57934158E+02
0.53246373E+02	0.14590115E+04	0.18670735E+04	0.23695300E+04	0.51994400E+02
0.63895637E+02	0.92493652E+03	0.93012671E+03	0.13117322E+04	0.45160294E+02

## 2. 一组简单多项式计算子程序

这里我们集中提供一组简单多项式计算子程序。这组子程序可以作为程序设计的例子来阅读,也可在实际程序设计中被引用。这组多项式计算子程序包括:

PADD	两个多项式相加
FORM	两个多项式加权和
SUBP	两个多项式之差
PMUL	两个多项式相乘
DIVP	两个多项式之商
NORMP	检查多项式系数
PVAL	计算当 $S = PR + jPI$ 时多项式 $A(S)$ 之值

现在分别介绍如下:

(1) **PADD 子程序** 用于将两个多项式  $X(S)$  和  $Y(S)$  相加,即

$$Z(S) = X(S) + Y(S)$$

程序中变量说明:

Z	多项式 $Z(S)$ 的系数数组,规定 $Z(1)$ 为常数项;
IZ	多项式 $Z(S)$ 的次数, $\leq 10$ ;
X	多项式 $X(S)$ 的系数数组;
IXA	多项式 $X(S)$ 的次数, $\leq 10$ ;
Y	多项式 $Y(S)$ 的系数数组;
IYA	多项式 $Y(S)$ 的次数, $\leq 10$ 。

子程序:

```
SUBROUTINE PADD (Z,IZ,X,IXA,Y,IYA)
  DIMENSION Z(11), X(11), Y(11)
  IX = IXA + 1
  IY = IYA + 1
  ND = IX
  IF (IX - IY) 10, 20, 20
10  ND = IY
20  IF (ND) 90, 90, 30
30  DO 80 I = 1, ND
    IF (I - IX) 40, 40, 60
40  IF (I - IY) 50, 50, 70
50  Z (I) = X(I) + Y(I)
    GO TO 80
60  Z (I) = Y (I)
    GO TO 80
70  Z(I) = X(I)
```

```

80 CONTINUE
90 IZ = ND - 1
   RETURN
   END

```

(2) **FORM 子程序** 用于计算两个多项式  $X(S)$  和  $Y(S)$  的加权和  

$$Z(S) = GX(S) + Y(S)$$

程序中变量说明:

G 标量权因子;  
X  $X(S)$  的系数数组,  $X(1)$  为常数项;  
N  $X(S)$  的次数,  $\leq 10$ ;  
Y  $Y(S)$  的系数数组,  $Y(1)$  为常数项;  
M  $Y(S)$  的次数,  $\leq 10$ ;  
Z  $Z(S)$  的系数数组;  
IZ  $Z(S)$  的次数.

子程序:

```

SUBROUTINE FORM (G,X,N,Y,M,Z,IZ)
DIMENSION X(11), Y(11), Z(11)
IF (N - M) 10, 20, 20
10 IZ = M + 1
   GO TO 30
20 IZ = N + 1
30 DO 40 I = 1, IZ
40 Z(I) = G * X(S) + Y(S)
   IZ = IZ - 1
   RETURN
END

```

(3) **SUBP 子程序** 用于计算两个多项式  $X(S)$  与  $Y(S)$  之差:  

$$Z(S) = X(S) - Y(S)$$

程序中变量说明:

Z  $Z(S)$  的系数数组,  $Z(1)$  为常数项;  
IZ  $Z(S)$  的次数,  $\leq 10$ ;  
X  $X(S)$  的系数数组;  
IXA  $X(S)$  的次数;  
Y  $Y(S)$  的系数数组;  
IYA  $Y(S)$  的次数.

子程序:

```

SUBROUTINE SUBP (Z,IZ,X,IXA,Y,IYA)
  DIMENSION X(11), Y(11), Z(11)
  IX = IXA + 1
  IY = IYA + 1
  ND = IX
  IF(IX - IY) 10, 20, 20
10 ND = IY
20 IF(ND) 90, 90, 30
30 DO 80 I = 1, ND
  IF(I - IX) 40, 40, 60
40 IF(I - IY) 50, 50, 70
50 Z(I) = X(I) - Y(I)
  GO TO 80
60 Z(I) = -Y(I)
  GO TO 80
70 Z(I) = X(I)
80 CONTINUE
90 IZ = ND - 1
  RETURN
END

```

(4) **PMUL 子程序** 用于两个多项式相乘:

$$Z(S) = X(S) \cdot Y(S)$$

程序中变量说明:

Z      Z(S) 的系数数组, Z(1) 为常数项;

IZ      Z(S) 的次数,  $\leq 10$ ;

X      X(S) 的系数数组;

IXA    X(S) 的次数,  $IXA + IYA \leq 10$

Y      Y(S) 的系数数组;

IYA    Y(S) 的次数,  $IXA + IYA \leq 10$

子程序:

```

SUBROUTINE PMUL(Z,IZ,X,IXA,Y,IYA)
  DIMENSION X(11), Y(11), Z(11)
  IX = IXA + 1
  IY = IYA + 1
  IF(IX*IY) 10, 10, 20
10 IZ = 0

```

```

        GO TO 50
20  IZ = IX + IY
    DO 30 I = 1, IZ
30  Z(I) = 0.
    DO 40 I = 1, IX
    DO 40 J = 1, IY
        K = I + J - 1
        Z(K) = X(I) * Y(J) + Z(K)
40  CONTINUE
    IZ = IZ - 2
50  RETURN
    END

```

(5) **NORMP 子程序** 用于逐项检查多项式的系数, 如果其系数不超过 EPS 值, 就将它置零, 直到有一项系数超过 EPS 为止. 使用本程序可消去多项式的一些次要项.

程序中变量说明:

X X(S) 的系数数组, X(1) 为常数项;

IX X(S) 的次数+1;

EPS 检查的控制参数.

子程序:

```

SUBROUTINE NORMP(X,IX,EPS)
    DIMENSION X(11)
1  IF(IX) 4, 4, 2
2  IF(ABS(X(IX)) - EPS) 3,3, 4
3  IX = IX - 1
    GO TO 1
4  RETURN
    END

```

(6) **DIVP 子程序** 用于求两个多项式之商:

$$P(S) = \frac{X(S)}{Y(S)}$$

本程序中还调用另一子程序 NORMP 来逐项检查多项式的系数, 把不超过容许值 TOL 的项舍去. 除法运算之前, 对 Y(S) 做一遍, 相除之后, 对 P(S) 做一遍; 如果 Y(S) 没有一项的系数超过 TOL, 则将信息标志 IER 置 1.

程序中变量说明:

P P(S) 的系数数组, P(1) 为常数项;

IP P(S) 的次数,  $\leq 10$ ;

X        X(S) 的系数数组;  
 IXA     X(S) 的次数,  $IXA - IYA \leq 10$ ;  
 Y        Y(S) 的系数数组;  
 IYA     Y(S) 的次数,  $IXA - IYA \leq 10$ ;  
 TOL     容许值  
 IER     信息标志, =0 时表示无误差, =1 表示有误差.  
 子程序:

```

SUBROUTINE DIVP(P,IP,X,IXA,Y,IYA,TOL,IER)
  DIMENSION X(11), Y(11), P(11)
  IX = IXA + 1
  IY = IYA + 1
  CALL NORMP (Y,IY,TOL)
  IF(IY) 50, 50, 10
10  IP = IX - IY + 1
  IF(IP) 20, 30, 60
20  IP = 0
30  IER = 0
40  RETURN
50  IER = 1
  GO TO 40
60  IX = IY - 1
  I = IP
70  II = I + IX
  P(I) = X(II)/Y(IY)
  DO 80 K = 1, IX
    J = K - 1 + I
    X(J) = X(J) - P(I)*Y(K)
80  CONTINUE
  I = I - 1
  IF(I) 90, 90, 70
90  CALL NORMP(X,IX,TOL)
  IP = IP - 1
  GO TO 30
END
  
```

(7) **PVAL 子程序**    用于计算当  $S = PR + jPI$  时实系数多项式  $A(S)$  之值.

程序中变量说明:

A         $A(S)$  的系数数组,  $A(1)$  为常数项;

NN     A(S) 的次数,  $\leq 20$ ;

PR     S 的实部;

PI     S 的虚部;

VR     A(S) 值的实部;

VI     A(S) 值的虚部;

子程序:

```
SUBROUTINE PVAL (A,NN,PR,PI,VR,VI)
  DIMENSION A(21)
  COMPLEX S, P
  S = CMPLX (PR,PI)
  P = CMPLX (A(NN + 1),0.)
  DO 100 J = 1, NN
100  P = P*S + A (NN + 1 - J)
  VR = REAL (P)
  VI = AIMAG (P)
  RETURN
END
```

### 3. 打印曲线子程序

把某些计算结果,在打印纸上打印成离散点组成的曲线,能使计算结果形象化。这里我们介绍几个打印曲线子程序,其基本思想都是根据已经“归一”(即最大值为1)的函数值 Y,按比例求出一个指定的整型数组的下标值 N,送入特定的符号,比如 \* 或其它可以组成曲线的符号,然后返回主调程序进行打印。设所指定的整型数组的长度为 L。子程序中不包含写语句,程序员可以在曲线左右插入文字和数字。

#### (1) SPLOT 子程序

在长度为 L 的数组 IA 中,按函数值 Y 的比例送入一个 \* 号。

```
SUBROUTINE SPLOT (Y,IA,L)
  DIMENSION IA(L)
  DO 2 I = 1, L
2  IA(I) = 1H_
  N = ABS(Y)*(L - 1) + 1.1
  IF(N - L) 3, 3, 4
3  IA(N) = 1H*
4  RETURN
END
```

#### (2) BPLOT 子程序

按函数 Y 的比例向数组 IA 的前若干个元素各送入一个 \* 号, 从而可打印出实心的图形.

```
SUBROUTINE BPLOT (Y,IA,L)
  DIMENSION IA(L)
  DO 10 I = 1, L
10  IA(I) = IH_
  N = ABS(Y) * (L - 1) + 1.1
  IF(N .GT. L) N = L
  DO 3 I = 1, N
  3  IA(I) = IH *
  RETURN
END
```

### (3) OPLOT 子程序

本子程序与 SPLOT 子程序连用. 它在 SPLOT 的基础上, 再根据另一个函数值向同一个数组送入比如·符号, 从而同时印出两条曲线.

```
SUBROUTINE OPLOT(Y,IA,L)
  DIMENSION IA(L)
  N = ABS(Y) * (L - 1) + 1.1
  IF(N - L) 3, 3, 4
  3  IA(N) = IH ·
  4  RETURN
END
```

下面, 我们以打印正弦曲线和余弦曲线为例, 说明上述子程序的用法.

程序清单:

```
      DIMENSION I(55)
      X = 0.0
      DO 10 J = 1, 48
      Y = SIN(X)
      Z = COS(X)
      CALL SPLOT(0.5 * (Y + 1.), I, 55)
      CALL OPLOT(0.5 * (Z + 1.), I, 55)
      WRITE(6, 500) X, Y, Z, I
10  X = X + 0.125
      STOP
```



0.00	0.00	1.00
0.12	0.12	0.99
0.25	0.25	0.97
0.37	0.37	0.93
0.50	0.48	0.83
0.62	0.59	0.81
0.75	0.58	0.73
0.87	0.77	0.64
1.00	0.84	0.54
1.12	0.90	0.43
1.25	0.95	0.32
1.37	0.98	0.19
1.50	1.00	0.07
1.62	1.00	-0.05
1.75	0.98	-0.13
1.87	0.95	-0.30
2.00	0.91	-0.42
2.12	0.85	-0.53
2.25	0.78	-0.63
2.37	0.69	-0.72
2.50	0.60	-0.80
2.62	0.49	-0.87
2.75	0.38	-0.92
2.87	0.26	-0.95
3.00	0.14	-0.99
3.12	0.02	-1.00
3.25	-0.11	-0.99
3.37	-0.23	-0.97
3.50	-0.35	-0.94
3.62	-0.46	-0.89
3.75	-0.57	-0.82
3.87	-0.67	-0.74
4.00	-0.76	-0.65
4.12	-0.83	-0.55
4.25	-0.89	-0.45
4.37	-0.94	-0.33
4.50	-0.98	-0.21
4.62	-1.00	-0.09
4.75	-1.00	0.04
4.87	-0.99	0.16
5.00	-0.96	0.28
5.12	-0.92	0.40
5.25	-0.86	0.51
5.37	-0.79	0.62
5.50	-0.71	0.71
5.62	-0.61	0.79
5.75	-0.51	0.85
5.87	-0.40	0.92

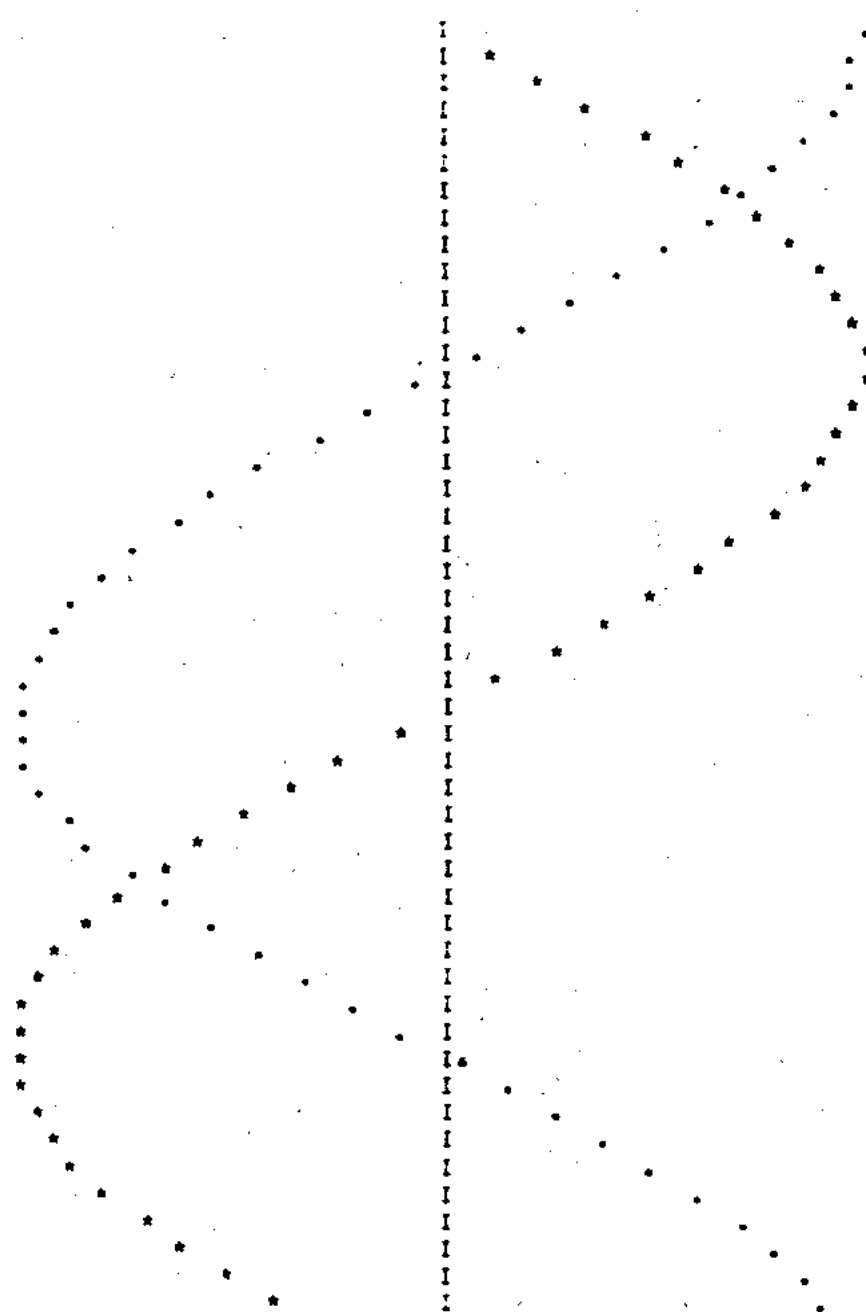


图 3-1

```

500  FORMAT(1X,3F5.2,2X,55A1)
      END

```

C

```

      SUBROUTINE OPLOT (Y,IA,L)
      DIMENSION IA(L)
      N = ABS(Y) * (L - 1) + 1.1
      IF(N - L) 3, 3, 4
3     IA(N) = IH.
4     IA(28) = IH1
      RETURN
      END

```

C

```

      SUBROUTINE BPLOT (Y,IA,L)
      DIMENSION IA(L)
      DO 10 I = 1, L
10    IA(I) = IH.
      N = ABS(Y) * (L - 1) + 1.1
      IF(N.GT.L) N = L
      DO 3 I = 1, N
3     IA(I) = IH *
      RETURN
      END

```

输出曲线如图 3-1 所示。

### 习 题 三

1. 指出下列各数中哪些是 FORTRAN 的双精度数:

3.141592653587793D0	3.141592653589793D-0
1234567890.123456D+29	123456789.12345678901
1,234,567,890,123	2.8D-99
+ 1.D5	1D10
3.14159	1.5E29

2. 指出下列各数中哪些是 FORTRAN 的复常数:

(7,06)	(12.39,-150)	(,8,9)
(2.13)	(0.04,1.88)	(.12345,1)
42,5	(-7,88)	(0012,001)
3-5i	(-79+i)	(0.4,5i)

3. 指出下列各数为何不能表示复型常数:

(0.0,64,28)	907.6,-2.14E-4
(2-7,0.1E-7)	(0,0.23F03)
(89634-2.1E04)	(E-03,601.3D2)

4. 把下列各数写成复型常数:

(1) -6013.1	(2) -12.9 + 12.8i
(3) -3i <sup>3</sup>	(4) -16.7 × 10 <sup>4</sup> i <sup>3</sup>
(5) 49.1 + i	(6) 8998.6 × 10 <sup>4</sup>
(7) 23i - 6i <sup>3</sup>	(8) 0.6 <sup>-3</sup> + 25i

5. 指出下列哪些常数是逻辑常数或字符常数:

(1) 0.276E-4	(2) 676000127
(3) .FALSE.	(4) 4HSTOP
(5) 6H25ABS4	(6) 0.64E+6
(7) .TRUE.	(8) -0.01D-8
(9) .F.	(10) .T.
(11) 'A + 2107B'	(12) (-0.102,-24E6)
(13) 10HFORTRAN001	(14) '406. + 24.'
(15) 2HNO	(16) 6HAB...Z

6. 已知逻辑变量 L1 与 L3 为 .TRUE., L2 与 L4 为 .FALSE., 实型变量 G = 4.5, H = 2.1, 试说明下列逻辑表达式的运算步骤及结果:

(1) L1.AND.(L2.OR.L3).AND.L4  
 (2) G.GT.H.AND..NOT.L1.OR.L3

7. 下列数组说明错在哪里:

(1) D 和 R 都是 30 个元素的数组:

DIMENSION D,R(30)

(2) 数组 L 是逻辑的, 数组 Q 是双精度型的, 但都是一维 10 个元素的数组:

DIMENSION L(10),Q(10)

(3) 数组 IM 和 RN 都是 15 行 8 列的数组:

DIMENSION IM(8,15),RN(8,15)

8. 写出下列数组的说明语句:

- (1) 8 行 6 列的双精度型数组 DNN.
- (2) 5 行 9 列的复型数组 CWN.
- (3) 10 个元素的逻辑型数组 LPN.

9. 试用较少的语句,完成下列所指的工作:

- (1) 求一复数 A, 其实部为实数 B 与 C 之和,虚部为 B 与 C 之积.
- (2) 将一复数 D 的虚部存于整变量 K, 实部与虚部之和存于实变量 A.
- (3) 设 A,B,C,D,E 均为逻辑变量,当 A,B 均为真值,或 C,D 均为假值时令 E 为真;否则令 E 为假.

10. 写出下列结构的 FORTRAN 程序语句:

- (1) `loop while x > 0 do`  
 $\{ \text{分别求 } x \text{ 和 } x^2 \text{ 累加和, 读入 } x \}$
- (2) `loop for n = 1, 3, ..., 99 do`  
 $\left\{ \text{加 } \frac{x^2}{n(n+1)} \text{ 到 SUM} \right\}$

11. 假设矩阵  $A \in R^{10 \times 10}$  的元素已输入,写出 FORTRAN 语句计算:

- (1) 矩阵 A 的最大元素.
- (2) 具有负值的元素的个数.
- (3) 正值元素之和.
- (4) 最大和值的行之和.

12. 分析下列输入/输出语句:

- (1) 假定输入卡片记录为  
 第一张    12345678  
 第二张    13579135  
 第三张    24680246

执行语句

```
READ (5,300) I,J,K
300 FORMAT (I4/I6/I5)
```

后 I,J,K 的值是什么?

(2) 指出下列程序语句输出什么结果:

```
WRITE (6,100)
WRITE (6,200)
100 FORMAT (5H_X = 10)
200 FORMAT (5H + _/_/_)
```

(3) 指出执行下面语句后打印的格式:

```
WRITE (6,555) (1,A(I),I = 1,6)
555 FORMAT(3(13,F8.2/))
```

(4) 指出下面程序段执行后的打印格式:

```
DIMENSION K(10),J(40),W(40)
```

```

      :
      WRITE (6,666)K, (J(I), W(I), I = 1,40)
      666 FORMAT (1H1,10I12/10 (1X,I4,F8.2))

```

(5) 设程序片断为

```

      READ (5,100) I,J,K
      100 FORMAT (3I3)
      WRITE (6,200) I,J,K
      200 FORMAT (I4,I2,I5)

```

又设当前输入记录为

102304506\_\_789\_\_10

指出执行程序后 I,J,K 之值和打印结果。

(6) 设程序片断为

```

      READ (5,5) (X(I), I = 1,3)
      WRITE (6,6) (X(I), I = 1,3)
      5 FORMAT (2F6.2,F3.2)
      6 FORMAT (1X,3F4.1)

```

又设当前输入记录为

21E-0360.215\_\_12

指出执行程序后 X(I) 之值和打印结果。

13. 考察下列程序语句:

```

      SUBROUTINE SUN(N)
      COMMON/GR1/IT(3,3)
      :
      END

      LOGICAL FUNCTION TE(L)
      LOGICAL X(6)
      COMMON/GR1/K(9)
      EQUIVALENCE (K16),X)
      :
      END

```

指出公用块 GR1 中各公用元素结合的情况。又如果把上面程序中的等价语句改写成

EQUIVALENCE (X(3), K)

其它语句不变,这时情况将会怎么样?

14. 指出下列语句中的错误:

- (1) COMMON A,B(5,8),R(20),
- (2) COMMON R,S,T/LAB,/U,V,W//P,QR/
- (3) COMMON P,Q,R,S,X,Y  
EQUIVALENCE (A,P,J), (R,H,E,Y)

```
(4) COMMON A(9),B,C(10)
    DIMENSION D(15),F(5)
    EQUIVALENCE (F(1),A(8)),(B,D(12)),G)
```

```
(5) DOUBLE PRECISION C
    COMMON A(6),B,C(5)
    EQUIVALENCE (A(5),D(2)),(D(6),C(3))
```

```
(6) DIMENSION X(4)
    COMMON A,B,X(4)
```

15. 指出以下每个变量赋予之值是什么:

(1) DATA A,B,C,D/1.2, 2\*3.4,5.6/

(2) DIMENSION V(8,5)  
DATA I,J,V,K/3,4,20\*6.8,20\*9.6,7/

16. 已知数组 A(50), B(40,30), C(70) 和变量 X,Y,Z,W,P,Q, 写出各存放语句:

(1) 数组 A,C,Z 和 W 在无名公用区, 数据 B,X,Y,P,Q 在公用区 K1.

(2) 数据 B,P,Q 在公用区 K1, 数据 A,Z,W 在公用区 K2, 数据 X,C,Y 在无名公用区.

(3) 数组 A,B 和 C 在无名公用区, 使得变量 A(1), B(1) 和 C(1) 分在同一个存储区.

(4) 变量 X,Y,Z,W,P 在公用区 K1, 使得它们的每一个都存于同一个存储区.

(5) 数据 A,B,X,Y,Z 在公用区 K1, 使变量 X 和 A(5) 存在同一个存储区, 变量 Z 和 Y 同在一个存储区.

17. 执行下面程序后, 确定变量 D 所得到的值:

```
    DIMENSION A(5,5)
    DO 1 K = 1,5
    DO 1 J = 1,5
1  A(K,J) = K + J
    D = DT(A,8,23)
    WRITE (6,2)D
2  FORMAT (1X,F6.1)
    STOP
    END
    FUNCTION DT(B,N1,N2)
    DIMENSION B(N2)
    DT = 0.0
    DO 10 K = N1,N2
10  DT = DT + B(K)
    RETURN
    END
```

18. 执行完下面程序后, 该打印出什么数值:

```
    DIMENSION A(10)
    COMMON X,Y,A(10)
    X = 2.3
```

```

Y = -3.7
DO 1 K = 1,10
1 A(K) = K + 0.5
CALL P
STOP
END

```

```

SUBROUTINE P
DIMENSION B(10)
COMMON P1,Q,B(10)
WRITE (6,1)P1,Q,B
1 FORMAT (1X,4F5.1)
RETURN
END

```

19. 定义语句函数来计算下列各式:

- (1)  $f_1 = \sqrt{3a + m}$ .
- (2)  $f_2 = \sqrt{3e^x + \sin x}$ .
- (3)  $f_3 = \frac{1}{4\sqrt{3y+x}} \sqrt{9a^2 + m^2}$

20. 用语句函数方法,计算下列各式:

- (1)  $\frac{x^3 + y^3 + a}{(x^2 + y^2)^{1/2}} \sin(x^3 + y^3)^{1/2}$ .
- (2)  $b \sqrt{x^2 - a^2} / (2a) + \ln|x - \sqrt{x^2 - a^2}|$

21. 写一个函数子程序 FIBONA(N), 计算和返回第  $n$  个斐波那奇 (Fibonacci) 数  $f_n$ , 其中

$$f_n = f_{n-1} + f_{n-2}; \quad f_1 = f_2 = 1.$$

22. 已知函数

$$y = \sqrt{x-1} + \frac{1}{x-1}$$

$x$  由键盘输入, 编写子程序以判别函数在  $x$  点是否有实数值.

23. 编写计算二项式系数

$$c_m^n = \frac{n!}{m!(n-m)!}$$

的过程. 提示: 采用递推公式

$$c_0^n = 1, \\ c_{i+1}^n = \frac{(n-i)c_i^n}{i+1}$$

24. 编写定义

$$f(x) = \sin x + \cos^2 x$$

的外部函数, 然后再编写计算  $y = f(x)/f'(x)$  ( $0.01 \leq x \leq 0.1$ ) 的主程序, 其中  $f'(x)$  是  $f(x)$  在  $x$  处的一阶导数.

25. 晶体二极管的伏-安特性由下式描述

$$I = I_s(e^{QV/KT} - 1)$$

其中  $I_s = 0.72 \times 10^{-12}$  (安),  $Q = 1.6 \times 10^{-19}$  (库仑),  $K = 1.38 \times 10^{-23}$  (焦耳/度). 要求对不同温度

$T_1, T_2, \dots, T$ , 及不同电压  $V_1, V_2, \dots, V_n$ , 计算并制表输出电流  $I$ 。

26. 某反馈系统的频率相关函数  $G(i\omega)$  由公式

$$G(i\omega) = \frac{-15(i\omega + 20)(i\omega + 30)(i\omega - 40)(i\omega - 10)}{\omega^2(i\omega + 5)(i\omega + 8)(i\omega + 50)(i\omega + 60)}$$

确定。试写出一个 FORTRAN 程序, 对于

$$\omega = \omega_0, h\omega_0, h^2\omega_0, h^3\omega_0, \dots, \omega_i$$

制表输出  $G$  的实部和虚部,  $G$  的大小和  $G$  的振幅。

27. 先写一个函数子程序, 它能够计算一维数组  $X$  ( $X$  的元素设最多有 10 个) 的前  $N$  个元素总和的某种函数值; 然后, 利用此函数子程序计算数组  $A$  ( $A$  的元素个数少于等于 10) 的前 7 个元素总和之绝对值及平方根。

28. 写一个子程序, 名为 MERGE, 它接受两个已从小到大排列好的, 各有  $M, N$  个元素的数列  $X$  和  $Y$ , 然后依下面方法把  $X, Y$  合并为  $Z$ : 若  $x_i < y_j$  时就把  $x_i$  移到  $Z$  中, 一直到某一个  $x_k > y_j$  为止, 再把  $y_j$  移到  $Z$  中, 一直到某一个  $y_l$  使得  $x_k < y_l$  为止, 再反复这个过程。



## 第4章 从 FORTRAN IV 到 FORTRAN 77

### 4-1 FORTRAN 77 的目标

在本书第2章的开头,我们曾经列出了 FORTRAN 发展过程中的各种标准文本。在计算机语言专家们看来,1966年的 FORTRAN 标准仍是一个令人失望的文本。主要原因有两个:一是,没有完全摆脱早期编译程序和硬件配置带来的不合逻辑的限制;另一个是, FORTRAN 66 标准没有采用在 ALGOL 60 中成功采用的严格的形式化描述,而仍采用英语散文式说明,这就潜在着多义性和误解的机会,也给编译程序工作留下了多方面的困难。

为此,早在1970年,美国国家标准化协会的 X3J3 技术委员会便着手新的 FORTRAN 标准的形式化工作。委员会遵循的主要研制准则是:

- 1) 程序的可移植性;
- 2) 与现有的标准和惯例的兼容性;
- 3) 加上希望有的新性能;
- 4) 对用户的连贯性和简单性;
- 5) 高效率实现的能力;
- 6) 允许进一步的扩充;
- 7) 对用户的可接受性。

所谓程序的移植性,就是在一种计算机上编出的 FORTRAN 程序可以不作或仅作很少的改变就可在另一种型号的计算机上运行。

由于委员会的积极工作,终于在1977年达成一个新的标准协议,并在1978年4月3日正式公布了美国国家标准程序设计语言 FORTRAN ANSI X3.9—1978 文本,这就是现在通称的标准 FORTRAN 77。

标准 FORTRAN 77 又描述了两级 FORTRAN 语言,分别称为 FORTRAN 与子集 FORTRAN。其中 FORTRAN 是一个全集语言,子集 FORTRAN 是全集语言的一个子集,两者之间是兼容的。

简单说来, FORTRAN 77 是 FORTRAN 66 (或者更笼统的说是 FORTRAN IV) 的修订本。沿着使 FORTRAN 具备现代程序设计语言必须考虑的某些特性的道路上, FORTRAN 77 的确比 FORTRAN IV 前进了值得欢迎的一步。FORTRAN 77 对 FORTRAN IV 的一些实质性改善主要表现在下列几方面:

1) 增加了字符数据类型,在原来文字常数的基础上,引进了字符型变量、字符型数组、子字符串和字符型赋值语句等,从而既增强了字符数据的处理能力,也提高了程序的可移植性。

2) 把库函数中的内部函数和基本外部函数都归并为内部函数,还增加了近20个新

的内部函数。如正切函数,反正弦函数,双曲正弦函数等等。

3) 改进了判断和控制结构,特别是其中引进了IF-THEN-ELSE结构,使FORTRAN 77成为较好的结构程序设计的工具。此外,对循环变量的初值、终值和步长的限制也放宽了。

4) 在非执行语句方面增加了参数语句、隐含语句、内部语句、入口语句和保留语句等等。

5) 在输入/输出中增加表控制格式和不少格式描述符,还增加了文件的处理功能等。

我们将在FORTRAN IV的基础上来解释FORTRAN 77的新内容。但必须指出,要对FORTRAN IV与FORTRAN 77作一个详详细细的比较,并对FORTRAN 77作一个系统的介绍,即使在文字叙述上(至少不要重重复复),也是相当困难的,而且事实上也是不必要的。对于面向科学计算的用户来说,重要的是掌握程序设计思想方法,抓住语法规则的基本部分,与其去硬背一大堆语法细节,不如进机房去调试一个并不需太复杂的实际程序。

## 4-2 源程序格式与基本概念的扩充

### 1. 源程序书写格式的新规定

FORTRAN 77与FORTRAN IV在源程序书写方面只有微小的差别:

1) FORTRAN IV规定在第1列写有字母C的程序行为注解行。FORTRAN 77新规定在第1列写有字母C、或字符\*、或整行空白的程序行,均为注解行。

2) FORTRAN IV允许一个续行的第1列到第5列也可以包含FORTRAN字符集中的任意字符,只是第1列不得为C,否则,这一行就会被理解为注解行。FORTRAN 77规定续行的第1列到第5列必须是空白,在第6行中可含有除了空白符或数字0以外的FORTRAN字符集的任意字符。在FORTRAN 77全集语言中,续行不能超过19行,在FORTRAN 77子集语言中,续行不能超过9行。

3) FORTRAN 77源程序与FORTRAN IV源程序一样,还是块状结构。在FORTRAN 77中,“程序块”更多的是称为“程序单位”或“程序单元”。一个可执行程序,也即一个FORTRAN 77源程序,它必须包含而且仅包含一个主程序。主程序由PROGRAM语句开始而以END结束。PROGRAM语句的一般形式为

PROGRAM (主程序名)

其中,(主程序名)最多用六个字母数字组成,且第一个字符必须是字母;还不得与本程序单位中的任何其它名字相重。

然而,对于一个可执行程序,主程序的PROGRAM语句也可以省略。因此,一个程序单位只要不是以FUNCTION语句、SUBROUTINE语句或BLOCK DATA语句开头,它就是主程序了。

4) 在FORTRAN 77中,每一个程序单位也都必须以END结尾。但END是一个可执行语句,而不再象FORTRAN IV中那样,END是结束行,单纯地表示程序单位

的结束。在这里, END 还具有 STOP 语句或 RETURN 语句的功能。

最后,顺便指出,与 FORTRAN IV 一样, FORTRAN 77 源程序的输入方式可以用卡片输入、纸带输入,也可以用终端键盘输入。正如我们前面说过的,在现代计算机系统中,由用户终端输入源程序和提供初始数据,这是更常用和方便的作业方式。

## 2. 字符集与字符数据类型

FORTRAN IV 标准规定的字符集已有 47 个字符(26 个大写字母, 10 个数字, 11 个专用字符)。FORTRAN 77 增加了两个:

(撇号)

: (冒号)

FORTRAN 77 与 FORTRAN IV 一样有六种数据类型,即整型、实型、双精度型、复型、逻辑型和字符型;而且同样有六种类型的常数。只是在 FORTRAN IV 中,字符型数据有且仅有字符型常数。

在 FORTRAN IV 中,字符型也称 Hollerith 型;字符型常数的标准表示形式是

$$nHh_1 h_2 \cdots h_n$$

非标准表示形式是

$$'h_1 h_2 \cdots h_n'$$

其中  $h_1, h_2, \cdots, h_n$  是任何特定的语言处理系统中能表示的字符。

由于 FORTRAN IV 中没有专门的字符型变量和数组,所以字符型常数的使用途径是:或者用 READ 语句给变量输入字符常数;或者用数据初值语句 DATA 把字符常数赋给任何类型的变量或数组;或者作为实元出现在调用语句中。例如

```
INTEGER S(3)
```

```
DATA S/12HABCDE12345**/
```

由于在不同系统中,一个变量的机器字长度与一个字符的机器字长的比例  $k$  是不同的,因此上述程序对不同的处理系统就会产生不同的结果。如某处理系统中的  $k=4$ ,则结果是

```
S(1) = 4HABCD
```

```
S(2) = 4HE123
```

```
S(3) = 4H45**
```

如另一处理系统的  $k=6$ ,则结果是

```
S(1) = 6HABCDE1
```

```
S(2) = 6H2345**
```

```
S(3) 未赋值
```

可见,这种字符数据类型是 FORTRAN IV 最不可移植部分之一。同时, FORTRAN IV 还允许字符与数值可以同时存放在一个数组的不同元素中,这样也极易引起混乱。

针对这些问题, FORTRAN 77 作出如下一些新规定:

(1) 字符常数只允许用单引号括着的字符串表示,而不能用 Hollerith 常数形式表示。如 'ABCDE',不能表示为 5HABCDE。字符常数本身的撇号则用两个相邻的撇号表

示,即凡连续两个撇号即代表字符串有效的一个撇号,如字符常数 'IT'S ME.' 表示长度为 8 的字符串

IT'S ME.

(2) 引进了字符变量和字符数组,它们由说明语句 CHARACTER 加以定义.

例如:

CHARACTER\*4 PC\*5, NAME(10)\*15, X, Y, Z

它说明 PC 是一个字符型变量,长度为 5 (即最多可放 5 个字符); NAME 是一个字符型一维数组,含有 10 个元素,每个元素的长度为 15; X, Y, Z 也为字符型变量,其后面不带 \* 号和数字,则由 CHARACTER 后面的 \*4 说明它们的长度均为 4.

又如:

CHARACTER BOOK, TV

不明显指定长度,则表示变量 BOOK 与 TV 的长度均为 1,即只能放一个字符.

引进了字符变量和字符数组以后,字符型常数就只能赋给字符型变量或字符型数组.

例如:

PC = 'APPLE'

NAME(1) = 'APPLESOFT\_BASIC'

由于每个字符变量和字符数组元素的长度是以字符为单位,而不是以一个变量为单位,因而可以适应不同的处理系统.

(3) 建立字符子串的概念.子串就是字符串中的一部分.在字符变量名或字符数组元素名后面跟一个圆括号,用圆括号中的两个以冒号隔开的整数(也可以是整型算术表达式)来确定子串在字符串中的位置.例如:

若 PC 是 'APPLE',则 PC(1:3) 是 'APP'.

若 NAME(1) 是 'APPLESOFT\_BASIC',则 NAME(1)(6,9) 是 'SOFT'.又如:

NAME (10) (1:1) = '\*'

则表示数组元素 NAME(10) 的第 1 个字符赋予 \* 号.

用以确定子串在字符串中位置的那两个数可以使用省略形式.如上述的字符数组元素,NAME(10)(1:5) 可写成 NAME(10)(:5); NAME(10)(1:15) 可写成 NAME(10)(1:).又如 PC(:),相当于 PC.

(4) 增设了一个字符运算符,其形式为//,即以两个斜杠组成,表示把左右两个字符串连接起来,从而组成了字符表达式.

例如:

'FOR'// 'TRAN'// '\_77'

所得结果是 FORTRAN 77.字符运算符两边允许是字符常数、字符变量名、子串名等字符基本元素.

有了字符表达式,进一步又引入字符赋值语句,其一般形式为

S = <字符表达式>

其中 S 可以是字符变量名、字符数组元素名或字符子串名.

字符赋值语句实际上我们已经在上面用上了。如：

```
PC = 'APPLE'
NAME(1) = 'APPLESOFT_BASIC'
```

字符赋值语句的执行过程是，先计算(连接)右边的字符表达式，然后把所得的值(结果)赋给 S。当 S 的长度比字符表达式的值的长度大时，在表达式值的右边不足之处填以空白；反之，当表达式的值的长度大于 S 的长度时，则舍去表达式右边多余的部分。例如：

```
CHARACTER ST1*6, ST2*3, ST3*8
ST1 = 'ABCDEF'
ST2 = ST1
ST3 = ST1
.....
```

这时，ST2 得到的是 ABC，ST3 得到的是 ABCDEF\_\_。

必须注意的是，在一个字符赋值语句中，赋值号左边字符串的任何部分都不能是右边表达式中的一部分。

(5) 字符串可以作为哑元和实元传递给函数和子程序或者作为函数值返回。例如

```
FUNCTION TWO(X,Y)
CHARACTER TWO*2, X*(*), Y*(*)
TWO = X(1:1)//Y(LEN(Y):LEN(Y))
END
```

其中哑元 X, Y 均为长度未定的字符变量，LEN 是内部函数，它的值等于其字符变量的长度。这个函数 TWO(X,Y) 的值是两个字符的字符串，这个串由实元 X 的第一个字符，后接实元 Y 的最后一个字符组成。

有关字符数据的新规定，主要有上述几方面，但使用时还要注意一些限制，例如：

字符数据不能与其它类型的数据保存在同一公用块中；

字符数据不能与其它类型的数据等价。

### 3. 数组与数组元素

我们知道，数组由 DIMENSION 语句加以说明，数组说明符包括数组名、圆括号、圆括号内的维说明符。在 FORTRAN IV 中，最多允许定义三维数组；在 FORTRAN 77 中，最多允许定义七维数组，而且每一维的大小由下列形式

$$d_1: d_n$$

来指定维的上限和下限，其中  $d_1$  是维的下限表达式， $d_n$  是维的上限表达式。在主程序中， $d_1$  和  $d_n$  必须是整常数表达式，在外部函数和子程序的可调数组中， $d_1$  和  $d_n$  可以是不包含函数及数组元素的整数表达式。可调数组名必须在哑元表中出现。哑数组的最后一维说明符的上限可以是 \* 号，这时，表示哑数组的大小与所对应的实数组的大小相同。维的上限值必须大于等于维的下限值。当维的下限值(连同冒号)省略时，认为下限值为 1。

例：

在主程序中可写:

```
DIMENSION ARR(-10:0,10,10:10+5)
DIMENSION MON(1:12)
```

在子程序中可写:

```
DIMENSION YL (K:K1 - J1,K + 10:*)
```

关于数组元素的引用,我们知道,在 FORTRAN IV 中,数组元素的下标表达式只允许是以下五种形式:

```
k
v
v ± k
c * v
c * v ± k
```

其中  $k$ ,  $c$  是整常数,  $v$  是整变量。在 FORTRAN 77 中,数组元素的下标表达式可以是任何算术表达式。当下标表达式的计算结果不为整型数时,按赋值语句的规则转化为整型数作为下标值。下面是数组元素引用的例子:

```
DAL(-5,3)    表示二维数组 DAL 的一个元素;
NN(K(2))     表示下标为另一数组 K 的元素;
BB(P * Q + 5) 表示下标是一般算术表达式。
```

在 FORTRAN IV 中,多维数组均可作为一维数组看待,故允许某一维的下标表达式之值超过其相应维的上界值,只要不超过此数组的最大下标值即可。但在 FORTRAN 77 中却不允。如

```
DIMENSION ARRAY(10,10)
B = ARRAY(11,1)
```

在 FORTRAN IV 中允许,在 FORTRAN 77 却不允许。可见 FORTRAN 77 中一个数组的各个维的界线是很严格的。

#### 4. 表达式

在 FORTRAN 77 中,表达式由操作数(常数、常数名、变量、数组元素、子串和函数调用)、运算符和括号组成,经过运算后得到一个值。

操作数分为三种类型:

- 数值型(整型、实型、双精度型和复型);
- 字符型;
- 逻辑型。

对应地,便有三种表达式:

- 算术表达式;
- 字符表达式;

· 逻辑表达式(其中简单情形就是关系表达式)。

字符表达式上面已经简单讨论过, 这里讨论算术表达式和逻辑表达式。它们的组成和运算规则与在 FORTRAN IV 中介绍过的大致相同, 下面讨论主要的差别。

FORTRAN 77 允许算术表达式中不同类型的数值量进行混合运算, 只有个别情形例外。详见表 4-1(a) 与表 4-1(b)。

表 4-1(a) + - \* / 运算规则的运算结果

右操作数 左操作数	整 型	实 型	双精度型	复 型
整 型	(整)	(实)	(双)	(复)
实 型	(实)	(实)	(双)	(复)
双精度型	(双)	(双)	(双)	(复)
复 型	(复)	(复)	(复)	(复)

表 4-1(b) 指数 \*\* 运算规则的运算结果

指数类型 底数类型	整 型	实 型	双精度型	复 型
整 型	(整)	(实)	(双)	(复)
实 型	(实)	(实)	(双)	(复)
双精度型	(双)	(双)	(双)	(禁止)
复 型	(复)	(复)	(禁止)	(复)

在关系表达式运算中, 不允许双精度型与复型值进行比较。当关系运算符两边的算术表达式为复型值时, 关系运算符只能使用 .EQ. 或 .NE. 两种。

当关系运算符两边为字符型表达式时, 首先计算出它们的值, 然后按字符排列序列(取决于具体的计算机系统)进行比较, 其值在前者, 便认为该表达式为小。若比较的两个字符串长短不同时, 则用空白符补齐后再比较。

FORTRAN 77 增加两个逻辑运算符:

.EQV. (同或)

.NEQV. (异或)

它们在逻辑运算中的优先级最低, 运算规则如下表 4-2, 表中包括了 FORTRAN IV 原来已经使用的三个逻辑运算符

.NOT. .AND. .OR.

的运算规则(见表 2-5)。

表 4-2 五种逻辑运算符真值表

$c_1$	$c_2$	$\text{.NOT. } c_1$	$c_1 \text{ .AND. } c_2$	$c_1 \text{ .OR. } c_2$	$c_1 \text{ .EQV. } c_2$	$c_1 \text{ .NEQV. } c_2$
真	真	假	真	真	真	假
真	假	假	假	真	假	真
假	真	真	假	真	假	真
假	假	真	假	假	真	假

综上所述, FORTRAN 77 中共有 18 个运算符, 它们也有类型之分, 还分为 0 级到 8

级的相对优先级。详见表 4-3。

表 4-3 FORTRAN 77 运算符表

序 号	运 算 符	类 型	相对优先级	意 义
1	**	算术	8	指数运算
2	-	算术	7	变号(一元运算)
3	/	算术	6	除法
4	*	算术	6	乘法
5	-	算术	5	减法
6	+	算术	5	加法
7	//	字符	5	字符连接
8	.NE.	关系	4	$\neq$
9	.GE.	关系	4	$\geq$
10	.GT.	关系	4	$>$
11	.EQ.	关系	4	$=$
12	.LE.	关系	4	$\leq$
13	.LT.	关系	4	$<$
14	.NOT.	逻辑	3	逻辑非
15	.AND.	逻辑	2	逻辑与
16	.OR.	逻辑	1	逻辑或
17	.EQV.	逻辑	0	逻辑相等
18	.NEQV.	逻辑	0	逻辑不等

### 4-3 新增加和修改的几个语句

FORTRAN 77 增加了几个新的语句,对原有的一些语句也作了部分改动或补充。分述如下。

#### 1. PARAMETER 语句(参数语句)

在 FORTRAN IV 中,常数是没有名字的,由其书写形式自动表明它的类型。为了方便, FORTRAN 77 增加了参数语句,用于为一些常数指定常数名,从而,在该程序单位中,该常数名就如同常数一样,可以多次出现在表达式或数据语句(DATA)中。

参数语句的一般形式为

$$\text{PARAMETER } (P_1 = e_1, P_2 = e_2, \dots)$$

其中,  $e_i$  是算术常数表达式、逻辑常数表达式或字符常数表达式,  $P_i$  是相应的数值型、逻辑型或字符型常数的符号名。符号名的取法与变量名的取法相同,其类型可用隐式规则说明,也可在第一次出现参数语句前,用显式类型说明语句或隐含语句予以说明。括号中的赋值语句  $P_1 = e_1, P_2 = e_2, \dots$  的个数,原则上没有限制,也可以只是一个。

参数语句的例子:

```
DOUBLE PRECISION DD
COMPLEX CC
```



```

LOGICAL TF
CHARACTER*3 A
PARAMETER(X = 1.414, PI = 3.1415926, K = 1)
PARAMETER(DD = 0.135792468D2, TF = K .GT. 2)
PARAMETER(BE = 2.*X*PI, CC = (1., -5.))
PARAMETER(A = 'TS'S_FILE_DAY.')
.....

```

使用 PARAMETER 语句需注意下列几点:

- 1) 一般式中的  $P_i$  不能作为复常数的一部分;
- 2)  $c_i$  可以含有常数符号名, 但该符号名必须在此之前的另一个 PARAMETER 语句中定义过;
- 3) 同一个符号名不能在 PARAMETER 语句中定义两次或两次以上;
- 4) 常数符号名的类型必须在 PARAMETER 语句出现之前定义, 且其后不得再改变。

## 2. IMPLICIT 语句(隐含类型说明语句)

隐含类型说明语句可用来改变本程序单位中的类型隐式说明。隐含类型说明语句的一般形式为

IMPLICIT <类型指定> ( $a, a, \dots$ ), <类型指定> ( $a, a, \dots$ ), ...

其中, <类型指定> 是 INTEGER, REAL, COMPLEX, LOGICAL, DOUBLE PRECISION, CHARACTER 之一,  $a$  代表单个字母或按字母顺序的单个字母范围, 如 A-E 表示 A, B, C, D, E 五个字母。

隐含类型说明语句的例子:

```

IMPLICIT REAL(K), INTEGER(P, X-Z)
IMPLICIT DOUBLE PRECISION(D, Q, U-W)
IMPLICIT LOGICAL(M), CHARACTER*10(A)

```

以上语句说明在本程序单位内, 凡以 K 开头的符号名隐含为实型; P, X, Y, Z 开头的符号名隐含为整型; D, Q, U, V, W 开头的符号名隐含为双精度型, M 开头的符号名隐含为逻辑型; 最后, A 开头的符号名隐含为长度是 10 的字符型。除此之外, 其它字母开头的符号名仍服从系统规定的隐式说明规则。

对 IMPLICIT 语句补充说明如下:

- 1) IMPLICIT 语句必须出现在除 PARAMETER 语句之外的所有其它说明语句之前;
- 2) 所有其它的显式类型说明语句的说明, 都优先于 IMPLICIT 语句的隐含说明;
- 3) IMPLICIT 语句并不能改变内部函数的类型。

## 3. IF-THEN-ELSE 结构

FORTRAN 77 在保持了 FORTRAN IV 的算术 IF 语句和逻辑 IF 语句的同时, 增加了所谓 IF-THEN-ELSE (如果-则-否则) 结构。按照这种结构组织的程序, 将更接近

于人们的思维和表达方式。

所谓 IF-THEN-ELSE 结构由下列语句:

```
块 IF 语句;  
ELSE IF 语句;  
ELSE 语句;  
END IF 语句
```

和其它一些可执行语句组成。

一种简单的 IF-THEN-ELSE 结构如下所示:

```
IF (<逻辑表达式>) THEN  
.....  
ELSE  
.....  
END IF
```

其中:

IF (<逻辑表达式>) THEN	称为块 IF 语句;
ELSE	称为 ELSE 语句;
END IF	称为 END IF 语句。

而

块 IF 语句与 ELSE 语句间的所有可执行语句称为 IF 块;

ELSE 语句与 END IF 语句间的所有可执行语句称为 ELSE 块。

上述这种 IF-THEN-ELSE 结构的意义就是:

如果<逻辑表达式>之值为真,则执行 IF 块中所有可执行语句,否则执行 ELSE 块中所有可执行语句。

上述结构也可用流程图 4-1 表示。

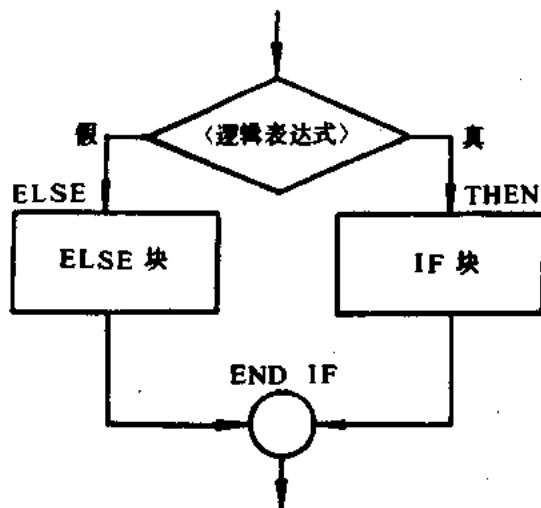


图 4-1

例1 编写下列算式的程序语句:

$$y = \begin{cases} \sqrt{x} & x \geq 0 \\ x^2 & x < 0 \end{cases}$$

用 IF-THEN-ELSE 结构:

```
.....  
IF (X .GE. 0) THEN  
    Y = SQRT (X)  
ELSE  
    Y = X * X  
END IF  
.....
```

下面我们对组成 IF-THEN-ELSE 结构的各语句及有关问题作进一步论述。

(1) 块 IF 语句

块 IF 语句是 IF-THEN-ELSE 结构的开头语句,其一般形式为

IF ((逻辑表达式)) THEN

其执行步骤是:

- 1) 计算<逻辑表达式>之值;
- 2) 若<逻辑表达式>之值为真,则执行 IF 块中所有可执行语句,然后控制转到 END IF 语句;

3) 若<逻辑表达式>的值为假,则控制转到 ELSE 语句。在这里,IF 块可以为空。

(2) END IF 语句

END IF 语句起结束块 IF 语句的作用,因而,每一个块 IF 语句必须在同一个程序单位内对应一个 END IF 语句。END IF 语句的一般形式为

END IF

执行 END IF 语句的意义是,结束对应的块 IF 语句,并继续执行 END IF 语句后面的语句。

(3) ELSE 语句

ELSE 语句的一般形式为

ELSE

执行 ELSE 语句实际上就是执行 ELSE 块中所有可执行语句,然后执行 END IF 语句。

例2 回忆我们一开始就给出的求二次方程

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

实根的 FORTRAN 程序(第2章2-1节中的示例)。现在用 FORTRAN 77 中的 IF-THEN-ELSE 结构来重新编写这个程序。但为了问题简单,这里我们不考虑把求根公式加以变形。

FORTRAN 77 程序:

```

*      FINDING THE REAL ROOTS OF QUADRATIC EQUATION
      READ(5,100) A, B, C
100  FORMAT (3F8.2)
      D = B*B - 4.0*A*C
      IF(D .GE. 0.0) THEN
          S = SQRT(D)
          A2 = 2.0*A
          X1 = (-B + S)/A2
          X2 = (-B - S)/A2
          WRITE(6,200) X1, X2
      ELSE
          WRITE (6,300)
      END IF
200  FORMAT (1X,F8.2,10X,F8.2)
300  FORMAT (1X,'THE REAL ROOT IS NOT')
      END

```

如果我们考虑二次方程根的所有情形,即在复数范围内求方程的两个根,那么,二次方程求根判别式  $D = b^2 - 4ac$ :

当  $D > 0$  时,方程有两个不同的实根

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a},$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a};$$

当  $D = 0$  时,方程有两个相同的实根

$$x_1 = x_2 = -\frac{b}{2a};$$

当  $D < 0$  时,方程有两个共轭的复根

$$x_1 = -\frac{b}{2a} + i \frac{\sqrt{4ac - b^2}}{2a},$$

$$x_2 = -\frac{b}{2a} - i \frac{\sqrt{4ac - b^2}}{2a}.$$

这里,出现多个条件的判定情形。因此。采用下面马上就要讨论的 ELSE IF 语将会更加有效。

(4) ELSE IF 语句

ELSE IF 语句的一般形式为

ELSE IF ((逻辑表达式)) THEN

ELSE IF 语句执行的步骤是:

1) 计算(逻辑表达式)之值;

2) 若(逻辑表达式)之值为真,则执行 ELSE IF 语句之后,直到 ELSE 语句(不包括 ELSE 语句)的所有可执行语句,然后,控制转移到 END IF 语句;

3) 若(逻辑表达式)为假,则控制转移到 ELSE 语句。

在这里,从 ELSE IF 语句之后,直到 ELSE 语句(不包括 ELSE 语句)的所有可执行语句称为 ELSE IF 块。

现在,利用 ELSE IF 语句,我们来编写上述在复数范围内求根的 FORTRAN 77 程序。程序中将  $x_1, x_2$  说明为复型变量 X1, X2。每一个复型变量由实部与虚部两部分组成,当这两部分中任一部分用表达式表示时,必须通过内部函数 CMPLX 把实部和虚部两个表达式合成复数,然后再赋给复型变量。内部函数 CMPLX 有两个自变量。每个自变量均可写成表达式。前一个自变量代表实部,后一个自变量代表虚部,若只写出一个自变量,则它代表实部,而虚部是 0.0。

FORTRAN 77 程序:

```
      COMPLEX X1, X2
      READ(5,100) A, B, C
100  FORMAT(3F10.4)
      D = B*B - 4.0*A*C
      A2 = 2*A
10  IF(D .EQ. 0.0) THEN
          X1 = CMPLX(-B/A2)
          X2 = X1
      } IF 块
20  ELSE IF(D .GT. 0.0) THEN
          S = SQRT(D)
          X1 = CMPLX((-B + S)/A2)
          X2 = CMPLX((-B - S)/A2)
      } ELSE IF 块
30  ELSE
          S = SQRT(-D)
          X1 = CMPLX(-B/A2, S/A2)
          X2 = CMPLX(-B/A2, -S/A2)
      } ELSE 块
40  END IF
50  WRITE(6,200) X1, X2
200  FORMAT(1X,'X1=',2F10.4/'X2=',2F10.4)
      STOP
      END
```

在上述程序中,若语句 10 中的表达式为真,则执行 IF 块,然后控制转到语句 40,接着执行语句 50。

若语句 10 中的表达式为假,而语句 20 中的表达式为真,则绕过 IF 块,而执行 ELSE IF 块,然后控制转到语句 40,接着执行语句 50。

若语句 10 和语句 20 中的表达式都为假,则跳过 IF 块和 ELSE IF 块而执行 ELSE

块，并顺序执行语句 40 和语句 50。

以上程序是建立在实数运算的基础上的。事实上，FORTRAN 语言允许作直接的复数运算，如加、减、乘、除、乘方，也可直接用 CSQRT 函数对复数开平方，从而使程序设计简单多了。

二次方程求根的复数运算程序如下：

```
*      PROGRAM COMPLEX
      COMPLEX A, B, C, S, X1, X2
      READ (5,100) A, B, C
100  FORMAT (3F10.4)
      S = CSQRT (B*B - 4.0 * A * C)
      A2 = 2 * A
      X1 = (-B + S)/A2
      X2 = (-B - S)/A2
      WRITE (6,200) X1, X2
200  FORMAT(1X,'X1=',X1/'X2=',X2)
      STOP
      END
```

采用含有 ELSE IF 语句的 IF-THEN-ELSE 结构还可以处理更多路的判定问题。其语法结构形式如下：

```
IF ((逻辑表达式1)) THEN
    :      (表达式 1 真时执行的 IF 块)
ELSE IF ((逻辑表达式 2)) THEN
    :      (表达式 2 真时执行 ELSE IF 块)
    :      :
ELSE IF ((逻辑表达式 n)) THEN
    :      (表达式 n 真时执行的 ELSE IF 块)
ELSE
    :      (n 个表达式均为假时执行的 ELSE 块)
END IF
```

这种结构的流程图如图 4-2。

#### (5) IF-THEN 结构

在 IF-THEN-ELSE 结构中，可以没有 ELSE 语句，这就是 IF-THEN 结构，其结构形式如下：

```
IF ((逻辑表达式)) THEN
    :      (执行 IF 块)
END IF
```

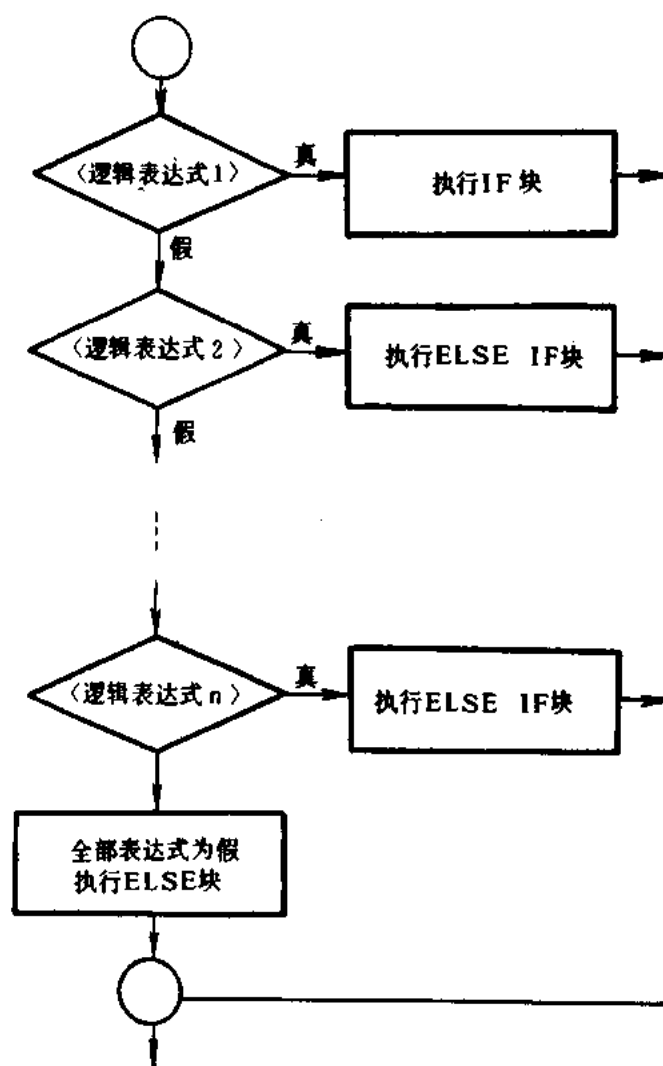


图 4-2

用流程图表示如图 4-3.

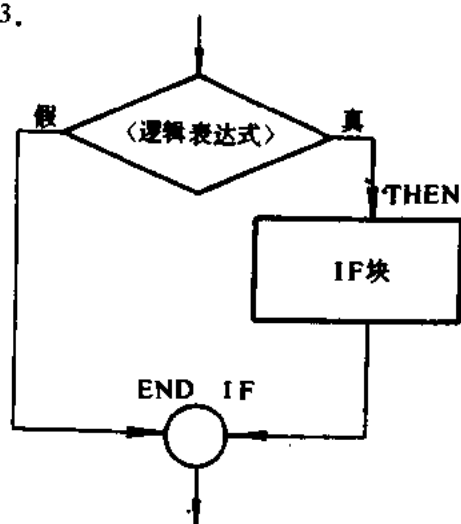


图 4-3

**例 3** 比较下列两段程序(其中 PRINT \*,... 详见下一节):

(I)

```
IF (BELOW .NE. 0) THEN
    AVEBEL = SUMBEL/REAL(BELOW) } (a)
    PRINT *, ...
END IF
IF (ABOVE .NE. 0) THEN
    AVEABV = SUMABV/REAL (ABOVE) } (b)
    PRINT *, ...
END IF
STOP
```

(II)

```
IF (BELOW .NE. 0) THEN
    AVEBEL = SUMBEL/REAL (BELOW) } (a)
    PRINT *, ...
ELSE
    AVEABV = SUMABV/REAL (ABOVE) } (b)
    PRINT *, ...
END IF
STOP
```

对于 (I), 当 BELOW  $\neq 0$  时做 (a), 否则, 当 ABOVE  $\neq 0$  时做 (b)。显然, 有可能 (a) 和 (b) 均不做的情况。

对于 (II), 当 BELOW  $\neq 0$  时做 (a), 否则便做 (b)。这就是说, 执行第 (II) 段程序, (a) 和 (b) 两者必做其一。

(6) 嵌套块 IF 语句与 IF 级的概念

在块 IF 语句中可以再嵌套块 IF 语句, 从而可以描述多层判定问题。一种只有两层嵌套结构的嵌套块 IF 语句如下所示:

```

      IF ((逻辑表达式)) THEN
外层  [ IF ((逻辑表达式)) THEN
        内层 [
              :
              END IF
              :
        ]
      ]
      END IF

```

还可以嵌套更多个块 IF 语句, 因而也就是更多个一一对应的 END IF 语句。每个嵌套块 IF 语句内还可能使用一些 ELSE 语句或 ELSE IF 语句, 从而构成相当复杂的 IF-THEN-ELSE 结构。

为了区分嵌套的 IF-THEN-ELSE 结构中各个语句的匹配关系, FORTRAN 77 引



入了 IF 级的概念。程序中每一个语句都赋予一个 IF 级, 这个 IF 级是这样一个数值, 其定义如下:

$$\text{语句 } S \text{ 的 IF 级} = n_1 - n_2$$

其中:  $n_1$  是从  $S$  语句所在的程序单位开头直至  $S$  之间的, 并不包含  $S$  在内的块 IF 语句的个数;

$n_2$  是从  $S$  语句所在的程序单位开头直到  $S$  之间的, 但不包含  $S$  在内的 END IF 语句的个数。

每一个语句的 IF 级必须是一个正数或零; 每一个块 IF 语句、ELSE 语句、ELSE IF 语句或 END IF 语句的 IF 级都必须为正数; 每个程序单位的最后一个语句 END 语句的 IF 级必须为零。

下面举一个嵌套块 IF 语句的说明性例子, 程序中各语句的 IF 级列于程序的右端。程序中使用 FORTRAN 77 新提供的输入/输出语句, 详见本章下一节。

**例 4** 写一个程序, 读入一个整型数  $K$ 。若  $K = 1$ , 则将  $K$  放大 10 倍; 若  $K = 2$ , 则将  $K$  放大 20 倍; 若  $K = 3$ , 则将  $K$  放大 30 倍; 若  $K$  是其它值, 则打印信息 “The value of  $K$  is incorrect”。

行号	语句部分	IF 级
1	READ *, K	0
2	IF((K .EQ. 1) .OR. (K .EQ. 2) .OR. (K .EQ. 3)) THEN	1
3	IF(K .EQ. 1) THEN	2
4	K = K * 10	2
5	ELSE IF(K .EQ. 2) THEN	2
6	K = K * 20	2
7	ELSE	2
8	K = K * 30	2
9	END IF	2
10	PRINT*, K	1
11	ELSE	1
12	PRINT*, 'THE VALUE OF K IS INCORRECT'	1
13	END IF	1
14	END	0

从这个例还可以看出, 块 IF 语句的 IF 级(值)正好标志该嵌套块 IF 语句的层次。因此, 利用 IF 级这个工具就可把嵌套的判定结构各层之间的关系分划清楚。

利用 IF 级这个概念还可以给出 IF 块、ELSE IF 块和 ELSE 块的正式定义:

1) IF 块 一个 IF 块包括接着块 IF 语句出现的且直到下一个与块 IF 语句有同样 IF 级的 ELSE IF 语句、ELSE 语句或 END IF 语句为止的所有可执行语句, 但不包括上述 ELSE IF 语句、ELSE 语句或 END IF 语句。IF 块可以是空。

2) ELSE IF 块 一个 ELSE IF 块包括接着 ELSE IF 语句出现的直到下一个与 ELSE IF 语句有相同 IF 级的 ELSE IF 语句、ELSE 语句或 END IF 语句为止的所有可执行语句, 但不包括上述 ELSE IF 语句, ELSE 语句或 END 语句。

3) ELSE 块 一个 ELSE 块包括接着 ELSE 语句出现的直到下一个与 ELSE 语句有相同 IF 级的 END IF 语句为止的所有可执行语句, 但不包括 END IF 语句。

使用 IF-THEN-ELSE 结构时还需注意的是,无论是 IF 块,ELSE IF 块还是 ELSE 块,都不允许从块外把控制转移到块内; ELSE IF 语句和 ELSE 语句都不应该有语句标号,即使有也不允许任何语句引用。

#### 4. DO 语句的改动

在 FORTRAN IV 中,对于 DO 语句

$$\text{DO } L \text{ } 1 = n_1, n_2, n_3$$

规定: 1)  $n_1, n_2, n_3$  均为无符号的整型常数或整型变量;

2) 必须满足  $n_2 \geq n_1 > 0$ , 且  $n_3 > 0$ , 又当  $n_3 = 1$  时可省略。

由于上述规定,因而执行一个 DO 语句,至少执行一次循环体。

在 FORTRAN 77 中, DO 语句改为

$$\text{DO } L, 1 = c_1, c_2, c_3$$

其中: 1) 标号 L 后面的逗号为可选项,即可要可不要;

2) 允许循环变量 1 是整型、实型或双精度型变量;

3)  $c_1, c_2, c_3$  可以是整型、实型或双精度型的算术表达式,还可以是不同类型,以及包括负值或零值 ( $c_1$  的值为零除外),当  $c_2$  的值为 1 时,  $c_3$  可省略。

执行 DO 语句的过程是:

1) 由  $c_1, c_2, c_3$  分别求出循环初值  $v_1$ , 终值  $v_2$ , 增值  $v_3$ , 必要时按算术转换规则转换成循环变量的类型;

2) 把初值  $v_1$  赋给变量 1;

3) 按下式确定循环次数

$$N = \text{MAX}(\text{INT}((v_2 - v_1 + v_3)/v_3), 0)$$

其中内部函数 MAX 和 INT 分别用来求最大值和取整,显然,当  $v_1 > v_2$  且  $v_3 > 0$ , 或  $v_1 < v_2$  且  $v_3 < 0$  时,循环次数为零;

4) 检查循环次数  $N$  是否为零,若是,则跳过循环体,执行终值语句后面规定的动作。否则,执行循环体;

5) 当循环体正常执行到终端语句时,循环变量的值加上增值  $v_3$ ;

6) 循环次数减 1,重新转第 4)步执行。

无论循环是正常结束还是非正常结束,循环变量保持退出循环时具有的值。

看下面例子:

$$\text{DO } 10 \text{ } V = 1.3, 5.0, 1.2$$

可以算出  $N = 4$ , 即循环体依次执行循环变量为 1.3, 2.5, 3.7, 4.9 四次。

再看下面例子:

$$\text{DO } 100 \text{ } K = 5, 1$$

易见,  $N = 0$ , 故循环体一次也不执行。

我们知道,在 FORTRAN IV 中,循环允许带扩充域,即当有一个从循环体内向循环

体外的转移后,在循环体外执行的语句序列并没有改变循环变量  $i$  和循环参量  $n_1, n_2, n_3$ , 那么,再从循环体外向循环体内转移是允许的。

在 FORTRAN 77 中,取消了扩充域的设置;也就是说,在任何情况下都不允许控制从 DO 循环体外转移到循环体内。

另外,如果 DO 语句出现在 IF 块、ELSE IF 块或 ELSE 块中,则整个循环体必须分别包含在 IF 块、ELSE IF 块或 ELSE 块中。反过来,如果块 IF 语句出现在一个 DO 循环的范围内,则相应的 END IF 语句必须出现在该 DO 循环范围内。

**例 5** 看一个负增量的循环例,其中关于输入/输出语句的用法下一节我们就要详讲,这里不妨初步认识一下。

```
INTEGER TIME
REAL VELOC, DIST
READ *, VELOC
PRINT *, 'VELOCITY=', VELOC
DO 50 TIME = 20, 10, -1
    DIST = VELOC * REAL(TIME)
    PRINT *, 'DISTANCE =', DIST, 'TIME =', REAL(TIME)
50 CONTINUE
STOP
END
```

运行这个程序时,首先我们输入初始数据,比如,通过终端键盘输入

VELOCITY = 20.0

(其中下加横线的数字表示由程序员键入的数字),那么,20.0 将赋给变量 VELOC。程序输出结果是:

VELOCITY = 20.0000000000	
DISTANCE = 400.0000000000	TIME = 20.0000000000
DISTANCE = 380.0000000000	TIME = 19.0000000000
DISTANCE = 360.0000000000	TIME = 18.0000000000
DISTANCE = 340.0000000000	TIME = 17.0000000000
DISTANCE = 320.0000000000	TIME = 16.0000000000
DISTANCE = 300.0000000000	TIME = 15.0000000000
DISTANCE = 280.0000000000	TIME = 14.0000000000
DISTANCE = 260.0000000000	TIME = 13.0000000000
DISTANCE = 240.0000000000	TIME = 12.0000000000
DISTANCE = 220.0000000000	TIME = 11.0000000000
DISTANCE = 200.0000000000	TIME = 10.0000000000

## 5. DATA 语句功能的增加

在 FORTRAN 77 中,数据初值语句即 DATA 语句的功能有一些扩充。这就是,不

但允许在 DATA 语句中出现数组名,同时还允许出现隐 DO 循环,也就是说,DATA 语句也可用来给整个数组赋初值,以及可以使用隐 DO 循环。这里,隐 DO 循环的规则与输入/输出语句中的隐 DO 循环规则相同。

例如,要给  $50 \times 50$  方阵 A 的对角线元素赋初值 1,可写作:

```
DIMENSION A(50,50)
DATA (A(K,K),K = 1,50)/50*1.0/
```

要注意的是,在 DATA 语句中出现数组元素和子字符串时,它的下标表达式必须是整常数表达式。

再看一个例子。

**例 6** 给数值变量、字符变量和字符数组赋初值,可写作:

```
CHARACTER*5 A, W(-10:5)*4
DATA A,X,Y,W/'AAAAA',2*1.5,10*'POS', 'ZERO', 5*'NEGA'/'
```

赋值结果:

```
A = 'AAAAA'
X = Y = 1.5
W = 'POS', ..., 'POS', 'ZERO', 'NEGA', ..., 'NEGA'
      10 个元素          5 个元素
```

## 4-4 输入/输出功能的扩充

比起 FORTRAN IV, FORTRAN 77 的输入/输出功能有较多的扩充。

这是因为,随着科学技术的发展和进步,一方面,在程序中往往要处理大量的数据,这些数据用原有的输入/输出语句是不够的,它们必须使用各种形式的文件;另一方面,在现代大型计算机系统中,不仅需要频繁地使用各种输入/输出设备,如卡片阅读器、磁带机、磁盘机、宽行打印机以及分时终端上的键盘和显示器,而且还需要在这些设备上建立文件,而这些文件在形式和存取方式方面又都各有不同。因此,在 FORTRAN 77 中,输入/输出语句的功能,不仅考虑了设备的选择、文件的建立,而且还引进了设备与文件的连接与不连接的概念;不仅指出输入/输出数据的对象、编辑的格式,而且还注意到数据传输过程中发生各种情况的处理。此外,还注意文件重新定位等问题。

在 FORTRAN 77 中,可以把输入/输出语句分成两大类:

一类是处理数据传输的输入/输出语句,它用来把数据从外部介质传输到内存或从内部文件传输到内存;或相反地,用来把数据从内存传输到外部介质或从内存传输到内部文件。

另一类是辅助输入/输出语句,它用来建立文件、实现文件与设备的连接或断开、询问文件和设备的特性、操纵外部介质实现文件定位等。

对于数据传输输入/输出语句也可再分为:

- 带格式输入/输出语句;

- 无格式输入/输出语句;
- 表控输入/输出语句。

其中,表控输入/输出语句也可算作带格式输入/输出语句的一种特殊形式。

对于辅助输入/输出语句,也可把其中一部分语句划分为文件定位语句。

这一节我们先讨论文件与记录等概念,然后分别讨论带格式输入/输出语句、表控输入/输出语句和辅助输入/输出语句。无格式输入/输出语句已在第3章讲过,这里没有更多的补充。我们讲基本的部分,其它可以随时从具体机器的使用说明书查得。

## 1. 记录、文件与部件

在上一章中,我们已经初步介绍了记录与文件的概念。事实上,在 FORTRAN IV 中记录和文件的概念没有得到很多的强调,应用也很简单。例如,对于记录,在那里只研究了这样一种格式记录,它由表示数据的字符序列所组成,根据包含在格式说明中的控制信息实现字符与计算机内部值之间的互相转换,并且格式记录的长度用字符来度量,而当格式记录被写出时,它的长度由放置在记录里面的字符数目来确定。对于文件,在那里也主要是使用卡片文件和宽行打印文件而已。

在 FORTRAN 77 中,输入/输出是在现代各种存贮设备广泛使用的基础上来进行的。因此,记录与文件的概念得到充分的强调和应用。下面我们进一步介绍记录与文件的概念。同时介绍与文件相联系的部件的概念

### (1) 记录

一般地,记录是值或字符的序列,所谓序列就是与数  $1, 2, 3, \dots$  一直到  $n$  一一对应的有序集合。

显然,一行数据是一个个数值或一个个字符的有序集合,所以一行数据可以看成是一个记录;一行表头的文字是一个字符的有序集合,所以一行表格头也可看成是一个记录。

一个记录可以包括若干个记录项,如一行表格头中有若干个项目名称一样。这称为记录的数据项或字段。因此,也可以说,记录是相关的数据项的集合。

在 FORTRAN 77 中,记录分三种:格式记录、无格式记录和结束文件记录。

格式记录如上所述,它由字符序列组成,其长度以字符个数来度量;它也可以改变,这取决于写入记录的字符个数,还取决于处理系统以及所使用的外部介质。由于格式记录的数据传输,是通过包含格式说明的控制信息来实现字符序列与数据的计算机内部表示之间的相互转换,故格式记录由带格式输入/输出语句来读和写。

无格式记录由计算机内部表示的值序列组成,其长度以计算机系统所使用的存储单位来度量,并且取决于写出此记录时所使用的输出表以及记录存储在其上面的外部介质。无格式记录中的数据传输无需进行格式的编辑,因此,传输的速度非常快。一般来说,无格式记录用于磁盘和磁带一类的磁性介质。

结束文件记录是所谓顺序存取文件(下面就要讲到)中最后的记录,它仅是一种结束标志,无实在内容,所以没有长度特性。

### (2) 文件

由于在各种设备上存储着不同的记录序列,因此,为了区分比如在同一个设备中同时存储的不同记录序列,便给每个记录序列起一个名字,这就形成了文件的概念。

一般地,文件是记录的序列,或者说,文件是相关的记录的集合。一个文件由一个特定的文件名标识。

一叠穿孔卡片可以构成一个卡片文件,一张印着一行行数字和字符的宽行打印纸可以构成一个行印文件,一份排列了若干中间计算结果的记录可以看成是一个中间结果文件,等等。

在 FORTRAN 77 中,文件分为内部文件和外部文件两种。

先讲内部文件。

内部文件是指其内容被存储在计算机系统内存存储器里的文件。内部文件具有这样一些特性:

1) 内部文件可以是一个字符变量、字符数组元素、字符子串或一个字符数组,这时它们的符号名即为内部文件名。

2) 如果内部文件的记录是一个字符变量、字符数组元素或字符子串,这时,记录的长度分别是变量的长度、数组元素的长度或子串的长度。

3) 如果内部文件是一个字符变量、字符数组元素或字符子串,则此内部文件由一个记录组成。如果内部文件由一个字符数组组成,则文件的记录是数组元素。

4) 只有当作为内部文件记录的变量、数组元素或子串已经被定义时,才可以读入这个记录。对于这些作为记录的变量、数组元素或子串,不仅可以通过写此记录使它们成为有定义,而且可以通过诸如赋值语句或 READ 语句的其它常规手段使它们成为有定义。

5) 内部文件只允许以顺序方式存取。每次数据传送之前,内部文件的指针总是定位于第一个记录的开始处。当在格式说明中扫描到/时,指针即指向下一个记录的开始处,并继续执行输入/输出操作,当然这是指内部文件是一个字符数组的情况。

总之,内部文件提供了从内存到内存传送和转换数据的一种手段。特别是可以实现把数字字符变换成量,或把量变换成数字字符的转换。为了在读/写语句中指明一个内部文件,可以在外部设备号的位置上使用内部文件名。

应用内部文件的场合并不多。看下面示例。

**例 1** 考虑下列语句:

```
CHARACTER * 10 BAS, ARR(5)
.....
READ (BAS,100) X, Y, Z
READ (ARR, '(I5///F10.0)') I, R
.....
```

执行前一个 READ 语句时,将字符变量 BAS (作为内部文件)中各字符按标号为 100 的格式语句的格式转换后赋给变量 X, Y, Z。执行后一个 READ 语句时,将字符数组 ARR 中第一个元素 ARR(1) 中的前五个字符以及第四个元素 (根据格式说明中的 ///) ARR(4) 中的前十个字符分别转换成整型量和实型量后赋给变量 I 和 R。当然,这里 ARR(1) 和 ARR(4) 必须是那些可以转换成整型量和实型量的字符。

现在讲外部文件。

外部文件是指其内容被存储在外部介质、并通过外部设备进行存取的文件。这里,外部介质有卡片、磁盘、磁带、打印纸等,外部设备有卡片阅读机、磁盘机、磁带机、打印机等。使用外部文件可以在外部介质上存取大批的数据。

在 FORTRAN 77 中,文件有两种存取方式:顺序存取方式与直接存取方式。

以顺序存取方式打开文件时,文件指针总是指在文件的初始点,即第一个记录前的位置,并且以后每次顺序存取的当前记录总是上一次存取的记录的下一个记录。事实上,顺序存取的各个记录在写入前并无记录号,只是在写入文件后才以该记录在文件中的排列次序作为它的记录号。不过,在以后的读/写中并不指定这个记录号;顺序存取在于强调顺序。例如,要读第 3 个记录,必须先读过前 2 个记录。顺序存取文件的最后必须放一个文件结束记录。除了这个文件结束记录,其它记录必须全是格式记录或全是无格式记录。不难看出,某些文件只能用顺序存取,如卡片文件和打印文件。

以直接存取方式读/写每个记录时,在读/写语句中必须写明要存取的记录号,也就是说,记录可以按任何次序读和写。直接存取文件的记录规定由序列 1, 2, 3, ... 作为它的记录号,且每个记录的长度相同。一个直接存取的记录建立以后,就可以读,也可以重写,但不能删除。与顺序存取文件不同,读/写直接存取文件的一个记录可按任意次序进行,只需指出该记录号。例如,要读记录号为 100 的记录,可直接读出,而并不需要先读过前面 99 个记录。但与顺序存取文件一样,直接存取文件的记录,或者全是格式记录,或者全是无格式记录。

### (3) 部件

在 FORTRAN 77 中,外部设备通常被称为外部部件,或简称部件。显然,外部文件必须通过部件进行实际存取。例如,磁带文件必须由磁带机实现存取。所以,可以说部件是引用文件的手段。

当然,文件也可以不依赖于这些部件而存在。例如,一个卡片文件可以不依赖于卡片阅读机而存在。但无论怎么样,当执行任何输入/输出语句时,文件却必须与一个部件相联系起来才行,这称为**连接**。所以说,部件有已连接和不连接的特性。如果已连接,就可以通过它来引用文件。在 FORTRAN 77 程序内,外部文件就用所连接的部件标识符来表示。部件标识符的形式是,或者一个非负整表达式,其值指明由计算机系统规定的某一部件;或者符号\*,标识一个交由计算机系统安排的特定部件。

可以通过所谓预连接或执行 OPEN 语句来使部件成为连接状态。后者我们将留在下面再讲。**预连接**指的是当可执行程序执行开始时,部件已被连接到文件上。例如,我们前面一直使用的、分别取设备号 5 和 6 的 READ 语句和 WRITE 语句,就可以看作两个预连接的例子。通常,我们总是假定有这样一个输入部件(如卡片阅读机或终端键盘)和这样一个输出设备(如行式打印机或终端显示器)作为这类预连接的部件,并且以\*号作为这个特定部件的标识符。

必须注意的是,一个部件不能同时连接多个文件,一个文件也不能同时连接多个部件。

## 2. 带格式输入/输出语句

FORTRAN 77 中的带格式输入/输出语句具有如下几种一般形式:

- 1) READ (<控制信息表>) <输入表>
- 2) READ *f*, <输入表>
- 3) WRITE (<控制信息表>) <输出表>
- 4) PRINT *f*, <输出表>

其中各部分的意义说明如下:

<输入表>与<输出表>可以包含用逗号隔开的变量名、数组名、数组元素名和字符子串名,也可以包含隐 DO 循环;对<输出表>还可以是表达式;此外,<输入表>与<输出表>可以不出现,这时,连同前面的逗号也省略。

*f* 是格式说明符,可以是同一程序单位内的一个 FORMAT 语句标号;可以用 ASSIGN 语句对它赋与本程序单位内的一个 FORMAT 语句标号的整型变量名;可以是放入格式说明的字符数组名,还可以是字符表达式。此外,在 *f* 的位置也可换为符号 \*, 这时,上述第 2)、4) 种形式将成为表控输入/输出语句的形式,我们将在下一小节讲。

这里主要介绍<控制信息表>。<控制信息表>包括下列各项(全部或部分):

<设备说明符>,<格式说明符>,<记录说明符>,

<错误说明符>,<文件结束说明符>,<输入输出状态说明符>

其中:<设备说明符>指明所使用的外部设备或内部文件。形式为

UNIT = *u*

当 *u* 是设备标识符时,它可以是取值为零或正数的整表达式;当 *u* 是内部文件标识符时,它可以是字符变量、字符数组元素、字符数组或字符子串的名字。每个<控制信息表>必包含一个设备说明符,但 "UNIT=" 几个字符可以省略,仅有一个 *u*。此外,*u* 这个位置也可换为符号 \*, 这时表示所用的外部设备由系统自行指定。

<格式说明符>用来标识一个格式,其形式为

FMT = *f*

*f* 是格式标识符,它可以是同一程序单位内的一个 FORMAT 语句标号,可以是 ASSIGN 语句对它赋与本程序单位内的一个 FORMAT 语句标号的整型变量;可以是放入格式说明的字符数组名;也可以是字符表达式(特殊情况是字符常数),其值就是格式说明。此外,<格式说明符>可以不出现,这时表示本语句是无格式的输入/输出语句;<格式说明符>也可换为符号 \*, 这时表示本语句是表控输入/输出语句。表控输入/输出语句也将在下一小节讲。

<记录说明符>的一般形式为

REC = *r*

*r* 是取正值的整表达式。在<控制信息表>中,<记录说明符>可有可无。如果有一个的话,表示本语句是直接存取输入/输出语句,这时设备应与直接存取的文件连接,*r* 就是直接存取连接的文件中读/写的记录号;如果没有,则表示这是顺序存取输入/输出语句,这时设备应与顺序存取的文件连接。

<错误说明符>用来处理当本语句执行过程中出现错误时程序的执行去向。输入/输出错误条件取决于系统的规定。<错误说明符>的一般形式为



$$\text{ERR} = 1$$

L 是本程序单位内一个可执行语句的标号。在〈控制信息表〉中〈错误说明符〉不是非要不可。如果〈控制信息表〉包含了〈错误信息说明符〉,则当执行本语句遇到错误条件时,停止本输入/输出语句的执行,并转到标号为 L 的语句去继续执行;否则在遇到错误条件时即停止程序执行。

〈文件结束说明符〉用来指定读到顺序存取文件结束时的处理方式，即只对顺序存取文件有意义，对直接存取文件无作用，因而它不应与记录说明符同时出现，WRITE 语句中的〈控制信息表〉也不应包含这个说明符，〈文件结束说明符〉的一般形式为

END = L

L 是同一程序单位内的一个可执行语句的标号。当输入语句执行读一个顺序存取文件时遇到结束文件记录,或当希望读一个内部文件末端之后的记录,便会产生文件结束条件。如果包含有〈文件结束说明符〉的输入语句遇到了结束条件,便终止输入语句的执行,而转到标号为 L 的语句去继续执行。如果这个说明符不出现,则遇到文件结束条件时便终止可执行程序的执行。

〈输入/输出状态说明符〉用来反映输入/输出语句的执行状态, 其一般形式为

$$\text{IOSTAT} = i_{os}$$

*ios* 是整变量或整数组元素。包含有这个说明符的输入/输出语句,在执行中若未遇到错误条件和文件结束条件,则 *ios* = 0; 若遇到错误条件,则 *ios* 等于由系统决定的一个正值;若遇到文件结束条件而未遇到错误条件,则 *ios* 等于由系统决定的一个负值。如果不需要了解输入/输出执行的状态,也可不写入这个说明符。

综上所述,可知〈记录说明符〉、〈错误说明符〉、〈文件结束说明符〉、〈输入/输出状态说明符〉以及〈输入表〉和〈输出表〉均为可选项,而〈控制信息表〉中各说明符的次序也可以任意,只是当〈部件说明符〉省去符号“UNIT=”时,它一定要放在〈控制信息表〉的第一项;当〈格式说明符〉省去符号“FMT=”时,它一定要放在〈控制信息表〉的第二项,且这时第一项是省去“UNIT=”的〈部件说明符〉。对后面这种情况,实际上就是我们在FORTRAN IV中已经熟悉了的形式(不选用其它说明符)。

下面是带格式输入/输出语句的一些示例:

- ```

1) READ (UNIT = 5,FMT = 100) R, S, T
2) READ (K,LL,IOSTAT = KK,ERR = 110,END = 300) X, Y, M
3) READ ('F10.0,3I6)', R, I, J, K
4) READ (*, '(I6,F10.2)') R1, R2, X1, X2
5) READ (UNIT = 9, FMT = 100, REC = 2*N, IOSTAT = K2,ERR =
    50,END = 99) ((A(I,J),J = 1,100),I = 1,100)
6) WRITE (UNIT = 6,FMT='("___",3I4,I8)') I, J, K, KK
7) WRITE (40,90,IOSTAT = IOERR) ((X(I,J),J = 1,N),I = 1,N)
8) PRINT ('("___",I5,2I10)', I, M1, M2, J, N1, N2
9) PRINT ('("___FORTRAN 77___EXAMPLE1")')

```

10) WRITE(\*,FMT = M) P, Q, L, K

### 3. 表控输入/输出语句

FORTRAN 77 的表控输入/输出语句,是指在语句中不明显地给出记录的格式,输入/输出由输入/输出表进行控制,因而叫做表控输入/输出。FORTRAN 77 中的表控制输入/输出语句类似于某些非标准的 FORTRAN IV 中的自由格式输入语句和固定格式输出语句。有时我们也把表控格式称为自由格式。

就语句的形式而言,只需在带格式输入/输出语句中放格式标识符的地方换为\*,就表示按自由格式输入或按固定格式输出,这便成为表控输入/输出语句。

表控输入/输出处理的记录,由表示值和值分隔符序列的字符所组成。值通常是常数,值分隔符可以是逗号也可以是一个或多个连续空格。

下面是几个正确的表控记录例子:

1) 693.1,1.13,70.65,502

2) 37.41\_510\_4410.23\_9.7

3) 422\_12.20,39510

对任何允许带格式输入/输出的文件,均允许表控输入/输出。我们分析一些具体例子。

#### 例1 读语句

```
READ *, X, SUM, L1, L2, L3
```

表示从已连接的输入部件的文件上从下一个记录开始读入5个值并把它们依次赋给实型变量 X, SUM 和整型变量 L1, L2, L3。这里,要求输入记录的值的类型应与输入表变量的类型相匹配。

#### 例2 读语句

```
READ (*,*) A, B, C
```

在预连接的特定部件(如卡片或终端屏幕)上输入数值  $A = 1.1$ ,  $B = 2.2$ ,  $C = 3.3$ , 则下列构成的多种输入记录形式都是允许的:

1.1, 2.2, 3.3

数据 1.1 从卡片或屏幕上第 1 列写起,两数据之间用逗号隔开,中间不插入空格符及斜线。

\_1.1, \_2.2\_3.3

数据之间的分隔也可用空格符;记录间也可插入空格符。

\_1.1, \_2.2, \_3.3/

采用斜线作为数据记录的结束符号。

#### 例3 设程序中有两个读语句

.....

```
READ *, A, (B(I), I = 1, 10), C, D, E, M, N
```

.....

```
READ(*,*, END = 99) L, (Y(I), I = 1, 20)
```

.....

其中 B, Y 定义为数组, C 为复型变量, D 为字符型变量, L 为逻辑型变量, 其它按隐式说明。

提供的三个数据记录为

3.1416\_5\*0.0, 5\*, (1.5, -0.3)

'YES', 100, 200,

F, 10\*-1.0/

则执行上述两个读语句的结果是:

A 取得值 3.1416;

B(1)~B(5) 各取得值 0.0;

B(6)~B(10) 保持原值不变(输入五个空项目);

C取得值 (1.5, -0.3);

D取得值 'YES';

E取得值 100.0;

M取得值 200;

N保持原值不变;

L取得逻辑值 .FALSE.;

Y(1)~Y(10)各取得值 -1.0;

Y(11)~Y(20)保持原值不变(无对应项目)。

例中采用了数据的缩写形式, 如用 5\*0.0 表示连续 5 个 0.0; 用 5\* 表示连续五个空项目, 连续两个逗号也表示一个空项目。此外, 当数据是逻辑常数时, 用 T 表示真 (.TRUE.), 用 F 表示假 (.FALSE.).

下面看表控输出的例子。首先要强调指出, 表控输出与计算机的处理系统密切相关, 也就是说, 表控输出的格式, 对不同的处理系统各有不同的规定。

**例 4** 设  $A = 74.0$ ,  $B = 3500.0$ ,  $C = 123.4$ , 考虑下列表控输出语句

PRINT \*, A, B, C

假设所用处理系统规定以格式 F8.2 输出实型数, 即以 8 个字符位输出一个实常数, 其中小数点后占 2 位数字, 那么, 执行上述语句输出结果(打印或显示)。

\_\_\_74.00\_\_\_3500.00\_\_\_123.40

**例 5** 表控输出语句

PRINT \*, 'X1=', X1, 'X2=', 1.0/X1

WRITE(\*, \*) '\_\_\_X=', X, '\_\_\_Y=', Y, '\_\_\_X + Y=', X + Y

其中, 假设  $X1 = 4$ ,  $X = 5$ ,  $Y = 6$ , 则在特定的已连接部件上输出两个记录是(按处理系统的规定):

X1 = \_\_\_4.0000X2 = \_\_\_\_.2500

\_\_\_X = \_\_\_5.0000\_\_\_Y = \_\_\_6.0000\_\_\_X + Y = \_\_\_11.0000

其中附加的符号 0 由具体处理系统按具体规定插入。此外, 作为记录第一个字符的空白符, 也由编译程序插入, 作为控制纵向间隔。

**例 6** 考虑下列表控输入/输出语句

READ \*, A0, A1, A2, AN, T, N

```

PRINT *, A0, A1, A2, AN, T, N
PRINT *, 'FOR T=', T, 'X=', A0 + A1 * T + A2 * T * * 2 + AN
      * T * * N

```

当输入记录为

12.2, 16.1, 3.2, 4.6, 3.0, 3

时, 在某处理系统上将可输出如下:

```

      12.2000      16.1000      3.20000      4.60000      3.00000
              3
FOR T=  3.00000      X=  213.500

```

除上述各例所涉及的情形之外, 我们还强调如下几点:

1) 输入表中包含的项目数可以比记录中包含的值的个数多, 即一个输入语句可连续读多个记录; 但若整个文件包含的值的个数不足输入表的需要, 则程序的执行将停止. 反之, 若记录中值的个数比输入表中的项目多, 则除了输入表中项目被赋值外, 记录中余下的值将被忽略.

2) 输入语句每次执行总是读入一个新记录, 对文件来说就是从下一个记录开始读入. 相仿, 输出语句每次执行总是从一个新的记录开始, 并且由具体的处理系统的需要输出一个或多个记录.

3) 输出整型数值时, 按整型数打印, 其宽度可能是统一大小, 也可能按所输出的数值大小分别给定, 这取决于处理系统; 输出实型数值(包括双精度数值)时, 可能用基本实常数形式, 或用含指数的实常数形式, 这也取决于处理系统.

4) 特别地, 没有输出表的语句 PRINT \*, 执行结果是输出一个空记录, 即输出一个空行.

再看两个完整的简单例子.

**例 7** 产生三角函数表.

C PROGRAM PRODUCES A TRIGONOMETRIC TABLE

```

      INTEGER ANGLE
      REAL CONV, RADIAN
      CONV = 3.141593/180.0
      PRINT *, 'ANGLE      RADIAN      SINE      COSINE      TAN'
      DO 100 ANGLE = 0, 90, 15
          RADIAN = CONV * REAL (ANGLE)
          SINE = SIN (RADIAN)
          COSINE = COS (RADIAN)
          TANG = TAN (RADIAN)
          PRINT *, REAL(ANGLE), RADIAN, SINE, COSINE, TANG
100  CONTINUE
      STOP

```

END

输出结果如下:

| ANGLE   | RADIAN  | SINE    | COSINE       | TAN          |
|---------|---------|---------|--------------|--------------|
| 0       | 0       | 0       | 1.00000      | 0            |
| 15.0000 | .261799 | .258819 | .965926      | .267949      |
| 30.0000 | .523599 | .500000 | .866025      | .577350      |
| 45.0000 | .785398 | .707107 | .707107      | 1.00000      |
| 60.0000 | 1.04720 | .866025 | .500000      | 1.73205      |
| 75.0000 | 1.30900 | .965926 | .258819      | 3.73205      |
| 90.0000 | 1.57080 | 1.00000 | -1.73205E-07 | -5.77350E+06 |

例8 打印\*程序:

```
INTEGER I, N
READ *, N
PRINT *, M, 'LINES OF XS WILL BE PRINTED'
DO 10 I = 1, N
    PRINT *, '*****'
10 CONTINUE
STOP
END
```

假设我们给N输入的数据是5,即输入记录为

5

则输出结果是

```
5 LINES OF XS WILL BE PRINTED
*****
*****
*****
*****
*****
```

#### 4. 新增加的格式说明符

FORTRAN 77 在格式说明符方面也有所增加。格式说明符用来指定文件中每一个值应该以怎样的格式出现。格式说明符分字段描述符和字段分隔符,在 FORTRAN 77 中,字段描述符常称为编辑描述符。编辑描述符指定文件下一部分的格式的某些方面。其中,指定文件中下一个数据的值的类型和格式的编辑描述符都是可重复的描述符;指定诸如保留空白或跳过不要求的数据那样的编辑描述符都是不可重复描述符。

FORTRAN 77 新增加的编辑描述符如下:

rlw.m    r 为重复数, w 为字段宽度, lw 的意义同前, m 表示输出时数字必须有 m 位,否则在左边补足零;输入时的处理方式与 lw 同。

rEw.dEc    r 为重复数, Ec 表示输出时指数部分占 c 位,即至少  $w \geq d + c + 5$ ; 输入时与 Ew.d 同。

|        |                                                                                                                  |
|--------|------------------------------------------------------------------------------------------------------------------|
| A      | 表示以输入/输出表中元素的字符长度作为格式长度。                                                                                         |
| Tn     | 确定从记录开头的第 n 个位置作为下一次转送的字符位置,即将文件指针拨到当前记录的第 n 个字符。                                                                |
| TLn    | 确定从记录当前位置向左移回 n 个字符,即文件指针向左移 n 个字符。                                                                              |
| TRn    | 确定从记录当前位置向右移 n 个字符,即文件指针向右移 n 个字符。                                                                               |
| SP, SS | 符号控制说明符,用于输出语句的格式控制。在格式说明中,从遇到 SP 至下一个符号控制说明符之间输出的正数值均印刷+号。与 SP 同一类的说明符。在格式说明中,从遇到 SS 至下一个符号控制说明符之间输出的正数值均不印刷+号。 |
| S      | 与 SP, SS 同一类的说明符。从遇到 S 直至下一个符号控制说明符之间输出的正数值是否带+号由具体系统规定。                                                         |
| BN     | 空白控制说明符,只用于输入语句。在输入格式控制中,从遇到 BN 直至下一个空白控制符之间,输出的数据中的空白当作零来处理。                                                    |
| BZ     | 与 BN 同一类的说明符,只是对所指输入数据中的空白均被忽略不计,向右压缩对齐。                                                                         |
| :      | 作格式控制符用。当输入/输出表已用完时,冒号:作终止控制,否则冒号:只起一般分隔作用。                                                                      |

## 5. 辅助输入/输出语句

除了上面讲的数据传输语句 READ, WRITE 和 PRINT 以外, FORTRAN 77 还提供了六个辅助性的输入/输出语句,用来操纵外部介质或描述外部介质的有关性质。FORTRAN 77 中的辅助输入/输出语句包括:

- OPEN 语句(打开语句),
- CLOSE 语句(关闭语句),
- INQUIRE 语句(询问语句),

和另外三个文件定位语句:

- REWIND 语句(重绕语句),
- BACKSPACE 语句(回退语句),
- ENDFILE 语句(文件结束语句)。

后三个语句在第 3 章中已有简单的介绍,这里主要介绍前三种语句。

### (1) OPEN 语句

OPEN 语句也称打开语句,主要用来建立一个新文件并把它连接到部件上,或者将一个已经存在了的文件连接到部件上。

OPEN 语句的一般形式为

OPEN (<说明符表>)

其中<说明符表>包括用逗号隔开的下列九种说明符(全部或部分):

- <部件说明符>, <文件名说明符>, <记录长度说明符>,
- <文件状态说明符>, <存取说明符>, <格式类型说明符>,

〈I/O 状态说明符〉, 〈错误说明符〉, 〈空白说明符〉

下面对这些说明符加以详细介绍:

〈部件说明符〉的形式为

$$\text{UNIT} = u$$

$u$  是外部部件标识符, 是一个非负整表达式, 但不能是 \* 号, 它用来指定外部文件要连接的外部部件. 部件说明符必定要有, 但符号 “UNIT=” 可以省略, 这时, 部件标识符必是说明符表的第一项. 而且, 除了这种情形外, 所有说明符在说明符表中的次序可以任意.

〈文件名说明符〉的形式为

$$\text{FILE} = fin$$

$fin$  是字符表达式, 其值就是要连接到所指部件的外部文件名. 文件名说明符可以不出现, 这时连接到部件的文件由计算机系统决定. 需要注意的是, 出现在说明符表中的字符表达式, 其值是指去掉尾随空白符以后所得的值.

〈记录长度说明符〉的形式为

$$\text{RECL} = rc$$

$rc$  是正整表达式, 其值就是所连接的文件中每一个记录的长度. 只有当一个文件作为直接存取连接时, 才需要写出这个说明符. 当文件连接格式输入/输出时, 则此长度是字符个数; 当文件连接无格式输入/输出时, 则此长度按处理系统规定的单位来计量.

〈文件状态说明符〉的形式为

$$\text{STATUS} = sta$$

$sta$  是字符表达式, 当所指文件已存在外部部件时,  $sta$  应取得使其值为 ‘OLD’ (旧的); 当所指的文件不存在外部部件时,  $sta$  应取得使其值为 ‘NEW’ (新的), 从而使得文件被建立, 并使  $sta$  给出值 ‘OLD’; 当文件属于中间暂用性质 (称为 “废” 的), 或准备在执行这个部件的 CLOSE 语句或在程序结束时将本文件取消, 这时  $sta$  应取得使其值为 ‘SCRATCH’ (暂时); 当不确定文件是否存在,  $sta$  可取 ‘UNKNOWN’ (未知), 这时计算机检索文件目录, 若所指文件存在, 则状态取 ‘OLD’, 否则, 状态取 ‘NEW’. 若这个说明符未写出时, 便认为文件状态是 ‘UNKNOWN’.

〈存取说明符〉的形式为

$$\text{ACCESS} = acc$$

$acc$  是字符表达式. 当  $acc$  取值 ‘SEQUENTIAL’ 时, 文件连接为顺序存取; 当  $acc$  取值 ‘DIRECT’ 时, 文件连接为直接存取; 如果这个说明符省略, 则当作顺序存取处理.

〈格式类型说明符〉的一般形式为

$$\text{FORM} = fm$$

$fm$  是字符表达式. 当  $fm$  取值 ‘FORMATTED’ 时, 文件连接成格式输入/输出; 当  $fm$  取值 ‘UNFORMATTED’ 时, 文件连接成无格式输入/输出; 如果这个说明符省略, 则当文件是顺序存取时, 便采用格式输入/输出, 当文件是直接存取时, 便采用无格式输入/输出.

〈I/O 状态说明符〉的形式为

IOSTAT = *ios*

*ios* 是整变量或整数组元素。如果 OPEN 语句执行时不发生错误, 则 *ios* 被赋予 0 值, 否则由处理系统赋予一个特定的正整数值。在无需了解输入/输出状态时, 这个说明符可省略。

〈错误说明符〉的形式为

ERR = L

L 是包括 OPEN 语句在内的同一程序单位的一个可执行语句标号。当执行 OPEN 语句出错时, 程序便转向标号 L 的语句去继续执行。如果没有写这个说明符, 则当出现错误条件时, 程序便终止执行。

〈空白说明符〉的形式为

BLANK = *blank*

*blank* 是字符表达式。当 *blank* 取值 'ZERO' 时, 除前导空白符外的所有空白符均作零; 当 *blank* 取值 'NULL' 时, 除了包含全部空白符的输入字段作为零值处理时, 所指部件上的数值格式输入字段中的空白符均被忽略。如果这个说明符不出现, 则当作取值 'NULL' 的情况。

最后需指出的是, OPEN 语句可以出现在一个源程序中的任意程序单位。OPEN 语句一经执行, 则所作的连接在随后的整个程序执行过程中均有效, 除非遇到断开连接的操作。

**例 5** 为了将一个已建立了的文件 ZHENG 作为顺序格式输入/输出方式连接到部件 12 上, 可写

OPEN (12, FILE = 'ZHENG', STATUS = 'OLD')

为了将一个中间暂用的文件作为顺序无格式输入/输出方式连接到部件 21 上, 可写:

OPEN (STATUS = 'SCRATCH', FORM = 'UNFORMATTED', UNIT = 21)

为了将未知状态的文件 FUZFIL 作为顺序格式输入/输出方式连接到部件 13 上, 并且当读入数值时, 空白符将解释为零值, 可写:

OPEN (13, FILE = 'FUZFIL', BLANK = 'ZERO')

为了建立一个这样的文件: 这个文件名为 AAI, 记录长度有 256 个字符, 以直接格式输入/输出, 连接到部件 90, 当读入数值时, 空白符将被忽略, 当执行 OPEN 语句出现错误条件时, KK 将被赋予某一正值, 而程序转向标号为 100 的语句继续执行, 否则, KK 将赋予零值。可写 OPEN 语句如下(其中+号为续行标志):

OPEN(90, FILE = 'AAI', RECL = 256, STATUS =  
+ 'NEW', FORM = 'FORMATTED', ACCESS = 'DIRECT',  
+ IOSTAT = KK, ERR = 100, BLANK = 'NULL')



## (2) CLOSE 语句

CLOSE 语句也称关闭语句,用来从部件断开文件。

CLOSE 语句的一般形式为

CLOSE (<说明符表>)

其中<说明符表>类似于 OPEN 语句中说明符表的意义,可以包括用逗号隔开的下列全部或部分说明符:

<部件说明符>: UNIT = <外部部件说明符>

<I/O 状态说明符>: IOSTAT = <整变量或整数组元素>

<错误说明符>: ERR = <语句标号>

<文件状态说明符>: STATUS = <字符表达式>

这里,需另外说明的是,<文件状态说明符>用来指出文件未来的状态,即当文件不是暂用文件且不被删除时,STATUS 取值 'KEEP';当文件将被删除时,STATUS 取值 'DELETE';如果这个说明符省略,则除了暂用文件取值 'DELETE' 外,其它文件的取值 'KEEP'。

**例 6** 相应例 5 中的四个 OPEN 语句,我们可分别写

CLOSE(12)

这时,文件 ZHENG 断开与部件 12 的连接并被保存起来。

写

CLOSE(21)

因文件是废文件,又文件状态说明符不出现,故它被关闭和删除。

写

CLOSE(13,STATUS = 'KEEP')

文件 FUZFIL 断开与部件 13 的连接并被保存。

写

CLOSE (STATUS = 'DELETE',UNIT = 90)

文件 AA1 断开与部件 90 的连接并被删除。

## (3) INQUIRE 语句

INQUIRE 语句也称询问语句,用于询问文件或部件的某些特性。

有两种一般形式:

INQUIRE(FILE = <字符表达式>,<询问说明符表>)

INQUIRE(UNIT = <外部部件标识符>,<询问说明符表>)

前者是按文件询问,后者是按部件询问。其中,<字符表达式>的值是文件名;符号 "UNIT=" 可省略;<询问说明符表>由询问说明符组成。询问说明符有十多个,大部分与 OPEN 语句中的说明符类似,这里不一一列出,可查阅所用计算机的 FORTRAN 77 使用手册。下面看简单例子。

**例 7** 三个询问:

询问名为 AA2 的文件是否存在,可写

INQUIRE (FILE = 'AA2', EXIST = L)

这时,若文件存在,则逻辑变量L的值为真,否则为假.

询问部件 18 所连接的文件的名称,可写

INQUIRE (18, NAME = H)

这时,若文件有名字,则字符变量H的值就是该文件的名称,否则H没有定义.

询问部件 20 是否连接到直接存取的文件,可写

INQUIRE (20, NEXTREC = N)

这时,如果部件 20 已连接到直接存取文件,则整变量N赋值为  $n + 1$ ,  $n$  是已连接直接存取文件读/写的最后记录号,即N是下一个要传输的记录号;若文件连接后从未读/写过,则整变量N的值为 1.

关于三个文件定位语句,在第 3 章中已有简单的介绍. 在 FORTRAN 77 中,三个语句都有两种形式,列出如下:

重绕语句

REWIND  $u$

或

REWIND (UNIT =  $u$ , IOSTAT =  $ios$ , ERR = S)

回退语句

BACKSPACE  $u$

或

BACKSPACE (UNIT =  $u$ , IOSTAT =  $ios$ , ERR = S)

文件结束语句

ENDFILE  $u$

或

ENDFILE (UNIT =  $u$ , IOSTAT =  $ios$ , ERR = S)

其中各语句中的  $u$  是外部部件标识符,说明符 UNIT =  $u$ , IOSTAT =  $ios$ , ERR = S 的意义及用法与 OPEN 语句中相应的说明符的意义及用法类似.

## 4-5 过程·函数与子程序

在标准 FORTRAN 77 中,过程的类型以及所包括的函数与子程序如下:

|    |   |              |   |      |
|----|---|--------------|---|------|
| 过程 | { | 内部函数         | } | 函数过程 |
|    |   | 语句函数         |   |      |
|    |   | 外部函数 (函数辅程序) |   | 外部过程 |
|    |   | 子程序 (子程序辅程序) |   |      |

与 FORTRAN IV 比较,这里没有太多的本质区别.除了 FORTRAN 77 把 FORTRAN IV 中的内部函数和基本外部函数归并为内部函数外,主要是 FORTRAN 77 增加了三个可供辅程序利用的功能,也就是三个语句: SAVE 语句、ENTRY 语句和选择 RETURN 语句.再有,就是这里我们把 FORTRAN IV 中称为函数子程序和子例程子程序的名字分别称为函数辅程序和子程序辅程序.

值得指出的是,外部函数和子程序还可以是非 FORTRAN 外部函数和非 FORTRAN 子程序,它们可由 ALGOL 语言或汇编语言等写成。但这已超出本书的讨论范围,而且也超出标准 FORTRAN 77 的范围。

### 1. 内部函数

在 FORTRAN IV 中,由编译程序提供的函数分为内部函数和基本外部函数两种,如第 2 章附录 A 的 FORTRAN IV 标准函数表所示。

在 FORTRAN 77 中,把由编译程序提供的“内部构造”的函数均称为 FORTRAN 77 的内部 (INTRINSIC) 函数,并将具有同一功能的内部函数分成属名 (*Generic name*) 和专用名 (*Specific name*) 两种,且在内部函数表中给予标明。FORTRAN 77 内部函数表见本章附录 A。

对于属名内部函数允许其自变量的类型多于一种。例如,属名内部函数 SIN,自变量可以是实型、双精度型或复型,而其函数值具有与自变量一样的类型。对于专有名内部函数,如 SIN 有三个专有名 SIN, DSIN, CSIN,对于每一个,如 DSIN 的自变量只能是双精度型,而函数值也只能是双精度型。

属名内部函数给编写含有未确定自变量类型的内部函数提供了方便。例如

```
FUNCTION WON(T)
REAL WON, T
WON = SIN(T) + 5.0
END
```

这时,当 T 为实型时, SIN(T) 也是实型;当 T 为双精度型时, SIN(T) 也是双精度型。

一般地,我们推荐使用属名。但 FORTRAN 77 规定,若内部函数作为辅程序引用中的实元时,则必须使用合适的专有名,而且还必须在内部函数说明语句中说明。

### 2. 内部函数说明语句 (INTRINSIC) 与外部过程说明语句 (EXTERNAL)

在 FORTRAN IV 中,已经有了外部过程说明语句 EXTERNAL 语句,它用来通知 FORTRAN 编译程序,在本程序单位中,在调用函数辅程序和子程序辅程序时,实元中哪些名字是外部过程名。

FORTRAN 77 增加规定,当调用一个外部过程时,作为实元使用的每一个内部函数的专用名,必须在内部函数说明语句 INTRINSIC 中加以说明。

内部语句的一般形式为

```
INTRINSIC <内部函数名>, <内部函数名>,...
```

因此,如果用 FORTRAN IV 写的某段主程序中有说明语句

```
EXTERNAL SIN, COS
```

则用 FORTRAN 77 写这段主程序时就应当改写为

```
INTRINSIC SIN, COS
```

可见, FORTRAN IV 与 FORTRAN 77 在这里有很大的差别, 务必注意。

另外, 内部函数的属名也可出现在 INTRINSIC 语句中, 它在本程序单位中仍保持属名的特性, 而作为实元还是按专有名来传送。只是如下几类内部函数:

类型转换函数, 如 INT, FLOAT 等;

选取最大最小值函数, 如 MAX, AMIN1 等;

字符关系函数, 如 LGE, LLE 等,

均不得作为实自变量使用。

还有, 在 INTRINSIC 语句中出现的名字不能重复出现, 也不能再在 EXTERNAL 语句中出现。

对于外部语句 EXTERNAL, 其形式在 FORTRAN IV 和 FORTRAN 77 中都是一样的。只有当外部过程名或数据块子程序名作为实元引用时, 这些名字才必须在 EXTERNAL 语句中加以说明。

如果内部函数名出现在 EXTERNAL 语句中, 则在此程序单位内, 该名字便失去内部函数的特性, 从而认为它是使用者提供的外部过程名或数据块子程序名。

**例 1** 使用内部函数作实元的例。

印刷一张指数函数  $y = e^x$  和对数函数  $y = \ln x$  在区间  $(0, 10]$  以步长 0.1 函数值表。

\*

```
PROGRAM EXPLOG
INTRINSIC EXP, ALOG
PRINT 100
100 FORMAT ('1', 8X, 'X', 10X, 'EXP(X)' 10X, 'LOG(X)')
CALL SUBPRI (EXP, ALOG)
STOP
END
```

\*

```
SUBROUTINE SUBPRI (F1, F2)
X = 0.0
40 X = X + 0.1
IF(X .GE. 10.0) GO TO 50
PRINT 200, X, F1(X), F2(X)
200 FORMAT ('0', 5X, F6.1, 2(10X, F10.3))
GO TO 40
50 RETURN
END
```

### 3. SAVE 语句、ENTRY 语句和选择 RETURN 语句

#### (1) SAVE 语句

SAVE 是存储起来的意思。SAVE 语句的一般形式为

SAVE  $a, a, \dots, a$

其中  $\alpha$  可以是变量名、数组名或夹在斜线中间的公用区名，但不能是哑元、过程名或公用区中的变量名、数组名。SAVE 后面的  $\alpha$  可以只有一个，还可全没有，即全省略。在  $\alpha$  全省略时，就相当于在该程序单位中，凡可以作为  $\alpha$  指定的实体均需要作 SAVE（存储）。

SAVE 语句在主程序中不起作用，只在其它程序单位中起作用。

SAVE 语句的作用是：

1) 对SAVE语句中指定的变量和数组，如果在执行外部过程的RETURN语句或END语句时，它们的值是确定的，则当再次引用该外部过程时，它们仍保持原来已确定的值。

2) 若在 SAVE 语句中指定公用区名，则属于该公用区的所有变量和数组均保持原确定的值。

3) 若在辅程序中无 SAVE 语句，则在执行该辅程序中的 RETURN 语句或 END 语句时，除了在 DATA 语句中赋了初值而又不曾重新定义过的变量和数组外，非公用区的变量和数组，或者虽属于公用区但在引用该过程的那个程序单位中并没有在公用区中出现的那些变量和数组，均变成无确定的值。

由于很少有机会来存储这种辅程序的局部实体，所以，SAVE 语句的作用有限。

看一个例子。

**例 2** 按关键字 KEY 检索一个具有 N 个元素的向量 LIST（假定 LIST 中至少有一个 KEY），并用 INDEX 送回这个关键字的索引。采用简单的线性检索法，即一次检索是从前一次已完成检索的地方开始的，所以完成的点必须存储起来供下次检索用。

实现这项工作的子程序如下：

```
SUBROUTINE SEARCH (LIST,N,KEY,INDEX)
  INTEGER LIST(N), KEY, INDEX, LOOK
  SAVE LOOK
  DATA LOOK /-1/
  IF(LOOK .EQ. -1) THEN
    LOOK = 1
  END IF
  IF (LIST (LOOK) .NE. KEY) THEN
    LOOK = MOD (LOOK,N) + 1
  END IF
  INDEX = LOOK
END
```

## (2) ENTRY 语句

ENTRY（入口）语句用来指定进入外部过程的自选入口点。ENTRY 语句的一般形式为

ENTRY <入口名> (<哑元>)

其中，<入口名>的取名规则与变量名相同，<哑元>的规定与外部过程中对于<哑元>的规定相同。

表面看来, ENTRY 语句似乎为程序员打开了方便之门, 然而, 实际上却往往带来了祸害。按照结构程序设计的观点, 每个“模块”应该只有一个入口点。

看一个使用 ENTRY 语句的例。

**例 3** 设计一个函数辅程序, 可以分别对下列情形求三角形的面积:

- 1) 已知三条边;
- 2) 已知一条边和三个角;
- 3) 已知两条边及夹角;
- 4) 已知三个顶点的笛卡儿坐标。

函数辅程序如下:

```
FUNCTION AREA (A,B,C)
PARAMETER (PI = 3.1415926, FAC = PI/180)

*
S = 0.5 * (A + B + C)
AREA = SQRT (S * (S - A) * (S - B) * (S - C))
RETURN

*

ENTRY AREA1 (A, ANGA, ANGB, ANGC)
X = A * SIN (FAC * ANGB) / SIN (FAC * ANGA)
AREA1 = 0.5 * A * X * SIN (FAC * ANGC)
RETURN

*

ENTRY AREA2 (A, B, ANGC)
AREA2 = 0.5 * A * B * SIN (FAC * ANGC)
RETURN

*

ENTRY AREA3 (X1, Y1, X2, Y2, X3, Y3)
AREA3 = 0.5 * ABS (X1 * (Y2 - Y3) + X2 * (Y3 - Y1) + X3 * (Y1 - Y2))
END
```

对这个函数辅程序可以出现如下一些调用。例如:

AREA (U, V, W)

这将从正常入口处进入 AREA 辅程序, 并导致实元 U, V, W 与哑元 A, B, C 的结合, 随后执行

```
S = 0.5 * (A + B + C)
AREA = SQRT (S * (S - A) * (S - B) * (S - C))
RETURN
```

又例如

AREA3 (2.2, 0.9, -1.0, -3.0, 1.1, -2.0)

这将从“最后”一个入口处进入实际上以 AREA3 为名的函数辅程序，并导致实元 2.2, 0.9, -1.0, -3.0, 1.1, -2.0 与哑元 X1, Y1, X2, Y2, X3, Y3 的结合,随后执行

```
AREA3=0.5*ABS(X1*(Y2-Y3)+X2*(Y3-Y1)+X3*(Y1-Y2))
END
```

还可以有其它情形的调用,如

```
AREA2 (R, S, PHI)
AREA1 (A, BET1, BET2, BET3)
```

等.

### (3) 选择 RETURN 语句

选择 RETURN 语句用来从子程序辅程序返回到调用该子程序的程序单位中的某一指定语句,其形式为

```
RETURN <整表达式>
```

<整表达式>之值就用来确定控制应该从被调单位返回到调用单位的哪一个语句.

与 ENTRY 语句一样,这里又一次违背结构程序设计的基本原则,因此,程序员最好不要轻率地使用这类多出入口语句.

下面的示意性例子说明选择 RETURN 语句的用法.

**例 4** 考虑主程序 MMM 和子程序 SSS.

```
SUBROUTINE SSS (X, *, Y, *, K)
  IF(X)10, 20, 30
10  RETURN 1
20  K = K + Y
   RETURN INT (X + 1)
30  RETURN
   END

*   PROGRAM MMM
   .....
   CALL SSS (P, *111,Q, *222, KK)
99  TA = P + KK
   GO TO 555
222 TA = TA + Q
   GO TO 555
111 TA = TA + P
555 CONTINUE
   .....
   END
```

从例中可见,子程序中的每个哑元“\*”,在 CALL 语句中,将用实元“\*<语句标号>”

代替。执行选择 RETURN 语句的作用是：计算出 RETURN 后边〈整表达式〉之值；如果该值小于 1 或大于在子程序语句(或 ENTRY 语句)中 \* 号的个数，则选择 RETURN 将按平常的 RETURN 执行；如果该值大于等于 1 且小于等于子程序语句中的 \* 号个数，则控制返回到 CALL 语句里次序对应的 \* 号所带的标号的语句上去。

## 4-6 FORTRAN 77 程序例

### 1. 含双精度、逻辑、字符和复数功能的程序例

#### 例 1 双精度内积程序。

标准 FORTRAN 77 中提供了一个双精度乘积内部函数 DPROD (见本章附录 A)，它由两个单精度实型变量得出一个双精度型乘积值(函数值)。

在数值计算中，我们常常需要计算两个向量，比如

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

的内积

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

其中  $x_1, x_2, \cdots, x_n; y_1, y_2, \cdots, y_n$  均为单精度实型量。为此，可以利用 DPROD 函数来减少内积运算中舍入误差的影响。这就是，把内积项作为双精度量来累加，再把最后累加结果舍入为单精度形式。

实现这一运算的通用函数辅程序如下：

```

FUNCTION DDPROD (X,Y,N)
REAL X(N), Y(N)
DOUBLE PRECISION SUM
SUM = 0
DO 10 I = 1, N
    SUM = SUM + DPROD (X(I),Y(I))
10 CONTINUE
DDPROD = SUM
END
    
```

**例 2** 为了检测每一个非零坐标的点  $(x, y)$  落在哪一个坐标象限内，且按习惯对应送回 1, 2, 3 或 4，即如果  $x > 0$  且  $y > 0$  时返回值 1；如果  $x < 0$  且  $y > 0$  时返回值 2，等等。

函数辅程序如下：

```

INTEGER FUNCTION QUAD(X,Y)
    
```



```

LOGICAL XPOS, YPOS
XPOS = X .GT. 0.0
YPOS = Y .GT. 0.0
IF(XPOS .AND. YPOS) THEN
    QUAD = 1
ELSE IF(.NOT. XPOS .AND. YPOS) THEN
    QUAD = 2
ELSE IF(.NOT. XPOS .AND. .NOT. YPOS) THEN
    QUAD = 3
ELSE
    QUAD = 4
END IF
END

```

**例3** 插入排序算法及其在字符排序中的应用例.

在第2章中我们已经介绍过排序算法中的“气泡漂起”算法和逐一比较算法. 这里再介绍另一种简单插入排序算法.

设要排序的对象为  $a_1, a_2, \dots, a_n$ . 插入排序算法如下:

```

loop for  $j = 2, 3, \dots, n$  do
    {  $a_j \rightarrow s$ 
       $j - 1 \rightarrow i$ 
      loop while  $i > 0$  与  $s < a_i$  do
          {  $a_i \rightarrow a_{i+1}$ 
             $i - 1 \rightarrow i$  }
           $s \rightarrow a_{i+1}$  }
    }

```

现在应用这个算法来编写一个字符数组  $C(0:N)$  的排序程序. 设一维字符数组  $C$  除了第一个元素  $C(0)$  外, 其它每个元素均包含一个不带连字符且字符数不超过15的英文字(必要时每个字的右边补上空白符), 要求排列这些字成英文字典的次序.

子程序辅程序如下:

```

SUBROUTINE CSORT (C,N)
CHARACTER C(0:N)*(*), S*15
C(0) = '_'
DO 100 J = 2, N
    S = C(J)
    I = J - 1
200 IF (I .GT. 0 .AND. T .LT. C(I)) THEN
        C(I + 1) = C(I)
        I = I - 1

```

```

        GO TO 200
    END IF
    C(I + 1) = S
100 CONTINUE
    END

```

**例4** 解复系数二元联立线性方程组程序.

已知复系数二元联立线性方程组为

$$\begin{cases} a_1 z_1 + b_1 z_2 = c_1 \\ a_2 z_1 + b_2 z_2 = c_2 \end{cases}$$

其中  $a_1, b_1, c_1, a_2, b_2, c_2$  均为复数. 根据代数可知, 当  $|a_1 b_2 - a_2 b_1| \neq 0$  时, 方程组的解存在, 且

$$z_1 = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}, \quad z_2 = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

为了标志解存在与否的情况, 程序中设置逻辑变量 TF, 当解存在时, TF 置 .TRUE. (真), 当解不存在时, TF 置 .FALSE. (假) 且  $z_1$  与  $z_2$  赋零值.

下面给出解此方程组的通用子程序, 以及引用此子程序解几组上述形式的不同方程组的完整源程序.

子程序:

```

SUBROUTINE CLEAN2(A1,B1,C1,A2,B2,C2,Z1,Z2,TF)
COMPLEX A1,B1,C1,A2,B2,C2,Z1,Z2,D
LOGICAL TF
D = A1 * B2 - A2 * B1
IF(ABS(D) .GT. 1E - 6) THEN
    TF = .TRUE.
    Z1 = ((1 * B2 - (2 * B1))/D
    Z2 = (A1 * C2 - A2 * C1)/D
ELSE
    TF = .FALSE.
    Z1 = 0
    Z2 = 0
END IF
RETURN
END

```

解三组不同方程组的源程序如下所示(其中左端加程序行号).

```

0210 * PROGRAM SOLVER
0220 LOGICAL TF
0230 COMPLEX A1, B1, C1, A2, B2, C2, Z1, Z2
0240 READ *, N

```

```

0250 DO 10 I = 1, N
0260 READ *, A1, B1, C1, A2, B2, C2
0270 WRITE (10, '( " * COEFFICIENTS: " )')
0280 WRITE(10, '(6F12.6/6F12.6)') A1, B1, C1, A2, B2, C2
0300 CALL CLINE2 (A1,B1,C1,A2,B2,C2,Z1,Z2,TF)
0310 IF (TF) THEN
0320 WRITE (10, '( " * SOLUTION : " )')
0330 WRITE (10, '(4F12.6//)') Z1, Z2
0340 ELSE
0350 WRITE (10, '( " * NO UNIOUE SOLUTION "/" )')
0360 END IF
0370 10 CONTINUE
0380 END
0390C
0400 SUBROUTINE CLINE2 (A1,B1,C1,A2,B2,C2,Z1,Z2,TF)
0410 COMPLEX A1, B1, C1, A2, B2, C2, Z1, Z2, D
0420 LOGICAL TF
0430 D = A1*B2 - A2*B1
0440 IF (ABS(D) .GT. 1E-5) THEN
0450 TF = .TRUE.
0460 Z1 = (C1*B2 - C2*B1)/D
0470 Z2 = (A1*C2 - A2*C1)/D
0480 ELSE
0490 TF = .FALSE.
0500 Z1 = 0
0510 Z2 = 0
0520 END IF
0530 END

```

假设为程序准备的输入数据的记录如下:

```

3
(1.0,1.0),(1.0,-1.0),(1.0,-1.0),(2.0,1.0),(3.0,7.0),(7.0,6.0)
(3.0,4.0),(6.0,3.0),(1.0,12.0),(4.0,-3.0),(0.0,1.0),(3.0,8.0)
(1.0,-1.0),(3.0,-1.0),(7.0,2.0),(-0.8,1.4),(2.0,3.0),(5.0,4.0)

```

则输出结果如下.

```

* COEFFICIENTS:
  1.000000    -1.000000    4.000000    7.000000    39.000000    16.000000
  2.000000     5.000000     3.000000    -2.000000    -2.000000    35.000000
* SOLUTION:

```

```

      7.522777      4.906725      3.639943      -1.715835
* COEFFICIENTS:
      3.000000      4.000000      5.000000      -3.000000      1.000000      12.000000
      4.000000      -3.000000      0.000000      1.000000      3.000000      8.000000
* SOLUTION:
      -0.157241      1.726897      0.620640      1.551724
* COEFFICIENTS:
      1.000000      1.000000      3.000000      -1.000000      7.000000      2.000000
      -0.800000      1.400000      2.000000      3.000000      3.000000      3.000000
* NO UNIQUE SOLUTION

```

## 2. 大批数据处理与数据文件的使用

FORTRAN 77 强有力的输入/输出功能特别显示在处理大批数据场合，大批数据的处理与数据文件的使用密切相关。

磁带最适合于存储具有不同长度记录的顺序存取文件，但注意下面事实是非常重要的，这就是许多计算机系统把数据写到磁带或从磁带读入数据，往往与 FORTRAN 77 程序里出现的读/写操作并不一样，原因是，支持计算机的操作系统总是企图最佳地利用磁带，它可能把两个或更多个记录写到一个磁带块上，不过，程序员是难以觉察这种现象的，也并不影响程序的设计。

磁盘适合于存储直接存取文件，也适合于存储顺序存取文件，通常磁盘被分成大小固定的区段，数据就被存储在磁盘上直接可编地址的区段上，虽然直接存取文件的记录的大小是固定的，但一个操作往往能读/写多个区段，故记录的长度应该是区段长度的倍数，与磁带的情况类似，操作系统通常对记录进行成块处理。

数据文件的使用实际包括文件的建立、检索、转存和修改等几方面，下面将通过简单例子来说明这些方面的程序设计方法。

建立数据文件的过程是，从卡片阅读机或终端键盘将记录输入内存，再从内存将记录写入文件，对于顺序存取文件最后写入一个文件结束记录。

**例 5** 建立一个机器测试数据文件 MACH11，采用有格式顺序存取连接方式，每个记录(为简单计)假设包括以下五个数据项：

| 机器序号<br>MNO<br>(整型) | 机器名称<br>NAME<br>(字符型) | 参数 1<br>MP1<br>(实型) | 参数 2<br>MP2<br>(实型) | 参数 3<br>MP3<br>(实型) |
|---------------------|-----------------------|---------------------|---------------------|---------------------|
| 1                   | 4 5                   | 14 15               | 20 21               | 26 27               |
|                     |                       |                     |                     | 32                  |

FORTRAN 77 程序：

```

*      PROGRAM AAA
      INTEGER U
      CHARACTER*10 NAME
      REAL MP1, MP2, MP3
10  FORMAT (I4,A,3F6.2)
      READ *, U
      OPEN(UNIT = U, FILE = 'MACH11', STATUS = 'NEW',
+ ACCESS = 'SEQUENTIAL', FORM = 'FORMATTED')

```

```

      READ *, MNO, NAME, MP1, MP2, MP3
20  IF (NAME .NE. 'EOF') THEN
      WRITE(UNIT = U, FMT = 10) MNO, NAME, MP1, MP2, MP3
      READ *, MNO, NAME, MP1, MP2, MP3
      GO TO 20
    END IF
  END FILE U
  CLOSE U
END

```

注：最后一个有效记录输入完毕，再输入一个记录：

```
0, 'EOF', 0.00, 0.00, 0.00
```

作为文件结束记录。

**例 6** 建立内容同 MACHI1，但要求用格式直接存取，且以机器序号作为文件记录号的文件 MACHI2。

FORTRAN 77 程序：

```

*      PROGRAM BBB
      INTEGER U
      CHARACTER *10 NAME
      REAL MP1, MP2, MP3
10  FORMAT (A,3F6.2)
      READ *, U
      OPEN (UNIT = U, FILE = 'MACHI2', STATUS = 'NEW',
+ ACCESS = 'DIRECT', FORM = 'FORMATTED', RECL = 28)
      READ *, MNO, NAME, MP1, MP2, MP3
20  IF (MNO .GT. 0) THEN
      WRITE(UNIT = U, FMT = 10, REC=MNO) NAME, MP1, MP2, MP3
      READ *, MNO, NAME, MP1, MP2, MP3
      GO TO 20
    END IF
  CLOSE U
END

```

注：因为机器序号已作为记录号写入文件，故 OPEN 语句的说明符表中记录长度为 28；序号作为记录号出现在 WRITE 语句中，WRITE 语句的执行将按记录号写入记录；在有效记录写完之后，输入 0 序号标志记录已输完。

使用数据文件的过程是，将记录从文件读入内存，必要时进行加工处理，再从内存传输到终端显示器或行打印机。对于按顺序存取连接的文件，可以按记录顺序逐个读入内存，并由文件结束记录判定读入结束。

**例 7** 把已建立的有格式顺序存取文件 MACHI1 制表输出，并求出其中三个参数

的平均值。

```
*      PROGRAM CCC
      INTEGER U
      CHARACTER*10 NAME
      REAL MP1, MP2, MP3
10  FORMAT (I4, A, 3F6.2)
20  FORMAT ('1', 'RECNO', T13, 'MNO', T23, 'NAME',
+ T34, 'MP1', T44, 'MP2', T54, 'MP3', T64, 'AVERA')
30  FORMAT (1X, I5, T12, I4, T20, A,
+ T32, F6.2, T42, F6.2, T52, F6.2, T63, F6.2)
      READ *, U
      OPEN (UNIT = U, FILE = 'MACH11', STATUS = 'OLD',
+ ACCESS = 'SEQUENTIAL', FORM = 'FORMATTED')
      PRINT 20
      N = 0
40  READ (UNIT = U, FMT = 10, END = 50) MNO, NAME,
+ MP1, MP2, MP3
      N = N + 1
      AVERA = (MP1 + MP2 + MP3)/3
      PRINT 30, N, MNO, NAME, MP1, MP2, MP3, AVERA
      GO TO 40
50  PRINT *, 'FILE DISPLAY COMPLETED'
      REWIND U
      CLOSE (U, STATUS = 'KEEP')
      END
```

以上几个例子说明了对文件进行存取和进行简单计算的程序设计方法。在大批数据处理中,往往还要对数据进行更动(或叫修改)。

通常,把包含基本信息的记录集中在一起,组织成所谓主文件,而把频繁变动的信息集中在一起,组织成所谓处理文件。然后用处理文件定期修改主文件。

**例8** 假设在银行存户的帐目处理中,已在磁盘上建立主文件 ABASE1,其中每个记录包括:帐户号(整型数)、帐户姓名(30个字符)、存款余额(实型数)和允许透支数(实型数)四项。又已在磁带上建立了处理文件 TRANS,其中每个记录包含对应上述某些帐户的帐号(整型数)和对帐目的实际修改值(实型数,对于存款,该值为正,对于取款,该值为负)。另外,假定两个文件的记录号按帐号升序排列,且两个文件均是顺序无格式存取,两者所连接部件分别为12和27。又后者中的帐户是唯一的且必对应于前者中的某一个。

现在,我们来设计如下的算法及其程序:在磁盘(部件13)上建立经过TRANS修改的ABASE1的更新文件ABASE2,并在指定部件上打印输出被更新且超过透支数的那些记录。

**【算法】**

```

读处理文件 TRANS 的一个记录
loop while 不是 TRANS 的结束文件记录 do
    {读主文件 ABASE1 的一个记录
    loop while 主帐号≠处理帐号 do
        {主记录写到 ABASE2 读主文件 ABASE1 的一个记录}
    按对应项更新余额
    if 允许的透支数已超过, then 打印更新的记录
    被更新的记录写到 ABASE2
    读处理文件 TRANS 的一个记录}
读主文件 ABASE1 的一个记录
loop while 不是 ABASE1 的结束文件记录 do
    {主记录写到 ABASE2
    读主文件 ABASE1 的一个记录}

```

程序变量说明:

```

MANO——主帐号
TRNO——处理帐号
NAME——帐户姓名
BAL  ——余额数
OVR  ——透支限(数)
CHA  ——支出数

```

程序清单:

```

*      PROGRAM DDD
      INTERGER MANO, TRNO, NU1, NU2, NU3
      REAL BAL, OVR, CHA
      CHARACTER *30 NAME
      CHARACTER*6 MASF, TRAF, NEWF
      PARAMETER(MASF='ABASE1',TRAF='TRANS', NEWF='ABASE2',
+ NU1=12, NU2=27, NU3=13)
      OPEN (UN1, FILE = MASF, FORM = 'UNFORMATTED', STATUS =
+ 'OLD')
      OPEN (UN2,FILE=TRAF,FORM='UNFORMATTED',STATUS='OLD')
      OPEN(UN3, FILE = NEWF, FORM = 'UNFORMATTED', STATUS =
+ 'NEW')
      PRINT 100
100  FORMAT ('1 ACC NO', 3X, 'NAME', 30X, 'BALANCE', 4X,
+ 'OVERDRAFT')
*-----

```

```

200 READ(NU2, END = 400) TRNO, CHA
* -----
      READ (NU1, END = 600) MANO, BAL, OVR
300 IF (MANO .NE. TRNO) THEN
      WRITE (NU3) MANO, NAME, BAL, OVR
      READ (NU1, END = 600) MANO, NAME, BAL, OVR
      GO TO 300
END IF
* -----
      BAL = BAL + CHA
      IF (BAL .LT. OVR) THEN
        PRINT '(“_”, I7, 3X, A30, 2 (3X, F9.2)', MANO, NAME,
          BAL, OVR
      END IF
      WRITE (UN3) MANO, NAME, BAL, OVR
      GO TO 200
* -----
400 READ (NU1, END = 500) MANO, NAME, BAL, OVR
* -----
      WRITE (UN3) MANO, NAME, BAL, OVR
      GO TO 400
* -----
500 ENDFILE (UN3)
      CLOSE (UN3)
      CLOSE (UN1)
      REWIND (UN2)
      CLOSE (UN2)
      STOP
* -----
600 PRINT '(11 "UNEXPECTED EOF DETECTED IN FILE", A6)', MASF
      END

```

## 习 题 四

1. 请对你正在使用的 FORTRAN IV 或/和 FORTRAN 77, 查清下述内容:

- (1) 正确的整数值范围;
- (2) 正确的实数值范围;
- (3) 正确的双精度范围;
- (4) 实算术运算的精度;
- (5) 双精度算术运算精度;



(6) 字符常数的长度界限(如果有的话)。

2. 在下列字符类型语句中,所命名的字符变量的长度是多少:

1) CHARACTER L\*2,M\*6,N\*4

2) CHARACTER\*8 ANAME,BNAME,X\*6,Y\*6,VAL

3) CHARACTER CHAR,C1,C2,P\*6,Q\*8,CNEW

3. 假设字符变量 SUBSO 的值为

'THE EARTH'IS ROUND.'

试求出 SUBSO(1:3),SUBSO(4:8),SUBSO(11:12),SUBSO(:12),SUBSO(16:),SUBSO(:) 之值。

4. 说明下列 FORTRAN 77 语句的作用:

```
CHARACTER*12 PW1*8,PW2*(*),X,Y,NAME*6,
```

```
+ A*4,B*5,C(8)*30
```

```
PARAMETER (PW1 = 'SONNET',PW2 = 'CARSON')
```

```
DATA X,Y/'CARPET TILE','LINOLEUM'/
```

```
NAME = 'ALBERT'
```

```
A = X(8:11)
```

```
B = NAME(1:2)//NAME(4:6)
```

```
C(1) = X(:11)//'S'/'ARE IN FASHION'
```

```
C(2) = NAME/'/'/'PW2'/'IS'/'C(1)(17:20)'//'BED'
```

```
I = INDEX(C(2),PW1(1:3))
```

5. 请把你的名字(汉语拼音)用下述语句告诉计算机:

(1) 赋值语句;

(2) 参数语句;

(3) 数据语句;

(4) 直接用字符型说明语句;

(5) 表格格式输入语句;

(6) 有格式输入语句。

6. 12 个温度读数已存入变量 TEMP1,TEMP2,...,TEMP12. 按下列每一种方式写出在指定输出部件上打印这些温度的 PRINT 语句格式说明:

(1) 印成一行;

(2) 印成一列;

(3) 印成四行,每一行三个读数为—组。

7. 写出下列判定结构的 FORTRAN 77 语句:

(1) if 偿还期限 < 一年 then 置利率为 20%

(2) if 收入数 ≤ 800 then 税款为 0

else 收入超过 800 纳税 20%

8. 考察下列程序:

```
IF (YEAR .EQ. 2000) THEN
```

```
IF (MONTH .NE. 2) THEN
```

```
PRINT *'MONTH NOT FEBRUARY'
```

```
ELSE IF (DAY .EQ. 29) THEN
```

```
MONTH = 3
```

```
DAY = 1
```

```

ELSE
DAY = DAY + 1
END IF
ELSE
PRINT * 'YEAR NOT 200'
END IF

```

- (1) 指出所有的 IF 块, ELSE IF 块和 ELSE 块.
- (2) 写出对于关系表达式的值的全部组合要执行的哪些语句.
- (3) 用直观易读的布局重写上述语句.

9. 求出下列每一个 DO 语句的重复执行次数:

- (1) DO 10 I = 1, 30
- (2) DO 20 J = 64, 1, -7
- (3) DO 30 K = 1 - 10, 2 \* J + 15, M \* \* 2

10. 假定 *fs* 是正确的格式说明, 下列输入/输出语句哪些是错误的:

- (1) READ (9, 'fs') I, J, X, Y
- (2) READ ('fs') A, B
- (3) READ( 10, 'fc') F, G, M-N
- (4) READ(18, fs) W, Z
- (5) READ 'fs', U, V
- (6) WRITE (\*, 'fs') A, B, C
- (7) WRITE (6, 'fs') I, J, 2 \* I + J, A - B, SQRT(D)
- (8) WRITE ('fs') M + N, F \* G + cos(Z + 3 \* Y)
- (9) WRITE 'fs', P, Q, R
- (10) PRINT 'fs'

11. 在连接到指定输入部件上的一个记录包含:

\_\_\_\_8371216.305\_\_\_\_967\_\_\_\_203.1897

在读入这个记录的下列 READ 语句中输入表的变量赋与什么值:

- (1) READ '(14,I3,F5.2,I1,I5,F10.4)' I, J, X, K, L, Y
- (2) READ '(3I3,F4.3,2I5,F4.3,I1)' I, J, K, F, J1, IA, HA, NU

12. 假定

I = -10, J = 5284, X = -2.5, Y = 49.887, Z = 7859.3751821

准确地写出用下列 PRINT 语句在指定输出部件上打印的结果:

- (1) PRINT '(2I6/2F7.2/F12.5)' I, J, X, Y, Z
- (2) PRINT('"THE NUMBER OF APPLES IN BARREL"', I5,  
+ "IS", I2/"THE AVERAGE PER BARREL IS",  
+ F6.1, "WITH STANDARD DEVIATION", F8.3) J, I, X, Y

13. 解释下面语句的意义:

- (1) READ(50, FMT = '(14,I5,I11)' I, J, K
- (2) READ(50, '(14,I5,I11)', IOSTAT = INFAIL) I, J, K
- (3) WRITE(UNIT = 75, FMT = 11, ERR = 50) R, S
- (4) WRITE (75, \*) A, B, C
- (5) READ (\*, 25, ERR = 40, END = 17) L, M, N
- (6) READ (30) X, Y, Z

- (7) WRITE(FMT = 29,UNIT = 7,END = 12)I,L,N  
 (8) WRITE (25,ERR = 4) (A(I),I = 1,5000)  
 (9) WRITE (UNIT = 25,FMT = '(" ",2I6,I8)')K,M,N  
 (10) WRITE (ERR = 20,END = 10,FMT = 55, IOSTAT = J,  
 + UNIT = 19) (X(I),I = 1,15)

14. 指出下列 OPEN 语句的错误:

- (1) OPEN(FILE = 'FILE1',STATUS = 'OLD')  
 (2) OPEN(FILE = 'STUDEN',STATUS = 'NEW',5)  
 (3) OPEN(32,FILE = 'JOBS',ACCESS = 'DIRECT')  
 (4) OPEN(24,FILE = 'UUU',RECL = 25)  
 (5) OPEN(12,FILE = 'TEMP',STATUS = 'SCRATCH')  
 (6) OPEN(65,FILE = 'CLASS',FORM = 'UNFORMATTED',  
 + RLANK = 'ZERO')

15. 考察下列程序片段:

- (1)  
 READ(\*,100)X,Y,Z  
 PRINT 200,X,Y,Z  
 100 FORMAT(E1.1,E9.3,E10.3E3)  
 200 FORMAT (1X,E1.1,E9.3,E10.3E3)

设当前输入记录为

1\_123\_F - 02\_ - 12.3\_ + 02

指出执行上述语句后 X,Y,Z 之值和打印结果.

- (2)  
 LOGICAL LOG  
 READ (5,100)X,LOG,INTEG  
 100 FORMAT (3G9.3)  
 WRITE(\*,'(1X,G9.2,G2.0,G10.0)')(X,LOG,INTEG

设当前输入记录

-- 1234E - 03\_ - TFILLER\_12345678

指出执行上述语句后 X, LOG, INTEG 之值和打印结果.

- (3)  
 COMPLEX C/(7.8, - 14.2)/  
 WRITE(6,'(1X,2F6.1)')C

指出执行上述语句后打印结果.

- (4)  
 LOGICAL LOG1, LOG2  
 CHARACTER\*7 A,B  
 A = '(L5,L2)'  
 READ (5,A) LOG1, LOG2  
 WRITE(\*,A) LOG1,LOG2

设当前输入记录为

.TRUE. \_F

指出执行上述语句后 LOG1, LOG2 之值和打印结果.

```
CHARACTER*2A(3)
READ(5,'(3A2)')A
WRITE(6,100)(A(J),J=3,1,-1)
FORMAT(1X,A2)
```

123456

(6)

```
WRITE (6,100)23.8,391.3,81,345
100 FORMAT(1X,SP,2F5.1,2(13,SS))
```

16. 以自由格式输入一组考生的考试成绩,其中每个输入记录包含五个整数,依次是考生的准考证号码,数学、物理、计算机科学和英语的分数。最后的输入记录包含-1, -1, -1, -1, -1, 作为数据结束标志。现在要求输出满足下列所有条件的考生的准考证号码和各科成绩:

- (1) 至少有一科成绩超过 90 分;
- (2) 没有任一科成绩低于 60 分;
- (3) 四科的平均成绩超过 75 分.

17. 编写程序计算长方体对角线长度 DIA

$$DIA = \sqrt{A^2 + B^2 + C^2}$$

其中边长  $A = 3$ ,  $B = 4$ ,  $C = 5$ 。要求用表控输入  $A, B, C$  之值;用有格式输出  $A, B, C$  DIA 之值;用表控输出打印表头  $A, B, C, DIA$  字符。

$$DO_{30} = 1,5$$

DO 10 J = 1,5

DO 20  $K = 1,5$ 
$$V = I * J * K$$

PRINT \*, V

10 CONTINUE

20 CONTINUE

30 CONTINUE

19. 编写产生下列两个图案的程序:

```

*               *
* *           * *
* * *       * * *
* * * *   * * * *
* * * * * * * * *

```

(b)

20. 分别编写下列两种要求的程序:

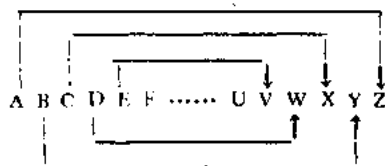
- (1) 输入一字符串包含数字 1,2,3,4. 要求能够将 1 翻译成 A, 2 翻译成 B, 3 翻译成 C, 4 翻译成

D. 例如

输入 1244, 输出 ABDD;

输入 4332, 输出 DCCB; 等等.

(2) 能够作这样的翻译: A 译成 Z, B 译成 Y, C 译成 X, ... 等等, 例如如下所示



例如, 单词 BAD 将译成 YZW. 这个程序以单词作为输入, 然后打印译出后的结果.

21. 计算下列级数之和

$$1 + \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \frac{1}{3 \times 4} + \cdots + \frac{1}{N \times (N+1)} + \cdots$$

所取累加项数为直至最后一项的值小于  $10^{-3}$ , 并且直至第 20 项时, 第 20 项的值还不小于  $10^{-3}$ , 也不再累加下去.

22. 已知以自由格式输入的数据是关于一个整数  $n$  和两个向量

$$x = [x_1, x_2, \dots, x_n]^T$$

$$y = [y_1, y_2, \dots, y_n]^T$$

的各元素的值. 写一个 FORTRAN 77 程序计算和打印内积

$$(x, y) = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

其中假设  $n \leq 100$ .

23. 按照 4-6 节例 5 中提出的机器测试数据的记录形式, 编写下列程序:

(1) 建立一个数据文件 MACH14, 采用无格式顺序存取方式.

(2) 按照序号输入顺序, 把已建立的格式直接存取文件的全部记录制表输出.

24. 编写计算下列各题的辅程序:

$$(1) u = \sum_{k=1}^N a_k x_k y_k.$$

(2)  $A, B \in R^{n \times n}$ , 计算  $A + B$ .

(3)  $x \in R^n, A \in R^{n \times n}$ , 求  $Ax$  和  $x^T A$ .

$$(4) S(N) = \sum_{k=1}^{N^2} \frac{N}{N^2 + k^2}$$

25. 利用上题编写的辅程序, 写出计算下列各题的完整的 FORTRAN 77 程序:

(1) 已知

$$a = [1.3, 0.25, 3.0, +.77, 5.01]^T$$

$$b = [0.14, 0.35, 9.11, 1.05, 3.04]^T$$

$$c = [10.5, 28.37, 30.05, 41.4, 50.88]^T$$

$$\text{求 } R = \sum_{k=1}^5 a_k b_k c_k.$$

(2) 设

$$C = \begin{bmatrix} 0.03 & 2.0 & 0.0 & 1.0 \\ 0.0 & 1.4 & 4.0 & 6.0 \\ 3.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}, \quad D = \begin{bmatrix} 1.0 & 0.0 & 5.0 & 1.0 \\ 3.0 & 0.0 & 2.0 & 1.0 \\ 0.0 & 9.0 & 0.4 & 0.3 \end{bmatrix}$$

求  $C + D$ .

(3) 设

$$u = \begin{bmatrix} 1.47 \\ 3.59 \\ -7.8 \\ 0.074 \end{bmatrix} \quad Q = \begin{bmatrix} -0.177 & 89.01 & -1.119 & 10^2 \\ 1.23 & 4.91 & 10 & 10 \\ 0.03 & 3.98 & 5.21 & 10^2 \\ -3.14 & -6.16 & 3.14^2 & 2.0 \end{bmatrix}$$

求  $Qu$  和  $u^T Q$ .

(4) 对指定的正整数  $n$ , 计算

$$S(n) = \sum_{k=1}^{n^2} \frac{n}{n^2 + k^2}$$

26. 用简单的迭代法(或称逐次逼近法)求方程

$$x^2 - \ln x - 1.5 = 0$$

的根. 方法如下: 将方程变形为

$$x = \sqrt{\ln x + 1.5}$$

并令  $X = F(X)$

先假定原方程的近似根为  $X_1$ ;

代入  $F(X)$  得  $F(X_1)$ ;

因为  $X_1$  是近似根, 故  $X_1 \neq F(X_1)$ ;

再令近似根  $X_2 = F(X_1)$ ;

再代入  $F(X)$  得  $F(X_2)$ ;

再令近似根  $X_3 = F(X_2)$ ;

$\vdots$

这样迭代  $N$  次后有  $X_N = F(X_{N-1})$ .

若有  $|X_N - X_{N-1}| < \varepsilon$

即前后两次根的差小于精度  $\varepsilon$ , 则认为  $X_N$  就是原方程的近似根.

若对  $F(X)$  迭代计算  $N$  次后仍不能满足精度, 则认为  $X_1, X_2, \dots, X_N$  发散, 方程无根.

现设  $\varepsilon = 0.001, X_1 = 2, N = 50$ , 试编 FORTRAN 77 程序.

## 附录 A FORTRAN 77 内部函数表

| 内部函数 | 定 义             | 属 名  | 专用名                              | 变元个数 | 类 型                          |                            |
|------|-----------------|------|----------------------------------|------|------------------------------|----------------------------|
|      |                 |      |                                  |      | 变 元                          | 函 数                        |
| 类型转换 | 转换为整型<br>[见注 2] | INT  | —<br>INT<br>IFIX<br>IDINT<br>—   | 1    | 实型<br>实型<br>实型<br>双精度型<br>复型 | 整型<br>整型<br>整型<br>整型<br>整型 |
|      | 转换为实型<br>[见注 3] | REAL | REAL<br>FLOAT<br>—<br>SINGL<br>— | 1    | 整型<br>整型<br>实型<br>双精度型<br>复型 | 实型<br>实型<br>实型<br>实型<br>实型 |

续附录(1)

| 内部函数  | 定 义                                                          | 属 名   | 专用名                         | 变元个数     | 类 型                    |                              |
|-------|--------------------------------------------------------------|-------|-----------------------------|----------|------------------------|------------------------------|
|       |                                                              |       |                             |          | 变 元                    | 函 数                          |
| 类型转换  | 转换为双精度型<br>「见注4」                                             | DBLE  | ——<br>——<br>——<br>——        | 1        | 整型<br>实型<br>双精度型<br>复型 | 双精度型<br>双精度型<br>双精度型<br>双精度型 |
|       | 转换为复型<br>「见注5」                                               | CMPLX | ——<br>——<br>——              | 1 或 2    | 整型<br>实型<br>双精度型<br>复型 | 复型<br>复型<br>复型<br>复型         |
|       | 转换为整型<br>「见注6」                                               | ——    | ICHAR                       | 1        | 字符型                    | 整型                           |
|       | 转换为字符型<br>「见注6」                                              | ——    | CHAR                        | 1        | 整型                     | 字符型                          |
| 截 断   | INT (x)<br>「见注2」                                             | AINT  | AINT<br>DINT                | 1        | 实型<br>双精度型             | 实型<br>双精度型                   |
| 最近整数  | INT (x+.5)<br>当 $x \geq 0$<br>INT (x-.5)<br>当 $x < 0$        | ANINT | ANINT<br>DNINT              | 1        | 实型<br>双精度型             | 实型<br>双精度型                   |
| 最近整数  | INT (x+.5)<br>当 $x \geq 0$<br>INT (x-.5)<br>当 $x < 0$        | NINT  | NINT<br>1DNINT              | 1        | 实型<br>双精度型             | 整型<br>整型                     |
| 绝对值   | $ x $ 或者<br>$\sqrt{(x_r)^2 + (x_i)^2}$<br>当 $x$ 为复型<br>「见注7」 | ABS   | IABS<br>ABS<br>DABS<br>CABS | 1        | 整型<br>实型<br>双精度型<br>复型 | 整型<br>实型<br>双精度型<br>实型       |
| 求 余   | $x_1 - \text{INT}$<br>$(x_1/x_2) \times x_2$<br>「见注2」        | MOD   | MOD<br>AMOD<br>DMOD         | 2        | 整型<br>实型<br>双精度型       | 整型<br>实型<br>双精度型             |
| 符号传递  | $ x_1 $ 当 $x_1 \geq 0$<br>$- x_1 $<br>当 $x_1 < 0$            | SIGN  | ISIGN<br>SIGN<br>DSIGN      | 2        | 整型<br>实型<br>双精度型       | 整型<br>实型<br>双精度型             |
| 正 差   | $x_1 - x_2$ 当 $x_1 > x_2$<br>0 当 $x_1 \leq x_2$              | DIM   | 1DIM<br>DIM<br>DDIM         | 2        | 整型<br>实型<br>双精度型       | 整型<br>实型<br>双精度型             |
| 双精度乘积 | $x_1 \times x_2$                                             | ——    | DPROD                       | 2        | 实型                     | 双精度型                         |
| 选最大值  | $\max(x_1, x_2, \dots)$                                      | MAX   | MAX0<br>AMAX1<br>1DMAX1     | $\geq 2$ | 整型<br>实型<br>双精度型       | 整型<br>实型<br>双精度型             |
|       |                                                              | ——    | AMAX0<br>MAX1               |          | 整型<br>实型               | 实型<br>整型                     |

法附录(2)

| 内部函数      | 定 义                                        | 属 名   | 专用名                    | 变元个数     | 类 型              |                  |
|-----------|--------------------------------------------|-------|------------------------|----------|------------------|------------------|
|           |                                            |       |                        |          | 变 元              | 函 数              |
| 选最小值      | $\min(x_1, x_2, \dots)$                    | MIN   | MIN0<br>AMINI<br>DMINI | $\geq 2$ | 整型<br>实型<br>双精度型 | 整型<br>实型<br>双精度型 |
|           |                                            |       | AMIN0<br>MINI          |          | 带型<br>实型         | 实型<br>整型         |
| 长 度       | 字符串长度                                      | ---   | LEN                    | 1        | 字符型              | 带型               |
| 子串下标      | 子字符串 $x_2$ 在字符串 $x_1$ 中的位置<br>[见注11]       | ---   | INDEX                  | 2        | 字符型              | 整型               |
| 串 比 较     | $x_1 \geq x_2$<br>[见注13]                   | ---   | LGE                    | 2        | 字符型              | 逻辑型              |
|           | $x_1 > x_2$<br>[见注13]                      | ---   | LGT                    | 2        | 字符型              | 逻辑型              |
|           | $x_1 \leq x_2$<br>[见注13]                   | ---   | LLE                    | 2        | 字符型              | 逻辑型              |
|           | $x_1 < x_2$<br>[见注13]                      | ---   | LLT                    | 2        | 字符型              | 逻辑型              |
| 取复型变元的虚部  | 取 $x_1 + ix_2$ 中的 $x_2$<br>[见注7]           | ---   | AIMAG                  | 1        | 复型               | 实型               |
| 求复型变元的共轭数 | 取 $x_1 + ix_2$ 的共轭复数 $x_1 - ix_2$<br>[见注7] | ---   | CONJG                  | 1        | 复型               | 复型               |
| 求平方根      | $\sqrt{x}$                                 | SQRT  | SQRT<br>DSQRT<br>CSQRT | 1        | 实型<br>双精度型<br>复型 | 实型<br>双精度型<br>复型 |
| 指 数       | $e^x$                                      | EXP   | EXP<br>DEXP<br>CEXP    | 1        | 实型<br>双精度型<br>复型 | 实型<br>双精度型<br>复型 |
| 自然对数      | $\log(x)$                                  | LOG   | ALOG<br>DLOG<br>CLOG   | 1        | 实型<br>双精度型<br>复型 | 实型<br>双精度型<br>复型 |
| 常用对数      | $\log_{10}(x)$                             | LOG10 | ALOG10<br>DLOG10       | 1        | 实型<br>双精度型       | 实型<br>双精度型       |
| 正 弦       | $\sin(x)$                                  | SIN   | SIN<br>DSIN<br>CSIN    | 1        | 实型<br>双精度型<br>复型 | 实型<br>双精度型<br>复型 |
| 余 弦       | $\cos(x)$                                  | COS   | COS<br>DCOS<br>CCOS    | 1        | 实型<br>双精度型<br>复型 | 实型<br>双精度型<br>复型 |



| 内部函数  | 定 义                | 属 名   | 专 用 名           | 变元个数 | 类 型        |            |
|-------|--------------------|-------|-----------------|------|------------|------------|
|       |                    |       |                 |      | 变 元        | 函 数        |
| 正 切   | $\tan(x)$          | TAN   | TAN<br>DTAN     | 1    | 实型<br>双精度型 | 实型<br>双精度型 |
| 反 正 弦 | $\arcsin(x)$       | ASIN  | ASIN<br>DASIN   | 1    | 实型<br>双精度型 | 实型<br>双精度型 |
| 反 余 弦 | $\arccos(x)$       | ACOS  | ACOS<br>DACOS   | 1    | 实型<br>双精度型 | 实型<br>双精度型 |
| 反 正 切 | $\arctan(x)$       | ATAN  | ATAN<br>DATAN   | 1    | 实型<br>双精度型 | 实型<br>双精度型 |
|       | $\arctan(x_1/x_2)$ | ATAN2 | ATAN2<br>DATAN2 | 2    | 实型<br>双精度型 | 实型<br>双精度型 |
| 双曲正弦  | $\sinh(x)$         | SINH  | SINH<br>DSINH   | 1    | 复型<br>双精度型 | 实型<br>双精度型 |
| 双曲余弦  | $\cosh(x)$         | COSH  | COSH<br>DCOSH   | 1    | 实型<br>双精度型 | 实型<br>双精度型 |
| 双曲正切  | $\tanh(x)$         | TANH  | TANH<br>DTANH   | 1    | 实型<br>双精度型 | 实型<br>双精度型 |

**注 1** 调用内部函数时,可使用属名或专用名。当使用属名调用内部函数时,除了类型转换、最近整数、求复型数的绝对值的内部函数外,函数值的类型与给出的实元类型相同。

当使用专用名时,变元的类型必须严格按表中所给出的对应规则,求得的函数值类型也如表中所示。

当要用一个内部函数名作为实元去调用外部过程时,只允许用其专用名。

**注 2** 对于整型  $x$ ,  $\text{INT}(x) = x$ ; 对于实型或双精度型  $x$ , 则:

当  $|x| < 1$  时,  $\text{INT}(x) = 0$ ;

如果  $|x| \geq 1$ ,  $\text{INT}(x)$  就是这样的整数,它的绝对值不超过  $x$  的绝对值的最大整数,它的符号和  $x$  的符号相同。如:

$$\text{INT}(7.4) = 7$$

$$\text{INT}(-6.7) = -6$$

对复型量  $x$ ,  $\text{INT}(x)$  是对  $x$  的实部应用上述规则得到的值。

对实型的  $x$ ,  $\text{FIX}(x)$  与  $\text{INT}(x)$  相同。

**注 3** 对实型的  $x$ ,  $\text{REAL}(x)$  即  $x$ 。对整型或双精度型的  $x$ ,  $\text{REAL}(x)$  的精度和  $x$  作为实数据时可包含的有效部分的精度一样。对复数型  $x$ ,  $\text{REAL}(x)$  是  $x$  的实部。

对整型的  $x$ ,  $\text{FLOAT}(x)$  和  $\text{REAL}(x)$  一样。

**注 4** 对双精度型的  $x$ ,  $\text{DBLE}(x)$  就是  $x$ 。对整型或实型的  $x$ ,  $\text{DBLE}(x)$  的精度和  $x$  作为一个双精度数据可包含的有效部分的精度一样。对复型的  $x$ ,  $\text{DBLE}(x)$  的精度和  $x$  的实部作为一个双精度数据可包含的有效部分的精度一样。

**注 5**  $\text{CMPLX}$  可以有一个或两个变元。有一个变元的情况,它可以是整型、实型、双精度型或复型。有两个变元的情况,这两个变元必须相同类型,可以是整型、实型或双精度型。

对复型的  $x$ ,  $\text{CMPLX}(x)$  就是  $x$ 。对整型、实型或双精度型的  $x$ ,  $\text{CMPLX}(x)$  即实部为  $\text{REAL}(x)$ , 虚部为零的复数值。

$\text{CMPLX}(x_1, x_2)$  是实部为  $\text{REAL}(x_1)$ , 虚部为  $\text{REAL}(x_2)$  的复数值。

**注6**  $\text{ICHAR}$  提供了从字符到整数的转换手段, 转换是根据处理系统的排序序列 (参见附录 D) 中的字符位置 (即排序序号) 进行的。在排序序列中第一个字符对应于位置, 最后一个字符对应于位置  $n-1$  其中  $n$  是排序序列中的字符数目。

$\text{ICHAR}(x)$  的值是范围  $0 \leq \text{ICHAR} \leq n-1$  内的整数, 其中  $x$  为长度是 1 的字符类型的一个变元。 $x$  的值必须是处理系统中能够表示的字符。该字符在排序序列中的位置即  $\text{ICHAR}$  的值。

对处理系统中能够表示的任意字符  $c_1$  和  $c_2$  当且仅当  $(\text{ICHAR}(c_1), \text{LE}, \text{ICHAR}(c_2))$  为真时,  $(c_1, \text{LE}, c_2)$  为真; 当且仅当  $(\text{ICHAR}(c_1), \text{EQ}, \text{ICHAR}(c_2))$  为真时,  $(c_1, \text{EQ}, c_2)$  为真。

$\text{CHAR}(i)$  所返回的值是处理系统排序序列中第  $i$  个位置上的字符。值为字符类型, 长度为 1。  $i$  必须是一整表达式, 其值必须处于下述范围:  $0 \leq i \leq n-1$ 。

$\text{ICHAR}(\text{CHAR}(i)) = i$  对  $0 \leq i \leq n-1$ 。

$\text{CHAR}(\text{ICHAR}(c)) = c$  对任一处理系统中能表示的字符  $c$ 。

**注7** 复数值用一对有序实数  $(x, y)$  表示, 其中:  $x$  是实部,  $y$  是虚部。

**注8** 所有角度都用弧度表示。

**注9** 复型函数的结果是其主值。

**注10** 内部函数调用中所有变元必须具有相同的类型。

**注11**  $\text{INDEX}(x_1, x_2)$  返回一个整数值, 它指出在字符串  $x_1$  中与串  $x_2$  一样的子串的开始位置。如  $x_2$  在  $x_1$  中不止出现一次, 则返回值为其第一次出现的开始位置。

若  $x_2$  不在  $x_1$  中出现, 则返回值为零。注意, 若  $\text{LEN}(x_1) < \text{LEN}(x_2)$ , 也返回零。例如, 设

```
A = 'ABCDEFG'
B = 'DEF'
C = 'ABCDEFGDEF'
```

则

$\text{INDEX}(A, B) = 4$

$\text{INDEX}(C, B) = 4$

$\text{INDEX}(A, C) = 0$

**注12**  $\text{LEX}$  函数的变元值, 在该函数调用执行时不一定是定义的。

**注13** 函数  $\text{LGE}$ ,  $\text{LGT}$ ,  $\text{LLE}$ ,  $\text{LLT}$  被用来按美国国家信息交换标准代码 (ASCII) 的顺序比较两个字符串。下面列出这四个函数的函数值为  $\text{.TRUE.}$  与  $\text{.FALSE.}$  的条件, 其中所指前后是在 (ASCII) 码排列顺序上的前后 (参见附录 D):

当  $x_1$  等于  $x_2$  或  $x_1$  跟在  $x_2$  之后时,  $\text{LGE}(x_1, x_2)$  为  $\text{.TRUE.}$ , 否则为  $\text{.FALSE.}$ 。

当  $x_1$  跟在  $x_2$  之后时,  $\text{LGT}(x_1, x_2)$  为  $\text{.TRUE.}$ , 否则为  $\text{.FALSE.}$ 。

当  $x_1$  等于  $x_2$  或  $x_1$  在  $x_2$  之前时,  $\text{LLE}(x_1, x_2)$  为  $\text{.TRUE.}$ , 否则为  $\text{.FALSE.}$ 。

当  $x_1$  在  $x_2$  之前时,  $\text{LLT}(x_1, x_2)$  为  $\text{.TRUE.}$ , 否则为  $\text{.FALSE.}$ 。

若  $\text{LGE}$ ,  $\text{LGT}$ ,  $\text{LLE}$  和  $\text{LLT}$  的运算对象是不等长的, 则要在长度短的那个运算对象右端补以空白, 使其长度与长度长的那个运算对象等长。

**注14** 各内部函数调用 (包括对用属名或专用名调用) 时, 变元与函数值的取值范围:

1) 求余: 当第二个变元的值为 0 时,  $\text{MOD}$   $\text{AMOD}$  及  $\text{DMOD}$  的结果无定义。

2) 符号传送: 如果  $\text{ISIGN}$ ,  $\text{SIGN}$  或  $\text{DSIGN}$  的第一个变元的值为 0, 则结果为 0, 它既非正也非负。

3) 平方根:  $\text{SQRT}$  及  $\text{DSQRT}$  的变元值必须大于等于零,  $\text{CSQRT}$  的结果是实部大于等于零的那个主值。当结果的实部为零, 其虚部应大于等于零。

4) 对数:  $\text{ALOG}$ ,  $\text{DLOG}$ ,  $\text{ALOG10}$  和  $\text{DLOG10}$  的变元值必须大于零,  $\text{CLOG}$  的变元值不能为  $(0, 0, \dots)$ ,  $\text{CLOG}$  结果的虚部范围为:  $-\pi < \text{虚部} \leq \pi$ 。仅当变元的实部小于零并且变元的虚部为零时, 结果的虚部为  $\pi$ 。

5) 正弦、余弦及正切: SIN, DSIN, COS, DCOS, TAN 及 DTAN 的变元的绝对值并不限制小于  $2\pi$ 。

6) 反正弦: ASIN 及 DASIN 的变元的绝对值必须小于或等于1;结果的范围是:  $0 \leq \text{结果} \leq \pi$ 。

7) 反余弦: ACOS 和 DACOS 的变元的绝对值必须小于等于1;结果的范围是:  $0 \leq \text{结果} \leq \pi$ 。

8) 反正切: ATAN 及 DATAN 的结果的范围是:  $-\frac{\pi}{2} \leq \text{结果} \leq \frac{\pi}{2}$ 。如果 ATAN2 或 DATAN2 的第一个变元的值为正,则结果为正。如果第一个变元的值为零,第二个变元为正,则结果为零。如果第一个变元值为零,而第二个变元为负,则结果为  $\pi$ 。如果第一个变元的值为负,则结果为负。若第二个变元的值为零,则结果的绝对值为  $\frac{\pi}{2}$ 。变元的值不允许均为零。ATAN2 及 DATAN2 的结果的范围是:  $-\pi < \text{结果} \leq \pi$ 。

## 附录 B 程序单位中注解行与语句的次序

FORTRAN 77 标准要求在程序单位中注解行和各种类型的语句,必须遵守一定的次序,这个次序如下表所示:

|        |                                              |                 |                       |
|--------|----------------------------------------------|-----------------|-----------------------|
| 注解行    | PROGRAM, FUNCTION, SUBROUTINE, BLOCK DATA 语句 |                 |                       |
|        | FORMAT 语句<br>与<br>ENTRY 语句                   | PARAMETER<br>语句 | IMPLICIT 语句<br>其它说明语句 |
|        |                                              | DATA 语句         | 语句函数语句<br>执行语句        |
| END 语句 |                                              |                 |                       |

说明:

- 1) 注解行可以在程序单位中 END 语句之前的任何地方出现(包括在第一个语句之前)。
- 2) 程序单位中的第一个语句,在主程序是 PROGRAM, 在辅程序或者是 FUNCTION, 或者是 SUBROUTINE, 或者是 BLOCK DATA。
- 3) 程序单位的最后一行必须是 END 语句。
- 4) FORMAT 语句可以出现在程序单位内的任何地方。
- 5) 所有说明语句(包括 PARAMETER 语句)必须放在所有 DATA 语句、语句函数语句和可执行语句之前。
- 6) 所有语句函数必须放在所有可执行语句之前。
- 7) DATA 语句必须出现在说明语句之后,但可以散置在语句函数和可执行语句中间。
- 8) 除了与 IMPLICIT 语句和其它说明语句交迭散置的 PARAMETER 语句外, IMPLICIT 语句先于所有其它说明语句。
- 9) 除了在块 IF 语句和与之相对应的 END IF 语句之间,或 DO 语句和 DO 循环的终结语句之间, ENTRY 语句可以出现在程序单位的其它任何地方。

## 附录 C FORTRAN 77 语句一览表

FORTRAN 77 中的每一个语句被分属于可执行语句或非执行语句。可执行语句指明动作,并构成执行序列。非执行语句指明数据的特性。排列和初值;包含编辑信息;指

明语句函数;程序单位分类等.

(1) 可执行语句

赋值语句:

算术赋值语句  
逻辑赋值语句  
字符赋值语句  
语句标号赋值语句

控制语句:

无条件 GO TO 语句  
计算 GO TO 语句  
赋值 GO TO 语句  
块 IF 语句  
ELSE IF 语句  
ELSE 语句  
END IF 语句  
逻辑 IF 语句  
算术 IF 语句  
DO 语句  
CONTINUE 语句  
STOP 语句  
PAUSE 语句  
END 语句  
CALL 语句  
RETURN 语句

输入/输出语句:

READ 语句  
WRITE 语句  
PRINT 语句  
OPEN 语句  
CLOSE 语句  
INQUIRE 语句  
BACKSPACE 语句  
ENDFILE 语句  
REWIND 语句

(2) 非执行语句

说明语句:

PARAMETER 语句  
DIMENSION 语句  
类型语句:

INTEGER  
 REAL  
 DOUBLE PRECISION  
 COMPLEX  
 LOGICAL  
 CHARACTER  
 EQUIVALENCE 语句  
 COMMON 语句  
 EXTERNAL 语句  
 INTRINSIC 语句  
 SAVE 语句  
 IMPLICIT 语句  
 程序单位语句:  
 PROGRAM 语句  
 FUNCTION 语句  
 SUBROUTINE 语句  
 ENTRY 语句  
 BLOCK DATA 语句  
 赋初值语句:  
 DATA 语句  
 格式语句:  
 FORMAT 语句  
 函数语句:  
 语句函数语句

## 附录 D FORTRAN 77 排序序列

在包含字符信息的许多问题中,需要把字符信息按某种方式排序。例如,利用字母顺序并假定空白符( )排在最前面的排序,就是常用的一种排序。用更准确的术语说,这里使用排序序列

$$\_ < A < B < C < \dots < X < Y < Z$$

来排序。

FORTRAN 77 标准不定义一个特定的排序序列,但要求 FORTRAN 77 任何实现的排序序列必须具有以下性质:

- 1)  $0 < 1 < 2 < \dots < 9$ ;
- 2)  $A < B < C < \dots < X < Y < Z$ ;
- 3)  $9 < A$  或者  $Z < 0$ ;
- 4)  $\_ < A$  且  $\_ < 0$

另外,在 FORTRAN 77 里还提供了四个串比较内部函数 LGE, LGT, LLE, LLT

(参见附录 A 及其注 13), 以便利用 ASCII 排序序列来比较字符串。

标准 FORTRAN 77 字符集的 ASCII 排序序列是:

|   |    |   |   |   |   |   |   |   |   |
|---|----|---|---|---|---|---|---|---|---|
| _ | \$ | . | ( | ) | * | + | , | - | / |
| 0 | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| = | A  | B | C | D | E | F | G | H | I |
| J | K  | L | M | N | O | P | Q | R | S |
| T | U  | V | W | X | Y | Z |   |   |   |

## 第5章 插值与拟合

### 5-1 引言

假定某物理过程遵从从一个未知的函数规律  $y = f(x)$ ; 又我们通过实验或其它手段获得这个物理过程的一组观测数据(或称离散样点):

$$(x_i, y_i) \quad i = 0, 1, 2, \dots, n \quad (5.1)$$

问题是, 要求根据这组数据来估计出未知函数  $y = f(x)$  的一个近似的、且比较简单的具体表达式  $y = F(x)$ .

可能有两种情形:

(1) 观测数据的误差比较小, 以致于我们可以把它作为准确值来处理, 即认为  $f(x_i) = y_i (i = 0, 1, 2, \dots, n)$ , 因而可以要求所估计出的函数  $F(x)$  通过这些样点, 即满足条件

$$F(x_i) = y_i, \quad i = 0, 1, 2, \dots, n \quad (5.2)$$

(2) 观测数据的误差比较大, 以致于我们不可能、也不必要把它作为准确值来处理, 因而我们只要求所估计出的函数  $F(x)$  尽可能地靠近这些样点就可以了, 比如在某种意义下总的偏差为最小.

与(1)相关的一类数学问题就是所谓插值问题, 它适宜于观测数据是精确的或可靠度较高的情况. 由于要满足条件(5.2), 故这时待求函数  $F(x)$  的待定参数的个数(自由度)必须与给定的条件(5.2)的个数相同. 插值问题的实际应用往往是从求得的函数表达式计算  $x_i$  以外的点的函数值, 或函数的微分或积分. 因此, 所求的函数自然是希望便于在计算机上计算的函数, 如多项式等.

与(2)相关的一类数学问题则称为拟合问题, 它适宜于观测数据本来就含有不可避免的误差(无妨称为噪音)的情况. 不要求所作函数  $F(x)$  严格地通过样点, 而只是尽可能地靠近样点, 这常常可能达到滤去噪音的目的. 相反, 强求所作的曲线通过样点, 反而有可能使曲线保留误差, 影响拟合的精度. 对这种情况, 我们总是尽量多提供一些样点, 因此拟合问题中样点的个数远多于待求函数的待定参数的个数.

一般地, 插值与拟合的理论属于数学中“函数构造论”或称“逼近论”这个分支. 近似函数  $F(x)$  称为逼近函数, 而称  $f(x)$  为被逼近函数.

本章讨论插值问题中的多项式插值、分段多项式插值、样条函数插值、差分插值和拟合问题中的最小二乘法.

### 5-2 多项式与分段多项式插值

**定义1** 已知定义在  $[a, b]$  上的某一函数  $y = f(x)$  在

$$a \leq x_0 < x_1 < \dots < x_n \leq b$$

处的值为

$$f_0, f_1, \dots, f_n$$

(必要时还引用导数值), 如果有另一函数  $F(x)$ , 满足条件

$$F(x_i) = f_i \quad i = 0, 1, \dots, n \quad (5.3)$$

则  $F(x)$  称为  $f(x)$  的插值函数;  $x_0, x_1, \dots, x_n$  称为插值节点;  $[a, b]$  称为插值区间;  $f(x) - F(x)$  称为插值余项.

特别, 若插值函数  $F(x)$  是次数不超过  $n$  的代数多项式

$$F(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (a_i \text{ 为实数}) \quad (5.4)$$

则  $F(x)$  称为插值多项式, 并称这种插值为多项式插值.

插值函数  $F(x)$  也可以是分段多项式(包括所谓样条函数)或有理函数等等. 这一节我们研究多项式插值中的两种基本形式: 拉格朗日插值和埃尔米特插值. 我们的任务就是根据已知条件, 构造所需的插值函数.

### 1. 拉格朗日插值

先讨论几种简单情形, 然后再推广到一般形式.

#### (1) 一点零次插值(水平插值)

已知一个样点  $(x_0, f_0)$ , 求作一个零次多项式(即常数, 也即水平线)  $F(x)$ , 使得

$$F(x_0) = f_0$$

显然, 所求的零次插值函数为

$$F(x) = f_0$$

如图 5-1 所示.

#### (2) 两点一次插值(线性插值)

已知两个样点  $(x_0, f_0), (x_1, f_1)$ , 求作一次多项式(直线)  $F(x)$  满足

$$F(x_0) = f_0, \quad F(x_1) = f_1$$

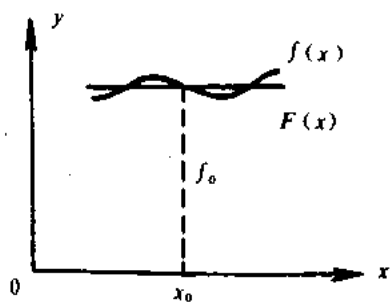


图 5-1

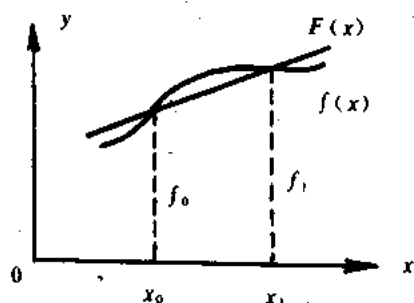


图 5-2

容易验证, 所求一次插值多项式(即过两点  $(x_0, f_0), (x_1, f_1)$  的直线)为

$$F(x) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1$$

或

$$F(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0} (x - x_0)$$

如图 5-2. 为了便于推广, 我们记



$$l_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad l_1(x) = \frac{x - x_0}{x_1 - x_0}$$

这是一次函数,且具有性质:

$$l_0(x_0) = 1, \quad l_0(x_1) = 0$$

$$l_1(x_0) = 0, \quad l_1(x_1) = 1$$

$l_0(x)$  与  $l_1(x)$  称为线性插值基函数. 于是线性插值函数可以表示为  $f_0, f_1$  与线性基函数的线性组合

$$F(x) = f_0 l_0(x) + f_1 l_1(x)$$

### (3) 三点二次插值(抛物线插值)

已知三个样点  $(x_0, f_0), (x_1, f_1), (x_2, f_2)$ , 求作一个二次多项式(抛物线)  $F(x)$ , 满足条件

$$F(x_0) = f_0, \quad F(x_1) = f_1, \quad F(x_2) = f_2$$

采用基函数方法,仿(2)作三个基函数(二次函数)

$$l_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \quad l_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)},$$

$$l_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

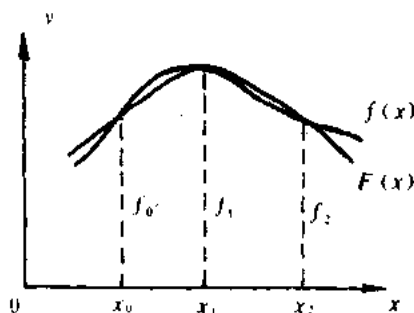


图 5-3

显然它们具有性质:

$$l_0(x_0) = 1, \quad l_0(x_1) = 0, \quad l_0(x_2) = 0$$

$$l_1(x_0) = 0, \quad l_1(x_1) = 1, \quad l_1(x_2) = 0$$

$$l_2(x_0) = 0, \quad l_2(x_1) = 0, \quad l_2(x_2) = 1$$

于是容易验证,满足条件的二次插值多项式(抛物线)为

$$F(x) = f_0 l_0(x) + f_1 l_1(x) + f_2 l_2(x)$$

如图 5-3 所示.

### (4) 一般多项式插值

现在推广到一般情形. 已知  $n+1$  个样点  $(x_i, f_i), i = 0, 1, 2, \dots, n$ , 求作  $n$  次多项式  $F(x)$ , 满足条件

$$F(x_i) = f_i, \quad i = 0, 1, \dots, n$$

仍仿上述基函数方法,作  $n+1$  个  $n$  次插值基函数( $n$  次函数)

$$l_i(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}$$

$$= \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}, \quad i = 0, 1, \dots, n$$

容易看出,它们具有性质:

$$l_0(x_0) = 1, \quad l_0(x_1) = 0, \quad \dots, \quad l_0(x_n) = 0$$

$$l_1(x_0) = 0, \quad l_1(x_1) = 1, \quad \dots, \quad l_1(x_n) = 0$$

$$\dots \dots \dots$$

$$l_n(x_0) = 0, \quad l_n(x_1) = 0, \quad \dots, \quad l_n(x_n) = 1$$

如果用所谓克罗内克尔 (Kronecker) 符号:

$$\delta_{ij} = \begin{cases} 1 & \text{当 } i = j \\ 0 & \text{当 } i \neq j \end{cases}$$

则基函数的性质可表示为

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{当 } i = j \\ 0 & \text{当 } i \neq j \end{cases}$$

于是可得  $n$  次插值多项式为

$$F(x) = \sum_{i=0}^n f_i l_i(x)$$

或写成便于在计算机上实现的紧凑格式

$$F(x) = \sum_{i=0}^n \left[ \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(x - x_j)}{(x_i - x_j)} \right] f_i$$

下面写出这个计算格式的 FORTRAN 子程序段。

在 FORTRAN IV 中, 下标值必须是大于零的整数, 因此下面程序中的下标均由 1 开始。

```

SUBROUTINE LAGRAN (X0, F0, N, X, F)
  DIMENSION X0(N), F0(N)
  F = 0.0
  DO 30 I = 1, N
    P = 1.0
    DO 20 J = 1, N
      IF (I - J) 10, 20, 10
10    P = P * (X - X0(J)) / (X0(I) - X0(J))
20    CONTINUE
30    F = F + P * F0(I)
  RETURN
END

```

为了便于理论分析, 有时还引用记号

$$\omega(x) = \prod_{j=0}^n (x - x_j)$$

并由对数求导法得

$$\omega'(x_i) = \prod_{\substack{j=0 \\ j \neq i}}^n (x_i - x_j)$$

把基函数表示成

$$l_i(x) = \frac{\omega(x)}{(x - x_i)\omega'(x_i)}, \quad i = 0, 1, \dots, n$$

于是  $n$  次插值多项式也可以表示成

$$F(x) = \sum_{i=0}^n \frac{\omega(x)}{(x-x_i)\omega'(x_i)} f_i$$

上述各种情形都是以函数值  $f_0, f_1, \dots, f_n$  为基础构成的插值, 通常称这种插值为**拉格朗日 (Lagrange) 插值**.

**例 1** 用线性插值与抛物线插值计算  $\sin 0.3367$  的值, 假定使用表距为 0.02 弧度的正弦表, 其中已知  $\sin 0.32 = 0.314567$ ,  $\sin 0.34 = 0.333487$ ,  $\sin 0.36 = 0.352274$ .

**解** 由题意取

$$(x_0, f_0) = (0.32, 0.314567)$$

$$(x_1, f_1) = (0.34, 0.333487)$$

$$(x_2, f_2) = (0.36, 0.352274)$$

用线性插值计算有

$$\begin{aligned} \sin 0.3367 &\approx F(0.3367) \\ &= 0.314567 + \frac{0.333487 - 0.314567}{0.34 - 0.32} \times (0.3367 - 0.32) \\ &= 0.330365 \end{aligned}$$

用抛物线插值计算有

$$\begin{aligned} \sin 0.3367 &\approx F(0.3367) \\ &= \frac{(0.3367 - 0.34)(0.3367 - 0.36)}{(0.32 - 0.34)(0.32 - 0.36)} \times 0.314567 + \\ &\quad + \frac{(0.3367 - 0.32)(0.3367 - 0.36)}{(0.34 - 0.32)(0.34 - 0.36)} \times 0.333487 + \\ &\quad + \frac{(0.3367 - 0.32)(0.3367 - 0.34)}{(0.36 - 0.32)(0.36 - 0.34)} \times 0.352274 \\ &= 0.330374 \end{aligned}$$

后面这个结果与六位有效数字的正弦函数表相符, 说明查表时用二次插值精度已相当高了. 若表距较小, 查表时用线性插值, 也可达到相当的精度.

以上我们根据已知条件, 把插值函数具体构造出来了. 这就是说, 我们事实上已经证明了满足插值条件 (5.3) 的解——即插值多项式 (5.4) 是存在的.

不仅如此, 我们还可以证明这个解是唯一的. 事实上, 假设还有次数不超过  $n$  次的多项式  $G(x)$  也满足条件 (5.3), 则  $F(x) - G(x)$  是次数不超过  $n$  次的多项式, 且它有  $n+1$  个零点. 但由于次数不超过  $n$  次的多项式的根的个数不能超过  $n$  个, 故只有一种可能, 就是  $F(x) - G(x) \equiv 0$ , 这即证明了  $F(x) \equiv G(x)$ . 由此我们得

**定理 1** 满足插值条件 (5.3) 的插值多项式 (5.4) 是存在唯一的.

关于插值余项的估计, 我们有

**定理 2** 假设  $f(x)$  在  $[a, b]$  上存在  $n$  阶连续导数, 在  $(a, b)$  内存在  $n+1$  阶导数, 则  $n$  次插值多项式  $F(x)$  对任何  $x \in [a, b]$  有插值余项

$$R(x) = f(x) - F(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \omega(x) \quad (5.5)$$

其中

$$\omega(x) = \prod_{i=0}^n (x - x_i), \quad a < \xi < b \quad \text{且一般依赖于 } x.$$

证 由插值条件

$$R(x_i) = f(x_i) - F(x_i) = 0, \quad i = 0, 1, 2, \dots, n$$

可知  $R(x) = f(x) - F(x)$  含有因式  $\omega(x)$ , 故可设

$$R(x) = f(x) - F(x) = K(x)\omega(x)$$

我们来确定  $K(x)$ . 为此, 把  $x$  看成  $[a, b]$  上一个固定点, 作辅助函数

$$G(t) = f(t) - F(t) - K(x)\omega(t)$$

则  $x_0, x_1, \dots, x_n$  及  $x$  均为  $G(t)$  的零点, 共有  $n+2$  个. 如果将  $x_0, x_1, \dots, x_n$  和  $x$  按大小顺序排列, 应用罗尔 (Rolle) 定理, 则在  $(a, b)$  内  $G'(t)$  至少有  $n+1$  个零点.  $G''(t)$  至少有  $n$  个零点,  $\dots, G^{(n+1)}(t)$  至少有一个零点. 设此零点为  $\xi$ , 则有

$$G^{(n+1)}(\xi) = f^{(n+1)}(\xi) - K(x)(n+1)! = 0$$

解之可得

$$K(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}$$

其中  $\xi$  与  $x_0, x_1, \dots, x_n$  及  $x$  有关, 且  $a < \xi < b$ , 于是可得 (5.5) 式. 证毕.

根据 (5.5) 式, 我们可得各种简单情形的余项估计:

对零次插值  $R(x) = f(x) - F(x) = f'(\xi)(x - x_0)$

对一次插值  $R(x) = f(x) - F(x) = \frac{1}{2} f''(\xi)(x - x_0)(x - x_1)$

对二次插值  $R(x) = f(x) - F(x) = \frac{1}{3!} f'''(\xi)(x - x_0)(x - x_1)(x - x_2)$

可见, 零次插值函数在插值点附近有一阶精度, 但导数值则无逼近性; 一次插值函数有二阶精度, 导数值也有了逼近性; 二次插值有三阶精度, 且直到二阶导数都有逼近性. 这就是说, 随着插值次数的提高, 插值函数的逼近程度也有所提高. 不过, 切不可以为插值次数越高, 插值函数的逼近程度就越高. 这一节的第三小节我们将讨论这个问题.

**例 2** 试估计例 1 中用线性插值与抛物线插值计算  $\sin 0.3367$  的误差.

**解** 用线性插值时, 误差估计是

$$|R(0.3367)| \leq \frac{1}{2} |f''(\xi)|(0.3367 - 0.32)(0.3367 - 0.34)|$$

其中  $f(x) = \sin x$ ,  $f'(x) = \cos x$ ,  $f''(x) = -\sin x$ , 因而

$$|f''(\xi)| \leq \max_{x_0 \leq x \leq x_1} |f''(x)| = \max_{x_0 \leq x \leq x_1} |-\sin x| \leq 0.5$$

故有

$$|R(0.3367)| \leq \frac{1}{2} \times 0.5 \times 0.0167 \times 0.0033 < \frac{3}{2} \times 10^{-4}$$

用抛物线插值时, 误差估计是

$$|R(0.3367)| \leq \frac{1}{3!} |f'''(\xi)|(0.3367 - 0.32)(0.3367 - 0.34)(0.3367 - 0.36)|$$

其中

$$|f'''(\xi)| \leq \max_{x_0 \leq x \leq x_1} |f'''(x)| = \max_{x_0 \leq x \leq x_1} |-\cos x| \leq 1$$

故有

$$|R(0.3367)| \leq \frac{1}{6} \times 1 \times 0.0167 \times 0.0033 \times 0.0233 < 0.215 \times 10^{-6}$$

## 2. 埃尔米特插值

如果已知在节点上的函数值外,还掌握导数值;或者说,问题不但要求在节点上函数值相等,而且还要求一阶甚至高阶导数值也相等,这时,我们采用的所谓带导插值,或称埃尔米特(Hermite)插值。这种插值不但要求“过点”,而且要求“相切”,因此密合程度更好。

这里我们主要讨论只涉及一阶导数的埃尔米特插值: 设  $y = f(x)$  在节点

$$a \leq x_0 < x_1 < \cdots < x_n \leq b$$

上的函数值和导数值分别为

$$\begin{aligned} f_0, f_1, \cdots, f_n \\ f'_0, f'_1, \cdots, f'_n \end{aligned}$$

要求作一个次数不超过  $2n+1$  的多项式  $F(x)$ , 满足  $2n+2$  个条件:

$$F(x_i) = f_i, \quad F'(x_i) = f'_i, \quad i = 0, 1, \cdots, n \quad (5.6)$$

容易看出,对于一点一次埃尔米特插值,有

$$F(x) = f_0 + (x - x_0)f'_0$$

我们不再逐一去构造其它简单情形,下面构造一般情形的埃尔米特插值。仿照拉格朗日插值的做法,取  $2n+2$  个基函数( $2n+1$  次函数):

$$\begin{cases} \alpha_i(x) = \left[ 1 - 2(x - x_i) \sum_{\substack{k=0 \\ k \neq i}}^n \frac{1}{x_i - x_k} \right] (l_i(x))^2 \\ \beta_i(x) = (x - x_i)(l_i(x))^2 \end{cases}$$

其中

$$l_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{x - x_k}{x_i - x_k}, \quad i = 0, 1, \cdots, n$$

可以验证,  $\alpha_i(x)$  与  $\beta_i(x)$  确具有基函数的性质:

$$\begin{aligned} \alpha_i(x_i) &= 1, & \alpha'_i(x_i) &= 0 \\ \beta_i(x_i) &= 0, & \beta'_i(x_i) &= 1 \end{aligned}$$

因此可得满足插值条件(5.6)的埃尔米特插值函数为

$$F(x) = \sum_{i=0}^n [f_i \alpha_i(x) + f'_i \beta_i(x)] \quad (5.7)$$

类似地,可证得埃尔米特插值函数的余项估计

$$f(x) - F(x) = \frac{1}{(2n+2)!} f^{(2n+2)}(\xi)(\omega(x))^2 \quad (5.8)$$

其中  $\xi$  在  $x_0, x_1, \cdots, x_n$  及  $x$  所确定的区间  $[a, b]$  内,且假定  $f(x)$  在  $[a, b]$  上有  $2n+2$  阶导数存在。

作为一种重要的情形,当  $n = 1$  时,由上可得满足条件

$$\begin{aligned} F(x_0) &= f_0, & F(x_1) &= f_1 \\ F'(x_0) &= f'_0, & F'(x_1) &= f'_1 \end{aligned}$$

的两点三次埃尔米特插值函数为

$$\begin{aligned} F(x) &= \left(1 + 2 \frac{x-x_0}{x_1-x_0}\right) \left(\frac{x-x_1}{x_0-x_1}\right)^2 f_0 + \left(1 + 2 \frac{x-x_1}{x_0-x_1}\right) \left(\frac{x-x_0}{x_1-x_0}\right)^2 f_1 \\ &\quad + (x-x_0) \left(\frac{x-x_1}{x_0-x_1}\right)^2 f'_0 + (x-x_1) \left(\frac{x-x_0}{x_1-x_0}\right)^2 f'_1 \end{aligned} \quad (5.9)$$

相应的插值余项为

$$R(x) = f(x) - F(x) = \frac{f^{(4)}(\xi)}{4!} (x-x_0)^2 (x-x_1)^2 \quad (5.10)$$

其中  $\xi$  在  $x_0, x_1, x$  所确定的区间内.

**例 3** 求满足插值条件

|      |   |    |
|------|---|----|
| $x$  | 1 | 2  |
| $y$  | 2 | 3  |
| $y'$ | 1 | -1 |

的埃尔米特插值多项式.

**解** 根据 (5.9) 式可得所求埃尔米特插值多项式为

$$\begin{aligned} H(x) &= 2 \times \left(1 + 2 \times \frac{x-1}{2-1}\right) \left(\frac{x-2}{1-2}\right)^2 + 3 \times \left(1 + 2 \times \frac{x-2}{1-2}\right) \left(\frac{x-1}{2-1}\right)^2 \\ &\quad + 1 \times (x-1) \left(\frac{x-2}{1-2}\right)^2 + (-1)(x-2) \left(\frac{x-1}{2-1}\right)^2 \\ &= -2x^3 + 8x^2 - 9x + 5 \end{aligned}$$

以上我们讨论的是节点上的函数值和对应的一阶导数值均点已知情况,实际上也可能有节点上的函数值和导数值不是全部都已知情况;也可能有已知二阶和二阶以上的导数值的情况.我们通过例子来说明这时如何求埃尔米特插值函数.

**例 4** 已知函数表如下:

|      |        |       |
|------|--------|-------|
| $x$  | $x_0$  | $x_1$ |
| $f$  | $f_0$  | $f_1$ |
| $f'$ | $f'_0$ |       |

求一插值多项式  $F(x)$ , 使其满足条件:

$$F(x_0) = f_0, \quad F(x_1) = f_1, \quad F'(x_0) = f'_0$$

**解** 先根据样点  $(x_0, f_0), (x_1, f_1)$  作线性插值函数

$$F_1(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0} (x - x_0)$$

再注意到要求的  $F(x)$  与  $F_1(x)$  在节点  $x_0, x_1$  上函数值相同,即

$$F(x_0) = F_1(x_0) = f_0, \quad F(x_1) = F_1(x_1) = f_1$$

于是,它们之差可设为

$$F(x) - F_1(x) = K(x - x_0)(x - x_1)$$

或

$$F(x) = F_1(x) + K(x - x_0)(x - x_1)$$

$K$  为待定常数. 为确定  $K$ , 对上式求导得

$$F'(x) = F_1'(x) + K(x - x_0 + x - x_1)$$

令  $x = x_0$  代入上式, 并注意到  $F'(x_0) = f'_0$ , 得

$$\begin{aligned} F'(x_0) &= F_1'(x_0) + K(x_0 - x_1) \\ &= \frac{f_1 - f_0}{x_1 - x_0} + K(x_0 - x_1) = f'_0 \end{aligned}$$

于是有

$$K = \frac{1}{x_0 - x_1} \left( f'_0 - \frac{f_1 - f_0}{x_1 - x_0} \right) = \frac{1}{x_1 - x_0} \left( \frac{f_1 - f_0}{x_1 - x_0} - f'_0 \right)$$

代入  $F(x)$  可得

$$F(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0} (x - x_0) + \frac{1}{x_1 - x_0} \left( \frac{f_1 - f_0}{x_1 - x_0} - f'_0 \right) (x - x_0)(x_1 - x_1)$$

**例 5** 已知函数表如下:

| $x$   | $x_0$ | $x_1$   | $x_2$ |
|-------|-------|---------|-------|
| $f$   | $f_0$ |         | $f_2$ |
| $f'$  |       | $f'_1$  |       |
| $f''$ |       | $f''_1$ |       |

求一个三次埃尔米特插值函数  $F(x)$ , 使它满足条件

$$\begin{aligned} F(x_0) &= f_0, & F(x_2) &= f_2 \\ F'(x_1) &= f'_1, & F''(x_1) &= f''_1 \end{aligned}$$

**解** 类似例 4, 可设所求多项式  $F(x)$  为

$$F(x) = F_1(x) + K(x)(x - x_0)(x - x_2)$$

其中

$$F_1(x) = f_0 + \frac{f_2 - f_0}{x_2 - x_0} (x - x_0)$$

$$K(x) = ax + b, \quad a, b \text{ 为待定常数.}$$

对  $F(x)$  求一阶导数得

$$F'(x) = F_1'(x) + K'(x)(x - x_0)(x - x_2) + K(x)(x - x_0 + x - x_2)$$

再求一次导数得

$$F''(x) = F_1''(x) + 2K'(x)(2x - x_0 - x_2) + 2K(x)$$

令  $x = x_1$  代入  $F'(x)$  和  $F''(x)$  且注意插值条件得

$$F'(x_1) = F_1'(x_1) + K'(x_1)(x_1 - x_0)(x_1 - x_2) + K(x_1)(2x_1 - x_0 - x_2) = f'_1$$

$$F''(x_1) = F_1''(x_1) + 2K'(x_1)(2x_1 - x_0 - x_2) + 2K(x_1) = f''_1$$

其中

$$F_1'(x_1) = \frac{f_2 - f_0}{x_2 - x_0}, \quad F_1''(x_1) = 0, \quad K'(x_1) = a$$

解上二式得

$$\begin{cases} a = \frac{\frac{2x_1 - x_0 - x_2}{2} f_1'' - f_1' + \frac{f_2 - f_1}{x_2 - x_0}}{(x_1 - x_0)^2 + (x_1 - x_0)(x_1 - x_2) + (x_1 - x_2)^2} \\ b = \frac{f_1''}{2} - a(3x_1 - x_0 - x_2) \end{cases}$$

代入  $F(x)$ , 整理得

$$F(x) = f_0 + \frac{f_1 - f_0}{x_2 - x_0}(x - x_0) + \frac{f_1''}{2}(x - x_0)(x - x_2) + a(x - x_1 + x_0 + x_2 - 2x_1)(x - x_0)(x - x_2)$$

其中  $a$  如前式所确定。

### 3. 插值过程的稳定性分析

(1) 插值过程中有两种误差: 一是上面所说的插值余项, 即截断误差; 另一是由节点数据  $f_i$  本身的误差(包括实验的误差或计算过程中舍入的误差)引起的初值误差。以拉格朗日插值为例, 设真值  $f_i$  被代以含有误差的  $\tilde{f}_i = f_i - \delta f_i$ ,  $\tilde{F}(x)$  为以  $\tilde{f}_i$  作成的插值多项式, 则最后误差为

$$f(x) - \tilde{F}(x) = [f(x) - F(x)] + [F(x) - \tilde{F}(x)]$$

右端第一项是插值余项; 第二项根据插值公式可写成

$$F(x) - \tilde{F}(x) = \sum_{i=0}^n f_i l_i(x) - \sum_{i=0}^n (f_i - \delta f_i) l_i(x) = \sum_{i=0}^n \delta f_i l_i(x)$$

可见数据误差  $\delta f_i$  通过插值基函数  $l_i(x)$  被扩散或放大, 即插值基函数是数据误差的“影响函数”。因此当  $n$  趋大时, 插值过程对于样点的数据误差非常敏感, 从而有可能引起了高次插值的数值不稳定性。图 5-4 表示一个  $l_i(x)$ , 它在基本区间即  $x_0$  到  $x_n$  之内呈波动状; 在基本区间之外则按距离的  $n$  次幂放大。当计算点落在插值区间之内时叫内插, 否则叫外插。有时我们仅在变量的一定范围内掌握数据和规律, 需要据此外推在该范围之外的行为, 这就要用外插。

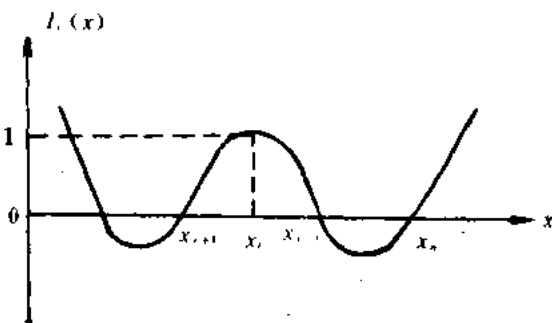


图 5-4

显然, 作外插计算时要特别小心, 在次数较高和距离较远时结果是很不可靠的。

(2) 理论可以证明: 如果函数  $f(x)$  在整个数轴  $-\infty < x < \infty$  上解析, 且作为复变函数  $f(z)$  在整个  $Z$  平面也解析, 则在实轴的任意区间  $a \leq x \leq b$  上采用等距节点且逐次加密的插值过程收敛于原来函数  $f(x)$ 。这就是说, 在这种情况下精确度是随次数升高而提高的。对于不光滑的函数, 或者不是在整个  $Z$  平面上解析的函数, 情况就不相同了。早在本世纪初, 龙格 (Runge) 就给出一个例子:

$$f(x) = \frac{1}{1+x^2}$$

它在  $[-5, 5]$  上各阶导数均存在, 但在  $[-5, 5]$  上取  $n+1$  个等距节点  $x_i$  所作成的拉



格朗日插值多项式, 当  $n \rightarrow \infty$  时, 只在  $|x| \leq 3.63$  内收敛, 而在这个区间外是发散的, 如图 5-5.

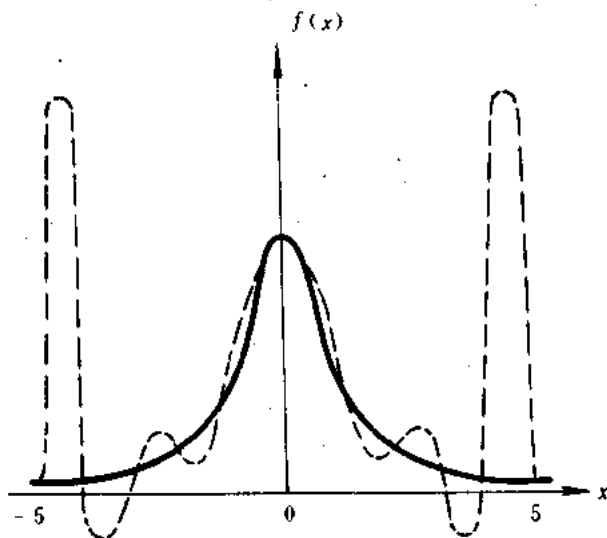


图 5-5

这个例子被称为“龙格现象”. 现在可以看到, 这里  $f(x)$  在复平面内的延拓  $f(z) = \frac{1}{1+z^2}$  在虚轴上有两个奇点  $z = \pm i$ , 发散性正是这两个“隐藏”的奇点引起的.

(3) 多项式是一种具有最高度光滑性的解析函数, 因此它只适宜于作为高光滑函数的插值函数. 以区间  $[-1, 1]$  上的函数  $f(x) = 1 - |x|$  为例, 若用  $x = -1, 0, 1$  三点作抛物线插值, 如图 5-6 中的虚线, 情况显然不好. 若逐次等距加密节点作高次插值, 则情况更糟. 然而, 若取  $x = -1, 0, 1$ , 分二段分别作线性插值, 则恰好得到原来的函数.

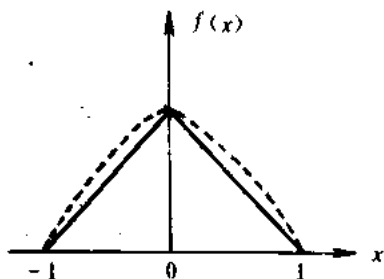


图 5-6

综上所述, 可见盲目提高插值次数的做法不是可取的. 实际计算中次数高于 5、6 次就很少用了. 提高精度的方向在于采用分段低次插值, 特别是样条插值.

#### 4. 几种分段多项式插值公式

由于高次多项式插值数值不稳定, 而低次多项式插值又简单方便, 因此, 当节点较多时, 自然设想把插值区间分成若干小段, 在每一小段中使用低次插值, 而在分点处保持一定的连续性, 这就是所谓分段多项式插值. 这种处理的算法并不复杂, 但却具有较好的收敛性和稳定性.

我们讨论几种简单的分段多项式插值公式.

##### (1) 分段零次插值(阶梯插值)

以两节点间的中点为分点, 把插值区间分成  $n+1$  个子区间  $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ , 其中  $i = 0, 1, \dots, n$ ,  $x_{0-\frac{1}{2}} = x_0$ ,  $x_{n+\frac{1}{2}} = x_n$ . 在每个子区间  $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$  内取

$$F(x) = f_i, \quad x \in [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$$

这就是分段零次插值(水平插值). 如图 5-7, 它是阶梯形, 在分点处间断. 如果利用基函

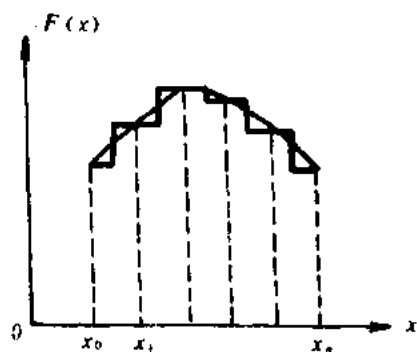


图 5-7

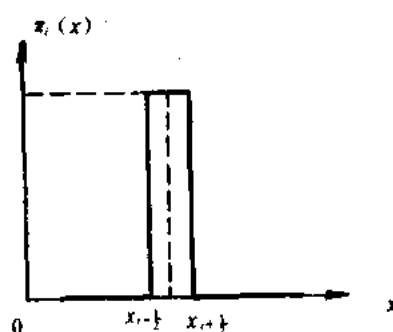


图 5-8

数(如图 5-8)

$$\pi_i(x) = \begin{cases} 1, & x_{i-\frac{1}{2}} \leq x \leq x_{i+\frac{1}{2}} \\ 0, & \text{其它处} \end{cases} \quad i = 0, 1, \dots, n$$

则分段零次插值函数也可写成

$$F(x) = \sum_{i=0}^n f_i \pi_i(x), \quad a \leq x \leq b$$

(2) 分段一次(线性)插值(折线插值)

在每个子区间  $[x_i, x_{i+1}]$  上依次过样点  $(x_i, f_i)$  作连接折线

$$F(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} f_i + \frac{x - x_i}{x_{i+1} - x_i} f_{i+1}, \quad x_i \leq x \leq x_{i+1}, \quad i = 0, 1, \dots, n-1$$

这就是分段一次插值函数,如图 5-9.

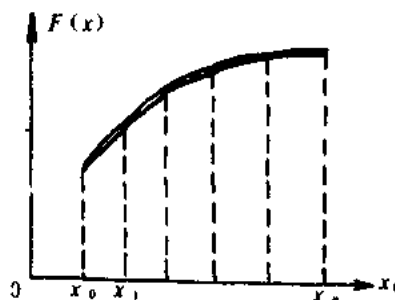


图 5-9

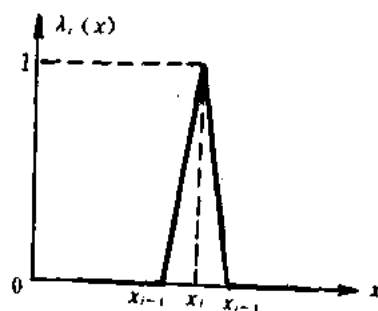


图 5-10

它在  $[a, b]$  上分段一次,总体连续,在  $x_i$  处导数间断。如果作基函数

$$\lambda_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} \leq x \leq x_i \\ \frac{x - x_{i+1}}{x_i - x_{i+1}}, & x_i \leq x \leq x_{i+1} \\ 0, & \text{其它处} \end{cases}$$

其中  $i = 0, 1, \dots, n$ ,  $x_{-1} = x_0$ ,  $x_{n+1} = x_n$ , 如图 5-10. 则分段一次插值函数也可写成

$$F(x) = \sum_{i=0}^n f_i \lambda_i(x), \quad a \leq x \leq b$$

### (3) 分段二次(抛物线)插值

为了讨论方便,假设在两节点间的半点  $x_{i+\frac{1}{2}}$  或  $x_{i-\frac{1}{2}}$  处,我们还已知样点  $(x_{i-\frac{1}{2}}, f_{i-\frac{1}{2}})$  或  $(x_{i+\frac{1}{2}}, f_{i+\frac{1}{2}})$ 。分段二次插值就是在子区间  $[x_i, x_{i+1}]$  上过三个样点  $(x_i, f_i)$ ,  $(x_{i+\frac{1}{2}}, f_{i+\frac{1}{2}})$ ,  $(x_{i+1}, f_{i+1})$  作抛物线

$$F(x) = \sum_{k=i, i+\frac{1}{2}, i+1} \left( \prod_{\substack{j=i, i+\frac{1}{2}, i+1 \\ j \neq k}} \frac{x-x_j}{x_k-x_j} \right) f_k, \quad x_i \leq x \leq x_{i+1} \quad i=0, 1, \dots, n-1$$

如图 5-11, 这里分段二次, 总体连续, 但在  $x_i, x_{i+1}$  处导数间断。如果作基函数

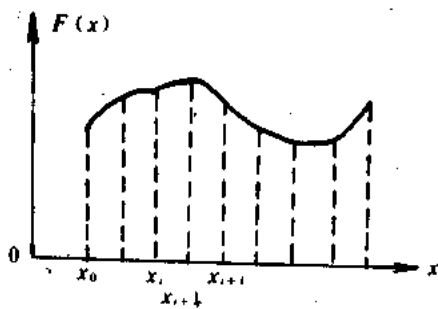


图 5-11

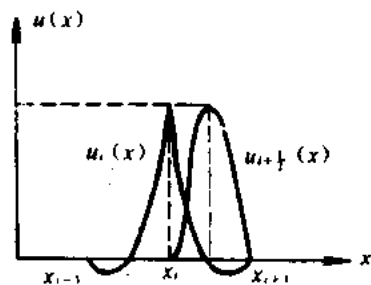


图 5-12

$$\begin{cases} \mu_i(x) = 2\lambda_i^2 - \lambda_i, \\ \mu_{i+\frac{1}{2}}(x) = 4\lambda_i\lambda_{i+1} \end{cases}$$

其中  $\lambda_i(x)$  是分段一次插值的基函数, 如图 5-12. 则可以验证

$$\begin{aligned} \mu_i(x_i) &= \delta_{ii}, & \mu_i(x_{i+\frac{1}{2}}) &= 0 \\ \mu_{i+\frac{1}{2}}(x_i) &= 0, & \mu_{i+\frac{1}{2}}(x_{i+\frac{1}{2}}) &= \delta_{ii} \end{aligned}$$

于是分段二次插值函数也可写成

$$F(x) = \sum_{i=0}^n f_i \mu_i(x) + \sum_{i=0}^{n-1} f_{i+\frac{1}{2}} \mu_{i+\frac{1}{2}}(x), \quad a \leq x \leq b$$

### (4) 分段三次埃尔米特插值

回忆两点三次埃尔米特插值函数 (5.9), 我们容易得到分段三次埃尔米特插值函数

$$F(x) = \sum_{k=i}^{i+1} [f_k \alpha_k(x) + f'_k \beta_k(x)], \quad x_i \leq x \leq x_{i+1} \quad i=0, 1, \dots, n-1$$

其中

$$\begin{cases} \alpha_i(x) = \left(1 + 2 \frac{x-x_i}{x_{i+1}-x_i}\right) \left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right)^2 \\ \alpha_{i+1}(x) = \left(1 + 2 \frac{x-x_{i+1}}{x_i-x_{i+1}}\right) \left(\frac{x-x_i}{x_{i+1}-x_i}\right)^2 \\ \beta_i(x) = (x-x_i) \left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right)^2 \\ \beta_{i+1}(x) = (x-x_{i+1}) \left(\frac{x-x_i}{x_{i+1}-x_i}\right)^2 \end{cases}$$

这时,  $F(x)$  分段三次, 总体直至一阶导数连续, 但二阶导数在  $x_i$  处间断。如果在整个区间上作基函数

$$\alpha_i(x) = \begin{cases} \left(1 + 2 \frac{x - x_{i-1}}{x_i - x_{i-1}}\right) \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2, & x_{i-1} \leq x \leq x_i \quad (i \neq 0) \\ \left(1 + 2 \frac{x - x_i}{x_{i+1} - x_i}\right) \left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)^2, & x_i \leq x \leq x_{i+1} \quad (i \neq n) \\ 0, & \text{其它处} \end{cases} \quad (5.11)$$

$$\beta_i(x) = \begin{cases} (x - x_i) \left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)^2, & x_{i-1} \leq x \leq x_i \quad (i \neq 0) \\ (x - x_i) \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2, & x_i \leq x \leq x_{i+1} \quad (i \neq n) \\ 0, & \text{其它处} \end{cases} \quad (5.12)$$

则分段三次埃尔米特插值可写成

$$F(x) = \sum_{i=0}^n [f_i \alpha_i(x) + f'_i \beta_i(x)], \quad a \leq x \leq b \quad (5.13)$$

### 5. 一元三点不等距成组插值的 FORTRAN 程序

下面提供分段二次(抛物线)插值,即通常称为一元三点不等距成组插值的FORTRAN程序及其上机算例。构造插值函数常用于计算节点以外的点 $x$ (一个或多个)对应的函数值。

为了便于 FORTRAN 程序设计,这里节点的下标改为从 1 开始,即设已知  $n$  个插值节点  $x_1 < x_2 < \cdots < x_n$  及其对应的函数值  $y_1, y_2, \cdots, y_n$ 。采用所谓一元三点插值计算点  $x$  处的函数值  $y(x)$ ,即分别选取最靠近  $x$  的三个节点  $x_i, x_{i+1}, x_{i+2}$ ,按分段二次插值公式(也称三点公式):

$$y(x) = \sum_{k=i}^{i+2} \left( \prod_{\substack{j=i \\ j \neq k}}^{i+2} \frac{x - x_j}{x_k - x_j} \right) y_k$$

计算  $y(x)$ ,其中

$$i = \begin{cases} 1 & x \leq x_1 \\ s & x_i < x \leq x_{i+1}, x - x_i \geq x_{i+1} - x, s = 2, 3, \cdots, n-2 \\ s-1 & x_i < x \leq x_{i+1}, x - x_i < x_{i+1} - x, s = 2, 3, \cdots, n-2 \\ n-2 & x > x_{n-1} \end{cases}$$

下面程序将从给定的  $N$  个插值节点中,用上述三点插值公式对  $L$  个一元列表函数进行成组插值<sup>[61]</sup>。

设置子程序哑元如下:

N 整型变量,输入参数,插值节点的个数( $\geq 3$ )。

L 整型变量,输入参数,插值的函数的个数。

X 实型变量,输入参数,插值点。

A  $N$  个元素的一维实型数组,输入参数,存放给定的插值节点  $x_1 < x_2 < \cdots < x_n$ 。

B  $N \times L$  个元素的二维实型数组,输入参数;存放给定的插值节点处的  $L$  组函数值,排列顺序为:  $y^{(1)}(x_1), y^{(1)}(x_2), \cdots, y^{(1)}(x_n); y^{(2)}(x_1), y^{(2)}(x_2), \cdots, y^{(2)}(x_n); \cdots; y^{(L)}(x_1), y^{(L)}(x_2), \cdots, y^{(L)}(x_n)$ 。

Y  $L$  个元素的一维实型数组,输出参数;存放  $L$  个插值结果。

FORTTRAN 子程序:

```

SUBROUTINE LAQ (N, L, X, A, B, Y)
  DIMENSION A(N), B(N, L), Y(L)
  N1 = N - 2
  DO 10 I = 1, N1
    IF (X. LE. A(I + 1)) GO TO 20
  10 CONTINUE
  I = N1
  20 IF (I. EQ. 1) GO TO 30
    IF (X - A(I). GE. A(I + 1) - X) GO TO 30
    I = I - 1
  30 A1 = A(I)
    A2 = A(I + 1)
    A3 = A(I + 2)
    U = (X - A2) * (X - A3) / ((A1 - A2) * (A1 - A3))
    V = (X - A1) * (X - A3) / ((A2 - A1) * (A2 - A3))
    W = (X - A1) * (X - A2) / ((A3 - A1) * (A3 - A2))
    DO 40 J = 1, L
  40 Y(J) = U * B(I, J) + V * B(I + 1, J) + W * B(I + 2, J)
    RETURN
  END

```

**例 1** 作为说明性的算例, 我们简单地取平方根表和常用对数表中已知的 6 个点作观测数据:

| $x$              | 1.20    | 1.24    | 1.28    | 1.32    | 1.36    | 1.40    |
|------------------|---------|---------|---------|---------|---------|---------|
| $y_1 = \sqrt{x}$ | 1.09545 | 1.11355 | 1.13137 | 1.14891 | 1.16619 | 1.18322 |
| $y_2 = \lg x$    | 0.07918 | 0.09342 | 0.10721 | 0.12057 | 0.13354 | 0.14613 |

然后利用上述子程序, 求出当

$$x = 1.22, 1.26, 1.30, 1.34, 1.38$$

时相应于函数  $y_1$  和  $y_2$  的值。

解题程序如下(假定已知数据用自由格式读入):

```

DIMENSION A(6), B(6, 2), Y(2)
READ *, A, B
X=1.22
DO 15 I=1, 5
  CALL LAQ (6, 2, X, A, B, Y)
  WRITE (6, 20) X, Y

```

```

      X=X+0.04
15  CONTINUE
20  FORMAT (5X, 3E20.10)
      STOP
      END

```

C

```

SUBROUTINE LAQ (N, L, X, A, B, Y)
: (本子程序段体)
END

```

插值结果如下:

| X             | Y1             | Y2             |
|---------------|----------------|----------------|
| 0.1220000E+01 | 0.11044925E+01 | 0.86356250E-01 |
| 0.1260000E+01 | 0.11227500E+01 | 0.10036875E+00 |
| 0.1300000E+01 | 0.11403000E+01 | 0.11393876E+00 |
| 0.1340000E+01 | 0.11575813E+01 | 0.12710251E+00 |
| 0.1380000E+01 | 0.11747363E+01 | 0.13988251E+00 |

此结果也可与精确值作比较如下:

| x    | y <sub>1</sub> |         | y <sub>2</sub> |         |
|------|----------------|---------|----------------|---------|
|      | 计算机插值          | 精 确 值   | 计算机插值          | 精 确 值   |
| 1.22 | 1.10449        | 1.10454 | 0.08635        | 0.08636 |
| 1.26 | 1.12275        | 1.12250 | 0.10036        | 0.10037 |
| 1.30 | 1.14030        | 1.14018 | 0.11393        | 0.11394 |
| 1.34 | 1.15758        | 1.15758 | 0.12710        | 0.12710 |
| 1.38 | 1.17473        | 1.17473 | 0.13988        | 0.13988 |

**例 2** 这是来自半导体物理研究中的一组有关参数。我们把它归结为如下样点:

| x              | 0.001  | 0.5    | 1.5    | 2.0    | 3.0    | 4.0    | 5.0    | 6.0    | 7.0    | 9.0    |
|----------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| y <sub>1</sub> | 8.42   | 8.78   | 8.82   | 8.95   | 8.32   | 8.16   | 7.43   | 6.42   | 6.80   | 6.48   |
| y <sub>2</sub> | 11.30  | 11.297 | 11.316 | 11.381 | 11.376 | 11.391 | 11.386 | 11.409 | 11.407 | 11.436 |
| y <sub>3</sub> | 3.13   | 3.17   | 3.19   | 3.26   | 3.26   | 3.40   | 3.40   | 3.40   | 3.47   | 3.65   |
| y <sub>4</sub> | 16.619 | 16.630 | 16.639 | 16.607 | 16.573 | 16.550 | 16.538 | 16.513 | 16.535 | 16.540 |

现在要求当

$x = 0.2, 1.0, 1.7, 2.5, 3.5, 4.5, 5.5, 6.5, 8.0$

时对应各函数  $y$  的值。

易见,可编写类似算例 1 的解题程序。我们可得计算结果如下:

| X       | Y1      | Y2       | Y3      | Y4       |
|---------|---------|----------|---------|----------|
| 0.20000 | 8.59120 | 11.29780 | 3.14840 | 16.61924 |
| 1.00000 | 8.76333 | 11.28800 | 3.16000 | 16.64667 |
| 1.70000 | 8.86320 | 11.33756 | 3.21320 | 16.62912 |

|         |         |          |         |          |
|---------|---------|----------|---------|----------|
| 2.50000 | 8.57625 | 11.37600 | 3.24250 | 16.58862 |
| 3.50000 | 8.31125 | 11.38600 | 3.34750 | 16.56013 |
| 4.50000 | 7.83000 | 11.38500 | 3.40000 | 16.54562 |
| 5.50000 | 6.75125 | 11.40063 | 3.39125 | 16.51962 |
| 6.50000 | 6.65500 | 11.40663 | 3.43333 | 16.52562 |
| 8.00000 | 6.82000 | 11.41600 | 3.55333 | 16.54400 |

## 5-3 样条函数与三次样条插值

### 1. 样条函数概念

样条是绘图员用于描绘光滑曲线的一种工具,它是由一些易弯曲材料做成的棒条(如木条)。对样条加上一些重量(如压铁),可以强制它通过或接近图表上确定的描绘点,于是,绘图员沿着样条描下曲线,这样的曲线便称为样条曲线。样条曲线在样点处是比较光滑的。

从数学概念来看,分段低次插值函数有很好的收敛性,但在样点处的光滑性较差。如分段二次插值函数,虽然它在整个插值区间是连续的,但在分段的连接点处,一阶导数就不连续了。实际问题往往要求分段插值函数要具备相当的光滑性,如机翼形线,船体放样等型值线,就要求有二阶光滑度,即有二阶连续导数。为此,人们在分段插值中进一步发展了样条插值。

下面先引进样条函数的概念。

**定义1** 设区间  $[a, b]$  由节点  $a = x_0, x_1, \dots, x_n = b$  ( $x_0 < x_1 < \dots < x_n$ ) 划分为  $n$  个子区间  $[x_i, x_{i+1}]$ , 若函数  $S(x)$ :

- 1) 在每个子区间上是分段  $m$  次多项式;
- 2) 在节点处有直至  $m-1$  阶的连续导数,则  $S(x)$  称为  $m$  次多项式样条函数。

样条函数常简称样条 (Spline)。“样条函数”这个术语意味着这种函数的图象与按“真实”样条画出的曲线相似。

根据定义,可知上节讨论的分段零次插值(阶梯插值)函数是零次样条;分段一次插值(折线插值)函数是一次样条。可是分段二次插值函数不是样条。很久以来,多项式曾最广泛地被用来逼近其它函数,这主要是因为它们有最简单的数学性质。然而,据一般的观察,拟合于许多已知点的其次数为中等大小的多项式,它的图象比起用样条或曲线板描绘的曲线往往有更多更厉害的波动。已经有大量的材料说明,在许多情形下,一个样条函数比一个所含参数与它差不多的多项式更适用于做逼近函数。许多“最优”逼近问题的解答,实际上是样条函数。样条函数不仅应用于象定积分和常微分方程的近似解这样一些数值分析领域,而且也应用于象最优控制、系统识别等领域。

样条函数的一个特别简单的形式可通过所谓截尾幂函数

$$x_+^m = \begin{cases} x^m, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

来表示,即对任何  $m$  次多项式样条函数  $S(x)$ , 有唯一表达式

$$S(x) = p(x) + \sum_{j=0}^n c_j (x - x_j)_+^m$$

其中  $p(x)$  是  $m$  次或小于  $m$  次的多项式.

关于样条函数的理论和应用,可参见诸如 [A4], [C13] 等.

下面仅讨论样条函数在插值中的简单应用.

## 2. 三次样条插值

**定义 2** 若样条函数  $S(x)$  满足插值条件

$$S(x_i) = f_i, \quad i = 0, 1, \dots, n \quad (5.14)$$

其中  $(x_i, f_i)$ ,  $i = 0, 1, \dots, n$  是已知的样点,则  $S(x)$  称为样条插值函数.

这里我们主要讨论常用的三次样条插值函数,即要求在分段点具有直至二阶连续导数的分段三次多项式插值函数.

从定义可知,要在每个子区间  $[x_i, x_{i+1}]$  上作出三次样条函数  $S(x)$  (三次多项式),必须确定 4 个参数,因而在整个区间上作出  $S(x)$ ,一共需要确定  $4n$  个参数. 现在我们要看有多少个条件? 首先,插值条件 (5.14) 有  $n+1$  个. 其次,由于要求  $S(x)$  在  $[a, b]$  上二阶导数连续,故在内节点处应满足连续性条件:

$$\begin{cases} S(x_i - 0) = S(x_i + 0) \\ S'(x_i - 0) = S'(x_i + 0) \\ S''(x_i - 0) = S''(x_i + 0) \end{cases} \quad i = 1, 2, \dots, n-1 \quad (5.15)$$

这里有  $3n-3$  个条件. 以上两项共  $4n-2$  个条件,可知尚缺两个条件. 为此,通常就在  $[a, b]$  的端点  $a = x_0$  与  $b = x_n$  上再提供两个条件,称为边界条件.

边界条件可以有不同的提法,常见的三种是:

边界条件 I: 给定导数值  $S'(x_0) = f'_0$ ,  $S'(x_n) = f'_n$ . 当曲线在端点的斜率很明确时,如当端点是一个局部极值时,有水平斜率,因而可取  $f'_0 = 0$  或  $f'_n = 0$ .

边界条件 II: 给定二阶导数值  $S''(x_0) = f''_0$ ,  $S''(x_n) = f''_n$ . 当曲线在端点的行为近乎反折点时,可取  $f''_0 = 0$  或  $f''_n = 0$ . 特别,取  $S''(x_0) = S''(x_n) = 0$  称为自然边界条件.

边界条件 III: 也称为周期性边界条件,即认为函数以  $[a, b]$  为周期向两端周期性延拓,且在端点保持直到二阶导数连续,故

$$\begin{cases} S(x_0 + 0) = S(x_n - 0) \\ S'(x_0 + 0) = S'(x_n - 0) \\ S''(x_0 + 0) = S''(x_n - 0) \end{cases}$$

此时自然应有  $f_0 = f_n$ , 周期性边界条件适用于封闭曲线的插值.

理论已经证明,带边界条件之一的三次样条插值问题的解是存在且唯一的. 下面我们求作三次插值样条函数  $S(x)$  的表达式. 有两种具体做法: 一种是利用节点处的一阶导数来表示三次样条插值函数,即通过设一阶导数值  $S'(x_i) = m_i$  作待定参数,由解出  $m_i$  来作出  $S(x)$ , 这种方法导出了所谓三转角方程; 另一种是利用节点处的二阶导数来表示三次样条插值函数,即通过设二阶导数值  $S''(x_i) = M_i$  作待定参数,由解出  $M_i$  来作出  $S(x)$ , 这种方法导出了所谓三弯矩方程. 下面分别讨论.

### (I) 三转角方程

设  $S'(x_i) = m_i$  ( $i = 0, 1, \dots, n$ ) 为待定参数,则由分段三次埃尔米特插值公式



(5.13) 有

$$S(x) = \sum_{i=0}^n [f_i \alpha_i(x) + m_i \beta_i(x)], \quad a \leq x \leq b \quad (5.16)$$

其中  $\alpha_i(x)$  与  $\beta_i(x)$  分别为基函数 (5.11) 与 (5.12). 可知  $S(x)$  分段三次, 具有直至一阶连续导数, 且满足插值条件 (5.14); 只是  $m_i$  未知, 且作为样条函数在分段点处  $S(x)$  的二阶连续导数尚未保证. 为此, 我们利用连续性条件  $S''(x_i - 0) = S''(x_i + 0)$  和边界条件之一来解决上述问题.

考虑  $S(x)$  在  $[x_i, x_{i+1}]$  ( $i = 1, 2, \dots, n-1$ ) 上的表达式并记  $h_i = x_{i+1} - x_i$ :

$$\begin{aligned} S(x) = & \frac{(x - x_{i+1})^2 [h_i + 2(x - x_i)]}{h_i^3} f_i \\ & + \frac{(x - x_i)^2 [h_i + 2(x_{i+1} - x)]}{h_i^3} f_{i+1} \\ & + \frac{(x - x_{i+1})^2 (x - x_i)}{h_i^2} m_i + \frac{(x - x_i)^2 (x - x_{i+1})}{h_i^2} m_{i+1} \end{aligned}$$

求  $S(x)$  的二阶导数可得

$$\begin{aligned} S''(x) = & \frac{6x - 2x_i - 4x_{i+1}}{h_i^2} m_i + \frac{6x - 4x_i - 2x_{i+1}}{h_i^2} m_{i+1} \\ & + \frac{6(x_i + x_{i+1} - 2x)}{h_i^3} (f_{i+1} - f_i) \end{aligned}$$

于是有

$$S''(x_i + 0) = -\frac{4}{h_i} m_i - \frac{2}{h_i} m_{i+1} + \frac{6}{h_i^2} (f_{i+1} - f_i)$$

同理考虑  $S(x)$  在  $[x_{i-1}, x_i]$  ( $i = 1, 2, \dots, n-1$ ) 上的表达式及求其二阶导数  $S''(x)$ , 于是有

$$S''(x_i - 0) = \frac{2}{h_{i-1}} m_{i-1} + \frac{4}{h_{i-1}} m_i - \frac{6}{h_{i-1}^2} (f_i - f_{i-1})$$

再由条件  $S''(x_i - 0) = S''(x_i + 0)$  便得

$$\begin{cases} \frac{1}{h_{i-1}} m_{i-1} + 2\left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) m_i + \frac{1}{h_i} m_{i+1} = 3\left[\frac{f_{i+1} - f_i}{h_i^2} + \frac{f_i - f_{i-1}}{h_{i-1}^2}\right] \\ i = 1, 2, \dots, n-1 \end{cases}$$

用  $\left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right)$  除全式, 并记  $f[x_i, x_{i+1}] = \frac{f_{i+1} - f_i}{h_i}$ ,

$$\lambda_i = \frac{h_i}{h_{i-1} - h_i}, \quad \mu_i = \frac{h_{i-1}}{h_{i-1} + h_i}, \quad g_i = 3(\lambda_i f[x_{i-1}, x_i] + \mu_i f[x_i, x_{i+1}])$$

其中  $i = 1, 2, \dots, n-1$ , 则上述方程简化为

$$\begin{cases} \lambda_i m_{i-1} + 2m_i + \mu_i m_{i+1} = g_i \\ i = 1, 2, \dots, n-1 \end{cases} \quad (5.17)$$

这是包含  $n+1$  个未知数  $m_0, m_1, \dots, m_n$  的  $n-1$  个方程, 这时:

1) 若加上边界条件 I, 即已知  $S'(x_0) = f'_0 = m_0, S'(x_n) = f'_n = m_n$ , 则得求  $m_1, m_2, \dots, m_{n-1}$  的方程组为

$$\begin{bmatrix} 2 & \mu_1 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_1 & 2 & \mu_2 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 2 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \lambda_{n-2} & 2 & \mu_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & \lambda_{n-1} & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} g_1 - \lambda_1 f'_0 \\ g_2 \\ g_3 \\ \vdots \\ g_{n-2} \\ g_{n-1} - \mu_{n-1} f'_n \end{bmatrix} \quad (5.18)$$

2) 若加上边界条件 II, 即已知  $S''(x_0) = f'_0$ ,  $S''(x_n) = f'_n$ , 则由  $S''(x_0 + 0)$  与  $S''(x_n - 0)$  的表达式可导出

$$\begin{cases} f'_0 = -\frac{4}{h_0} m_0 - \frac{2}{h_0} m_1 + \frac{6}{h_0^2} (f_1 - f_0) \\ f'_n = \frac{2}{h_{n-1}} m_{n-1} + \frac{4}{h_{n-1}} m_n - \frac{6}{h_{n-1}^2} (f_n - f_{n-1}) \end{cases}$$

即

$$\begin{cases} 2m_0 + m_1 = 3f[x_0, x_1] - \frac{h_0}{2} f'_0 & (\text{记为 } g_0) \\ 2m_{n-1} + 2m_n = 3f[x_{n-1}, x_n] + \frac{h_{n-1}}{2} f'_n & (\text{记为 } g_n) \end{cases}$$

特别, 对于自然边界条件有

$$\begin{cases} 2m_0 + m_1 = 3f[x_0, x_1] = g_0 \\ m_{n-1} + 2m_n = 3f[x_{n-1}, x_n] = g_n \end{cases}$$

于是可得求  $m_0, m_1, \dots, m_n$  的方程组为

$$\begin{bmatrix} 2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_1 & 2 & \mu_1 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 2 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \lambda_{n-1} & 2 & \mu_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{n-1} \\ g_n \end{bmatrix} \quad (5.19)$$

3) 若加上边界条件 III (周期性边界条件), 即  $f'_0 = f'_n$ , 则得  $m_0 = m_n$ . 又仿 2) 有

$$\begin{cases} \frac{2}{h_0} m_0 + \frac{1}{h_0} m_1 = \frac{3}{h_0} f[x_0, x_1] - \frac{1}{2} f'_0 \\ \frac{1}{h_{n-1}} m_{n-1} + \frac{2}{h_{n-1}} m_n = \frac{3}{h_{n-1}} f[x_{n-1}, x_n] + \frac{1}{2} f'_n \end{cases}$$

二式相加 (注意  $f'_0 = f'_n$ ) 得

$$\begin{aligned} & \frac{1}{h_0} m_1 + \frac{1}{h_{n-1}} m_{n-1} + 2 \left( \frac{1}{h_0} + \frac{1}{h_{n-1}} \right) m_n \\ & = \frac{3}{h_0} f[x_0, x_1] + \frac{3}{h_{n-1}} f[x_{n-1}, x_n] \end{aligned}$$

用  $\left( \frac{1}{h_0} + \frac{1}{h_{n-1}} \right)$  除全式, 并记  $\mu_n = \frac{h_{n-1}}{h_0 + h_{n-1}}$ ,  $\lambda_n = \frac{h_0}{h_0 + h_{n-1}}$ ,  $g_n = 3(\mu_n f[x_0, x_1] + \lambda_n f[x_{n-1}, x_n])$ , 则上式化简为

$$\mu_n m_1 + \lambda_n m_{n-1} + 2m_n = g_n$$

于是可求得  $m_1, m_2, \dots, m_n$  的方程组为

$$\begin{bmatrix} 2 & \mu_1 & 0 & \cdots & 0 & 0 & 0 \\ \lambda_1 & 2 & \mu_2 & \cdots & 0 & 0 & 0 \\ 0 & \lambda_2 & 2 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \lambda_{n-1} & 2 & \mu_n \\ \mu_n & 0 & 0 & \cdots & 0 & \lambda_n & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_{n-1} \\ g_n \end{bmatrix} \quad (5.20)$$

由于  $m_i$  在力学上解释为细梁在  $x_i$  截面处的转角, 因此上述方程组 (5.18), (5.19) 和 (5.20) 称为三转角方程. 其系数矩阵中对角元素均为 2, 非对角元素  $\lambda_i + \mu_i = 1$ , 即系数矩阵具有所谓强对角优势, 在第 8 章中我们将看到, 这样的方程组解存在且唯一, 也可用追赶法解之. 从三转角方程解出  $m_i$  后, 代入 (5.16), 即得所求的三次样条插值函数  $S(x)$ .

## (2) 三弯矩方程

若取  $S''(x_i) = M_i (i = 0, 1, \dots, n)$  为待定参数, 并要求  $S(x)$  为分段三次多项式, 则  $S''(x)$  应为分段线性函数

$$S''(x) = \frac{x_{i+1} - x}{h_i} M_i + \frac{x - x_i}{h_i} M_{i+1}, \quad x_i \leq x \leq x_{i+1} \quad (5.21)$$

$i = 0, 1, \dots, n-1$ . 在每个子区间  $[x_i, x_{i+1}]$  上对上式积分两次, 并利用端点(插值)条件确定积分常数, 可得

$$\begin{aligned} S'(x) &= \frac{(x_{i+1} - x)^2}{2h_i} M_i + \frac{(x - x_i)^2}{2h_i} M_{i+1} \\ &\quad - \left( \frac{f_i}{h_i} - \frac{h_i M_i}{6} \right) + \left( \frac{f_{i+1}}{h_i} - \frac{h_i M_{i+1}}{6} \right) \end{aligned} \quad (5.22)$$

$$\begin{aligned} S(x) &= \frac{(x_{i+1} - x)^3}{6h_i} M_i + \frac{(x - x_i)^3}{6h_i} M_{i+1} \\ &\quad + (x_{i+1} - x) \left( \frac{f_i}{h_i} - \frac{h_i M_i}{6} \right) + (x - x_i) \left( \frac{f_{i+1}}{h_i} - \frac{h_i M_{i+1}}{6} \right) \end{aligned} \quad (5.23)$$

由此又可得

$$\begin{cases} S'(x_i + 0) = -\frac{h_i}{3} M_i - \frac{h_i}{6} M_{i+1} + \frac{f_{i+1} - f_i}{h_i} \\ S'(x_i - 0) = \frac{h_{i-1}}{6} M_{i-1} + \frac{h_{i-1}}{3} M_i + \frac{f_i - f_{i-1}}{h_{i-1}} \end{cases}$$

于是利用  $S'(x-0) = S'(x_i+0)$  再得

$$\begin{cases} \mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i \\ i = 1, 2, \dots, n-1 \end{cases} \quad (5.24)$$

其中  $\lambda_i, \mu_i$  与 (1) 中记号相同, 而

$$d_i = 6 \times \frac{f[x_i, x_{i+1}] - f[x_{i-1}, x_i]}{h_{i-1} + h_i} = 6f[x_{i-1}, x_i, x_{i+1}]$$

与 (1) 的情况一样, 只要加上边界条件之一, 即可获得两个端点方程. 如由边界条件 I, 可得

$$\begin{cases} 2M_0 + M_1 = \frac{6}{h_0}(f[x_0, x_1] - f'_0) \\ M_{n-1} + 2M_n = \frac{6}{h_{n-1}}(f'_n - f[x_{n-1}, x_n]) \end{cases}$$

由边界条件 II 可得

$$\begin{cases} M_0 = f''_0 \\ M_n = f''_n \end{cases}$$

这样与方程组 (5.24) 一起便构成求解  $M_0, M_1, \dots, M_n$  的方程组.  $M_i$  在力学上解释为细梁在  $x_i$  截面处的弯矩, 故称三弯矩方程. 同样地从三弯矩方程解出  $M_0, M_1, \dots, M_n$ , 代入 (5.23) 式即可得三次插值样条函数  $S(x)$ .

**例 1** 已知一组观测数据如下:

$$\begin{aligned} x &= 1, 2, 4, 5 \\ f(x) &= 1, 3, 4, 2 \end{aligned}$$

试求满足所给条件的自然样条函数, 并计算  $f(3), f(4.5)$  之值.

**解** 这是在自然边界条件  $S''(x_0) = S''(x_n) = 0$  下的插值问题, 故确定  $m_0, m_1, m_2, \dots, m_3$  的方程组为

$$\begin{bmatrix} 2 & 1 & & \\ \lambda_1 & 2 & \mu_1 & \\ & \lambda_2 & 2 & \mu_2 \\ & & 1 & 2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}$$

其中系数和自由项可计算如下:

$$h_0 = 2 - 1 = 1, \quad h_1 = 4 - 2 = 2, \quad h_2 = 5 - 4 = 1;$$

$$\lambda_1 = \frac{2}{1+2} = \frac{2}{3}, \quad \lambda_2 = \frac{1}{2+1} = \frac{1}{3};$$

$$\mu_1 = \frac{1}{1+2} = \frac{1}{3}, \quad \mu_2 = \frac{2}{2+1} = \frac{2}{3};$$

$$g_1 = 3 \left( \frac{2}{3} \times \frac{3-1}{1} + \frac{1}{3} \times \frac{4-3}{2} \right) = \frac{9}{2},$$

$$g_2 = 3 \left( \frac{1}{3} \times \frac{4-3}{2} + \frac{2}{3} \times \frac{2-4}{1} \right) = -\frac{7}{2},$$

$$g_0 = 3 \times \frac{3-1}{1} = 6, \quad g_3 = 3 \times \frac{2-4}{1} = -6.$$

于是可得

$$\begin{bmatrix} 2 & 1 & & \\ 2/3 & 2 & 1/3 & \\ & 1/3 & 2 & 2/3 \\ & & 1 & 2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 9/2 \\ -7/2 \\ -6 \end{bmatrix}$$

解之得  $m_0 = \frac{17}{8}, m_1 = \frac{7}{4}, m_2 = -\frac{5}{4}, m_3 = -\frac{19}{8}$ . 由此可知所求三次插值样条函数

$S(x)$  在  $[x_1, x_2]$  上的表达式为

$$\begin{aligned}
S_1(x) &= \frac{(x-x_2)^2[h_1+2(x-x_1)]}{h_1^3} f_1 + \frac{(x-x_1)^2[h_1+2(x_2-x_1)]}{h_1^3} f_2 \\
&\quad + \frac{(x-x_2)^2(x-x_1)}{h_1^2} m_1 + \frac{(x-x_1)^2(x-x_2)}{h_1^2} m_2 \\
&= \frac{(x-4)^2[2+2(x-2)]}{2^3} \times 3 + \frac{(x-2)^2[2+2(4-x)]}{2^3} \times 4 \\
&\quad + \frac{(x-4)^2(x-2)}{2^2} \times \frac{7}{4} + \frac{(x-2)^2(x-4)}{2^2} \times \left(-\frac{5}{4}\right) \\
&= -\frac{1}{8}x^3 + \frac{3}{8}x^2 + \frac{7}{4}x - 1
\end{aligned}$$

类似地可得  $S(x)$  在  $[x_0, x_1]$  与  $[x_2, x_3]$  上的表达式分别为

$$S_0(x) = -\frac{1}{8}x^3 + \frac{3}{8}x^2 + \frac{7}{4}x - 1$$

$$S_2(x) = \frac{3}{8}x^3 - \frac{45}{8}x^2 + \frac{103}{4}x - 33$$

或写成分段表达式

$$S(x) = \begin{cases} -\frac{1}{8}x^3 + \frac{3}{8}x^2 + \frac{7}{4}x - 1, & 1 \leq x \leq 2 \\ -\frac{1}{8}x^3 + \frac{3}{8}x^2 + \frac{7}{4}x - 1, & 2 \leq x \leq 4 \\ \frac{3}{8}x^3 - \frac{45}{8}x^2 + \frac{103}{4}x - 33, & 4 \leq x \leq 5 \end{cases}$$

利用  $S(x)$  可计算出

$$f(3) \approx S_1(3) = \frac{7}{4} = 4.25$$

$$f(4.5) \approx S_2(4.5) = 3.1406$$

在电子计算机上求解三次样条插值函数  $S(x)$  (包括计算插值、微商和积分)已经有各种现成的程序, 这些程序通常按照边界条件的不同提法来编制(参见 [C1]).

## 5-4 差分与均差·牛顿插值公式

### 1. 差分概念

差分在数值分析和其它学科中都有很多应用, 当插值节点是等距的情况时, 插值公式可以表示成简单的差分形式.

**定义 1** 设函数  $f(x)$  在等距节点  $x_i = x_0 + ih$  ( $i = 0, 1, \dots, n$ ) 上的值为  $y_i = f(x_i)$ , 则  $f(x)$  在每个小区间  $[x_i, x_{i+1}]$  上的改变量  $y_{i+1} - y_i$  称为  $f(x)$  在点  $x_i$  上的一阶差分或向前差分, 记为

$$\Delta y_i = y_{i+1} - y_i \quad (\text{或 } \Delta f(x_i) = f(x_i + h) - f(x_i))$$

对一阶差分再取一次差分就是二阶差分, 记为

$$\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i = (y_{i+2} - y_{i+1}) - (y_{i+1} - y_i) = y_{i+2} - 2y_{i+1} + y_i$$

一般地,  $n$  阶差分为

$$\Delta^n y_i = \Delta^{n-1} y_{i+1} - \Delta^{n-1} y_i$$

用数学归纳法可以证明有

$$\Delta^n y_0 = y_n - C_n^1 y_{n-1} + C_n^2 y_{n-2} - \cdots + (-1)^n y_0$$

式中的组合数  $C_n^i$  (有时记为  $C_n^i$  或  $\binom{n}{i}$ ) 为

$$C_n^i = \frac{n(n-1)\cdots(n-i+1)}{i!}$$

根据差分逐层递推的特点, 计算向前差分常用如下的向前差分表:

| $x_i$ | $y_i$ | $\Delta y_i$ | $\Delta^2 y_i$ | $\Delta^3 y_i$ | $\Delta^4 y_i$ |
|-------|-------|--------------|----------------|----------------|----------------|
| $x_0$ | $y_0$ |              |                |                |                |
|       |       | $\Delta y_0$ |                |                |                |
| $x_1$ | $y_1$ |              | $\Delta^2 y_0$ |                |                |
|       |       | $\Delta y_1$ |                | $\Delta^3 y_0$ |                |
| $x_2$ | $y_2$ |              | $\Delta^2 y_1$ |                | $\Delta^4 y_0$ |
|       |       | $\Delta y_2$ |                | $\Delta^3 y_1$ |                |
| $x_3$ | $y_3$ |              | $\Delta^2 y_2$ |                |                |
|       |       | $\Delta y_3$ |                |                |                |
| $x_4$ | $y_4$ |              |                |                |                |

**例 1** 设自变量  $x$  从  $-0.2$  变到  $0.2$ , 步长  $h$  为  $0.1$ , 则多项式  $f(x) = x^4 - 2x + 1$  的向前差分表为:

| $x_i$  | $y_i$  | $\Delta y_i$ | $\Delta^2 y_i$ | $\Delta^3 y_i$ | $\Delta^4 y_i$ |
|--------|--------|--------------|----------------|----------------|----------------|
| $-0.2$ | 1.4016 |              |                |                |                |
|        |        | $-2015$      |                |                |                |
| $-0.1$ | 1.2001 |              | $14$           |                |                |
|        |        | $-2001$      |                | $-12$          |                |
| $0.0$  | 1.0000 |              | $2$            |                | $24$           |
|        |        | $-1999$      |                | $12$           |                |
| $0.1$  | 0.8001 |              | $14$           |                |                |
|        |        | $-1985$      |                |                |                |
| $0.2$  | 0.6016 |              |                |                |                |

为了表示方便, 依惯例表中的差分值是以节点函数值最后一位小数做单位的.

**定义 2** 与向前差分 ( $\Delta y_i = y_{i+1} - y_i$ ) 相仿,  $f(x)$  在每个等长小区间  $[x_i, x_{i+1}]$  上的改变量  $y_{i+1} - y_i$  称为  $f(x)$  在  $x_{i+1}$  上的向后差分, 记为

$$\nabla y_{i+1} = y_{i+1} - y_i$$

$$\nabla^n y_{i+1} = \nabla^{n-1} y_{i+1} - \nabla^{n-1} y_i$$

向后差分表如下:

| $x_i$ | $y_i$ | $\nabla y_i$ | $\nabla^2 y_i$ | $\nabla^3 y_i$ | $\nabla^4 y_i$ |
|-------|-------|--------------|----------------|----------------|----------------|
| $x_0$ | $y_0$ |              |                |                |                |
|       |       | $\nabla y_1$ |                |                |                |
| $x_1$ | $y_1$ |              | $\nabla^2 y_1$ |                |                |
|       |       | $\nabla y_2$ |                | $\nabla^3 y_2$ |                |
| $x_2$ | $y_2$ |              | $\nabla^2 y_2$ |                | $\nabla^4 y_2$ |
|       |       | $\nabla y_3$ |                | $\nabla^3 y_3$ |                |
| $x_3$ | $y_3$ |              | $\nabla^2 y_3$ |                |                |
|       |       | $\nabla y_4$ |                |                |                |
| $x_4$ | $y_4$ |              |                |                |                |

关于差分有三个常用的运算符号,或称算子:

- 1) 恒等算子  $I \quad Iy_i = y_i$
- 2) 移位算子  $E \quad Ey_i = y_{i+1}$
- 3) 微分算子  $D \quad Dy_i = \frac{d}{dx} f(x) \Big|_{x=x_i}$

算子概念已经在现代科学和工程中经常出现,算子就是一种运算或变换,两个算子相等就是说它们作用于任意元素后结果都相等.

**定理** 算子  $I, E, D$  之间有如下关系:

- 1)  $\Delta = E - I$ .
- 2)  $E^n = (I + \Delta)^n = I + C_n^1 \Delta + \cdots + C_n^k \Delta^k + \cdots + \Delta^n$ .
- 3)  $E = e^{hD}$ .

**证** 1) 由  $(E - I)y_i = Ey_i - Iy_i = y_{i+1} - y_i = \Delta y_i$ , 又由  $i$  的任意性, 故知  $\Delta = E - I$ .

2) 以  $n = 2$  为例(一般情况可用数学归纳法), 由

$$E^2 y_i = E(Ey_i) = E(y_{i+1}) = y_{i+2},$$

又

$$\begin{aligned} (I + C_2^1 \Delta + \Delta^2)y_i &= Iy_i + C_2^1 \Delta y_i + \Delta^2 y_i \\ &= y_i + 2(y_{i+1} - y_i) + (y_{i+2} - 2y_{i+1} + y_i) = y_{i+2}, \end{aligned}$$

所以

$$E^2 = (I + \Delta)^2 = I + C_2^1 \Delta + \Delta^2.$$

$$\begin{aligned} 3) \quad Ey_i &= y_{i+1} = f(x_i) + [f(x_{i+1}) - f(x_i)] \\ &= f(x_i) + [f(x_i + h) - f(x_i)] \\ &= f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!} h^2 + \cdots + \frac{f^{(n)}(x_i)}{n!} h^n + \cdots \\ &= f(x_i) + hDf(x_i) + \frac{(hD)^2}{2!} f(x_i) + \cdots + \frac{(hD)^n}{n!} f(x_i) + \cdots \\ &= \left( I + hD + \frac{(hD)^2}{2!} + \cdots + \frac{(hD)^n}{n!} + \cdots \right) y_i \\ &= e^{hD} y_i. \end{aligned}$$

所以

$$E = e^{hD}.$$

## 2. 等距节点插值公式的差分形式

利用算子公式,可直接得等距节点插值公式的差分形式. 我们有

$$\begin{aligned} f(x_0 + nh) &= E^n y_0 \\ &= (I + C_n^1 \Delta + \cdots + C_n^k \Delta^k + \cdots + \Delta^n) y_0 \\ &= f(x_0) + \frac{n}{1!} \Delta f(x_0) + \cdots + \frac{n(n-1)\cdots(n-k+1)}{k!} \Delta^k f(x_0) \\ &\quad + \cdots + \Delta^n f(x_0) \end{aligned} \quad (5.25)$$

公式(5.25)称为**有限差分公式**. 当  $n$  取整数值即在节点处时,公式(5.25)给出  $f(x)$  的准确值,即有限差分公式通过节点. 当  $n$  取非整数值  $t$  时,公式(5.25)可写成

$$f(x_0 + th) = f(x_0) + \frac{t}{1!} \Delta f(x_0) + \cdots + \frac{t(t-1)\cdots(t-n+1)}{n!} \Delta^n f(x_0) + R_n(x) \quad (5.26)$$

$R_n(x)$  称为余项, 是插值公式的误差, 此插值公式称为牛顿 (Newton) 前插公式.

注意到过  $n+1$  个节点  $x_0, x_1, \cdots, x_n$  所对应的  $n$  次插值多项式的唯一性, 可知, 在等距节点的情况下, 牛顿前插公式与拉格朗日插值公式等同, 而且余项也应该等同. 在 (5.5) 中以  $x_0 + th$  代替  $x$ , 可得牛顿前插公式的余项为

$$R_n(x) = \frac{t(t-1)\cdots(t-n+1)}{(n+1)!} h^{n+1} f^{(n+1)}(\xi) \quad (5.27)$$

其中  $\xi$  在  $x_0$  与  $x_0 + nh$  之间.

除了牛顿前插公式, 还有牛顿后插公式. 将插值点顺序从大到小排列, 即

$$x_0, x_{-1} = x_0 - h, \cdots, x_{-k} = x_0 - kh, \cdots, x_{-n} = x_0 - nh$$

牛顿后插公式为

$$f(x_0 + th) = y_0 + \frac{t}{1!} \nabla y_0 + \frac{t(t+1)}{2!} \nabla^2 y_0 + \cdots + \frac{t(t+1)\cdots(t+n-1)}{n!} \nabla^n y_0 + R_n(x) \quad (5.28)$$

$$\text{式中余项} \quad R_n(x) = \frac{t(t+1)\cdots(t+n)}{(n+1)!} h^{n+1} f^{(n+1)}(\xi) \quad (5.29)$$

其中  $\xi$  在  $x_0 - nh$  与  $x_0$  之间,  $-n \leq t \leq 0$ .

根据插值公式的唯一性, 可知当插值点固定时, 前插与后插公式表示相同的多项式, 后插公式相同于倒过来的前插公式. 因此根据 (5.25) 计算终点附近的函数值变为计算初值附近的函数值.

**例 2** 六位三角函数表给出正弦值如下:

$$x = 0.6, \quad 0.7, \quad 0.8, \quad 0.9, \quad 1.0$$

$$\sin x = 0.564624, 0.644218, 0.717356, 0.783327, 0.841471$$

利用有限差分插值计算  $\sin 0.63$  的值, 并作余项估计.

**解** 先构造差分表:

| $x$ | $\sin x$ | $\Delta$ | $\Delta^2$ | $\Delta^3$ | $\Delta^4$ |
|-----|----------|----------|------------|------------|------------|
| 0.6 | 0.564624 |          |            |            |            |
|     |          | 79576    |            |            |            |
| 0.7 | 0.644218 |          | -6438      |            |            |
|     |          | 73138    |            | -729       |            |
| 0.8 | 0.717356 |          | -7167      |            | 60         |
|     |          | 65971    |            | -660       |            |
| 0.9 | 0.783327 |          | -7827      |            |            |
|     |          | 58144    |            |            |            |
| 1.0 | 0.841471 |          |            |            |            |

题中  $x_0 = 0.6$ ,  $t = 0.3$ ,  $h = 0.1$ , 由公式 (5.26) 可得



$$\begin{aligned}\sin 0.63 &\approx 0.564624 + 0.3 \times 0.079576 + \frac{1}{2} \times 0.3 \times (-0.7) \times (-0.006438) \\ &\quad + \frac{1}{6} \times 0.3 \times (-0.7) \times (-1.7) \times (-0.000729) + \frac{1}{24} \times 0.3 \\ &\quad \times (-0.7) \times (-1.7) \times (-2.7) \times 0.000069 \\ &= 0.589145\end{aligned}$$

注意到  $\sin x$  的五阶导数是  $\cos x$ , 即  $|\cos x| \leq 1$ , 因而有余项估计

$$10^{-5} \times 0.3 \times (-0.7) \times (-1.7) \times (-2.7) \times (-3.7) / 5! \leq 0.3 \times 10^{-6}$$

从结果可见, 当插值点靠近初值时, 牛顿前插公式效果良好. 但如果计算终点附近的函数值, 则由于从前算到后, 计算量大, 中间过程误差积累也多, 这时宜用后插公式.

牛顿前、后插公式还有这样的优点, 这就是在计算过程中需要增加节点时, 只要在原来的基础上增加相应的项即可. 例如, 利用  $x_0, x_1, x_2$  三个节点计算  $f(x)$  在  $\tilde{x} = x_0 + th$  上的函数值, 由牛顿前插公式知

$$f(\tilde{x}) \approx f(x_0) + \frac{t}{1!} \Delta f(x_0) + \frac{t(t-1)}{2!} \Delta^2 f(x_0)$$

如果算出上面结果 (记为  $S_2$ ) 后, 我们又希望增加一个节点  $x_3$ , 再计算函数值  $f(\tilde{x})$ , 则

$$\begin{aligned}f(\tilde{x}) &\approx f(x_0) + \frac{t}{1!} \Delta f(x_0) + \frac{t(t-1)}{2!} \Delta^2 f(x_0) + \frac{t(t-1)(t-2)}{3!} \Delta^3 f(x_0) \\ &= S_2 + \frac{t(t-1)(t-2)}{3!} \Delta^3 f(x_0)\end{aligned}$$

可见, 只要在原有基础上加上相应的项

$$\frac{t(t-1)(t-2)}{3!} \Delta^3 f(x_0)$$

而整个计算过程不必从头开始.

### 3. 均差概念与牛顿基本插值多项式

在插值节点  $x_0, x_1, \dots, x_n$  非等距的情况下, 我们不能用差分来简化插值多项式. 但是, 如果引入与差分概念相关的均差概念, 还是可以得到便于应用的插值多项式.

**定义 3** 函数  $y = f(x)$  在某个区间, 比如  $[x_i, x_{i+1}]$  上的平均变化率

$$\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

称为函数  $f(x)$  关于点  $x_i, x_{i+1}$  的一阶均差, 简称一阶均差, 并记为  $f[x_i, x_{i+1}]$ ①. 均差也称为差商.

相仿地, 一阶均差的平均变化率

$$\frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

称为函数  $f(x)$  的二阶均差, 并记为  $f[x_i, x_{i+1}, x_{i+2}]$ .

① 事实上, 上一节我们已用过这个记号. 在现代文献中, 也有的把均差记为  $[x_i, x_{i+1}]'$  的.

一般地,在定义了  $f(x)$  的  $m-1$  阶均差后,可定义  $f(x)$  的  $m$  阶均差为

$$f[x_0, x_1, \dots, x_m] = \frac{f[x_1, x_2, \dots, x_m] - f[x_0, x_1, \dots, x_{m-1}]}{x_m - x_0}$$

这就是说,高阶均差是由比它低一阶的两个均差组合而成.

例如,一阶均差

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \quad f[x_1, x_2] = \frac{f(x_2) - f(x_1)}{x_2 - x_1}, \dots$$

二阶均差

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}, \\ f[x_1, x_2, x_3] &= \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} \\ &\dots\dots \end{aligned}$$

类似于差分表,也可以构造均差表,如

| $x_k$    | $f(x_k)$ | 一阶均差          | 二阶均差               | 三阶均差                    |
|----------|----------|---------------|--------------------|-------------------------|
| $x_0$    | $f(x_0)$ |               |                    |                         |
| $x_1$    | $f(x_1)$ | $f[x_0, x_1]$ |                    |                         |
| $x_2$    | $f(x_2)$ | $f[x_1, x_2]$ | $f[x_0, x_1, x_2]$ |                         |
| $x_3$    | $f(x_3)$ | $f[x_2, x_3]$ | $f[x_1, x_2, x_3]$ | $f[x_0, x_1, x_2, x_3]$ |
| $\vdots$ | $\vdots$ | $\vdots$      | $\vdots$           |                         |

均差具有如下重要性质(证略):

1) 对称性,即任意调换节点的顺序,其值不变,如

$$f[x_0, x_1, x_2] = f[x_1, x_0, x_2] = f[x_2, x_1, x_0]$$

2) 均差与导数的关系有,当  $f(x)$  的  $m$  阶导数在包含节点比如  $x_0, x_1, \dots, x_m$  的某个区间上存在时,在  $x_0, x_1, \dots, x_m$  之间至少有一点  $\xi$ , 使得

$$f[x_0, x_1, \dots, x_m] = \frac{f^{(m)}(\xi)}{m!}$$

3) 均差与差分的关系有,当节点为等距节点时

$$f[x_0, x_1, \dots, x_m] = \frac{\Delta^m f(x_0)}{m! h^m}$$

其中,  $x_k = x_0 + kh$  ( $k = 0, 1, \dots, m$ ),  $h$  为步长,  $\Delta^m f(x_0) = \Delta^m y_0$  为函数  $y = f(x)$  在  $x_0$  处的  $m$  阶向前差分.

按照均差的定义及其对称性质,我们可以构造满足插值条件(5.3)的所谓牛顿基本插值多项式:

$$\begin{aligned} N_n(x) &= f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + \dots\dots + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1)\dots(x - x_{n-1}) \end{aligned} \quad (5.30)$$

事实上,容易看出,任何一个次数不超过  $n$  的多项式可以表示成形式

$$a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

其中  $a_i (i = 0, 1, \cdots, n)$  为待定系数. 这种形式的插值多项式称为牛顿插值多项式, 我们把它记为  $N_n(x)$ , 则根据插值条件 (5.3) 有  $N_n(x_0) = f_0 = f(x_0)$  立即可得

$$a_0 = f(x_0)$$

再由插值条件  $N_n(x_1) = f_1 = f(x_1)$ , 可得

$$f(x_1) = f(x_0) + a_1(x_1 - x_0)$$

故

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1]$$

又由插值条件  $N_n(x_2) = f_2 = f(x_2)$ , 可得

$$f(x_2) = f(x_0) + f[x_0, x_1](x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

故

$$\begin{aligned} a_2 &= \frac{f(x_2) - f(x_0) - f[x_0, x_1](x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{\frac{f(x_2) - f(x_0)}{x_2 - x_0} - f[x_0, x_1]}{x_2 - x_1} = \frac{f[x_0, x_2] - f[x_1, x_0]}{x_2 - x_1} \\ &= f[x_1, x_0, x_2] = f[x_0, x_1, x_2] \end{aligned}$$

一般地, 由插值条件  $N_n(x_i) = f_i = f(x_i) (i = 1, 2, \cdots, n)$ , 可得

$$a_i = f[x_0, x_1, \cdots, x_i] \quad (i = 1, 2, \cdots, n)$$

这就导出了牛顿基本插值多项式 (5.30).

牛顿基本插值多项式可用于求非等距节点下的函数值, 它还具有与牛顿前、后插公式类似的优点. 在利用电子计算机构造均差表时, 通常用二维数组, 比如  $T(N, M)$  来计算和存放各阶均差, 其中  $N$  是函数表中最后一个节点的下标  $n$ ,  $M$  是所要求出的均差的最高阶数, 数组元素  $T_{ji}$  则表示  $x_i$  为最后一个节点的  $j$  阶均差, 即

$$\begin{aligned} T_{ji} &= f[x_{i-i}, x_{i-i+1}, \cdots, x_i] \\ j &= 1, 2, \cdots, M; \quad i = j, j+1, \cdots, N \end{aligned}$$

**例 3** 已知  $f(x) = \operatorname{sh} x$  的函数表如下:

$$x = 0.40, \quad 0.55, \quad 0.65, \quad 0.80, \quad 0.90, \quad 1.05$$

$$f(x) = 0.41075, 0.57815, 0.69675, 0.88811, 1.02652, 1.25386$$

试作出牛顿基本插值多项式, 并计算  $f(0.596) = \operatorname{sh}(0.596)$  的值.

**解** 先构造如下均差表:

| $x_k$ | $f(x_k)$ | 一阶均差   | 二阶均差   | 三阶均差  | 四阶均差  |
|-------|----------|--------|--------|-------|-------|
| 0.40  | 0.41075  |        |        |       |       |
| 0.55  | 0.57815  | 1.1160 |        |       |       |
| 0.65  | 0.69675  | 1.1860 | 0.2800 |       |       |
| 0.80  | 0.88811  | 1.2757 | 0.3588 | 0.197 |       |
| 0.90  | 1.02652  | 1.3841 | 0.4336 | 0.214 | 0.034 |
| 1.05  | 1.25386  | 1.5156 | 0.5260 | 0.231 |       |

从表可看出,五阶均差将为零,因此,可作四次多项式.这时可选与  $x = 0.596$  靠近的五个节(前五个),于是得

$$\begin{aligned} N_4(x) = & 0.41075 + 1.1160(x - 0.40) + 0.2800(x - 0.40)(x - 0.55) \\ & + 0.197(x - 0.40)(x - 0.55)(x - 0.65) \\ & + 0.034(x - 0.40)(x - 0.55)(x - 0.65)(x - 0.80) \end{aligned}$$

而

$$f(0.596) \approx N_4(0.596) = 0.63192$$

## 5-5 曲线拟合的最小二乘法

### 1. 线性最小二乘拟合原理

在这一章的引言中已经说明了拟合问题的基本思想,现在给出拟合问题的一般提法:假设已知某物理过程(函数关系)  $y = f(x)$  的一组观测(实验)数据

$$(x_i, f_i) \quad i = 0, 1, \dots, m$$

要求在某特定函数类  $\{\varphi(x)\}$  (例如多项式)中找出一个函数  $F(x)$  作为  $y = f(x)$  的近似函数,使得在  $x_i$  上的误差(或称残差)

$$\delta_i = F(x_i) - f_i, \quad i = 0, 1, \dots, m$$

按某种度量标准最小,这就是拟合问题,也称为曲线拟合.

用范数表示,记向量  $\delta = [\delta_0, \delta_1, \dots, \delta_m]^T$ , 要求残差  $\delta_i$  按某种度量标准最小,即要求向量  $\delta$  的某种范数  $\|\delta\|$  最小.例如,可要求  $\delta$  的 1-范数  $\|\delta\|_1$  或最大范数  $\|\delta\|_\infty$ , 即

$$\|\delta\|_1 = \sum_{i=0}^m |\delta_i| = \sum_{i=0}^m |F(x_i) - f_i|$$

或

$$\|\delta\|_\infty = \max |\delta_i| = \max |F(x_i) - f_i|$$

最小.这本来都是很合适的,可惜计算并不方便,因此,通常要求  $\delta$  的欧几里德范数(2-范数)

$$\|\delta\|_2 = \sqrt{\sum_{i=0}^m \delta_i^2} = \sqrt{\sum_{i=0}^m [F(x_i) - f_i]^2}$$

即

$$\|\delta\|_2^2 = \sum_{i=0}^m \delta_i^2 = \sum_{i=0}^m [F(x_i) - f_i]^2 \quad (5.31)$$

最小.这种要求误差的平方和最小的拟合就称为曲线拟合的最小二乘法.

如果特定函数类  $\{\varphi(x)\}$  是指由  $n+1$  个在包含  $x_i (i = 0, 1, \dots, m)$  的一个区间上线性无关的有限函数系

$$\varphi_0, \varphi_1, \varphi_2, \dots, \varphi_n$$

(如幂函数系  $\{x^k\}$ 、指数函数系  $\{e^{kx}\}$ 、三角函数系  $\{\sin kx\}$  等)的线性组合,即所找的函数形式为

$$F(x) = a_0\varphi_0 + a_1\varphi_1 + \dots + a_n\varphi_n \quad (n \leq m) \quad (5.32)$$

它对于诸  $a_i$  而言是线性的,也即  $F(x)$  是线性模型,这时称为线性最小二乘拟合.

用最小二乘法求拟合曲线时,必须先选择函数类,即确定  $F(x)$  的形式.这与所讨论

问题的专门知识和经验有关. 有一种简单的处理方法是, 将数据  $(x_i, f_i)$  描绘在坐标纸上, 观察其分布规律, 从而设想  $F(x)$  的形式. 例如, 设想  $F(x)$  是一抛物线, 这时取幂函数系  $\varphi_0 = 1, \varphi_1 = x, \varphi_2 = x^2$ , 则  $F(x)$  具有形式  $F(x) = a_0 + a_1x + a_2x^2$ , 即二次多项式.

有时, 由于各数据的可靠性不同, 故需要权衡轻重而引进权函数, 即给定各点的比重  $\omega(x_i) > 0, i = 0, 1, \dots, m$  (例如,  $\omega(x_i)$  可用来表示数据在实验中重复出现的次数). 因此, 更一般的最小二乘准则是使

$$\|\delta\|_2^2 = \sum_{i=0}^m \omega(x_i) [F(x_i) - f_i]^2 \quad (5.33)$$

最小.

现在我们来求解线性最小二乘问题, 即求出形如 (5.32) 的函数  $F(x)$ , 使 (5.33) 取得最小值. 显然, 这相当于求多元函数

$$J(a_0, a_1, \dots, a_n) = \sum_{i=0}^m \omega(x_i) \left[ \sum_{j=0}^n a_j \varphi_j(x_i) - f_i \right]^2 \quad (5.34)$$

的极小点. 根据求多元函数极值的必要条件有

$$\begin{cases} \frac{\partial J}{\partial a_k} = 2 \sum_{i=0}^m \omega(x_i) \left[ \sum_{j=0}^n a_j \varphi_j(x_i) - f_i \right] \varphi_k(x_i) = 0 \\ k = 0, 1, \dots, n \end{cases} \quad (5.35)$$

采用内积记号

$$\begin{cases} (\varphi_j, \varphi_k) = \sum_{i=0}^m \omega(x_i) \varphi_j(x_i) \varphi_k(x_i) \\ (f_i, \varphi_k) = \sum_{i=0}^m \omega(x_i) f_i \varphi_k(x_i) = d_k \\ j, k = 0, 1, \dots, n \end{cases} \quad (5.36)$$

则 (5.35) 可写成

$$\begin{cases} \sum_{j=0}^n (\varphi_k, \varphi_j) a_j = d_k \\ k = 0, 1, \dots, n \end{cases} \quad (5.37)$$

这是包含  $n+1$  个未知数  $a_0, a_1, \dots, a_n$  的  $n+1$  阶线性代数方程组, 称为**法方程组**或**正则方程组**. 若记

$$\begin{aligned} \alpha &= [a_0, a_1, \dots, a_n]^T, & d &= [d_0, d_1, \dots, d_n]^T \\ G &= \begin{bmatrix} (\varphi_0, \varphi_0) & (\varphi_0, \varphi_1) & \cdots & (\varphi_0, \varphi_n) \\ (\varphi_1, \varphi_0) & (\varphi_1, \varphi_1) & \cdots & (\varphi_1, \varphi_n) \\ \cdots & \cdots & \cdots & \cdots \\ (\varphi_n, \varphi_0) & (\varphi_n, \varphi_1) & \cdots & (\varphi_n, \varphi_n) \end{bmatrix} \end{aligned}$$

法方程组还可写成矩阵形式

$$G\alpha = d \quad (5.38)$$

可以证明, 当  $\varphi_0, \varphi_1, \dots, \varphi_n$  线性无关时, 法方程 (5.38) 存在唯一解  $a_k^*(k = 0, 1, \dots, n)$ , 并且相应的函数

$$F(x) = a_0^* \varphi_0(x) + a_1^* \varphi_1(x) + \cdots + a_n^* \varphi_n(x) \quad (5.39)$$

就是要求的线性最小二乘解。

## 2. 多项式曲线拟合与指数曲线拟合

作为两种简单而常用的情形是多项式曲线拟合和指数曲线拟合。

以  $n$  次多项式作拟合函数,即在线性模型 (5.32) 中取函数系

$$\varphi_0 = 1, \quad \varphi_1 = x, \quad \varphi_2 = x^2, \quad \cdots, \quad \varphi_n = x^n$$

这时,相应的法方程组 (5.37) 为

$$\begin{bmatrix} \sum \omega_i & \sum \omega_i x_i & \cdots & \sum \omega_i x_i^n \\ \sum \omega_i x_i & \sum \omega_i x_i^2 & \cdots & \sum \omega_i x_i^{n+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum \omega_i x_i^n & \sum \omega_i x_i^{n+1} & \cdots & \sum \omega_i x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum \omega_i f_i \\ \sum \omega_i x_i f_i \\ \vdots \\ \sum \omega_i x_i^n f_i \end{bmatrix} \quad (5.40)$$

其中  $\sum$  均为  $\sum_{i=0}^m$  的简写,  $\omega_i = \omega(x_i)$ 。特别当  $\omega_i = 1, i = 0, 1, \cdots, m$  时, (5.40) 又可简化为

$$\begin{bmatrix} m & \sum x_i & \cdots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^{n+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum x_i^n & \sum x_i^{n+1} & \cdots & \sum x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum f_i \\ \sum x_i f_i \\ \vdots \\ \sum x_i^n f_i \end{bmatrix} \quad (5.41)$$

**例1** 已知一组观测数据如下:

$$x = 1, 3, 4, 5, 6, 7, 8, 9, 10$$

$$y = 10, 5, 4, 2, 1, 1, 2, 3, 4$$

求一代数多项式拟合曲线。

**解** 把已知数据描绘在坐标纸上,可以看出这些点大致形成一条抛物线。为此取

$$F(x) = a_0 + a_1 x + a_2 x^2$$

相应的法方程组为

$$\begin{bmatrix} a & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

计算各元素之值,代入可得

$$\begin{bmatrix} 9 & 53 & 381 \\ 53 & 381 & 3017 \\ 381 & 3017 & 25317 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 32 \\ 147 \\ 1025 \end{bmatrix}$$

解之得  $a_0 = 13.4597, a_1 = -3.6053, a_2 = 0.2676$  故所求拟合曲线为

$$F(x) = 13.4597 - 3.6053x + 0.2676x^2$$

关于指数曲线拟合,这里是指取拟合函数

$$F(x) = ae^{bx} \quad (a > 0)$$

其中  $a, b$  是待定参数。

对上述拟合函数两边取对数

$$\ln F = \ln a + bx$$

令

$$y = \ln F, \quad A = \ln a, \quad B = b$$

则得

$$y = A + Bx$$

这便成为标准线性函数,用多项式曲线拟合方法即可解此问题。一般来说,通过变换可以把某些非多项式曲线拟合化为多项式曲线拟合来求解。看例子。

**例2** 研究某化学反应。假设测得分解物的浓度与时间的关系有如下数据:

|                       |       |       |       |       |       |       |       |       |
|-----------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| 时间 $t$ (分)            | 1     | 2     | 3     | 4     | 5     | 6     | 7     | 8     |
| 浓度 $y \times 10^{-3}$ | 4.00  | 6.40  | 8.00  | 8.80  | 9.22  | 9.50  | 9.70  | 9.86  |
| 时间 $t$ (分)            | 9     | 10    | 11    | 12    | 13    | 14    | 15    | 16    |
| 浓度 $y \times 10^{-3}$ | 10.00 | 10.20 | 10.32 | 10.42 | 10.50 | 10.55 | 10.58 | 10.60 |

试建立浓度  $y$  与时间  $t$  的经验公式。

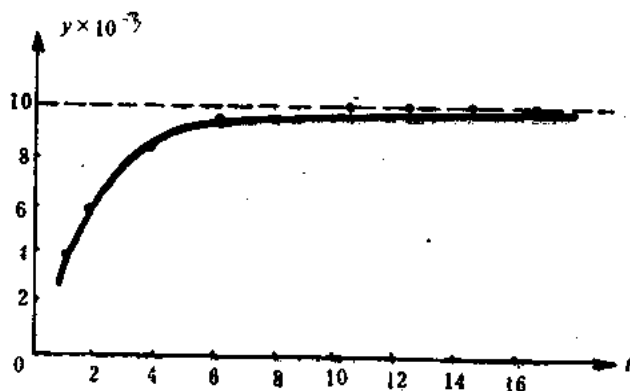


图 5-13

**解** 把已知数据  $(t, y_i), i = 1, 2, \dots, 16$  描绘在坐标纸上(图5-13),根据点的分布规律和问题的实际意义,可知拟合曲线  $y = F(t)$  具有下列大致特点:

- 1) 曲线单调升,上升速度由快到慢;
- 2) 当  $t \rightarrow \infty$  时,  $y$  趋于某常数,故有一水平渐近线;
- 3) 当  $t = 0$  时,反应未开始,浓度  $y = 0$ 。由此,我们可以估计  $y = F(t)$  是双曲型

$$y = \frac{t}{at + b}$$

上述模型是一个非线性模型,我们通过变量变换把它变为关于待定参数的线性模型。把上式写成

$$\frac{1}{y} = a + b \frac{1}{t}$$

令  $Y = \frac{1}{y}, T = \frac{1}{t}$ , 则上式化为线性函数

$$Y = a + bT$$

根据  $(t_i, y_i)$  算出对应的  $(T_i, Y_i)$ , 再按方程 (5.35) 解出  $a = 80.6621$ ,  $b = 161.6822$ , 于是可得时间与浓度的经验公式

$$y = \frac{t}{80.6621t + 161.6822}$$

然而, 根据数据的特点, 我们也可估计拟合曲线具有指数形式

$$y = ae^{\frac{b}{t}} \quad (a > 0, b < 0)$$

同样可把上式变换为线性函数。对上式两边取对数得

$$\ln y = \ln a + b \frac{1}{t}$$

令  $Y_i = \ln y$ ,  $T_i = \frac{1}{t}$ , 且记  $A = \ln a$ ,  $B = b$ , 则上式成为

$$Y_i = A + BT_i$$

类似地, 根据  $(t_i, y_i)$  算出对应的  $(T_i, Y_i)$ , 再按方程 (5.35) 解出  $A = -4.4807$ ,  $B = -1.0567$ , 进而算出  $a = e^A = 11.3253 \times 10^{-3}$ ,  $b = B = -1.0567$ , 于是可得时间与浓度的另一个经验公式为

$$y = 11.3253 \times 10^{-3} e^{-1.0567/t}$$

哪个公式好? 我们作下列比较:

| 经验公式                                               | 均方误差                  | 最大偏差                   |
|----------------------------------------------------|-----------------------|------------------------|
| $y = \frac{t}{80.6621t + 161.6822}$                | $1.19 \times 10^{-3}$ | $0.568 \times 10^{-3}$ |
| $y = 11.3253 \times 10^{-3} e^{-\frac{1.0567}{t}}$ | $0.34 \times 10^{-3}$ | $0.277 \times 10^{-3}$ |

可知后一个公式优于前一个公式。

多项式曲线拟合和指数曲线拟合都有专门的算法语言程序可供引用<sup>[63]</sup>; 后者也可化为前者来处理。对于多项式曲线拟合程序, 其中还要带一个解法方程组的子程序, 例如可用高斯消去法, 见第 8 章。

必须指出的是, 法方程组通常是“病态”的, 能处理的方程个数受到舍入误差的严格限制。在很多计算机上用单精度算法, 当  $n$  大于 5 以后就常常得出毫无意义的结果。双精度算法对保证最小二乘拟合的精度颇有助益, 可能时应采用双精度运算。另外, 为避免解法方程组, 建议使用下面将要介绍的正交多项式拟合程序。

### 3. 正交多项式曲线拟合及其 FORTRAN 程序

采用正交多项式作曲线拟合, 可以避免解“病态”的法方程组。先引入关于正交函数族的概念。为了便于编写程序, 下面所用记号与上一小节所用记号稍有不同。

**定义** 若函数族  $\{\varphi_j(x)\}$  ( $j = 0, 1, \dots, m$ ) 对于点集  $\{x_i\}$  ( $i = 1, 2, \dots, n$ ) 和权函数  $w(x_i) = w_i$  ( $i = 1, 2, \dots, n$ ) 具有性质

$$(\varphi_j, \varphi_k) = \sum_{i=1}^n w_i \varphi_j(x_i) \varphi_k(x_i) = \begin{cases} 0 & j \neq k \\ A_j & j = k \end{cases} \quad (5.42)$$



则  $\{\varphi_i(x)\}$  称为关于点集  $\{x_i\}$  带权  $w_i$  的正交函数族。特别当  $\varphi_i(x)$  是多项式时, 称为正交多项式。正交多项式下一章我们还要较详细地讲到。

设已知  $n$  对观测数据为

$$(x_i, y_i), \quad i = 1, 2, \dots, n \quad (5.43)$$

(其中限定  $x_i < x_{i+1}$ ); 与函数值的标准误差平方成反比的权(正数)为  $w_i, i = 1, 2, \dots, n$ ; 要拟合的线性模型为

$$F_m(x) = C_0\varphi_0(x) + C_1\varphi_1(x) + \dots + C_m\varphi_m(x) \quad (m+1 \leq n) \quad (5.44)$$

显然, 如果 (5.44) 中的有限函数系  $\varphi_i(x), i = 0, 1, \dots, m$  为正交函数族, 则法方程组的系数矩阵是对角线矩阵, 于是可立刻求得解为

$$C_j = \frac{\sum_{i=1}^n w_i y_i \varphi_j(x_i)}{\sum_{i=1}^n w_i \varphi_j^2(x_i)} \quad j = 0, 1, \dots, m \quad (5.45)$$

且可推导出平方误差

$$\|\sigma\|^2 = \|y\|^2 - \sum_{j=0}^m A_j (C_j)^2 \quad (5.46)$$

其中  $y = [y_1, y_2, \dots, y_n]^T, \sigma$  相当于 (5.33) 式中的  $\delta$ 。

这就是说, 求最小二乘解不必解法方程组, 而只需根据已知数据 (5.43) 和权函数  $w_i (i = 1, 2, \dots, n)$  去构造出正交函数族。

可以证明, 按递推公式

$$\begin{cases} P_0(x) = 1 \\ P_1(x) = (x - \alpha_1)P_0(x) \\ P_{j+1}(x) = (x - \alpha_{j+1})P_j(x) - \beta_j P_{j-1}(x) \\ j = 1, 2, \dots, m-1 \end{cases} \quad (5.47)$$

其中

$$\alpha_{j+1} = \frac{\sum_{i=1}^n w_i x_i P_j^2(x_i)}{\sum_{i=1}^n w_i P_j^2(x_i)}, \quad \beta_j = \frac{\sum_{i=1}^n w_i P_j^2(x_i)}{\sum_{i=1}^n w_i P_{j-1}^2(x_i)} \quad (5.48)$$

构造的  $\{P_j(x)\}$  是正交多项式。

这个事实用数学归纳法经过较冗长的运算可得到证明<sup>[A2]</sup>, 这里从略。

由此, 可得用正交多项式作曲线拟合的方法是, 在根据 (5.47) 和 (5.48) 逐步求出  $P_j(x)$  的同时, 按 (5.45) 算出  $C_j$ , 并再逐步把  $C_j P_j(x) (j = 0, 1, \dots, m)$  累加到  $F_m(x)$  中去, 最后就可得所求的拟合曲线(即广义多项式)

$$F_m(x) = C_0 P_0(x) + C_1 P_1(x) + \dots + C_m P_m(x) \quad (5.49)$$

或展开成等价的幂多项式

$$Q(x) = d_0 + d_1 x + \dots + d_m x^m$$

在这里, 可以事先指定拟合多项式的次数  $m$ , 也可在计算过程中, 通过估计误差来确定出所谓“最佳”次数  $h (h \leq m)$ 。其做法是, 先计算平均平方误差

$$\delta_h = \sigma_h^2 / (n - h - 1) \quad \left( \text{其中 } \sigma_h^2 = \sum_{i=1}^n w_i [F_h(x_i) - y_i]^2 \right)$$

如果当拟合多项式的次数逐次增加,  $h$  是第一次使  $\delta_h < \delta_{h+1}$  且  $0.6\delta_h < \delta_i (h+1 < i \leq m)$ , 则“最佳”次数是  $h$ ; 如果  $\delta_h < \delta_{h+1}$ , 但  $0.6\delta_h \geq \delta_i (h+1 < i)$ , 则“最佳”次数是  $i$ .

将多项式  $F_m(x)$  及其系数  $C_i$  代入平方误差  $\sigma_i^2$ , 就可得计算它的递推公式 (注意 (5.45) 式):

$$\begin{aligned}\sigma_i^2 &= \sum_{j=1}^n w_j [F_i(x_j) - y_j]^2 \\ &= \sum_{j=1}^n w_j [F_{i-1}(x_j) + C_i P_i(x_j) - y_j]^2 \\ &= \sigma_{i-1}^2 - 2C_i \sum_{j=1}^n w_j P_i(x_j) y_j + C_i^2 \sum_{j=1}^n w_j P_i^2(x_j) \\ &= \sigma_{i-1}^2 - C_i^2 \sum_{j=1}^n w_j P_i^2(x_j)\end{aligned}$$

下面是根据这种方法设计的一个 FORTRAN 子程序<sup>[C1, C2]</sup>.

设置哑元如下:

- N 整型变量, 输入参数; 观测数据  $(x_i, y_i)$  的个数.
- K 整型变量, 输入参数; 所要拟合的多项式的最高次数加 1.
- X 存放观测数据  $(x_i, y_i)$  的  $2 * N$  个元素的二维实型数组, 存放顺序是  $x_1, y_1, x_2, y_2, \dots, x_n, y_n$ ; 输入参数.
- W 存放相应的权函数  $w_1, w_2, \dots, w_n$  的  $N$  个元素的一维实型数组; 输入参数.
- SI 存放平均平方误差  $\delta$  的  $K$  个元素的一维实型数组; 输出参数.
- AF, BT 分别存放  $\alpha_i$  与  $\beta_i$  的  $K$  个元素的一维实型数组,  $i = 1, 2, \dots, K-1$ ; 工作单元.
- S, P 分别存放广义多项式系数和幂多项式系数的  $K$  个元素的一维实型数组; 输出参数.
- P1, P2, P3 均为  $K$  个元素的一维实型数组; 工作单元.
- P4, P5, P6 均为  $N$  个元素的一维实型数组; 工作单元.
- L 整型变量, 输入参数; 控制程序走向的信息, 当  $L = 1$  时, 用  $K-1$  次多项式进行拟合, 当  $L = 0$  时, 自动选取拟合多项式的“最佳”次数.

下面是 FORTRAN 子程序 LSFIT, 其中还将调用另一子程序 POLXY. 为了减少舍入误差的影响, 程序中把区间  $[x_1, x_n]$  变换成  $[-2, 2]$  进行计算. 子程序 POLXY 就用于把区间  $[-2, 2]$  上的幂多项式系数变换成  $[x_1, x_n]$  上的系数.

```
SUBROUTINE LSFIT (N, K, X, W, SI, S, P,
1 AF, BT, P1, P2, P3, P4, P5, P6, L)
    DIMENSION X(2, N), W(N), SI(K), S(K), P(K),
1 AF(K), BT(K), P1(K), P2(K), P3(K), P4(N), P5(N), P6(N)
    DO 10 I = 1, K
10 P1(I) = 0.0
    SIMIN = 0.0
```

```

LS = 1
BT(1) = 0.0
P4(1) = 0.0
P4(2) = 0.0
DE = 0.0
CM = 0.0
P2(1) = 1.0
TW = 0.0
LC = 0
DO 20 I = 1, N
P5(I) = 0.0
P6(I) = 1.0
DE = DE + W(I) * X(2, I) * * 2
CM = CM + W(I) * X(2, I)
20 TW = TW + W(I)
S(1) = CM/TW
P1(1) = S(1)
DE = DE - S(1) * CM
SI(1) = DE/(N - 1)
A = 4.0/(X(1, N) - X(1, 1))
B = - 2.0 - A * X(1, 1)
DO 30 I = 1, N
30 X(1, I) = A * X(1, I) + B
K1 = K - 1
DO 120 I = 1, K1
DU = 0.0
DO 40 J = 1, N
40 DU = DU + W(J) * X(1, J) * P6(J) * * 2
AF(I + 1) = - DU/TW
WL = TW
TW = 0.0
CM = 0.0
DO 50 J = 1, N
DU = BT(I) * P5(J)
P5(J) = P6(J)
P6(J) = (X(1, J) - AF(I + 1) * P6(J) - DU
TW = TW + W(J) * P6(J) * * 2
50 CM = CM + W(J) * X(2, J) * P6(J)
BT(I + 1) = TW/WL

```

```

      S(I + 1) = CM/TW
      DE = DE - S(I + 1) * CM
      SI(I + 1) = DE / (N - I - 1)
      IF (L .EQ. 1) GO TO 90
      IF (LC .EQ. 1) GO TO 120
      IF (LS .EQ. 0) GO TO 80
      IF (SI(I + 1) .LT. SI(I)) GO TO 90
      LS = 0
      SIMIN = SI(I)
      DO 60 J = 1, K
60    P3(J) = P1(J)
      GO TO 90
80    IF (SI(I + 1) .LT. 0.6 * SIMIN) LC = 1
90    DO 100 J = 1, I
      DU = P4(J + 1) * BT(I)
      P4(J + 1) = P2(J)
      P2(J) = P4(J) - AF(I + 1) * P2(J) - DU
100   P1(J) = P1(J) + S(I + 1) * P2(J)
      P1(I + 1) = S(I + 1)
      P2(I + 1) = 1.0
      P4(I + 2) = 0.0
      IF (LC .EQ. 1) GO TO 120
      IF (LS .EQ. 1) GO TO 120
      IF (I .NE. K - 1) GO TO 120
      DO 110 J = 1, K
110   P1(J) = P3(J)
120   CONTINUE
      CALL POLYX (A, B, P1, P, K, P4, P5)
      RETURN
      END
      SUBROUTINE POLYX (A, B, C, D, N, Z, W)
      DIMENSION C(N), D(N), Z(N), W(N)
      W(1) = 1.0
      Z(1) = 1.0
      D(1) = C(1)
      DO 1 I = 2, N
      W(I) = 1.0
      Z(I) = B * Z(I - 1)
1    D(I) = D(I) + C(I) * Z(I)

```

```

DO 2 J = 2, N
W(1) = W(1) * A
D(J) = C(J) * W(1)
K = 2
J1 = J + 1
DO 2 I = J1, N
W(K) = A * W(K) + W(K - 1)
D(J) = D(J) + C(I) * W(K) * Z(K)
2 K = K + 1
RETURN
END

```

**算例** 为了方便,我们姑且从函数  $y = e^x$  取其在  $x_i = -1 + 0.1i$  上的对应值  $y_i$  ( $i = 0, 1, \dots, 20$ ) 作为观测数据,求次数  $\leq 9$  的拟合多项式,其中取权  $w_i = 1$ .

解题程序将编写如下. 我们要求程序能自动选取拟合多项式的“最佳”次数,即程序中 L 取 0.

```

DIMENSION X(2, 21), W(21), SI(10), S(10), P(10),
1 AF(10), BT(10), P1(10), P2(10), P3(10), P4(21), P5(21), P6(21)
DO 200 I = 1, 21
X(1, I) = -1.0 + 0.1 * (I - 1)
X(2, I) = EXP(-1.0 + 0.1 * (I - 1))
200 CONTINUE
WRITE (6, 222)
222 FORMAT (1X, 2H_I, 8X, 2HAF, 16X, 2HBT, 16X, 2HSI,
1 16X, 1HP, 18X, 1HS)
CALL LSFIT (21, 10, X, W, SI, S, P, AF, BT,
1 P1, P2, P3, P4, P5, P6, 0)
DO 300 I = 1, 10
WRITE (6, 333) I, AF(I), BT(I), SI(I), S(I), P(I)
300 CONTINUE
333 FORMAT (1X 2I, 5E18.8)
STOP
END

```

C

```

SUBROUTINE LSFIT (N, K, X, W, SI, S, P, AF, BT,
1 P1, P2, P3, P4, P5, P6, L)
: (本子程序段体)
END
SUBROUTINE POLXY (A, B, C, D, N, Z, W)
: (本子程序段体)
END

```

计算和输出结果如下表所示. 从表中可见, 当  $l$  从 8 到 9 时,  $SI$  (即平均平方误差  $\delta$ ) 之值第一次使得前者小于后者, 即有  $SI(8) < SI(9)$ , 且有  $0.6 \times SI(8) < SI(10)$ , 故可知拟合多项式的“最佳”次数为 7.

| $l$ | AF              | BT             | SI             | S              | P              |
|-----|-----------------|----------------|----------------|----------------|----------------|
| 1   | 0.00000000E+00  | 0.00000000E+00 | 0.51153586E+00 | 0.11936518E+01 | 0.10000000E+01 |
| 2   | 0.34647417E-11  | 0.14666666E+01 | 0.35495936E-01 | 0.55701822E+00 | 0.99999999E+00 |
| 3   | 0.47246478E-11  | 0.11653333E+01 | 0.11019217E-02 | 0.13504659E+00 | 0.49999997E+00 |
| 4   | -0.16217326E-11 | 0.11108571E+01 | 0.19123321E-04 | 0.22120425E-01 | 0.16666666E+00 |
| 5   | -0.14598930E-11 | 0.10793650E+01 | 0.20945589E-06 | 0.27342756E-02 | 0.41666853E-01 |
| 6   | 0.94678359E-11  | 0.10505050E+01 | 0.15960379E-08 | 0.27129180E-03 | 0.83333312E-02 |
| 7   | 0.10943934E-10  | 0.10195804E+01 | 0.46723672E-10 | 0.22476463E-04 | 0.13883299E-02 |
| 8   | 0.34095480E-10  | 0.98502564E+00 | 0.41396492E-10 | 0.15982276E-05 | 0.19839081E-02 |
| 9   | 0.10191841E-09  | 0.94619607E+00 | 0.44810755E-10 | 0.99501674E-07 | 0.25472428E-04 |
| 10  | 0.30247942E-09  | 0.90278637E+00 | 0.48884354E-10 | 0.54739370E-08 | 0.28026557E-05 |

## 习 题 五

1. 已知函数表:

| $x$      | 1 | 3 |
|----------|---|---|
| $f_1(x)$ | 1 | 2 |

| $x$      | 1 | 3 | 4   |
|----------|---|---|-----|
| $f_2(x)$ | 1 | 2 | 1.5 |

试分别求其近似插值表达式, 并求  $f_1(1.5)$  和  $f_2(1.5)$  的近似值.

2. 证明:

(1) 若  $\omega(x) = (x - x_0)(x - x_1)$ , 则

$$|x - x_0| |x - x_1| \leq \frac{1}{4} (x_1 - x_0)^2$$

(2) 线性插值余项  $R_1(f, x)$  有估计式

$$|R_1(f, x)| \leq \frac{M_2}{8} (x_1 - x_0)^2 \quad (M_2 = \max_{x \in [a, b]} |f''(x)|)$$

3. 已知连续函数  $\varphi(x)$  的函数值表如下:

| $x$          | -1 | 0  | 1 | 2 |
|--------------|----|----|---|---|
| $\varphi(x)$ | -2 | -1 | 1 | 2 |

求方程  $\varphi(x) = 0$  在  $[-1, 2]$  内的近似根. [提示: 由于函数值严格单调下降, 故可用反插值法, 即把  $\varphi(x_i)$  看作节点,  $x_i$  看作函数表, 把零看成插值点].

4. 在  $[0, \pi]$  上取点  $0, \frac{\pi}{2}, \pi$  作  $\sin x$  的抛物插值多项式  $F_2(x)$ , 并画出  $y = F_2(x)$  的图形. 同

时也画出  $\sin x$  的级数近似值  $y = x$  和  $y = x - \frac{x^3}{6}$  的图形. 比较它们之间的近似情况.

5. 证明下列结果:

(1) 设  $l_i(x)$  是  $n$  次拉格朗日插值基函数, 则有

$$\sum_{i=0}^n l_i(x) = 1$$

(2) 设  $f(x)$  是次数不超过  $n$  的多项式,  $F(x)$  是  $f(x)$  的  $n$  次拉格朗日插值多项式, 则有

$$f(x) = F(x)$$

(3) 设  $l_i(x)$  是  $n$  次拉格朗日插值基函数, 又  $x_0 < x_1 < \cdots < x_n$ , 则有

$$\sum_{i=0}^n x_i^k l_i(x) = x^k \quad (0 \leq k \leq n)$$

6. 已知函数表

| $x$     | $x_0$ | $x_1$  | $x_2$ |
|---------|-------|--------|-------|
| $f(x)$  | $f_0$ |        | $f_2$ |
| $f'(x)$ |       | $f'_1$ |       |

求一个二次插值函数  $H(x)$ , 使其满足条件:

$$H(x_0) = f_0, \quad H(x_1) = f_1, \quad H'(x_1) = f'_1$$

7. 求满足条件

| $x$     | 1 | 2 | 3  |
|---------|---|---|----|
| $f(x)$  | 2 | 4 | 12 |
| $f'(x)$ |   | 3 |    |

的插值多项式及其余项.

8. 求  $f(x) = x^2$  在  $[a, b]$  上的分段线性插值函数, 并估计误差.

9. 给定节点  $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ , 设  $h = \max_{0 \leq i \leq n-1} (x_{i+1} - x_i)$ , 若  $f(x)$  在  $[a, b]$

上连续,  $F_h(x)$  是  $f(x)$  的分段线性插值函数, 证明:

$$\lim_{h \rightarrow 0} F_h(x) = f(x)$$

10. 在某处测得海洋不同深度处水温如下:

| 深度(米)                    | 466  | 714  | 950  | 1422 | 1634 |
|--------------------------|------|------|------|------|------|
| 水温( $^{\circ}\text{C}$ ) | 7.04 | 4.28 | 3.40 | 2.54 | 2.13 |

(1) 写出分段线性插值函数, 并求在深度 500 米、1000 米、1500 米处的水温.

(2) 用分段二次插值求深度 500 米、800 米、1000 米、1200 米和 1500 米处的水温 (用计算机程序求解).

(3) 用三次牛顿插值求深度 1000 米处的水深.

11. 已知  $f(x)$  的函数表

| $x$    | 0 | 1 | 2 | 3 |
|--------|---|---|---|---|
| $f(x)$ | 0 | 2 | 3 | 6 |

求在区间  $[0, 3]$  上、边界条件为  $f'(0) = 1$ ,  $f'(3) = 0$  的三次样条插值函数.

12. 已知  $f(x)$  的函数表

| $x$    | 0 | 1 | 2 | 3  |
|--------|---|---|---|----|
| $f(x)$ | 0 | 2 | 3 | 16 |

求其边界条件为  $f''(0) = 0$ ,  $f''(3) = 6$  的三次样条插值函数.

13. 设  $f(x) \in C^1[a, b]$ ,  $a = x_0 < x_1 < \dots < x_{n-1} < x_n = b$ ,  $S_f(x)$  为  $f(x)$  的三次样条插值提法 1 的解,  $S(x)$  为任一以  $x_i (i=0, 1, \dots, n)$  为样条节点的三次样条函数, 证明:

$$(1) \int_a^b (f''(x) - S''(x))^2 dx = \int_a^b (f''(x) - S''_f(x))^2 dx + \int_a^b (S''_f(x) - S''(x))^2 dx \\ + 2 \int_a^b (f''(x) - S''_f(x))(S''_f(x) - S''(x)) dx$$

$$(2) \int_a^b (f''(x) - S''_f(x))(S''_f(x) - S''(x)) dx = 0$$

$$(3) \int_a^b (f''(x) - S''_f(x))^2 dx \leq \int_a^b (f''(x) - S''(x))^2 dx$$

14. 已知函数表如下:

| $x$    | -1 | 0 | 1  | 2 | 3 |
|--------|----|---|----|---|---|
| $f(x)$ | 0  | 2 | -1 | 0 | 1 |

试选用合适的等距插值多项式计算  $f\left(-\frac{1}{2}\right)$  和  $f\left(\frac{5}{2}\right)$  的近似值.

15. 已知函数表

| $x$ | 0 | 1  | 3 | 4 | 6  |
|-----|---|----|---|---|----|
| $y$ | 0 | -7 | 5 | 8 | 14 |

求牛顿基本插值多项式.

16. 设  $y = f(x)$  在  $[x_0, x_n]$  上有连续的  $n$  阶导数, 求证在等距节点时, 差分与导数存在下列关系:

$$(1) \Delta^k y_0 = h^k f^{(k)}(\xi), \quad x_0 < \xi < x_n$$

$$(2) \lim_{h \rightarrow 0} \frac{\Delta^k y_0}{h^k} = f^{(k)}(x_0)$$

17. 证明下列等式:

(1) 用数学归纳法证明

$$\Delta^k y_i = y_{i+k} - C_k^1 y_{i+k-1} + C_k^2 y_{i+k-2} - \dots + (-1)^k y_i$$

(2) 按差分定义, 证明

$$\Delta^k y_0 + \Delta^k y_1 + \dots + \Delta^k y_n = \Delta^{k-1} y_{n+1} - \Delta^{k-1} y_0$$

18. 对于给定的函数表

| $x$    | 1 | 2  | 3  | 4   | 5   |
|--------|---|----|----|-----|-----|
| $f(x)$ | 6 | 10 | 46 | 138 | 430 |

分别构造向前差分表和向后差分表.

19. 证明:  $n$  次多项式  $P(x)$  的一阶差分是  $n-1$  次多项式, 它的  $n$  阶差分为常数, 而  $n+1$  阶差分等于零.

20. 证明均差的下列性质:

(1) 若  $F(x) = C_1 f_1(x) + C_2 f_2(x)$ , 其中  $C_1, C_2$  为任意常数, 则

$$F[x_0, x_1, \dots, x_n] = C_1 f_1[x_0, x_1, \dots, x_n] + C_2 f_2[x_0, x_1, \dots, x_n]$$

(2) 设  $f(x) = x^2 + x^4 + 3x + 1$ , 则

$$f[2^0, 2^1, \dots, 2^7] = 1; \quad f[2^0, 2^1, \dots, 2^7, 2^8] = 0$$

21. 设实验所得数据如下:



(20, 4.35), (40, 7.55), (60, 10.40)  
(80, 13.80), (100, 16.80)

试求适合上述关系的近似公式.

22. 已知观测数据如下:

| $x$ | 0   | 1   | 2   | 3   | 5   |
|-----|-----|-----|-----|-----|-----|
| $y$ | 1.1 | 1.9 | 3.1 | 3.9 | 4.9 |

试用最小二乘法求经验直线  $y = ax + b$

23. 已知观测数据

| $x$    | 0    | 0.50 | 1.25 | 2.00 | 2.70 | 3.00 | 3.50 | 3.90 | 4.75 | 5.25 |
|--------|------|------|------|------|------|------|------|------|------|------|
| $f(x)$ | 1.37 | 1.48 | 2.09 | 2.77 | 3.60 | 4.10 | 4.88 | 6.01 | 7.95 | 9.90 |

试求拟合曲线.

24. 利用最小二乘法, 求解矛盾方程组:

$$\begin{cases} x + y = 3.0 \\ 2x - y = 0.2 \\ x + 3y = 7.0 \\ 3x + y = 5.0 \end{cases}$$

25. 重复  $n$  次测量一个长度, 因而获得  $n$  个数值

$$x_1, x_2, \dots, x_n$$

通常, 我们取平均值

$$\bar{x} = \frac{1}{n} (x_1 + x_2 + \dots + x_n)$$

作为所求的长度值. 试解释这种做法的道理.

26. 使电流通过 2 欧姆的电阻, 用伏特表测量电阻两端的电压, 可得如下数据:

| $i$ (安培) | 1   | 2   | 4   | 6    | 8    | 10   |
|----------|-----|-----|-----|------|------|------|
| $v$ (伏特) | 1.8 | 3.7 | 8.2 | 12.0 | 15.8 | 20.2 |

试用最小二乘法建立  $i$  与  $v$  之间的经验公式.

27. 某实验需要观察水份的渗透速度. 今测得时间  $t$  与水的重量的数据如下:

| $t$ (秒) | 1    | 2    | 4    | 8    | 16   | 32   | 64   |
|---------|------|------|------|------|------|------|------|
| $W$ (克) | 4.22 | 4.02 | 3.85 | 3.59 | 3.44 | 3.02 | 2.59 |

已知  $t$  与  $W$  之间有关系  $W = Ct^\lambda$ , 试用最小二乘法确定参数  $C$  和  $\lambda$ .

28. 编写完成拉格朗日内插的计算机 FORTRAN 程序. 这个程序的输入包括任意数目 ( $n$  个或更少), 任意间隔的  $x$  值, 这些值的数目,  $f(x)$  的相应值, 以及所需内插  $f(x)$  处的  $x$  值. 应用此程序, 对下面函数

| $x$    | 0     | 1.2   | 1.7   | 2.8    | 4.4    | 5.8   | 7.0   | 8.0   |
|--------|-------|-------|-------|--------|--------|-------|-------|-------|
| $f(x)$ | 1.000 | 0.671 | 0.398 | -0.185 | -0.342 | 0.092 | 0.300 | 0.172 |

求出  $f(6.3)$ 。

29. 编写完成牛顿前插的计算机 FORTRAN 程序。这个程序的输入包括：等间距的  $x$  值若干个 (8 个或更少), 相应的  $f(x)$ , 待求的  $f(x)$  的  $x$ 。其输出应包括：差分表, 所得的  $f(x)$  内插值。应用此程序, 对下面函数

| $x$    | 0 | 1    | 2    | 3     | 4     | 5     | 6     | 7    |
|--------|---|------|------|-------|-------|-------|-------|------|
| $f(x)$ | 4 | -250 | -881 | -1667 | -2357 | -2493 | -1295 | 2450 |

求出  $f(1.3)$ 。

30. 已知下列观测数据:

|        |      |      |      |      |      |      |      |      |      |
|--------|------|------|------|------|------|------|------|------|------|
| $x$    | 0    | 1.0  | 1.5  | 2.3  | 2.5  | 4.0  | 5.1  | 6.0  |      |
| $f(x)$ | 0.2  | 0.8  | 2.5  | 2.5  | 3.5  | 4.3  | 3.0  | 5.0  |      |
| $x$    |      | 6.5  | 7.0  | 8.1  | 9.0  | 9.2  | 11.0 | 11.3 |      |
| $f(x)$ |      | 3.5  | 2.4  | 1.3  | 2.0  | -0.3 | -1.3 | -3.0 |      |
| $x$    | 12.1 | 13.1 | 14.0 | 15.5 | 16.0 | 17.5 | 17.8 | 19.0 | 20   |
| $f(x)$ | -4.0 | -4.9 | -4.0 | -5.2 | -3.0 | -3.5 | -1.6 | -1.4 | -0.1 |

试利用正交多项式拟合程序, 为这些数据进行适宜的多项式拟合。另外, 把这些数据画在坐标纸上, 观察数据可能遵循的规律, 并通过解法方程组的途径 (最好用双精度), 试就下列逼近函数形式:

$$F_1(x) = ax^3 + bx^2 + cx + d$$

$$F_2(x) = A + B \sin \frac{\pi}{10} x$$

$$F_3(x) = C \sin Dx$$

分别拟合上述数据, 然后再把所得的拟合曲线连同已知数据一起画在一张图上。

## 第6章 数值积分与数值微分

### 6-1 引言

我们知道,定积分

$$I = \int_a^b f(x) dx \quad (6.1)$$

的计算,在被积函数  $f(x)$  的原函数能由明显表达式  $F(x)$  给出时,可用牛顿-莱布尼兹公式

$$\int_a^b f(x) dx = F(b) - F(a)$$

来做。可惜,在许多实际问题中,被积函数的原函数不一定能由明显表达式给出,有些即使能,又由于过分复杂而不便计算。另外,在有些问题中,被积函数还可能只是一个函数表,或者定义为某个微分方程的解,而这个微分方程又不能明显解出。因此,实际计算中,人们常常采用另一类计算方法,这就是数值积分法。

数值积分法,就是从近似计算的角度,采用某种数值过程来求出定积分的近似值。

在微积分学的基础教程中,根据定积分的几何意义,通常导出一些简单的数值积分法,如矩形公式、梯形公式和抛物线公式。这一章我们将进一步研究一些计算机上常用的数值积分法,并在最后简单介绍常用的数值微分法。

根据积分中值定理,满足一定条件的定积分(6.1),在积分区间  $[a, b]$  内至少存在一点  $\xi$ ,使得

$$I = \int_a^b f(x) dx = (b-a)f(\xi) \quad (6.2)$$

即所求曲边梯形的面积恰好等于底为  $(b-a)$ 、高为  $f(\xi)$  的矩形面积(图 6-1),然而,这里的问题在于  $\xi$  的具体位置一般不知道,因而  $f(\xi)$  无从计算,从而导出的公式未有实用价值。不过,如果我们把  $f(\xi)$

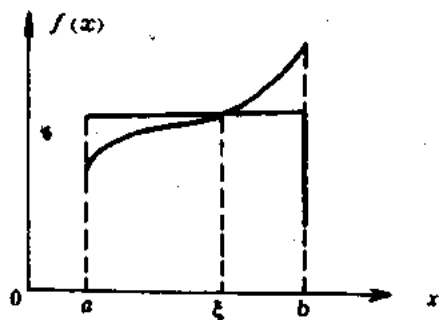


图 6-1

看作  $[a, b]$  上的平均高度,那么,只要能对平均高度  $f(\xi)$  提出一种算法,则根据公式(6.2)相应地便可得到一种数值积分方法。

例如,取  $f(a)$  与  $f(b)$  的算术平均值  $\frac{f(a) + f(b)}{2}$  作为平均高度  $f(\xi)$ ,便得梯形公式

$$\int_a^b f(x) dx \approx \frac{b-a}{2} [f(a) + f(b)]$$

取区间中点  $\frac{a+b}{2}$  的函数值  $f\left(\frac{a+b}{2}\right)$  作为平均高度  $f(\xi)$ ,便得(中)矩形公式

$$\int_a^b f(x) dx \approx (b-a)f\left(\frac{a+b}{2}\right)$$

一般地,在区间  $[a, b]$  上取若干个节点  $x_i (i = 0, 1, \dots, n)$ , 用  $f(x_i)$  的加权平均

$\frac{\sum_{i=0}^n A_i f(x_i)}{b-a}$  作为平均高度  $f(\xi)$ , 便得一般求积公式

$$\int_a^b f(x) dx \approx (b-a) \frac{\sum_{i=0}^n A_i f(x_i)}{b-a} = \sum_{i=0}^n A_i f(x_i) \quad (6.3)$$

其中  $x_i$  称为求积节点,  $A_i$  称为求积系数或相应于  $x_i$  的权. 显然, 为了构造这类求积公式, 原则上只是确定参数  $x_i$  和  $A_i (i = 0, 1, \dots, n)$  的问题, 利用这类积分公式, 便把积分计算转化为函数值计算.

构造求积公式的一种基本方法是: 设被积函数  $f(x)$  在一组节点

$$a = x_0 < x_1 < \dots < x_n = b$$

上的值为

$$f(x_0), f(x_1), \dots, f(x_n)$$

我们用  $n$  次拉格朗日插值多项式

$$L_n(x) = \sum_{i=0}^n f(x_i) l_i(x) \quad (6.4)$$

其中  $l_i(x)$  为  $n$  次插值基函数

$$l_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{x - x_k}{x_i - x_k} \quad (i = 0, 1, \dots, n) \quad (6.5)$$

代替被积函数  $f(x)$ , 于是得

$$\begin{aligned} \int_a^b f(x) dx &\approx \int_a^b L_n(x) dx = \int_a^b \left[ \sum_{i=0}^n f(x_i) l_i(x) \right] dx \\ &= \sum_{i=0}^n \left[ \int_a^b l_i(x) dx \right] f(x_i) = \sum_{i=0}^n A_i f(x_i) \end{aligned} \quad (6.6)$$

其中求积系数

$$A_i = \int_a^b l_i(x) dx = \int_a^b \prod_{\substack{k=0 \\ k \neq i}}^n \frac{x - x_k}{x_i - x_k} dx \quad (6.7)$$

通常, (6.6) 式称为由 (6.7) 式确定的插值型求积公式.

通过插值余项估计, 可以导出插值型求积公式的余项估计(截断误差)为

$$E_n = \int_a^b f(x) dx - \sum_{i=0}^n A_i f(x_i) = \int_a^b \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega(x) dx \quad (6.8)$$

其中  $\omega(x) = \prod_{i=0}^n (x - x_i)$ ,  $\xi \in (a, b)$  且依赖于  $x$ .

一般地, 引入代数精度的概念来表示求积公式的近似程度. 容易理解, 如果一个求积公式能对“较多”的函数(比如多项式)准确地成立(多项式的积分能准确求出), 自然就可认为它有“较好”的精度. 因此, 引入如下定义:

**定义** 如果一个求积公式对于次数小于等于  $m$  的多项式都能准确成立, 但对于  $m+1$  次多项式不能准确成立, 便称这个求积公式具有  $m$  次代数精度.

根据插值型求积公式的余项估计(6.8)式,可知由次数 $\leq n$ 的多项式 $f(x)$ 构成的插值型求积公式的余项为0,因此,可见这时求积公式至少具有 $n$ 次代数精度。

## 6-2 梯形求积公式与辛普生求积公式

梯形求积公式与辛普生求积公式是最基本的求积公式,它们都是等距节点的插值型求积公式。

### 1. 梯形求积公式

设所求积分为

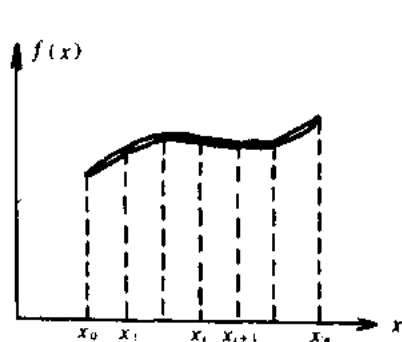


图 6-2

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

将积分区间 $[a, b]$ 分成 $n$ 个等长度小区间

$$[x_0, x_1], [x_1, x_2], \dots, [x_{n-1}, x_n]$$

其中 $x_i = a + ih; i = 0, 1, \dots, n; h = \frac{b-a}{n}$ 称为步长。用过点 $(x_i, f(x_i)), (x_{i+1}, f(x_{i+1}))$ 的线性插值函数 $L_1(x)$ 来近似被积函数 $f(x)$ 的弧段,即用小梯形面积来近似小曲边梯形面积(图 6-2),于是可得小区间上的积分近似值

$$\int_{x_i}^{x_{i+1}} f(x) dx \approx \frac{h}{2} [f(x_i) + f(x_{i+1})]$$

因为 $\int_a^b f(x) dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x) dx$ ,故得整个区间上的积分近似值

$$\begin{aligned} \int_a^b f(x) dx &\approx h \left[ \frac{f(a)}{2} + f(a+h) + f(a+2h) + \dots + f(a+(n-1)h) \right. \\ &\quad \left. + \frac{f(b)}{2} \right] \end{aligned} \quad (6.9)$$

这就是梯形求积公式。

梯形求积公式的截断误差分析如下:如果 $f(x)$ 本来就是线性函数或者是顶点在 $(x_i, f(x_i))$ 处的分段线性函数,则梯形求积公式的截断误差显然为0,即梯形求积公式对于线性函数是准确成立的,也即梯形求积公式至少具有一次代数精度。如果 $f(x)$ 是非线性函数,则梯形求积公式的截断误差一般不可能再是0。先看小区间上的截断误差

$$E_i = \int_{x_i}^{x_{i+1}} [f(x) - L_1(x)] dx$$

设 $f(x)$ 在 $[a, b]$ 上存在二阶连续导数,则根据插值多项式的余项公式可得

$$E_i = \frac{1}{2} \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1}) f''(\xi_i) dx$$

其中 $\xi_i$ 位于 $x_i$ 与 $x_{i+1}$ 之间,且随 $x$ 连续变化,即 $\xi_i$ 是 $x$ 的连续函数,因而 $f''(\xi_i)$ 是 $x$ 的连续函数;又因为 $(x - x_i)(x - x_{i+1})$ 在 $[x_i, x_{i+1}]$ 上不变号,故根据积分中值定理,在

$[x_i, x_{i+1}]$  内存在一  $\xi_i$ , 使得

$$\begin{aligned} E_i &= \frac{1}{2} f''(\xi_i) \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1}) dx \\ &= \frac{1}{2} f''(\xi_i) \cdot \frac{1}{6} (x_i - x_{i+1})^3 \\ &= -\frac{h^3}{12} f''(\xi_i) = -\frac{(b-a)^3}{12n^3} f''(\xi_i) \end{aligned}$$

将这些误差相加, 并注意到  $f''(x)$  在  $[a, b]$  上连续, 因此在  $[a, b]$  内存在一  $\xi$ , 使得

$$\begin{aligned} E &= -\frac{(b-a)^3}{12n^3} \sum_{i=0}^{n-1} f''(\xi_i) = -\frac{(b-a)^3}{12n^3} n f''(\xi) \\ &= -\frac{(b-a)^3}{12n^2} f''(\xi) \quad (a < \xi < b) \end{aligned} \quad (6.10)$$

这就是梯形积分法的截断误差公式。它按步长  $h^2$  或者说按  $\frac{1}{n^2}$  的速度下降。

此外, 把梯形求积公式 (6.9) 的右边写成

$$\frac{1}{2} \sum_{i=0}^{n-1} f(x_i) \times h + \frac{1}{2} \sum_{i=1}^n f(x_i) \times h$$

根据定积分的定义, 可知当  $n \rightarrow \infty$  时, 它趋于定积分  $\int_a^b f(x) dx$ , 这即证明梯形求积公式的收敛性。

## 2. 辛普生求积公式

如果不用线性函数而改用二次函数(抛物线)来近似被积函数  $f(x)$ , 则可得辛普生求积公式或称抛物线公式。

将积分区间  $[a, b]$  分成  $2n$  个等长度小区间

$$[x_0, x_1], [x_1, x_2], \dots, [x_{2n-1}, x_{2n}]$$

其中  $x_i = a + ih$ ;  $i = 0, 1, \dots, 2n$ ;  $h = \frac{b-a}{2n}$

称为步长。在每两个小区间  $[x_{2i}, x_{2i+1}]$ ,  $[x_{2i+1}, x_{2i+2}]$  上, 用过三点  $(x_{2i}, f_{2i})$ ,  $(x_{2i+1}, f_{2i+1})$ ,  $(x_{2i+2}, f_{2i+2})$  (这里记  $f_{2i} = f(x_{2i})$ , 等等) 的二次插值函数  $L_2(x)$  来近似  $f(x)$  的图形(图 6-3), 即用  $L_2(x)$  在区间  $[x_{2i}, x_{2i+2}]$  上的积分来近似  $f(x)$  在该区间上的积分, 于是得

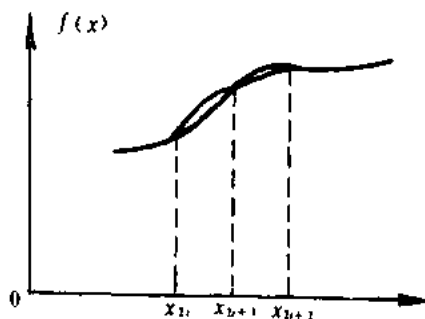


图 6-3

$$\begin{aligned} \int_{x_{2i}}^{x_{2i+2}} f(x) dx &\approx \int_{x_{2i}}^{x_{2i+2}} L_2(x) dx \\ &= \int_{x_{2i}}^{x_{2i+2}} \left[ \frac{(x - x_{2i+1})(x - x_{2i+2})}{(x_{2i} - x_{2i+1})(x_{2i} - x_{2i+2})} f_{2i} + \frac{(x - x_{2i})(x - x_{2i+2})}{(x_{2i+1} - x_{2i})(x_{2i+1} - x_{2i+2})} f_{2i+1} \right. \\ &\quad \left. + \frac{(x - x_{2i})(x - x_{2i+1})}{(x_{2i+2} - x_{2i})(x_{2i+2} - x_{2i+1})} f_{2i+2} \right] dx \\ &= \frac{1}{2h^2} \int_{x_{2i}}^{x_{2i+2}} [(x - x_{2i+1})(x - x_{2i+2}) f_{2i} \end{aligned}$$

$$\begin{aligned}
& -2(x-x_{2i})(x-x_{2i+1})f_{2i+1} + (x-x_{2i})(x-x_{2i+2})f_{2i+2} \} dx \\
& = \frac{1}{2h^2} \left[ \frac{2h^3}{3} f_{2i} + \frac{8h^3}{3} f_{2i+1} + \frac{2h^3}{3} f_{2i+2} \right] \\
& = \frac{h}{3} [f_{2i} + 4f_{2i+1} + f_{2i+2}]
\end{aligned}$$

因为  $\int_a^b f(x) dx = \sum_{i=0}^{n-1} \int_{x_{2i}}^{x_{2i+2}} f(x) dx$ , 故得整个区间上的积分近似值

$$\begin{aligned}
\int_a^b f(x) dx & \approx \frac{h}{3} [f_0 + 4(f_1 + f_3 + \cdots + f_{2n-1}) \\
& \quad + 2(f_2 + f_4 + \cdots + f_{2n-2}) + f_{2n}] \\
& = \frac{h}{3} \left[ f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=1}^n f(x_{2i-1}) \right] \quad (6.11)
\end{aligned}$$

这就是**辛普生 (Simpson) 求积公式**。

类似地,可推导出辛普生积分法的截断误差公式

$$E = -\frac{(b-a)^5}{2880n^4} f^{(4)}(\xi) \quad (a < \xi < b) \quad (6.12)$$

并且可以验证,辛普生公式具有 3 次代数精度。

此外,同样容易证明,当  $n \rightarrow \infty$  时,辛普生求积公式收敛于  $\int_a^b f(x) dx$ 。

由上述梯形求积公式和辛普生求积公式的推导过程可知,它们都是利用一插值多项式来近似被积函数  $f(x)$ ,这就是我们所说的插值型求积公式。一般地,用  $n$  次 ( $n=1, 2, 3, \cdots$ ) 插值多项式来近似被积函数  $f(x)$  所导出的插值型求积公式统称为**牛顿-柯特斯 (Newton-Cotes) 公式**。实际计算中常用  $n=1$  与  $n=2$  的两种情形,即梯形求积公式和辛普生求积公式。

### 3. 自动选步长梯形求积

梯形公式和辛普生公式的截断误差均随  $n$  的增大而减少。因此,自然想到最好先确定一个恰当的  $n$ ,即选取恰当的步长,使得积分近似值  $I_n$  与真值  $I$  之间的误差落在允许的误差范围之内。显然,可以在计算之前根据截断误差公式定出  $n$  值。但这需要涉及被积函数的高阶导数,往往是难做的。自动选步长的思想,就是按照精度要求,在积分计算过程中,自动确定需要的  $n$  值,即自动选步长  $h$ ,从而算出满足精度要求的积分近似值。这种方法称为自动积分法或变步长积分法。

这一段先讨论自动选步长梯形求积。

由梯形求积的截断误差公式 (6.10),当取区间为  $n$  等分时有

$$E_n = I - I_n = -\frac{(b-a)^3}{12n^2} f''(\xi_n) \quad (a < \xi_n < b)$$

这里,  $I$  为积分真值,  $I_n$  为区间分  $n$  等分时的近似值;当取区间为  $2n$  等分时有

$$E_{2n} = I - I_{2n} = -\frac{(b-a)^3}{12(2n)^2} f''(\xi_{2n}), \quad (a < \xi_{2n} < b)$$

上两式相减得

$$I_{2n} - I_n = -\frac{(b-a)^3}{12(2n)^2} [4f''(\xi_n) - f''(\xi_{2n})]$$

且从  $f''(\xi)$  的意义可知当  $n$  足够大时有  $f''(\xi_n) \approx f''(\xi_{1n})$ , 故得

$$\frac{1}{3} (I_{2n} - I_n) \approx (I - I_{2n})$$

这就提供了一个简单的误差判据. 计算过程中便可通过检验条件

$$|I_{2n} - I_n| < \varepsilon \quad (\varepsilon \text{ 为允许误差})$$

来判断积分近似值  $I_{2n}$  是否已满足精度要求, 从而决定是继续细分区间呢还是可以结束计算.

我们构造自动积分过程如下:

1) 取  $n = 1$  时, 计算

$$I_1 = h_0 \left[ \frac{f(a)}{2} + \frac{f(b)}{2} \right], \quad h_0 = b - a$$

2) 取  $n = 2$ , 即把区间对分为 2 等分, 计算

$$I_2 = h_1 \left[ \frac{f(a)}{2} + \frac{f(b)}{2} + f(x_1) \right] = \frac{I_1}{2} + h_1 f(x_1)$$

$$h_1 = (b - a)/2, \quad x_1 = a + h_1$$

3) 取  $n = 2^2 = 4$ , 即把区间再对分为 4 等分, 计算

$$I_4 = h_2 \left[ \frac{f(a)}{2} + \frac{f(b)}{2} + f(x_1) + f(x_2) + f(x_3) \right]$$

$$= \frac{I_2}{2} + h_2 [f(x_1) + f(x_3)]$$

$$h_2 = (b - a)/4, \quad x_i = a + ih_2 \quad (i = 1, 2, 3)$$

一般地, 每次总是在前一次的基础上再将小区间对分, 把分点加密一倍, 其中老分点上的函数值不需重复计算, 而积分值的计算公式是:

$$I_{2n} = \frac{I_n}{2} + h_n \sum_{i=1}^n f(a + (2i - 1)h_n)$$

$$h_n = (b - a)/2n$$

在计算  $I_1, I_2, I_4, \dots, I_n, I_{2n}, \dots$  的过程中, 每当算出一新的近似值  $I_{2n}$  时, 便检查条件:

$$\left. \begin{aligned} &|I_{2n} - I_n| < \varepsilon \quad (\text{当取绝对误差时}) \\ \text{或} \quad &\frac{|I_{2n} - I_n|}{|I_{2n}|} < \varepsilon \quad (\text{当取相对误差时}) \end{aligned} \right\} \quad (6.13)$$

如果条件满足, 即知  $I_{2n}$  是符合精度要求的积分近似值.

在实际计算中, 还需要防止假收敛. 如图 6-4 的振荡被积函数, 有  $I_2 - I_1 = 0$ , 这是假收敛.

为此, 通常给出关于  $n$  的一个下界(或  $h$  的上界), 只有当同时满足误差条件 (6.13) 和防止假收敛条件时, 才认为计算收敛.

关于方法的收敛速度, 类似于上述的推导, 我们有

$$I_n - I_2 = -\frac{(b-a)^3}{12(n)^2} [4f''(\xi_2) - f''(\xi_n)]$$

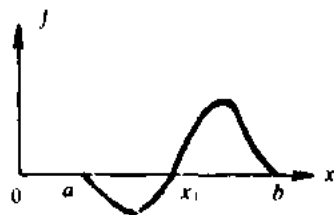


图 6-4



从而有

$$I_{2n} - I_n \approx \frac{1}{4} (I_n - I_{\frac{n}{2}}) \quad (6.14)$$

因此, 我们称方法具有收敛因子为  $\frac{1}{4}$  的线性收敛速度.

#### 4. 自动选步长辛普生求积及其 FORTRAN 程序

相仿地, 从辛普生公式的截断误差我们也可得

$$\frac{1}{15} (I_{2n} - I_n) \approx (I - I_{2n})$$

其中,  $I$  是积分真值,  $I_n$  和  $I_{2n}$  分别是区间分  $n$  等分和  $2n$  等分时的近似值. 由此, 可以通过  $(I_{2n} - I_n)$  来估计近似值  $I_{2n}$  的误差.

同样采用区间逐次对分的方法来计算辛普生积分序列  $I_2, I_4, \dots, I_n, I_{2n}, \dots$

$$I_{2n} = \frac{h_n}{3} [f(a) + f(b) + 2S_n + 4S'_n] \quad (6.15)$$

其中

$$S_n = f(x_2) + f(x_4) + \dots + f(x_{2n-2})$$

$$S'_n = f(x_1) + f(x_3) + \dots + f(x_{2n-1})$$

$$h_n = (b-a)/2n, \quad x_i = a + ih_n \quad (i = 1, 2, \dots, 2n-1)$$

就相邻两次对分而言,  $S_n$  是前一次分点上函数值之和,  $S'_n$  是后一次分点上函数值之和, 后一次计算时,  $S_n$  无需重复计算.

同样地, 每次要检验条件 (6.13), 也要设置防止假收敛的控制常数. 只有同时满足所述两个条件时, 才认为  $I_{2n}$  是符合精度要求的积分近似值.

关于方法的收敛速度, 类似于 (6.14) 可得

$$I_{2n} - I_n \approx \frac{1}{16} (I_n - I_{\frac{n}{2}})$$

这是收敛因子为  $\frac{1}{16}$  的线性收敛速度.

下面我们讨论自动选步长辛普生求积的 FORTRAN 程序设计.

第 1 次把区间  $[a, b]$  分成  $2^1$  等分, 积分近似值记为  $S_1$ ; 第 2 次分成  $2^2$  等分, 积分近似值记为  $S_2$ ;  $\dots$ , 每次把分点加密一倍.

在计算  $S_n$  时, 记下

$$RP = f(a) + f(b) + 2 \sum_{k=1}^{2^n-1} f(a + kh)$$

则在计算  $S_{n+1}$  时

$$S_{n+1} = (RP + 4 \times RC)h/6$$

其中  $RC$  为新增加的诸分点上函数值之和,  $h = (b-a)/2^n$ . 因为每次计算  $S_n (n=1, 2, \dots)$  时保留  $RP$ , 所以在计算  $S_{n+1}$  时只需计算  $RC$ , 而省去计算原有诸分点上的函数值.

对相邻两次所得的积分近似值  $S_n$  和  $S_{n+1}$ , 考察如下之  $d$ :

$$d = \begin{cases} S_{n+1} - S_n & \text{当 } |S_{n+1}| < 1 \text{ 时} \\ \frac{S_{n+1} - S_n}{S_{n+1}} & \text{当 } |S_{n+1}| \geq 1 \text{ 时} \end{cases}$$

若  $|d|$  小于允许的精确度  $\varepsilon$ , 则  $S_{n+1}$  便为所求的积分近似值; 否则, 继续将分点加密一倍, 直至  $|d| < \varepsilon$  为止。

为防止假收敛, 当  $n$  大于控制常数  $H0$  时, 不论精度满足与否, 都继续加密分点, 直至  $n$  小于等于  $H0$  后才考察相邻两次积分近似值是否满足精度要求。

上述算法可以写成一般子程序 (SUBROUTINE) 的形式, 也可写成函数子程序 (FUNCTION) 的形式。下面我们把它写成函数子程序 SIMP1, 其中设置哑元如下:

A, B 实型变量, 积分下限和上限。

F 子程序中调用的实型函数, 计算被积函数的值, 函数语句的形式为

FUNCTION F(X)

X 为实型变量, 是积分变量。

EPS 实型变量, 相邻两次结果的允许误差。

H0 为防止假收敛设置的实常数。

函数子程序:

```

FUNCTION SIMPSN (A, B, F, EPS, H0)
  H = B - A
  RP = F(A) + F(B)
  N = 1
2  X = A + 0.5 * H
  RC = 0.0
  DO 4 I = 1, N
    X = X + H
4  RC = F(X) + RC
  R2 = (RP + 4.0 * RC) * H / 6.0
  IF (N.EQ. 1 .OR. ABS(H).GT. H0) GO TO 6
  X = R2 - R1
  IF (ABS(R2).GE. 1.0) X = X/R2
  IF (ABS(X).LT. EPS) GO TO 8
6  R1 = R2
  RP = 2.0 * RC + RP
  N = N + N
  H = 0.5 * H
  GO TO 2
8  SIMPSN = R2
  RETURN
END

```

算例 计算定积分

$$I = \int_1^e \frac{\sin^2 x}{x} dx$$

设精度  $\varepsilon = 10^{-6}$ , 取  $H0 = 10^{-3}$ . 解题程序(包括主程序、子程序 SIMPSN 和计算被积函数值的子程序 F) 如下:

```
C      THE MAIN PROGRAM
      EXTERNAL F
      RI = SIMPSN (1.0, 5.0, F, 1E-6, 1E-2)
      WRITE (6, 100) RI
100  FORMAT (1X, 2H1 = , F10.6)
      STOP
      END

C      FUNCTION SIMPSN (A, B, F, EPS, H0)
      : (本子程序段体)
      END

C      FUNCTION F(X)
      F = SIN(X) ** 2 / X
      RETURN
      END
```

计算结果:

$$I = 1.038938$$

### 6-3 龙贝格积分法及其 FORTRAN 程序

龙贝格 (Romberg) 积分法是一种在逐次对分的基础上, 再采用加速技巧, 从而获得所谓超线性收敛速度的方法。其基本思想是: 按照逐次对分区间的步骤, 计算梯形和序列:

$$\begin{aligned} T_0^{(0)} &= (b-a) \frac{f(a) + f(b)}{2} \\ T_0^{(1)} &= \frac{b-a}{2} \left[ \frac{f(a)}{2} + f\left(a + \frac{b-a}{2}\right) + \frac{f(b)}{2} \right] \\ T_0^{(2)} &= \frac{b-a}{4} \left[ \frac{f(a)}{2} + f\left(a + \frac{b-a}{4}\right) + f\left(a + \frac{b-a}{2}\right) + f\left(a + \frac{3(b-a)}{4}\right) \right. \\ &\quad \left. + \frac{f(b)}{2} \right] \\ &\dots \dots \dots \end{aligned}$$

或写成一般形式

$$T_0^{(k)} = h_k \left[ \frac{f(a)}{2} + f(a + h_k) + \cdots + \frac{f(b)}{2} \right] \quad (6.16)$$

其中  $h_k = \frac{b-a}{2^k}$ ,  $k = 0, 1, 2, \dots$ .

另一方面, 根据梯形求积的截断误差公式 (6.10) 我们有

$$T_0 = I + \frac{(b-a)}{12} f''(\xi) h^2, \quad a < \xi < b.$$

其中  $I$  是积分真值,  $\xi$  与  $h$  有关。它表示梯形和 (6.16) 与步长  $h$  的关系。如果我们把  $T_0$  看作以  $h^2$  为自变量的函数, 如图 6-5 所示, 则所得曲线在  $T_0$  轴上的截距即为积分真值  $I$ , 而梯形和序列所对应的点

$$(h_0^2, T_0^{(0)}), (h_1^2, T_0^{(1)}), \dots, (h_k^2, T_0^{(k)}), \dots$$

是在曲线上以点  $(0, I)$  为极限的点列。如果我们通过相邻两点  $(h_k^2, T_0^{(k)})$  与  $(h_{k+1}^2, T_0^{(k+1)})$  作一直线并延长交  $T_0$  轴于  $T_1^{(k)}$ , 则  $T_1^{(k)}$  可望比  $T_0^{(k+1)}$  更趋近  $I$ 。这就是线性外插加速思想的几何意义。

现在我们来推导建立在这种思想上的龙贝格算法, 以及给出这个算法的 FORTRAN 程序。

过点  $(h_k^2, T_0^{(k)})$  与  $(h_{k+1}^2, T_0^{(k+1)})$  作直线方程

$$T_0 = T_0^{(k)} + \frac{T_0^{(k+1)} - T_0^{(k)}}{h_{k+1}^2 - h_k^2} (h^2 - h_k^2)$$

它在  $T_0$  轴上的截距为

$$T_1^{(k)} = T_0^{(k)} - \frac{T_0^{(k+1)} - T_0^{(k)}}{h_{k+1}^2 - h_k^2} h_k^2$$

注意到关系式  $h_{k+1} = \frac{h_k}{2}$ , 便得新的近似值

$$T_1^{(k)} = \frac{4T_0^{(k+1)} - T_0^{(k)}}{4 - 1} \quad (k = 0, 1, \dots)$$

把这个近似值展开并代入 (6.16) 的值, 将可以看到, 它等于以  $h_{k+1}$  为步长的辛普生公式, 而辛普生公式的截断误差为

$$T_1^{(k)} = I + \frac{(b-a)}{180} f^{(4)}(\xi_k) h_{k+1}^4, \quad a < \xi_k < b$$

可知新近似值  $T_1^{(k)}$  比  $T_0^{(k+1)}$  更接近于  $I$ 。

类似地推导, 我们把  $T_1^{(k)}$  视为  $h^4$  的函数, 再应用线性外插过程, 又可得新的近似值

$$T_2^{(k)} = \frac{4^2 T_1^{(k+1)} - T_1^{(k)}}{4^2 - 1} \quad (k = 0, 1, \dots)$$

此类推, 一般地, 由  $T_{L-1}^{(k+1)}$  和  $T_{L-1}^{(k)}$  可得新的近似值

$$T_L^{(k)} = \frac{4^L T_{L-1}^{(k+1)} - T_{L-1}^{(k)}}{4^L - 1} \quad (L = 1, 2, \dots; k = 0, 1, \dots) \quad (6.17)$$

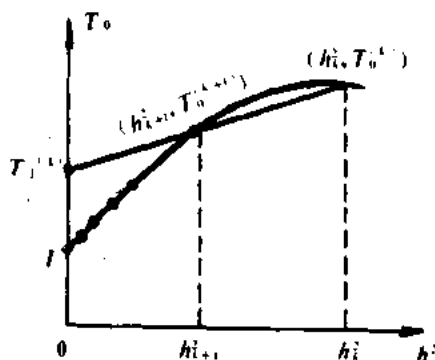
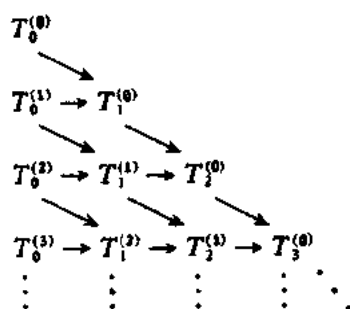


图 6-5

这个数值过程可构成一个所谓龙贝格表如下:



其中第1列是用梯形公式(6.16)从函数值直接算出的值,以后的每一列都是从前一列用公式(6.17)计算出来的。具体做法是沿着“左上到右下”的对角线计算表中的各元素,直至表中主对角线上的两相邻元素在所给定的误差范围内为止,即先计算  $T_0^{(0)}$ ,  $T_0^{(1)}$  和  $T_1^{(0)}$ , 判断: 若  $|T_1^{(0)} - T_0^{(0)}| < \epsilon$ , 则计算结束, 取  $T_1^{(0)}$  为积分近似值; 否则, 把步长缩小一半, 求得  $T_1^{(2)}$ , 然后计算  $T_1^{(1)}$ ,  $T_2^{(0)}$ , 再判断是否  $|T_2^{(0)} - T_1^{(0)}| < \epsilon$ , 如果是, 则停止计算, 取积分近似值为  $T_2^{(0)}$ ; 否则, 又对分步长并作相应计算, 如此继续。

在计算机上具体实现龙贝格算法(包括其 FORTRAN 程序)的方案很多<sup>[C1, C2, C3]</sup>, 它与上述过程往往略有不同。下面我们引用其中比较简单的一种:

先用梯形公式计算出

$$T_1 = \frac{b-a}{2} [f(a) + f(b)]$$

然后把求积区间  $[a, b]$  逐次折半, 即令区间长度依次为

$$h = \frac{b-a}{2^i} \quad (i = 0, 1, 2, \dots)$$

再逐次计算出

$$T_{2n} = \frac{1}{2} T_n + \frac{h}{2} \sum_{k=1}^n f\left(a + h\left(k - \frac{1}{2}\right)\right)$$

其中  $n = 2^i$ ,  $i = 0, 1, 2, \dots$ 。于是, 按(6.17)式推得的关系这里可得

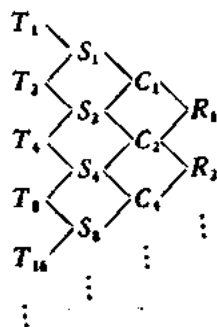
$$(1) \quad S_n = T_{2n} + (T_{2n} - T_n)/3$$

并且相仿地可得

$$(2) \quad C_n = S_{2n} + (S_{2n} - S_n)/15$$

$$(3) \quad R_n = C_{2n} + (C_{2n} - C_n)/63$$

其中(1)式实际就是辛普生求积公式; (2)式是所谓柯特斯公式; (3)式即为龙贝格公式, 而且这个计算过程也可以构造如下形状的龙贝格表



直至相邻两次积分近似值  $R_{1n}$  和  $R_n$  满足如下关系:

当  $|R_{1n}| \leq 1$  时, 有  $|R_{1n} - R_n| < \varepsilon$ ;

当  $|R_{1n}| > 1$  时, 有  $\left| \frac{R_{1n} - R_n}{R_{1n}} \right| < \varepsilon$ ,

其中  $\varepsilon$  为允许的误差限, 便取  $R_{1n}$  为所求的积分近似值。

下面给出 FORTRAN 子程序 ROMBG.

哑元设置如下:

- A, B 实型变量, 输入参数; 积分下限和上限.  
EPS 实型变量, 输入参数; 积分的精度, 即允许的误差限.  
R2 实型变量, 输出参数; 积分的结果.  
F 计算被积函数的函数子程序名, 形式为  
FUNCTION F(X)  
X 为实型变量; 本函数子程序由使用自编.

子程序:

```
SUBROUTINE ROMBG (A, B, EPS, R2, F)
  H = B - A
  T1 = 0.5 * H * (F(A) + F(B))
  N = 1
5  S = 0
  DO 10 K = 1, N
10  S = S + F(A + H * (K - 0.5))
    T2 = 0.5 * T1 + 0.5 * H * S
    S2 = T2 + (T2 - T1) / 3.
    IF (N .NE. 1) GO TO 20
15  N = N + N
    H = 0.5 * H
    T1 = T2
    S1 = S2
    GO TO 5
20  C2 = S2 + (S2 - S1) / 15.
    IF (N .NE. 2) GO TO 30
25  C1 = C2
    GO TO 15
30  R2 = C2 + (C2 - C1) / 63.
    IF (N .NE. 4) GO TO 40
35  R1 = R2
    GO TO 25
40  IF (ABS (R2) .GT. 1.) GO TO 45
```

```

        IF (ABS (R2 - R1) .GE. EPS) GO TO 35
        RETURN
    ) 45 IF (ABS ((R2 - R1)/R2) .GE. EPS) GO TO 35
        RETURN
        END

```

**算例 1** 计算下列三个积分(连同系数):

- (1)  $\int_0^1 \frac{4}{1+x^2} dx;$
- (2)  $\frac{2}{\sqrt{\pi}} \int_0^1 e^{-x^2} dx;$
- (3)  $\frac{1}{\sqrt{2\pi}} \int_{-3}^3 x^2 e^{-\frac{x^2}{2}} dx.$

解题程序:

```

C      THE MAIN PROGRAM
        EXTERNAL F1, F2, F3
        CALL ROMBG (0.0, 1.0, 1E-7, R2, F1)
        WRITE (6, 100) R2
        CALL ROMBG (0.0, 1.0, 1E-7, R2, F2)
        R2 = 2.2 * R2/SQRT (3.1415926)
        WRITE (6, 100) R2
        CALL ROMBG (-3.0, 3.0, 1E-7, R2, F3)
        R2 = R2/SQRT (6.2831852)
        WRITE (6, 100) R2
100  FORMAT (1X, 3HR2 = , F13.6//3HR2 = , F13.6//3HR2 = , F13.6)
        STOP
        END

C
        SUBROUTINE ROMBG (A, B, EPS, R2, F)
            : (本子程序段体)
        END

C
        FUNCTION F1 (X)
            F1 = 4./(1. + X*X)
            RETURN
        END
        FUNCTION F2 (X)
            F2 = EXP (-X*X)
            RETURN
        END

```

```

FUNCTION F3 (X)
F3 = X*X*EXP (-0.5*X*X)
RETURN
END

```

计算结果依次为:

$$R2 = 0.314159E + 01$$

$$R2 = 0.842701E + 00$$

$$R2 = 0.970709E + 00$$

**算例 2** 再研究一个实例.

已知流过某 1F 电容器的电流(非正弦)表达式为

$$i = \cos t + \sin 2t$$

我们用计算机来考察电容器两端电压  $u_c$  的变化情况. 这里我们设  $t \leq 0$  时  $u_c = 0$ , 并取  $0 < t \leq 6$  (秒)这段时间来加以研究.

因为电容器的端电压与流过其上的电流关系为  $u_c = u(0) + \frac{1}{C} \int_0^t i(t) dt$ , 据题意有  $u(0) = 0$ ,  $C = 1$ , 故问题归结为求积分

$$u(t) = \int_0^t (\cos t + \sin 2t) dt \quad (0 < t \leq 6)$$

我们取误差限  $\varepsilon = 10^{-5}$ , 取  $\Delta t = 0.1$ , 即每隔 0.1 秒计算一次电压值. 为了更直观地看出电容器端电压的变化情况, 程序中不仅输出各个时刻对应的端电压值, 而且通过绘图子程序 (GRAPH), 还描绘出端电压的变化曲线.

解题程序如下:

```

C      THE MAIN PROGRAM
      EXTERNAL F
      CHARACTER * 61 LINE
      A = 0.0
      Q = 0.1
      B = A + Q
      WRITE (6, 100)
00  FORMAT (1X, 'TIME', 6X, 'VOL', 8X, 'I = COSX + SIN2X')
      DO 130 KK = 1, 60
      CALL ROMBG (A, B, 1.0 E - 5, R2, F)
      CALL GRAPH (LINE, R2)

```



```

WRITE (6, 110) B, R2, LINE
110 FORMAT (1X, F6.4, 2X, F6.4, 2X, A61)
B = B + Q
130 CONTINUR
STOP
END

C
SUBROUTINE ROMBG (A, B, EPS, R2, F)
: (本子程序段体)
END

C
FUNCTION F(X)
F = COS(X) + SIN(2.0 * X)
RETURN
END

C
SUBROUTINE GRAPH (LINE, R2)
CHARACTER * 61 LINE
LINE = ' '
LINE (31:31) = 'I'
IS = INT (15.0 * (R2 + 2.0) + 1.5)
LINE (IS:IS) = '*'
RETURN
END

```

计算输出结果如下(见下页)。

## 6-4 正交多项式与高斯型求积公式

高斯型求积公式与正交多项式有关。事实上，正交多项式在许多场合也有用。这一节从介绍正交多项式的一些基本概念讲起。

### 1. 正交多项式

**定义** 若  $n$  次多项式  $g_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$  ( $a_n \neq 0$ ) 满足条件

$$\int_a^b \rho(x) g_i(x) g_j(x) dx = \begin{cases} 0 & i \neq j \\ A_i > 0 & i = j \end{cases} \quad (i, j = 0, 1, \cdots)$$

便称多项式序列  $g_0(x), g_1(x), \cdots$ , 在  $[a, b]$  上带权  $\rho(x)$  正交, 并称  $g_n(x)$  为  $[a, b]$  上带权  $\rho(x)$  的  $n$  次正交多项式.  $\rho(x)$  称为权函数, 在  $[a, b]$  上  $\rho(x) \geq 0$ ;  $\int_a^b \rho(x) dx > 0$ ;

| TIME   | VOL.   | $1 = \cos X + \sin 2X$ | 0 |
|--------|--------|------------------------|---|
| 0.1000 | 0.1098 | 1                      | * |
| 0.2000 | 0.2381 | 1                      | * |
| 0.3000 | 0.3829 | 1                      | * |
| 0.4000 | 0.5411 | 1                      | * |
| 0.5000 | 0.7093 | 1                      | * |
| 0.6000 | 0.8835 | 1                      | * |
| 0.7000 | 1.0592 | 1                      | * |
| 0.8000 | 1.2320 | 1                      | * |
| 0.9000 | 1.3969 | 1                      | * |
| 1.0000 | 1.5495 | 1                      | * |
| 1.1000 | 1.6855 | 1                      | * |
| 1.2000 | 1.8007 | 1                      | * |
| 1.3000 | 1.8920 | 1                      | * |
| 1.4000 | 1.9566 | 1                      | * |
| 1.5000 | 1.9925 | 1                      | * |
| 1.6000 | 1.9987 | 1                      | * |
| 1.7000 | 1.9751 | 1                      | * |
| 1.8000 | 1.9222 | 1                      | * |
| 1.9000 | 1.8418 | 1                      | * |
| 2.0000 | 1.7361 | 1                      | * |
| 2.1000 | 1.6083 | 1                      | * |
| 2.2000 | 1.4622 | 1                      | * |
| 2.3000 | 1.3018 | 1                      | * |
| 2.4000 | 1.1317 | 1                      | * |
| 2.5000 | 0.9566 | 1                      | * |
| 2.6000 | 0.7812 | 1                      | * |
| 2.7000 | 0.6100 | 1                      | * |
| 2.8000 | 0.4472 | 1                      | * |
| 2.9000 | 0.2965 | 1                      | * |
| 3.0000 | 0.1610 | 1                      | * |
| 3.1000 | 0.0433 | 1                      | * |
| 3.2000 | -.0550 | 1                      | * |
| 3.3000 | -.1329 | 1                      | * |
| 3.4000 | -.1902 | 1                      | * |
| 3.5000 | -.2277 | 1                      | * |
| 3.6000 | -.2467 | 1                      | * |
| 3.7000 | -.2491 | 1                      | * |
| 3.8000 | -.2375 | 1                      | * |
| 3.9000 | -.2147 | 1                      | * |
| 4.0000 | -.1841 | 1                      | * |
| 4.1000 | -.1487 | 1                      | * |
| 4.2000 | -.1119 | 1                      | * |
| 4.3000 | -.0768 | 1                      | * |
| 4.4000 | -.0461 | 1                      | * |
| 4.5000 | -.0220 | 1                      | * |
| 4.6000 | -.0063 | 1                      | * |
| 4.7000 | -.0001 | 1                      | * |
| 4.8000 | -.0038 | 1                      | * |
| 4.9000 | -.0172 | 1                      | * |
| 5.0000 | -.0394 | 1                      | * |
| 5.1000 | -.0687 | 1                      | * |
| 5.2000 | -.1030 | 1                      | * |
| 5.3000 | -.1396 | 1                      | * |
| 5.4000 | -.1756 | 1                      | * |
| 5.5000 | -.2078 | 1                      | * |
| 5.6000 | -.2328 | 1                      | * |
| 5.7000 | -.2474 | 1                      | * |
| 5.8000 | -.2487 | 1                      | * |
| 5.9000 | -.2341 | 1                      | * |
| 6.0000 | -.2013 | 1                      | * |

积分  $\int_a^b x^n \rho(x) dx$  ( $n = 0, 1, \dots$ ) 存在. 如无特别指明, 通常取  $\rho(x) \equiv 1$ .

例如, 可以验证,  $1, x, x^2 - \frac{1}{3}$  在  $[-1, 1]$  上带权  $\rho(x) = 1$  正交.

常用的正交多项式有勒让德(Legendre) 多项式、切比雪夫(Чебышев) 多项式、拉盖尔(Laguerre) 多项式和埃尔米特多项式. 这里我们简单地介绍前两种.

(1) 勒让德多项式 定义为

$$\begin{cases} P_0(x) = 1 \\ P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2 - 1)^n], \quad n = 1, 2, \dots \end{cases} \quad (6.18)$$

由于  $(x^2 - 1)^n$  是  $2n$  次多项式, 所以  $P_n(x)$  是  $n$  次多项式, 最高次幂的系数

$$a_n = \frac{1}{2^n n!} 2n(2n-1)(2n-2)\cdots(n+1) = \frac{(2n)!}{2^n (n!)^2}$$

从定义式可列出勒让德多项式的前几个表达式如下表:

|                                           |                                             |
|-------------------------------------------|---------------------------------------------|
| $P_0(x) = 1$                              | $P_1(x) = x$                                |
| $P_2(x) = \frac{1}{2}(3x^2 - 1)$          | $P_3(x) = \frac{1}{2}(5x^3 - 3x)$           |
| $P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$ | $P_5(x) = \frac{1}{8}(63x^5 - 70x^3 + 15x)$ |

**性质 1** 勒让德多项式  $P_n(x)$  是在  $[-1, 1]$  上带权  $\rho(x) = 1$  的  $n$  次正交多项式组, 即有

$$\int_{-1}^1 P_m(x) P_n(x) dx = \begin{cases} 0 & m \neq n \\ \frac{2}{2n+1} & m = n \end{cases}$$

**证** 对于  $m \neq n$  (不妨设  $m < n$ ), 使用  $n$  次分部积分有

$$\begin{aligned} 2^{m+n} m! n! \int_{-1}^1 P_m(x) P_n(x) dx &= \int_{-1}^1 \frac{d^m}{dx^m} [(x^2 - 1)^m] \frac{d^n}{dx^n} [(x^2 - 1)^n] dx \\ &= \left\{ \frac{d^m}{dx^m} [(x^2 - 1)^m] \frac{d^{n-1}}{dx^{n-1}} [(x^2 - 1)^n] \right\}_{-1}^1 \\ &\quad - \int_{-1}^1 \frac{d^{m+1}}{dx^{m+1}} [(x^2 - 1)^m] \frac{d^{n-1}}{dx^{n-1}} [(x^2 - 1)^n] dx \\ &= - \int_{-1}^1 \frac{d^{m+1}}{dx^{m+1}} [(x^2 - 1)^m] \frac{d^{n-1}}{dx^{n-1}} [(x^2 - 1)^n] dx \\ &= \dots\dots\dots \\ &= (-1)^n \int_{-1}^1 \frac{d^{m+n}}{dx^{m+n}} [(x^2 - 1)^m] (x^2 - 1)^n dx \end{aligned}$$

因为  $(x^2 - 1)^m$  为  $2m$  次多项式, 而求导次数  $m + n > 2m$ , 故上式为 0.

对于  $m = n$ , 仿上式可得

$$2^{2n} (n!)^2 \int_{-1}^1 P_n^2(x) dx = \int_{-1}^1 \frac{d^n}{dx^n} [(x^2 - 1)^n] \frac{d^n}{dx^n} [(x^2 - 1)^n] dx$$

$$\begin{aligned}
&= (-1)^n \cdot \int_{-1}^1 (2n)! (x^2 - 1)^n dx \\
&= (2n)! \int_{-1}^1 (1 - x^2)^n dx \quad (\text{令 } x = \sin u) \\
&= 2(2n)! \int_0^{\frac{\pi}{2}} \cos^{2n+1} u du \\
&= 2(2n)! \frac{(2n) \cdots 4 \cdot 2}{(2n+1) \cdots 3 \cdot 1} \\
&= 2(2n)! \frac{[(2n) \cdots 4 \cdot 2]^2}{(2n+1)(2n)!} \\
&= 2 \cdot \frac{(2^n \cdot n!)^2}{(2n+1)} = 2^{2n} \cdot (n!)^2 \cdot \frac{2}{2n+1}
\end{aligned}$$

所以 
$$\int_{-1}^1 P_n^2(x) dx = \frac{2}{2n+1}$$

**性质 2** 勒让德多项式具有递推关系

$$\begin{cases} P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x), & n = 1, 2, \dots \\ P_0(x) = 1, & P_1(x) = x \end{cases} \quad (6.19)$$

**证** 考虑  $n+1$  次多项式  $xP_n(x)$ :

$$xP_n(x) = C_0 P_0(x) + C_1 P_1(x) + \cdots + C_{n+1} P_{n+1}(x)$$

两边乘  $P_k(x)$ , 并从  $-1$  到  $1$  积分, 得

$$\int_{-1}^1 xP_n(x) P_k(x) dx = C_k \int_{-1}^1 P_k^2(x) dx$$

当  $k \leq n-2$  时,  $xP_k(x)$  次数小于等于  $n-1$ , 上式左端积分为 0, 故得  $C_k = 0$ ; 当  $k = n$  时,  $xP_n^2(x)$  为奇函数, 左端积分仍为 0, 故  $C_n = 0$ . 于是

$$xP_n(x) = C_{n-1} P_{n-1}(x) + C_{n+1} P_{n+1}(x)$$

其中

$$\begin{aligned}
C_{n-1} &= \frac{2n-1}{2} \int_{-1}^1 xP_n(x) P_{n-1}(x) dx \\
&= \frac{2n-1}{2} \frac{2n}{4n^2-1} = \frac{n}{2n+1}; \\
C_{n+1} &= \frac{2n+3}{2} \int_{-1}^1 xP_n(x) P_{n+1}(x) dx \\
&= \frac{2n+3}{2} \cdot \frac{2(n+1)}{(2n+1)(2n+3)} = \frac{n+1}{2n+1}
\end{aligned}$$

整理即得所证递推公式.

(2) **切比雪夫多项式** 定义为

$$T_n(x) = \cos(n \arccos x), \quad -1 \leq x \leq 1 \quad (6.20)$$

若令  $\cos \theta = x$ , 则可记为参数形式

$$\begin{cases} x = \cos \theta \\ T_n(x) = \cos n\theta \end{cases} \quad 0 \leq \theta \leq \pi \quad (6.21)$$

**性质 1** 切比雪夫多项式具有递推关系

$$\begin{cases} T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), & n = 1, 2, \dots \\ T_0(x) = 1, & T_1(x) = x \end{cases} \quad (6.22)$$

证 由三角恒等式

$$\cos(n+1)\theta = \cos n\theta \cos \theta - \sin n\theta \sin \theta$$

$$\cos(n-1)\theta = \cos n\theta \cos \theta + \sin n\theta \sin \theta$$

相加可得

$$\cos(n+1)\theta = 2\cos n\theta \cos \theta - \cos(n-1)\theta$$

由参数形式即可得递推关系。

根据递推关系可以列出前几个切比雪夫多项式如下表

---

|                                                                             |
|-----------------------------------------------------------------------------|
| $T_0 = 1$                                                                   |
| $T_1 = x$                                                                   |
| $T_2 = 2x^2 - 1$                                                            |
| $T_3 = 4x^3 - 3x$                                                           |
| $T_4 = 8x^4 - 8x^2 + 1$                                                     |
| $T_5 = 16x^5 - 20x^3 + 5x$                                                  |
| $T_6 = 32x^6 - 48x^4 + 18x^2 - 1$                                           |
| $T_7 = 64x^7 - 112x^5 + 56x^3 - 7x$                                         |
| $T_8 = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$                                |
| $T_9 = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$                              |
| $T_{10} = 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1$               |
| $T_{11} = 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x$          |
| $T_{12} = 2048x^{12} - 6144x^{10} + 6912x^8 - 3584x^6 + 840x^4 - 72x^2 + 1$ |

---

从表容易推得  $x^n$  可用  $T_0, T_1, \dots, T_n$  的线性组合表出。

**性质 2** 切比雪夫多项式是  $[-1, 1]$  上带权  $\rho(x) = \frac{1}{\sqrt{1-x^2}}$  的正交多项式。

证 容易证得

$$\begin{aligned} \int_{-1}^1 \frac{1}{\sqrt{1-x^2}} T_m(x) T_n(x) dx &= \int_0^\pi \cos m\theta \cos n\theta d\theta \\ &= \begin{cases} \pi & \text{当 } m=n=0 \text{ 时} \\ \pi/2 & \text{当 } m=n \neq 0 \text{ 时} \\ 0 & \text{当 } m \neq n \text{ 时} \end{cases} \end{aligned}$$

这就说明了切比雪夫多项式的正交性。

## 2. 高斯型求积公式

高斯 (Gauss) 型求积公式是各种数值积分法中精度较高的一种。它与梯形公式和辛普生公式一样, 也是一种插值型求积公式:

$$\int_a^b f(x) dx \approx \sum_{i=0}^n A_i f(x_i) \quad (6.23)$$

所不同的是, 它所选择的  $n+1$  个节点  $x_0, x_1, \dots, x_n$  并非等距节点, 也取消了  $x_0, x_n$  与积分限  $a, b$  重合的限制, 并且针对某些特殊的权函数, 还采用更有效的正交多项式逼近被积函数, 因而, 获得  $2n+1$  次代数精度。

对插值型求积公式来说,如果节点  $x_i$  确定了,则求积系数  $A_i$  原则上通过公式

$$A_i = \int_a^b l_i(x) dx = \int_a^b \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} dx$$

确定,因此构造插值型求积公式的关键在于确定节点  $x_i$ .

**定义** 若由节点  $x_0, x_1, \dots, x_n \in [a, b]$  构造的插值型求积公式 (6.23) 具有  $2n+1$  次代数精度,则称此组节点为**高斯点**,相应的求积公式 (6.23) 为**高斯型求积公式**.

由此,构造高斯型求积公式的关键在于求高斯点.有几种方法用来求高斯点以及相应的求积系数.

**方法 I.** 待定系数法.我们以实例来说明.

**例 1** 设要构造下列形式的高斯公式

$$\int_{-1}^1 f(x) dx \approx A_0 f(x_0) + A_1 f(x_1)$$

要求它具有  $2 \times 1 + 1 = 3$  阶代数精度,因而上式应对  $f(x) = 1, x, x^2, x^3$  都准确成立,即满足方程组

$$\begin{cases} A_0 + A_1 = 2 \\ A_0 x_0 + A_1 x_1 = 0 \\ A_0 x_0^2 + A_1 x_1^2 = \frac{2}{3} \\ A_0 x_0^3 + A_1 x_1^3 = 0 \end{cases}$$

解之得  $x_0 = -\frac{\sqrt{3}}{3}, x_1 = \frac{\sqrt{3}}{3}, A_0 = A_1 = 1$ , 故得

$$\int_{-1}^1 f(x) dx \approx f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

待定系数法通过解一个方程组(非线性)来求解高斯点和求积系数,在  $n$  较大的情况下是不容易做的.

**方法 II.** 利用正交多项式.它建立在如下定理的基础上.

**定理** 节点  $x_0, x_1, \dots, x_n$  为  $[a, b]$  上的高斯点的充分必要条件是由这些节点构成的  $n+1$  次多项式

$$\omega_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$$

在  $[a, b]$  上与任何次数不超过  $n$  的多项式均正交.

**证** 必要性: 设  $x_0, x_1, \dots, x_n$  是  $[a, b]$  上的高斯点,则相应的插值型求积公式是高斯型求积公式,具有  $2n+1$  次代数精度,于是对任何不高于  $n$  次的多项式  $P(x)$  与  $n+1$  次多项式  $\omega_{n+1}(x)$  的乘积  $P(x)\omega_{n+1}(x)$  有

$$\int_a^b P(x)\omega_{n+1}(x) dx = \sum_{i=0}^n A_i P(x_i)\omega_{n+1}(x_i) = 0$$

(因为  $\omega_{n+1}(x_i) = 0$ ) 这说明  $\omega_{n+1}(x)$  在  $[a, b]$  上与任何次数不超过  $n$  的多项式均正交.

充分性: 设  $n+1$  次多项式  $\omega_{n+1}(x)$  在  $[a, b]$  上与任何次数不超过  $n$  的多项式均正交,用它除任何不高于  $2n+1$  次的多项式  $f(x)$ ,记商为  $P(x)$ ,余式为  $q(x)$ ,则

$$f(x) = P(x)\omega_{n+1}(x) + q(x)$$

$$\text{或} \quad \int_a^b f(x) dx = \int_a^b P(x) \omega_{n+1}(x) dx + \int_a^b q(x) dx$$

其中  $P(x)$ ,  $q(x)$  都是不高于  $n$  次的多项式. 因此, 上式右端第一项应为 0, 故得

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

但由于求积公式是插值型的, 它至少具有  $n$  次代数精度, 即对  $q(x)$  准确成立, 故又得

$$\int_a^b f(x) dx = \int_a^b q(x) dx = \sum_{i=0}^n A_i q(x_i)$$

取从  $f(x) = P(x) \omega_{n+1}(x) + q(x)$  可知有  $f(x_i) = q(x_i)$ , 从而得

$$\int_a^b f(x) dx = \sum_{i=0}^n A_i f(x_i)$$

这表明, 求积公式对任何不高于  $2n+1$  次多项式  $f(x)$  都准确成立, 也就是说, 所作出的求积公式是高斯型求积公式, 其节点  $x_0, x_1, \dots, x_n$  是高斯点. 定理至此证毕.

由定理可知, 在  $[a, b]$  上的任一  $n+1$  次正交多项式的零点就是高斯点. 据此, 我们利用已知的正交多项式, 求出它的零点, 就可构造出各种具体的高斯型求积公式.

考虑勒让德多项式, 并先讨论积分区间为  $[-1, 1]$  的情形. 根据上述定理, 只要求出勒让德多项式的零点, 它就是高斯点, 就可构造出相应的高斯型求积公式(包括求出求积系数), 这种求积公式称为高斯-勒让德求积公式.

### 例 2 构造两点高斯-勒让德求积公式.

由于二次勒让德多项式

$$P_2(x) = \frac{1}{2^2 \cdot 2!} \frac{d^2}{dx^2} [(x^2 - 1)^2] = \frac{1}{8} (12x^2 - 4)$$

的零点为  $\pm \frac{\sqrt{3}}{3}$ , 故所求插值型公式形如

$$\int_{-1}^1 f(x) dx \approx A_0 f\left(-\frac{\sqrt{3}}{3}\right) + A_1 f\left(\frac{\sqrt{3}}{3}\right)$$

它对于  $f(x) = 1$  和  $f(x) = x$  都准确成立, 故有

$$\begin{cases} A_0 + A_1 = 2 \\ -\frac{\sqrt{3}}{3} A_0 + \frac{\sqrt{3}}{3} A_1 = 0 \end{cases}$$

解之得  $A_0 = A_1 = 1$ , 从而得两点高斯-勒让德求积公式

$$\int_{-1}^1 f(x) dx \approx f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

### 例 3 类似地, 可构造三点高斯-勒让德求积公式

$$\int_{-1}^1 f(x) dx \approx \frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right)$$

为了应用方便, 人们提供了对应各种正交多项式的高斯型求积公式的节点-系数表. 下面给出高斯-勒让德求积公式节点-系数表的前几行, 其它情况可参见[A1].

高斯-勒让德求积公式节点-系数表

| $n$ | $x_i$                                                                                                               | $A_i$                                                             |
|-----|---------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| 2   | $\pm 0.57735 \quad 02691 \quad 86926$                                                                               | 1.00000 00000 00000                                               |
| 3   | $0.00000 \quad 00000 \quad 00000$<br>$\pm 0.77459 \quad 66692 \quad 41483$                                          | 0.88888 88888 88889<br>0.55555 55555 55556                        |
| 4   | $\pm 0.33998 \quad 10435 \quad 84856$<br>$\pm 0.86113 \quad 63115 \quad 94053$                                      | 0.65214 51548 62546<br>0.34785 48451 37454                        |
| 5   | $0.00000 \quad 00000 \quad 00000$<br>$\pm 0.53846 \quad 93101 \quad 05683$<br>$\pm 0.90617 \quad 98459 \quad 38664$ | 0.56888 88888 88889<br>0.47862 86704 99366<br>0.23692 68850 56189 |

对于一般区间  $[a, b]$  上的积分, 可以先利用变量变换

$$x = \frac{b-a}{2}t + \frac{a+b}{2}$$

将它化为区间  $[-1, 1]$  上的积分

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

然后再用高斯-勒让德公式来计算。

方法 III. 针对带权函数  $\rho(x) \geq 0$  的一般积分  $\int_a^b \rho(x)f(x)dx$  构造的求积公式

$$\int_a^b \rho(x)f(x)dx \approx \sum_{i=0}^n A_i f(x_i) \quad (6.24)$$

显然, 当  $\rho(x) = 1$  时, 即为上述普通积分情况; 而任何普通积分  $\int_a^b f(x)dx$ , 都可写成  $\int_a^b \rho(x) \frac{f(x)}{\rho(x)} dx$ , 从而化成带权形式的积分。

若求积公式 (6.24) 具有  $2n+1$  次代数精度, 则称它为在  $[a, b]$  上带权  $\rho(x)$  的高斯型求积公式; 节点  $x_0, x_1, \dots, x_n$  仍称为高斯点。

带权函数的积分也可用待定系数法来构造高斯型求积公式。

例 4 构造下列形式的高斯型求积公式

$$\int_0^1 \sqrt{x} f(x) dx \approx A_0 f(x_0) + A_1 f(x_1)$$

令它对  $f(x) = 1, x, x^2, x^3$  准确成立, 得下列关系式

$$\begin{cases} A_0 + A_1 = \int_0^1 \sqrt{x} dx = \frac{2}{3} \\ x_0 A_0 + x_1 A_1 = \int_0^1 \sqrt{x} \cdot x dx = \frac{2}{5} \\ x_0^2 A_0 + x_1^2 A_1 = \int_0^1 \sqrt{x} \cdot x^2 dx = \frac{2}{7} \\ x_0^3 A_0 + x_1^3 A_1 = \int_0^1 \sqrt{x} \cdot x^3 dx = \frac{2}{9} \end{cases}$$

解之得  $x_0 = 0.821162, x_1 = 0.289949, A_0 = 0.389111,$

$A_1 = 0.27756$ , 故得高斯型求积公式为



$$\int_0^1 \sqrt{x} f(x) dx \approx 0.389111 f(0.821162) + 0.277556 f(0.289949)$$

对于某些特殊的权函数,也可利用正交多项式的零点来构造高斯型求积公式。这是因为仿前可以证明高斯点就是区间  $[a, b]$  上带权  $\rho(x)$  的  $n+1$  次正交多项式的零点。

**例 5** 构造下列形式的高斯型求积公式

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx \approx A_0 f(x_0) + A_1 f(x_1)$$

因为节点  $x_0, x_1$  是  $[-1, 1]$  上带权  $\rho(x) = \frac{1}{\sqrt{1-x^2}}$  的二次正交多项式的零点,而这个正交多项式就是二次切比雪夫多项式

$$T_2(x) = \cos(2 \arccos x)$$

故可得  $x_0 = -\frac{\sqrt{2}}{2}, x_1 = \frac{\sqrt{2}}{2}$ , 即所求公式形如

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx \approx A_0 f\left(-\frac{\sqrt{2}}{2}\right) + A_1 f\left(\frac{\sqrt{2}}{2}\right)$$

由于它对  $f(x) = 1, x$  准确成立,故又可得  $A_0 = A_1 = \frac{\pi}{2}$ . 于是所求高斯型求积公式为

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx \approx \frac{\pi}{2} f\left(-\frac{\sqrt{2}}{2}\right) + \frac{\pi}{2} f\left(\frac{\sqrt{2}}{2}\right)$$

由权函数  $\rho(x) = \frac{1}{\sqrt{1-x^2}}$  所建立的高斯型公式

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx \approx \sum_{i=0}^n A_i f(x_i) \quad (6.25)$$

特别地称为高斯-切比雪夫求积公式。容易得到, (6.25) 式的高斯点是  $n+1$  次切比雪夫多项式的零点, 即

$$x_i = \cos((2i+1)/2(n+1)), \quad i = 0, 1, \dots, n$$

最后, 我们不加证明地指出, 对于高斯型公式 (6.23), 其余项

$$R[f] = \int_a^b f(x) dx - \sum_{i=0}^n A_i f(x_i) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \int_a^b \omega^2(x) dx \quad (6.26)$$

这里  $\omega(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$ ,  $\xi \in (a, b)$  且依赖于  $x$ . 此外, 高斯型公式不但精度较高, 而且是数值稳定的; 且当  $n \rightarrow \infty$  时,  $\sum_{i=0}^n A_i f(x_i)$  收敛于  $\int_a^b f(x) dx$ .

### 3. 高斯-勒让德求积公式的 FORTRAN 程序

这里给出高斯-勒让德求积公式的 FORTRAN 程序。与上述符号略有不同, 高斯-勒让德求积公式是指插值型公式

$$\int_{-1}^1 f(x) dx \approx \sum_{i=1}^n A_i f(x_i)$$

中的节点  $x_1, x_2, \dots, x_n$  为勒让德多项式

$$L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2-1)^n]$$

的  $n$  个根,而系数

$$A_i = \frac{2}{(1-x_i^2)[L'_n(x_i)]^2}, \quad i = 1, 2, \dots, n$$

下面给出的程序的处理过程是,先指定高斯求积公式的阶数  $n$ ,然后把积分区间分割成小区间,并在小区间上按指定阶的高斯-勒让德求积公式来积分,使之自动满足给定的精度  $\varepsilon$  (即不满足时把积分区间再分小)。

精度检查的方法是,用相邻两次积分的结果  $S_m$  和  $S_{m+1}$  按如下关系来估计误差

$$d = \frac{|S_{m+1} - S_m|}{(|S_{m+1}| + 1)}$$

当  $d \leq \varepsilon$  时,  $S_{m+1}$  即为所求的积分近似值。这样,当积分值  $|S_{m+1}|$  很小,如  $|S_{m+1}| < 1$  时  $d$  相当于绝对误差;当积分值  $|S_{m+1}| \geq 1$  时  $d$  相当于相对误差。

需要注意的是,对于一般积分区间  $[a, b]$ ,则先作变换

$$x_i = \frac{b-a}{2} t_i + \frac{b+a}{2}, \quad i = 1, 2, \dots, n$$

从而化成  $[-1, 1]$  上的积分。

设置哑元如下:

A, B 实型变量,输入参数;积分下限和上限。

N 整型变量,输入参数;所用高斯公式的节点数。

XI, AI N 个元素的一维实型数组,前者存放  $N$  个高斯点  $x_i$  的值,后者存放  $N$  个求积系数  $A_i$  的值。

EPS 实型变量,积分精度;兼做输出信息标志:若 EPS 为正,表示积分满足精度,若 EPS 为负,表示步长已为机器零,但精度未满足。

S 实型变量,输出参数;积分结果。

F 计算被积函数  $f(x)$  的函数子程序名,形式为  
FUNCTION F(X)

X 为实型变量;本子程序由使用者自编。

子程序

```
SUBROUTINE LEGEND (A, B, N, XI, AI, EPS, S, F)
```

```
DIMENSION XI(N), AI(N)
```

```
M = 1
```

```
C = ABS(A) + ABS(B)
```

```
R = 0.
```

```
10 S = 0.
```

```
H = (B - A)/M
```

```
DO 15 I = 1, N
```

```
AA = A + H*(I - 1)
```

```
BB = A + H*I
```

```
DO 20 J = 1, N
```

```
X = 0.5*((BB - AA)*XI(J) + AA + BB)
```

```

20 S = S + AI(J)*F(X)
15 CONTINUE
   S = 0.5*S*H
   IF (EPS - ABS(S - R)/(ABS(S) + 1.)) 30, 40, 40
30 R = S
   M = M + 1
   IF (C + H .NE. C) GO TO 10
   EPS = - EPS
40 RETURN
   END

```

注：子程序中的第 7, 9, 10 三行涉及到整型量和实型量的混合运算，这在许多实际运行的 FORTRAN IV 编译中是通得过的。否则，需用 FLOAT 函数把整型量化为实型量参加运算。

#### 算例 计算积分

$$S = \int_3^{12} x^3 \cos x^2 dx$$

取节点数  $N = 5$ ，积分精度  $\epsilon = 10^{-7}$ 。高斯点及系数查表。

解题程序如下：

```

C      THE MAIN PROGRAM
      EXTERNAL F
      DIMENSION XI(5), AI(5)
      DATA XI/ -0.9061798, -0.5384669, 0.0, 0.5384669, 0.90617981
      DATA AI/0.2369269, 0.4786287, 0.5688889, 0.4786287, 0.23692691
      CALL LEGEND (3.0, 12.0, 0, 5, XI, AI, 1E-7, S, F)
      WRITE (6, 100) S, EPS
      FORMAT (1X, 2HS = , F15.7, 10X, 4HEPS = , E10.1)
      STOP
      END

C      SUBROUTINE LEGEND (A, B, N, XI, AI, EPS, S, F)
      : (本子程序段体)
      END

C      FUNCTION F(X)
      E = X**3 * COS (X*X)
      RETURN
      END

```

计算结果：

$$S = -36.3170424 \quad EPS = 0.1E-06$$

其中 EPS 为正,表示积分满足精确度。

从解题程序可见,使用高斯-勒让德积分程序可能出现的麻烦是需要输入相当多的节点值及系数值。在经常使用这种程序的场合,不妨考虑把高斯-勒让德求积公式的节点-系数表建立一个公用的数据文件,以供不同需要(不同的 N)时使用。

## 6-5 多重积分与广义积分计算

### 1. 求多重积分的高斯法及其 FORTRAN 程序

我们知道,多重积分,比如二重积分

$$I = \int_a^b \int_{y_1(x)}^{y_2(x)} f(x, y) dx dy$$

通常化成两个单积分来处理,即

$$I = \int_a^b \left[ \int_{y_1(x)}^{y_2(x)} f(x, y) dy \right] dx$$

现在我们考虑求下列  $n$  重积分的高斯法及其 FORTRAN 程序<sup>[1,63]</sup>。

$$\int_{FL_1}^{FU_1} f_1(x_1) dx_1 \int_{FL_2(x_1)}^{FU_2(x_1)} f_2(x_1, x_2) dx_2 \cdots \int_{FL_n(x_1, x_2, \dots, x_{n-1})}^{FU_n(x_1, x_2, \dots, x_{n-1})} f_n(x_1, x_2, \dots, x_n) dx_n$$

其中  $n$  为任意正整数,是积分重数。

根据高斯求积公式,计算  $\int_{-1}^1 f(t) dt$  时,在  $[-1, 1]$  上取  $mp$  个插值点,于是有

$$\int_{-1}^1 f(t) dt \approx \sum_{k=1}^{mp} A_k f(t_k)$$

其中插值点  $t_1, t_2, \dots, t_{mp}$  是勒让德多项式  $x_{mp}(x)$  的  $mp$  个根,即高斯点,而  $A_k$  为高斯系数。

当计算的是  $\int_a^b f(x) dx$ , 即积分区间为  $[a, b]$  时,则进行变换

$$x_k = \frac{b-a}{2} t_k + \frac{b+a}{2}$$

以求高斯点  $x_1, x_2, \dots, x_{mp}$ 。

对单积分  $\int_a^b f(x) dx$  的计算,则先将积分区间  $[a, b]$  划分为等长度的  $k_{s1}$  个子区间,在每个子区间上用  $mp$  个点高斯公式求积分近似值,然后求和得整个区间上的积分近似值。

对  $n$  重积分的计算,则分别将第  $1, 2, \dots, n$  层积分区间等分为  $k_{s1}, k_{s2}, \dots, k_{sn}$  个子区间,并按上述区间变换公式先求出各积分区间上的第一个子区间中的第一个高斯点  $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ , 然后暂时固定  $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n-1}$ , 按高斯公式计算最内层积分

$$\int_{FL_n(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n-1})}^{FU_n(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n-1})} f_n(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n-1}, x_n) dx_n$$

再顺次从内到外计算出各层积分值,最后得到所求积分的近似值。

这里,我们限定,第  $1, 2, \dots, n$  层积分区间所分成的子区间个数  $k_{s1}, k_{s2}, \dots, k_{sn}$  可以彼此不同,但在各子区间上使用高斯公式时的插值点个数  $mp$  必须相同。

设置哑元如下:

- N 整型变量,输入参数;积分重数.  
N1 整型变量,输入参数;积分重数加 1.  
KS N 个元素的一维整型数组,输入参数;表示各层积分区间所划分的子区间个数.  
MP 整型变量,输入参数;各子区间上插值点的个数.  
U, W 均为 MP 个元素的一维实型数组,输入参数;前者存放 MP 个高斯点,后者存放 MP 个高斯系数.

FUN, FL, FU 分别为计算被积函数、计算各层积分的下限值和上限值的哑过程名,形式依次为

FUNCTION FUN (J, X)

FUNCTION FL (J, X)

FUNCTION FU (J, X)

其中整型变量 J 为积分层数, X 为 J 个元素的一维实型数组. 这三个函数子程序由使用者自编.

- GS1 实型变量,输出参数;积分结果.  
C, X 均为 N 个元素的一维实型数组,工作单元.  
KK  $2 * N$  个元素的二维整型数组,工作单元.  
DD  $2 * N1$  个元素的二维实型数组,工作单元.

子程序

```
SUBROUTINE MULEGA (N, N1, KS, MP, U, W,  
1 C, X, KK, DD, GS1, FUN, FL, FU)  
  DIMENSION KS(N), U(MP), W(MP), C(N), X(N), KK(2, N),  
  DD (2, N1)  
  
  M = 1  
  DD (1, N1) = 1.  
  DD (2, N1) = 1.  
20 DO 10 J = M, N  
  A = FL (J, X)  
  B = KS (J)  
  DD (1, J) = 0.5 * (FU(J, X) - A)/B  
  C(J) = DD (1, J) + A  
  X(J) = DD (1, J) * U(1) + C(J)  
  DD (2, J) = 0.  
  KK(1, J) = 1  
10 KK (2, J) = 1  
  K = N  
30 F = FUN (K, X)
```

```

KB = KK (1, K)
DD(2, K) = DD(2, K + 1)*DD(1, K + 1)*F*W(KB) + DD(2, K)
KK(1, K) = KK(1, K) + 1
IF (KK(1, K) .LE. MP) GO TO 33
IF (KK(2, K) .LT. KS(K)) GO TO 32
K = K + 1
IF (K .NE. 0) GO TO 30
GSI = DD(2, 1)* * DD(1, 1)
RETURN
32 KK(2, K) = KK(2, K) + 1
C(K) = C(K) + DD(1, K) * 2.
KK(1, K) = 1
33 KC = KK(1, K)
X(K) = DD(1, K) * U(KC) + C(K)
IF (K .EQ. N) GO TO 30
M = K + 1
GO TO 20
END

```

**算例1** 计算下列三重积分

$$\int_0^2 dz \int_0^{\sqrt{2z-z^2}} dy \int_0^{\sqrt{2z-z^2-y^2}} (x^2 + y^2 + z^2)^{-\frac{1}{2}} dx$$

为了程序书写方便,上述积分略为换一下符号:

$$\int_0^2 dx_1 \int_0^{\sqrt{2x_1-x_1^2}} dx_2 \int_0^{\sqrt{2x_1-x_1^2-x_2^2}} (x_1 + x_2 + x_3)^{-\frac{1}{2}} dx_3$$

取 MP = 3, 并循环四次对各层积分区间作不同划分 (即 KS(1) ≠ KS(2) ≠ KS(3)) 进行计算。

解题程序:

```

C      THE MAIN PROGRAM
      DIMENSION L(3), U(3), W(3), C(3), X(3), KK(2, 3), DD(2, 4)
      EXTERNAL FUN, FL, FU
      U(1) = - 0.77459666
      U(2) = 0.
      U(3) = - U(1)
      W(1) = 0.55555556
      W(2) = 0.88888889
      W(3) = 0.55555556
      DO 200 I = 1, 4

```

```

      II = 2 * I
      L(1) = II
      L(2) = II + 2 * I
      L(3) = II + 4 * I
      CALL MULEGA (3, 4, L, 3, U, W, C, X, KK, DD,
1 GSI, FUN, FL, FU)
200 WRITE (6, 100) GSI
100 FORMAT (1X, 4H GSI = , E20.8)
      STOP
      END

```

C

```

      SUBROUTINE MULEGA (N, NI, KS, MP, U, W,
1 C, X, KK, DD, GSI, FUN, FL, FU)
      : (本子程序段体)
      END

```

C

```

      FUNCTION FUN (J, X)
      DIMENSION X(3)
      IF (J .LT. 3) GO TO 5
      FUN = 1.0/SQRT (X(1)*X(1) + X(2)*X(2) + X(3)*X(3))
      RETURN
5 FUN = 1.0
      RETURN
      END

```

C

```

      FUNCTION FL(J, X)
      DIMENSION X(3)
      EL = 0.0
      RETURN
      END

```

C

```

      FUNCTION FU(J, X)
      DIMENSION X(3)
      GO TO (1, 2, 3), J
1 FU = 2.0
      RETURN
2 FU = SQRT (2.0 * X(1) - X(1) * X(1))
      RETURN
3 FU = SQRT (2.0 * X(1) - X(1) * X(1) - X(2) * X(2))

```

RETURN  
END

上机计算,对于各层积分区间不同分法的四种计算结果依次为

GSI = 0.10532556E 01  
GSI = 0.10493395E 01  
GSI = 0.10479680E 01  
GSI = 0.10474765E 01

**算例 2** 计算六重积分

$$\int_{-1}^1 \int_{-(1-x_1^2)}^{1-x_1^2} \int_{-(1-x_1^2-x_2^2)}^{1-x_1^2-x_2^2} \int_{-(1-x_1^2-x_2^2-x_3^2)}^{1-x_1^2-x_2^2-x_3^2} \cdots \int_{-(1-x_1^2-x_2^2-x_3^2-x_4^2-x_5^2)}^{1-x_1^2-x_2^2-x_3^2-x_4^2-x_5^2} e^{x_1^2+x_2^2+\cdots+x_5^2+x_6^2} dx_1 dx_2 \cdots dx_6$$

如同算例 1, 取  $MP = 3$ . 但这里对三次计算, 每次分别取

$$KS(1) = KS(2) = \cdots = KS(6) = 2, 3, 4$$

类似于算例 1 的解题程序, 可得三次计算结果依次为

GBI = 4.65850  
GBI = 4.63330  
GBI = 4.63100

我们可以发现, 随着积分层数的增加, 所需花费的计算时间远比插值区间加密时所需花费的计算时间为多。对于高层积分, 可考虑采用更有效率的方法, 如数论网格方法<sup>[41]</sup>、蒙特卡罗 (Monte Carlo) 方法等。

## 2. 无穷区间上的广义积分计算

假定需要计算下列形式的广义积分

$$\int_a^\infty f(x) dx \quad \left( \text{它定义为 } \lim_{b \rightarrow \infty} \int_a^b f(x) dx \right)$$

我们已知这个积分存在, 并且有限。通常采用如下一些处理方法。

**方法 I** 在某些情况下, 可以通过适当的变量变换, 使它化为有穷区间上的积分, 然后再选用合适的数值方法计算它。

**例 1** 计算积分

$$I = \int_1^\infty \frac{dx}{(1+x)\sqrt{x}}$$

令  $x = 1/t$ , 则有

$$\begin{aligned} \int_1^\infty \frac{dx}{(1+x)\sqrt{x}} &= \int_1^0 \frac{-\frac{1}{t^2} dt}{\left(1 + \frac{1}{t}\right)\sqrt{\frac{1}{t}}} = \int_0^1 \frac{\sqrt{t} dt}{t(1+t)} = \int_0^1 \frac{d(2\sqrt{t})}{1+t} \\ &= \frac{2\sqrt{t}}{1+t} \Big|_0^1 + 2 \int_0^1 \frac{\sqrt{t}}{(1+t)^2} dt = 1 + 2 \int_0^1 \frac{\sqrt{t}}{(1+t)^2} dt \end{aligned}$$

最后一个积分已变成正常积分了, 这时便可用标准的数值方法来求它的数值。

**方法 II** 若积分值有限, 被积函数  $f(x)$  是光滑的, 则随着  $x \rightarrow \infty$ ,  $f(x)$  或其中某些项可能以某种方式渐近于零。于是把积分写成



$$\int_a^{\infty} f(x) dx = \int_a^c f(x) dx + \int_c^{\infty} f(x) dx$$

则右端第一个积分可用任一合适的数值方法求值, 第二个积分可用解析法近似求解. 看例子.

### 例 2 计算积分

$$I = \int_0^{\infty} \frac{dx}{e^x + e^{-x}}$$

注意到当  $x = 5$  时, 有  $e^5 = 148.413159$ ,  $e^{-5} = 0.006738$ . 因此, 当  $x = 5$  或更大时, 可能把  $e^{-x}$  略去. 把积分改写成

$$I = \int_0^5 \frac{dx}{e^x + e^{-x}} + \int_5^{\infty} \frac{dx}{e^x + e^{-x}} \approx \int_0^5 \frac{dx}{e^x + e^{-x}} + \int_5^{\infty} \frac{dx}{e^x}$$

第一个积分用辛普生公式可得结果 0.778659; 第二个积分用解析法得

$$\int_5^{\infty} \frac{dx}{e^x} = -e^{-x} \Big|_5^{\infty} = e^{-5} = 0.006738$$

于是得

$$I \approx 0.778659 + 0.006738 = 0.785397$$

这个积分的精确解是  $\frac{\pi}{4} = 0.785398$ .

**方法 III** 按某种极限过程来计算. 例如把积分写成

$$I = \int_0^{\infty} f(x) dx = \int_0^{a_1} f(x) dx + \int_{a_1}^{a_2} f(x) dx + \cdots + \int_{a_n}^{a_{n+1}} f(x) dx + \cdots$$

其中  $0 < a_1 < a_2 < \cdots < a_n < a_{n+1} < \cdots$  是一趋于无穷的序列. 右端每一个积分都是正常积分, 都可用标准的求积方法计算, 而当

$$\int_0^{a_n} f(x) dx \approx \int_0^{a_{n+1}} f(x) dx$$

时, 我们就认为

$$I = \int_0^{\infty} f(x) dx \approx I_{n+1} = \int_0^{a_{n+1}} f(x) dx$$

### 例 3 计算积分

$$I = \int_0^{\infty} \frac{e^{-x}}{1+x^4} dx$$

取  $a_n = 2^n$ , 则

$$I_n = \int_0^{a_n} \frac{e^{-x}}{1+x^4} dx$$

就  $n = 0, 1, \cdots, 4$  计算上述正常积分的结果如下:

| $n$ | $a_n$ | $I_n$    |
|-----|-------|----------|
| 0   | 1     | 0.572026 |
| 1   | 2     | 0.627460 |
| 2   | 4     | 0.630440 |
| 3   | 8     | 0.630478 |
| 4   | 16    | 0.630478 |

#### 例4 计算积分

$$I = \int_0^{\infty} e^{-x^2} dx$$

当  $x \geq k > 0$  时, 有  $x^2 \geq kx$ , 故有估计式

$$\int_k^{\infty} e^{-x^2} dx \leq \int_k^{\infty} e^{-kx} dx = \frac{e^{-k^2}}{k}$$

当  $k=4$  时,  $\frac{e^{-k^2}}{k} \approx 10^{-8}$ , 因此在  $10^{-7}$  的精度要求下, 可取

$$\int_0^{\infty} e^{-x^2} dx \approx \int_0^4 e^{-x^2} dx$$

此外, 还可利用无穷区间上高斯公式(如拉盖尔公式、埃尔米特公式)以及样条函数方法来求无穷区间上的广义积分, 可参考  $[A_1, A_4, A_7]$  等. 总之, 要根据具体问题作灵活、综合的处理.

#### 3. 无界函数的广义积分计算

考虑  $f(x)$  在  $x=0$  的近旁无界的广义积分

$$\int_0^1 f(x) dx \quad \left( \text{定义为 } \lim_{r \rightarrow 0^+} \int_r^1 f(x) dx \right)$$

处理这类积分的常用方法有

**方法 I** 用变量变换或分部积分法消去奇点. 如下列例子.

**例 1** 
$$I = \int_0^1 x^{-\frac{1}{n}} f(x) dx$$

其中  $n \geq 2$ ,  $f(x)$  在  $[0, 1]$  上连续. 若令  $x = t^n$ , 则

$$I = \int_0^1 t^{-1} f(t^n) \cdot n t^{n-1} dt = n \int_0^1 t^{n-1} f(t^n) dt$$

显然, 右端积分的被积函数在区间  $[0, 1]$  上连续.

**例 2** 
$$\begin{aligned} \int_0^1 \frac{\cos x}{\sqrt{x}} dx &= 2 \sqrt{x} \cos x \Big|_0^1 + 2 \int_0^1 \sqrt{x} \sin x dx \\ &= 2 \cos 1 + 2 \int_0^1 \sqrt{x} \sin x dx \end{aligned}$$

**方法 II** 截去使被积函数无限的区间.

**例 3** 计算积分

$$\int_0^1 \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx$$

其中  $f(x)$  在  $[0, 1]$  上连续, 且  $f(x) \leq 1$ . 由于在  $[0, 1]$  上  $\sqrt{x} \leq \sqrt[3]{x}$ ,

故 
$$\left| \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} \right| \leq \frac{1}{2\sqrt{x}},$$

从而 
$$\left| \int_0^{\delta} \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx \right| \leq \int_0^{\delta} \left| \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} \right| dx \leq \int_0^{\delta} \frac{1}{2\sqrt{x}} dx = \sqrt{x} \Big|_0^{\delta} = \sqrt{\delta}$$

即 
$$\left| \int_0^1 \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx - \int_{\delta}^1 \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx \right| \leq \sqrt{\delta}$$

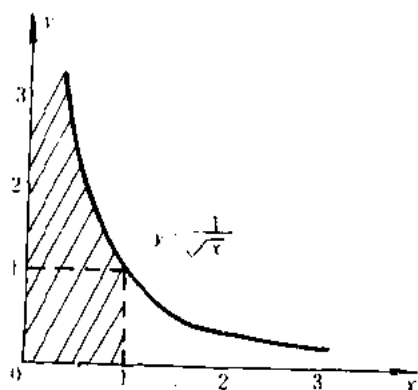


图 6-6

于是在误差不超过  $\sqrt{\delta}$  的精度要求下可取

$$\int_0^1 \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx \approx \int_0^1 \frac{f(x)}{\sqrt{x} + \sqrt[3]{x}} dx$$

方法 III 交换自变量和因变量的位置.

例 4 计算积分

$$I = \int_0^1 \frac{1}{\sqrt{x}} dx$$

由图 6-6 易得, 若把  $y$  看成自变量, 则有

$$I = \int_0^1 \frac{1}{\sqrt{x}} dx = 1 + \int_1^{\infty} \frac{1}{y^2} dy$$

最后一个积分显然可用解析法求值. 如果仍用数值方法来处理, 可把它写成

$$\int_1^{\infty} \frac{1}{y^2} dy = \int_1^{10} \frac{1}{y^2} dy + \int_{10}^{50} \frac{1}{y^2} dy + \int_{50}^{250} \frac{1}{y^2} dy + \int_{250}^{1000} \frac{1}{y^2} dy + \int_{1000}^{\infty} \frac{1}{y^2} dy$$

略去最后一项, 其余用辛普生方法求解, 可得

$$\int_1^{\infty} \frac{1}{y^2} dy \approx 0.900292 + 0.080019 + 0.016046 + 0.003003 = 0.999360$$

于是得

$$I = \int_0^1 \frac{1}{\sqrt{x}} dx = 1.999360$$

这个积分的精确值是 2.000000.

同样, 还可利用带权函数的高斯型求积公式来处理这类积分. 在参考书<sup>[47]</sup>中有许多处理广义积分计算的示例.

## 6-6 数值微分

如果已知某函数关系  $f(x)$  的一组样点

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

比如说某移动目标的观测值, 要求据此推算出  $f(x)$  的一阶、二阶导数  $f'(x)$ ,  $f''(x)$  或者其在各个时刻  $x_i$  的值, 即速度、加速度, 这就是数值微分的问题.

这里我们介绍两种基本而又常用的数值微分方法.

### 1. 用插值多项式求数值导数

先写出  $f(x)$  在某些点  $x_i$  ( $i = 0, 1, \dots, n$ ) 上的  $n$  次插值多项式  $F(x)$ , 然后用  $F(x)$  近似地代替  $f(x)$ , 并对  $F(x)$  求导, 就可得  $f(x)$  的导数的近似值. 需注意的是, 两个相差不大的函数, 其导数可能相差很大, 因此用这个方法求数值导数时要注意误差估计. 分几种情况讨论.

#### (1) 两点公式

已知  $f(x)$  在节点  $x_0, x_1 = x_0 + h$  的函数值为  $f(x_0), f(x_1)$ , 且  $f(x)$  的二阶导数存在, 作线性插值函数  $F(x)$ , 则

$$f(x) = F(x) + R(x)$$

$$\begin{aligned} &= \frac{x-x_1}{x_0-x_1}f(x_0) + \frac{x-x_0}{x_1-x_0}f(x_1) + \frac{f''(\xi)}{2!}(x-x_0)(x-x_1) \\ &= \frac{f(x_0)}{h}(x_1-x) + \frac{f(x_1)}{h}(x-x_0) + \frac{f''(\xi)}{2}(x-x_0)(x-x_1) \end{aligned}$$

其中  $\xi \in (x_0, x_1)$  且依赖于  $x$ 。于是可得

$$f'(x) = \frac{f(x_1) - f(x_0)}{h} + \frac{2x - x_0 - x_1}{2}f''(\xi) + \frac{(x-x_0)(x-x_1)}{2} \frac{d}{dx}[f''(\xi)]$$

由于上式很难对任意的  $x$  作出误差估计, 因此通常限定只求节点  $x_0, x_1$  上的导数值, 于是得两点公式

$$\begin{cases} f'(x_0) = \frac{f(x_1) - f(x_0)}{h} - \frac{h}{2}f''(\xi) \\ f'(x_1) = \frac{f(x_1) - f(x_0)}{h} + \frac{h}{2}f''(\xi) \end{cases} \quad \xi \in (x_0, x_1) \quad (6.27)$$

根据两点公式, 可知只要  $x_0, x_1$  充分近, 就可把  $\frac{f(x_1) - f(x_0)}{h}$  作为  $f'(x_0)$  或  $f'(x_1)$  的近似值, 其误差界为  $\frac{h}{2}M$ , 这里  $M = \max_{x \in [x_0, x_1]} |f''(x)|$ 。

## (2) 三点公式

已知  $f(x)$  在等距节点  $x_i$  的函数值为  $f(x_i)$ ,  $i = 0, 1, 2$ 。且  $f(x)$  的三阶导数存在, 作二次插值函数  $F(x)$  并仿照上述做法, 可得

$$f'(x) = F'(x) + R'(x)$$

$$\begin{aligned} \text{其中} \quad F'(x) &= \frac{2x-x_1-x_2}{(x_1-x_1)(x_0-x_2)}f(x_0) + \frac{2x-x_0-x_2}{(x_1-x_0)(x_1-x_2)}f(x_2) \\ &\quad + \frac{2x-x_0-x_1}{(x_2-x_0)(x_2-x_1)}f(x_1) \\ R'(x) &= \frac{f'''(\xi)}{3!}[(x-x_0)(x-x_1) + (x-x_0)(x-x_2) + (x-x_1)(x-x_2)] \\ &\quad + \frac{1}{3!}[(x-x_0)(x-x_1)(x-x_2)] \frac{d}{dx}[f'''(\xi)] \end{aligned}$$

( $\xi \in (x_0, x_2)$  且与依赖于  $x$ )

同样, 我们只限于求节点上的导数值, 于是可得三点求导公式

$$\begin{cases} f'(x_0) = \frac{1}{2h}[-3f(x_0) + 4f(x_1) - f(x_2)] + \frac{h^2}{3}f'''(\xi) \\ f'(x_1) = \frac{1}{2h}[-f(x_0) + f(x_2)] - \frac{h^2}{6}f'''(\xi) \\ f'(x_2) = \frac{1}{2h}[f(x_0) - 4f(x_1) + 3f(x_2)] + \frac{h^2}{3}f'''(\xi) \end{cases} \quad \xi \in [x_0, x_2] \quad (6.28)$$

类似地, 还可建立更高阶的求导公式。

(3) 一般地, 若已知  $f(x)$  在  $n+1$  个节点  $x_i \in [a, b]$  ( $i = 0, 1, \dots, n$ ) 上的函数值  $f(x_i)$ , 且  $f^{(n+1)}(x)$  在  $[a, b]$  上存在, 则通过类似的途径, 可得涉及  $n+1$  个节点的求

导公式

$$f'(x_i) = F'(x_i) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega'_{n+1}(x_i), \quad i = 0, 1, \dots, n$$

它表明,若取  $f'(x_i) \approx F'(x_i)$ , 则截断误差为  $\frac{f^{(n+1)}(\xi)}{(n+1)!} \omega'_{n+1}(x_i)$ .

需要特别指出的是,当插值多项式  $F(x)$  收敛于  $f(x)$  时,不能保证  $F'(x)$  也收敛于  $f'(x)$ ,而且当节点距离缩小时,虽然截断误差将逐步减小,但舍入误差却会增大,因此缩小步长不一定能提高计算精度.

## 2. 用三次样条插值函数求数值导数

在样条函数理论中可以证明,函数  $f(x)$  的三次样条插值函数  $S(x)$ ,在一定条件下,不仅有  $S(x)$ ,  $S'(x)$ ,  $S''(x)$ ,  $S'''(x)$  分别收敛于  $f(x)$ ,  $f'(x)$ ,  $f''(x)$ ,  $f'''(x)$ , 而且有

$$\begin{aligned} |f(x) - S(x)| &= O(h^4) \\ |f'(x) - S'(x)| &= O(h^3) \\ |f''(x) - S''(x)| &= O(h^2) \\ |f'''(x) - S'''(x)| &= O(h) \end{aligned}$$

其中  $h = \max_{1 \leq i \leq n} h_i$ . 可见,用三次样条插值函数求数值导数,比用插值多项式求数值导数要好.

用三次样条插值函数求数值导数的步骤大致如下:

首先,从已知的函数表

$$\begin{aligned} a = x_0, x_1, x_2, \dots, x_n = b \\ f_0, f_1, f_2, \dots, f_n \end{aligned}$$

和适当的边界条件出发,构造三次样条插值函数

$$\begin{aligned} S(x) = & \frac{(x - x_{i+1})^2[h_i + 2(x - x_i)]}{h_i^3} f_i + \frac{(x - x_i)^2[h_{i+1} + 2(x_{i+1} - x)]}{h_{i+1}^3} f_{i+1} \\ & + \frac{(x - x_{i+1})^2(x - x_i)}{h_i^3} m_i + \frac{(x - x_i)^2(x - x_{i+1})}{h_{i+1}^3} m_{i+1}, \\ & x \in [x_i, x_{i+1}], \quad i = 0, 1, \dots, n-1 \end{aligned}$$

然后,再对上式两边求导数,并用它近似代替  $f'(x)$ , 即得

$$\begin{aligned} f'(x) \approx S'(x) = & \frac{6}{h_i^2} \left[ \frac{(x - x_{i+1})^2}{h_i^2} + (x - x_{i+1}) \right] f_i \\ & + \frac{6}{h_{i+1}^2} \left[ (x - x_i) - \frac{(x - x_i)^2}{h_{i+1}} \right] f_{i+1} \\ & + \frac{1}{h_i} \left[ \frac{3(x - x_{i+1})^2}{h_i} + 2(x - x_{i+1}) \right] m_i \\ & + \frac{1}{h_{i+1}} \left[ \frac{3(x - x_i)^2}{h_{i+1}} - 2(x - x_i) \right] m_{i+1} \\ & x \in [x_i, x_{i+1}], \quad i = 0, 1, \dots, n-1 \end{aligned}$$

若只要求节点  $x_i$  上的导数, 则

$$f'(x_i) \approx m_i \quad i = 0, 1, \dots, n$$

若要求二阶导数  $f''(x)$ , 只要对前式再求一次导数, 即得

$$\begin{aligned} f''(x) \approx S''(x) &= \frac{6}{h_i^2} \left[ 1 + \frac{2(x-x_{i+1})}{h_i} \right] f_i + \frac{6}{h_i^2} \left[ 1 - \frac{2(x-x_i)}{h_i} \right] f_{i+1} \\ &\quad + \frac{2}{h_i} \left[ 1 + \frac{3(x-x_{i+1})}{h_i} \right] m_i + \frac{2}{h_i} \left[ \frac{3(x-x_i)}{h_i} - 1 \right] m_{i+1} \\ x &\in [x_i, x_{i+1}], \quad i = 0, 1, \dots, n-1 \end{aligned}$$

与插值多项式求导公式不同, 样条求导公式可用来计算插值范围内任何一点  $x$  (不仅是节点  $x_i$ ) 上的导数值.

## 习 题 六

1. 在区间  $[-1, 1]$  上, 取  $x_0 = -1, x_1 = 0, x_2 = 1$  构造插值型求积公式, 并求它的代数精度.
2. 确定下列公式中的参数, 使其代数精度尽量高, 并指明所得公式具有的代数精度:

$$(1) \int_{-h}^h f(x) dx \approx A_{-1}f(-h) + A_0f(0) + A_1f(h),$$

$$(2) \int_{-2h}^{2h} f(x) dx \approx A_{-1}f(-h) + A_0f(0) + A_1f(h),$$

$$(3) \int_{-1}^1 f(x) dx \approx \frac{1}{3} [f(-1) + 2f(x_1) + 3f(x_2)],$$

$$(4) \int_0^h f(x) dx \approx \frac{h}{2} [f(0) + f(h)] + ah^2[f'(0) - f'(h)].$$

3. 试证明:

$$A_0 + A_1 + \dots + A_n = b - a$$

其中  $A_i (i = 0, 1, \dots, n)$  是插值型求积公式中的求积系数,  $a$  与  $b$  是积分下、上限.

4. 在区间  $[-1, 1]$  上求三个不同的节点  $x_1, x_2, x_3$ , 使求积公式

$$\int_{-1}^1 f(x) dx \approx C[f(x_1) + f(x_2) + f(x_3)]$$

具有三次代数精度.

5. 用最简单的矩形公式、梯形公式和辛普生公式计算积分

$$I = \int_0^1 \frac{xe^x}{(1+x)^2} dx$$

精确到  $10^{-4}$ .

6. 用梯形公式和辛普生公式计算积分:

$$(1) \int_1^4 \frac{x}{4+x^2} dx \quad (\text{取 } n=6)$$

$$(2) \int_0^1 \frac{(1-e^{-x})^{1/2}}{x} dx \quad (\text{取 } n=10)$$

$$(3) \int_0^{\pi/6} \sqrt{4-\sin^2 x} dx \quad (\text{取 } n=6)$$

7. 用辛普生公式计算积分

$$\int_0^1 e^{-x} dx$$

并估计误差.

8. 用辛普生公式的计算机程序计算下列积分之值:

$$(1) I = \int_0^{\pi} \ln(5-4\cos x) dx,$$

$$(2) I = \int_0^{\pi/2} \frac{\sin x}{\sqrt{1 - 0.25 \sin^2 x}} dx.$$

9. 用自动选步长的梯形公式和辛普生公式计算定积分

$$(1) I = \int_0^1 \frac{1}{1+x^2} dx.$$

$$(2) I = \int_1^7 \frac{x}{\ln(1+x)} dx.$$

10. 证明辛普生公式当  $n \rightarrow \infty$  时收敛于  $\int_a^b f(x) dx$ .

11. 选用合适的积分公式(梯形公式或辛普生公式)计算积分

$$(1) I = \int_0^1 \frac{\sin x}{x} dx.$$

$$(2) I = \int_0^1 e^{-x^2} dx.$$

12. 用依次计算梯形和序列的方法计算积分

$$I = \int_1^2 \frac{\sin x}{x} dx$$

使误差不超过  $\frac{1}{2} \times 10^{-4}$ .

13. 用龙贝格方法计算下列积分:

$$(1) S_1 = 4 \int_0^{\pi/2} \frac{3c^2}{ab} \sin t \cos t \sqrt{b^2 \cos^2 t + a^2 \sin^2 t} dt$$

$$\text{其中 } a = 15, \quad b = 10, \quad c = \sqrt{a^2 - b^2}.$$

$$(2) S_1 = \int_0^{2\pi} a \sqrt{\varphi^2 + 1} d\varphi$$

$$\text{其中 } a = 15.$$

$$(3) S_1 = \int_a^b (e^{\frac{x}{c}} + e^{-\frac{x}{c}}) dx$$

$$\text{其中 } b = 55, \quad c = 15.$$

14. 用三点及五点高斯型求积公式计算积分

$$\int_1^2 \frac{1}{x} dx$$

15. 用高斯公式求积分

$$(1) \int_0^1 \frac{1}{1+x^2} dx.$$

$$(2) \int_0^1 \frac{\sqrt{x}}{(1+x)^2} dx.$$

16. 用高斯公式求积分

$$(1) I = \int_0^1 \frac{\arctg x}{x^{3/2}} dx.$$

$$(2) I = \int_0^1 \sqrt{1+2x} dx.$$

17. 证明高斯系数  $A_i$  之和等于 2:

$$\sum_{i=1}^n A_i = 2$$

18. 计算积分

$$I = \int_1^{\infty} \frac{dx}{(1+x)\sqrt{x}}$$

(提示: 先将  $I$  分成两部分)

$$I = I_1 + I_2 = \int_0^1 \frac{dx}{(1+x)\sqrt{x}} + \int_1^{\infty} \frac{dx}{(1+x)\sqrt{x}}$$

对  $I_1$  令  $x = u^{-1}$ , 可把  $I_1$  化为  $\int_0^1 \frac{dx}{(1+x)\sqrt{x}}$ . 但注意  $x=0$  是奇点, 先运用分部积分法化

$$\int_0^1 \frac{dx}{(1+x)\sqrt{x}} = 1 + 2 \int_0^1 \frac{\sqrt{x}}{(1+x)^2} dx$$

最后一个积分再用高斯公式求解.)

19. 试求关于区间  $[-1, 1]$ , 权函数  $\rho(x) = x^2$  的两点、三点高斯型求积公式.

20. 导出用辛普生公式计算重积分

$$\int_a^b \int_c^d f(x, y) dx dy$$

的公式, 并用它来计算(精确到  $10^{-4}$ )

$$\int_0^1 \int_{-1}^1 (x^2 + 2y^2) dx dy$$

21. 计算二重积分

$$\int_0^{1.2} \int_1^{e^{-x}} (x^2 + y^2)^{1/2} e^{-xy} dy dx$$

22. 计算下列积分

$$(1) \int_0^{\infty} \frac{e^{-x}}{4+x} dx,$$

$$(2) \int_0^{\infty} \ln \frac{e^x + 1}{e^x - 1} dx.$$

23. 计算下列积分

$$(1) \int_0^1 \frac{(1 - e^{-x})^{1/2}}{x} dx.$$

$$(2) \int_0^1 \frac{1}{x} \ln \frac{1+x}{1-x} dx$$

24. 已知函数表

| $x$    | 1.0  | 1.5   | 2.0   | 2.5   |
|--------|------|-------|-------|-------|
| $f(x)$ | 8.00 | 13.75 | 21.00 | 29.75 |

求  $f'(1)$  的近似值.

25. 用两点公式和三点公式求函数

$$f(x) = \frac{1}{(1+x)^2}$$

在  $x = 1.0, 1.2$  处的导数值, 并估计误差.  $f(x)$  的值给出如下:

| $x$    | 1.0    | 1.1    | 1.2    | 1.3    |
|--------|--------|--------|--------|--------|
| $f(x)$ | 0.2500 | 0.2268 | 0.2066 | 0.1890 |



## 第7章 方程求根与非线性方程组数值解

### 7-1 引言

由函数  $f(x)$  组成的方程

$$f(x) = 0 \quad (7.1)$$

当  $f(x)$  是  $x$  的  $n$  次多项式 ( $n > 2$ ) 时称为**高次代数方程**, 如

$$x^4 - 2x^3 + 2x^2 + x^3 + 6x^2 - 6x + 8 = 0$$

当  $f(x)$  是超越函数时称为**超越方程**, 如

$$(x^2 + 4)e^x + x \sin x - \cos x = 0$$

一般地,  $f(x)$  可能是实函数, 也可能是复函数. 当方程 (7.1) 中的  $f(x)$  是非线性函数时统称为**非线性方程**.

使方程 (7.1) 成为恒等式的数  $x^*$ , 即  $f(x^*) = 0$ , 称为方程的**解**或方程的**根**. 方程  $f(x) = 0$  的根也称为函数  $f(x)$  的**零点**.

方程的根可能是实数, 也可能是复数, 相应地称为**实根**和**复根**. 通常, 求复根又在解多项式方程时才加以考虑.

如果对于数  $x^*$ , 有  $f(x^*) = 0$ , 但  $f'(x^*) \neq 0$ , 则  $x^*$  称为**单根**; 如果有  $f(x^*) = f'(x^*) = \dots = f^{(k)}(x^*) = 0$ , 但  $f^{(k+1)}(x^*) \neq 0$ , 则  $x^*$  称为 **$k$ 重根**.

对于高次代数方程, 下面我们马上就要讲到, 其根(实的或复的)的个数与其次数相同. 对于超越方程, 其根可能是一个或几个, 也可能是无穷多个或无根存在.

实际问题中, 由于方程的物理背景不同, 应用目的也不同, 因此, 对解方程的要求往往也不一定相同. 例如, 有的要求求出方程所有的根, 有的只要求求出其中某一特殊的根; 有的是已知根的初始近似值要求精确化, 有的则只要判断右半平面是否有根, 或在单位圆外是否有根, 而并不要求把根求出来, 等等.

这一节我们先从理论上探讨方程根的存在性及其分布, 即研究方程有没有根? 如果有根, 有几个根? 哪儿有根? 求出有根的区域或平面区域, 把它们分成充分小的子区间或子平面域, 使在每个子区间或子平面域内或是没有根, 或是只有一个根或一对共轭复根, 这叫做根的**隔离**. 做好根的隔离工作有助于获得方程各根的初步近似值.

其次, 在这一节中, 我们还将研究迭代法的一般理论. 迭代法是迄今为止在方程求根以及数值分析各个领域中都有重要应用的方法.

#### 1. 方程根的存在性

我们考虑  $n$  次代数方程

$$a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n = 0 \quad (7.2)$$

其中, 左边是  $n$  次多项式,  $x$  是复变量,  $a_0, a_1, \dots, a_n$  ( $a_0 \neq 0$ ) 称为**系数**, 是实常数或复常数. 实际问题中经常遇到的是实系数代数方程.

代数方程的根的存在性已经解决了,它建立在下面有关定理的基础上.下面我们将叙述一系列与方程根的存在性和根的隔离有关的定理,略加注解,不予证明.

**定理1** (代数基本定理) 在复数范围内,  $n$  次代数方程至少有一个根.

在初等代数中,我们已经知道,一次代数方程有一个根,即

$$\text{方程: } a_0x + a_1 = 0 \quad (a_0 \neq 0)$$

$$\text{根: } x = -\frac{a_1}{a_0}$$

二次代数方程有两个根,即

$$\text{方程: } a_0x^2 + a_1x + a_2 = 0 \quad (a_0 \neq 0)$$

$$\text{根: } x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0a_2}}{2a_0}$$

在这里,当方程系数都是实数时,一次方程的根是实根,而二次方程的两个根可能是实根或一对共轭复根.类似地,我们也知道,三次和四次代数方程也分别有三个和四个根,而且也可通过根式来表示,只是形式复杂,很少有实用意义,可参见[A5].至于五次和五次以上的代数方程,已经知道,其根不能用根式来表示,但是否也有与其次数一样多的根呢?下面我们就来讨论这个问题.

设  $n$  次多项式

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n, \quad a_0 \neq 0$$

则它在任一点  $x_0$  的泰勒展开式为

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

由于它是恒等式,所以可用它来研究函数  $f(x)$  在  $x_0$  附近的变化情况.以  $n=3$  为例,我们有

$$\begin{aligned} f(x) &= a_0x^3 + a_1x^2 + a_2x + a_3 \\ &= \frac{f'''(x_0)}{3!}(x - x_0)^3 + \frac{f''(x_0)}{2!}(x - x_0)^2 + f'(x_0)(x - x_0) + f(x_0) \end{aligned}$$

或写成便于计算的链式形式

$$f(x) = \left\{ \left[ \frac{f'''(x_0)}{3!}(x - x_0) + \frac{f''(x_0)}{2!} \right] (x - x_0) + f'(x_0) \right\} (x - x_0) + f(x_0)$$

由此可得计算泰勒展开式系数的方法如下:

把  $f(x)$  除以  $(x - x_0)$ , 可得

$$\begin{cases} \text{商} & Q_1(x) = \left[ \frac{f'''(x_0)}{3!}(x - x_0) + \frac{f''(x_0)}{2!} \right] (x - x_0) + f'(x_0) \\ \text{余数} & f(x_0) \end{cases}$$

再把  $Q_1(x)$  除以  $(x - x_0)$ , 可得

$$\begin{cases} \text{商} & Q_2(x) = \left[ \frac{f'''(x_0)}{3!}(x - x_0) + \frac{f''(x_0)}{2!} \right] \\ \text{余数} & f'(x_0) \end{cases}$$

再把  $Q_2(x)$  除以  $(x - x_0)$ , 可得

$$\begin{cases} \text{商} & Q_1 = \frac{f''(x_0)}{3!} \\ \text{余数} & \frac{f'(x_0)}{2!} \end{cases}$$

可见,用除法就可以求出函数  $f(x)$  在  $x_0$  的泰勒展开式的各系数  $f(x_0)$ ,  $f'(x_0)$ ,  $\frac{f'(x_0)}{2!}$ ,  $\frac{f''(x_0)}{3!}$ . 这种方法称为秦九韶法.

现在我们回到关于代数方程根的存在性问题在讨论上来.

**引理**  $x_0$  是代数方程  $f(x) = 0$  的根的充分必要条件是  $f(x)$  可以被  $(x - x_0)$  整除.

显然,这由上述链式形式可以看出.

根据以上基本定理和引理可知,  $n$  次代数方程  $f(x) = 0$  至少有一个根,比如设为  $x_1$ , 则

$$f(x) = (x - x_1)Q_1(x)$$

其中  $Q_1(x)$  是  $n - 1$  次多项式;若  $n - 1 > 0$ , 方程  $Q_1(x) = 0$  又至少有一根  $x_2$ , 则

$$Q_1(x) = (x - x_2)Q_2(x)$$

其中  $Q_2(x)$  是  $n - 2$  次多项式. 依此类推,可得

$$f(x) = a_0(x - x_1)(x - x_2) \cdots (x - x_n)$$

其中  $x_1, x_2, \dots, x_n$  是  $f(x) = 0$  的  $n$  个根. 由此,我们可得

**定理 2**  $n$  次代数方程有  $n$  个根.

至此,我们便回答了  $n$  次代数方程根的存在性问题.

## 2. 关于根的隔离问题

根的隔离问题是比较复杂和麻烦的问题. 这一小节讲一些比较深刻的理论问题,篇幅也比较多,没有专门需要的读者可以跳过这一小节.

一种简单的根的隔离方法是图解法. 先画出函数  $y = f(x)$  的草图,然后观察曲线与  $x$  轴的交点,从而可以隔离出在哪些区间上存在方程  $f(x) = 0$  的实根.

另外,有两种关于确定代数方程正根上界的方法:

(1) 设实系数代数方程

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0$$

的最高次项的系数  $a_0 > 0$  (否则以  $-1$  相乘两端),  $k \geq 1$  是它的第一个负系数的下标,  $B$  为所有负系数的绝对值的最大值,则

$$R = 1 + \sqrt[k]{\frac{B}{a_0}}$$

是上述方程的正根上界.

(2) 设上述方程最高次项的系数  $a_0 > 0$ , 如果  $x = \alpha$  时方程本身及其各次导数  $f'(x)$ ,  $f''(x)$ ,  $\dots$ ,  $f^{(n)}(x)$  都取正值,则数  $\alpha$  为上述方程的正根上界.

确定正根上界的这两种方法的证明可参见比较详细的高等代数教程.

**例 1** 试确定下列方程的正根上界

$$f(x) = x^4 - 5x^2 + 6x - 8 = 0$$

这里  $k = 2$ ,  $B = 8$ . 由方法 (1) 可得方程正根上界大致为

$$R = 1 + \sqrt[3]{\frac{8}{1}} \approx 4$$

若按方法(2),由各次导数

$$f'(x) = 4x^3 - 10x + 6$$

$$f''(x) = 12x^2 - 10$$

$$f'''(x) = 24x$$

$$f^{(4)}(x) = 24$$

当  $x = 2$  时,它们均为正,可知方程正根的上界为 2.

由上例可见,方法(2)比方法(1)精确一些.

下面讨论方程实根(包括重根)隔离的另一些较复杂的方法.先引进关于一个实数序列变号次数的概念.

**定义 1** 设有限个实变函数序列

$$\{f_0(x), f_1(x), f_2(x), \dots, f_n(x)\}$$

其中  $x$  为实变数,当  $x = a$  时,它对应于数列

$$\{f_0(a), f_1(a), f_2(a), \dots, f_n(a)\}$$

若数列中有两相邻数  $f_k(a)$  与  $f_{k+1}(a)$  的符号相反,我们就说这两个数之间有一次变号,否则没有变号,或说变号次数等于零.整个数列相邻数变号次数之和称为这个数列的变号次数,记为符号

$$V_a = V\{f_0(a), f_1(a), f_2(a), \dots, f_n(a)\}$$

其中,若数列中出现零时则把零除去以后再计算变号的次数.

此外,我们还用符号

$$V_a^{p,q} = V\{f_p(a), f_{p+1}(a), \dots, f_q(a)\}$$

表示从  $f_p(a)$  到  $f_q(a)$  这一段的变号次数.显然有

$$V_a^{m,q} = V_a^{m,p} + V_a^{p,q} \quad (m < p < q)$$

**例 2** 设  $f(x) = x^3 - 2x - 5$ , 作函数序列  $\{f(x), f'(x), f''(x), f'''(x)\}$ , 其中  $f'(x) = 3x^2 - 2$ ,  $f''(x) = 6x$ ,  $f'''(x) = 6$ , 则此函数序列在  $x = 1, x = 2, x = 3$  的变号次数依次为

$$V_1 = V\{f(1), f'(1), f''(1), f'''(1)\} = V\{-6, 1, 6, 6\} = 1,$$

$$V_2 = V\{f(2), f'(2), f''(2), f'''(2)\} = V\{-1, 10, 12, 6\} = 1,$$

$$V_3 = V\{f(3), f'(3), f''(3), f'''(3)\} = V\{16, 25, 18, 6\} = 0.$$

若  $V_a - V_b = m > 0$ ,  $m$  是正整数,我们就说这个函数序列当  $x$  从  $a$  变到  $b$  时损失了  $m$  次变号.如例 1 的序列,当  $x$  从 2 变到 3 时损失了 1 次变号.下面定理就是根据例 1 中这种函数序列从  $a$  变到  $b$  损失的变号次数来判断  $(a, b)$  内存在方程  $f(x) = 0$  的几个实根的定理.

由  $n$  次代数方程  $f(x) = 0$  作函数序列

$$\{f(x), f'(x), f''(x), \dots, f^{(n)}(x)\} \quad (7.3)$$

再经过较长的推证,我们可得

**定理 3** (Budan-Fourier 定理) 设

1)  $f(a) \neq 0, f(b) \neq 0$ ;

2)  $f(x) = 0$  在  $(a, b)$  内的实根个数为  $N(a, b)$ ;

3)  $V_x$  为函数序列 (7.3) 的变号次数, 则有

$$0 \leq N(a, b) = (V_a - V_b) - \langle \text{正偶数或零} \rangle$$

其中  $V_a - V_b$  是函数序列 (7.3) 当  $x$  从  $a$  增加到  $b$  时损失的变号次数, 它是正整数或零.

这个定理实际上能提供的信息并不是很多. 因为定理中所说的实根个数是指  $(a, b)$  内各根的重数之和, 即一个  $m$  重根算作  $m$  个根. 另外, 定理中所说的正偶数也不明确, 它可能是任何小于或等于  $V_a - V_b$  的正偶数. 例如设  $V_a - V_b = 3$ , 则  $f(x) = 0$  在  $(a, b)$  可能有 3 个根也可能只有一个根. 只有当  $V_a - V_b = 0$  时,  $f(x) = 0$  在  $(a, b)$  没有实根; 当  $V_a - V_b = 1$  时,  $f(x) = 0$  在  $(a, b)$  有一个实根, 这两种情况定理给出了完全肯定的答案.

我们注意到方程  $f(x) = 0$  的正根个数就是方程在区间  $(0, +\infty)$  的根的个数. 现设方程的形式为

$$f(x) = x^n + a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1} x + a_n = 0 \quad (7.4)$$

其中  $a_n = f(0) \neq 0$ , 则

$$\begin{aligned} V_0 &= V\{f(0), f'(0), f''(0), \dots, f^{(n)}(0)\} \\ &= V\left\{f(0), f'(0), \frac{f''(0)}{2!}, \dots, \frac{f^{(n)}(0)}{n!}\right\} \\ &= V\{a_n, a_{n-1}, a_{n-2}, \dots, 1\} \\ &= V\{1, a_1, a_2, \dots, a_n\} \end{aligned}$$

即它是方程 (7.4) 系数序列的变号次数. 又显然有  $V_{+\infty} = 0$ , 因此有

$$V_0 - V_{+\infty} = V\{1, a_1, a_2, \dots, a_n\}$$

应用 Budan-Fourier 定理我们可得推算正根个数的法则

**定理 4** (笛卡尔 (Descartes) 法则) 方程 (7.4) 的正根个数等于方程系数序列的变号次数或比它少某个正偶数.

这个法则仍然没有超出 Budan-Fourier 定理的局限性. 它只有在  $V_0 = 0$  和  $V_0 = 1$  两种情况下有肯定的答案: 前者无正根, 后者有一个正根. 当  $V_0 \geq 2$  时仍无关于正根的进一步信息.

**例 3** 考察方程  $x^3 - 2x - 5 = 0$ .

因  $V_0 = V\{1, 0, -2, -5\} = 1$ , 所以方程有一个正根. 还有另两个根是什么呢? 令  $x = -y$ , 则  $y$  满足方程

$$y^3 - 2y + 5 = 0$$

这个方程的  $V_0 = 2$ , 故可能有两个正实根或无正实根, 所以原方程可能有两个负实根或无负实根. 事实上, 此方程仅有一个正实根和另外一对共轭复根.

**例 4** 考察方程  $x^{41} + x^3 + 1 = 0$ .

因  $V_0 = 0$ , 所以这个方程无正根. 令  $x = -y$ , 则有

$$y^{41} + y^3 - 1 = 0$$

现在  $V_0 = 1$ , 所以第二个方程有一个正根, 即原来方程有一个负根, 其余 40 个根就全是共轭复根了.

**例 5**  $U(\rho) = \rho^n - |a_1|\rho^{n-1} - |a_2|\rho^{n-2} - \cdots - |a_{n-1}|\rho - |a_n| = 0$ .

其中  $|a_n| \neq 0$ . 因为

$$V_0 = V\{1, -|a_1|, -|a_2|, \dots, -|a_{n-1}|, -|a_n|\} = 1$$

所以方程有一正根.

应用 Budan-Fourier 定理和秦九韶法还可以做根的进一步隔离工作. 如我们上节已经用过的, 由于计算数列的变号次数  $V_a$  只与数字的符号有关, 而与数字的数值大小无关, 因此有

$$\begin{aligned} V_a &= V\{f(a), f'(a), \dots, f^{(n-1)}(a), f^{(n)}(a)\} \\ &= V\left\{\frac{f^{(n)}(a)}{n!}, \frac{f^{(n-1)}(a)}{(n-1)!}, \dots, f'(a), f(a)\right\} \end{aligned}$$

后面这个数列就是  $f(x)$  在  $a$  点的泰勒展开式的系数, 可用秦九韶法求出.

**例 6** 考虑方程  $f(x) = x^3 - 6x^2 + 11x - 6 = 0$ .

应用秦九韶法算出在点  $x = 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5$  处的泰勒展开式系数, 然后将其符号列如下表:

| $x$ | $f(x)$ | $f'(x)$ | $\frac{f''(x)}{2!}$ | $\frac{f'''(x)}{3!}$ | $V_x$ |
|-----|--------|---------|---------------------|----------------------|-------|
| 0   | -      | +       | -                   | +                    | 3     |
| 0.5 | -      | +       | -                   | +                    | 3     |
| 1   | 0      | +       | -                   | +                    | (2)   |
| 1.5 | +      | -       | -                   | +                    | 2     |
| 2   | 0      | -       | 0                   | +                    | (1)   |
| 2.5 | -      | -       | +                   | +                    | 1     |
| 3   | 0      | +       | +                   | +                    | (0)   |
| 3.5 | +      | +       | +                   | +                    | 0     |

注意 Budan-Fourier 定理中要求  $f(a) \neq 0, f(b) \neq 0$ , 所以当  $x$  是根时, 算出的  $V_x$  不能用. 表中加括号算出. 从表我们便可得:

$V_{0.5} - V_{1.5} = 3 - 2 = 1$ , 故区间  $(0.5, 1.5)$  内有一根;

$V_{1.5} - V_{2.5} = 2 - 1 = 1$ , 故区间  $(1.5, 2.5)$  内有一根;

$V_{2.5} - V_{3.5} = 1 - 0 = 1$ , 故区间  $(2.5, 3.5)$  内有一根.

Budan-Fourier 定理和 Descartes 法则简单易用, 缺点是答案往往不明确. 下面进一步引入施多姆序列和施多姆定理.

设  $f_0(x)$  和  $f_1(x)$  是两个  $x$  的多项式, 其中  $f_1(x)$  的次数低于  $f_0(x)$ . 用  $f_1(x)$  除  $f_0(x)$  得商式  $q_0(x)$  和余式  $R_2(x)$ . 显然  $R_2(x)$  的次数低于  $f_1(x)$ . 若  $R_2(x) \neq 0$ , 令  $f_2(x) = -R_2(x)$  作为序列的下一个函数. 用  $f_2(x)$  除  $f_1(x)$  得商式  $q_1(x)$  和余式  $R_3(x)$ . 同样  $R_3(x)$  的次数低于  $f_2(x)$ . 若  $R_3(x) \neq 0$ , 又令  $f_3(x) = -R_3(x)$  作为序列的下一个函数. 如此辗转相除一直到除尽为止.

注意到  $f_0(x), f_1(x), f_2(x), \dots$  的次数逐渐降低, 至多降到零次多项式比如  $f_m(x)$ , 即非零常数, 它自然可以把任何多项式除尽.

**定义 2** 由多项式  $f_0(x), f_1(x)$  经过上述辗转相除的方法得出的  $m+1$  个非零多项式构成的序列

$$\{f_0, f_1, f_2, \dots, f_m\}, \quad f_{m+1} \equiv 0$$

称为以  $f_0, f_1$  为基的施多姆 (Sturm) 序列.

施多姆序列中三个相邻函数间的关系是

$$f_k(x) = q_k(x)f_{k+1}(x) - f_{k+2}(x), \quad k = 0, 1, \dots, m-1, f_{m+1}(x) \equiv 0$$

建筑在这个序列基础上的有

**定理 5** (施多姆定理) 设

1)  $f(a) \neq 0, f(b) \neq 0$ ;

2) 以  $f_0(x) = f(x), f_1(x) = f'(x)$  为基的施多姆序列为

$$\{f(x), f'(x), f_2(x), \dots, f_m(x)\};$$

3) 上述序列在  $x$  点的变号次数为  $V_x$ , 则方程  $f(x) = 0$  在  $(a, b)$  区间内共有  $V_a - V_b$  个不相同的实根. 若最后非零函数没有实零点, 则  $f(x) = 0$  的实根都是单根; 若  $f_m(x) = 0$  有实根, 则这些根都是  $f(x) = 0$  的重根, 其重数为  $f_m(x) = 0$  内的重数加 1.

**例 7** 仍考虑方程  $f(x) = x^3 - 6x^2 + 11x - 6 = 0$

其施多姆序列是

$$\begin{cases} f_0(x) = f(x) = x^3 - 6x^2 + 11x - 6, \\ f_1(x) = f'(x) = 3x^2 - 12x + 11, \\ f_2(x) = \frac{2}{3}(x-2), \\ f_3(x) = 1. \end{cases}$$

因为最后函数是一个常数, 所以可知这个方程没有重根. 再计算如下结果:

| $x$       | $f_0(x)$ | $f_1(x)$ | $f_2(x)$ | $f_3(x)$ | $V_x$ |
|-----------|----------|----------|----------|----------|-------|
| $-\infty$ | -        | +        | -        | +        | 3     |
| 0         | -        | +        | -        | +        | 3     |
| $+\infty$ | +        | +        | +        | +        | 0     |

由表可知, 方程有三个正实根, 没有负实根.

至此, 我们已经研究了实根分离的问题. 关于复根的情形, 限于本书的篇幅, 这里就不再讲了. 需要时可见参考书 [A5], 在那里还讲到关于根的最大最小模的界的问题.

### 3. 迭代法的一般理论

迭代法是逐次逼近的方法. 解方程 (7.1) 的迭代法的基本思想是, 从隔根区间  $(a, b)$  中的任一个初始近似值  $x_0$  出发, 按照某种格式构造一个序列

$$x_0, x_1, x_2, \dots \quad (7.5)$$

使得这个序列的极限就是方程 (7.1) 的根  $x^*$ , 即

$$\lim_{k \rightarrow \infty} x_k = x^*, \quad f(x^*) = 0$$

产生序列 (7.5) 的一种方法是把方程 (7.1) 写成等价形式

$$x = \varphi(x) \quad (7.6)$$

然后令

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \dots \quad (7.7)$$

这种方法称为比卡 (Picard) 迭代, (7.7) 称为迭代公式,  $\varphi(x)$  称为迭代函数,  $x_0$  称为初始近似值,  $x_k$  称为第  $k$  次近似值.

如果从初始近似值  $x_0$  出发, 按照迭代公式 (7.7) 依次计算出的序列 (7.5), 存在极限  $\lim_{k \rightarrow \infty} x_k = x^*$ . 则当  $\varphi(x)$  连续时, 由 (7.7) 式两边取极限得

$$x^* = \varphi(x^*) \quad \text{即} \quad f(x^*) = 0$$

这时便称迭代公式 (7.7) 是收敛的, 否则是发散的.

**例 1** 用迭代法求下列方程的根:

$$10^x - x - 2 = 0$$

**解** 若将方程改写为

$$x = 10^x - 2$$

即作迭代公式

$$x_{k+1} = 10^{x_k} - 2, \quad k = 0, 1, 2, \dots$$

取初值  $x_0 = 1$ , 迭代函数  $\varphi(x) = 10^x - 2$ , 则依次迭代可得  $x_1 = 8$ ,  $x_2 = 10^8 - 2, \dots$ , 当  $k$  增大时,  $x_k$  随之增大而不趋于任何极限值.

若将方程改写为

$$x = \lg(x + 2)$$

即作迭代公式

$$x_{k+1} = \lg(x_k + 2), \quad k = 0, 1, 2, \dots$$

取初值  $x_0 = 1$ , 迭代函数  $\varphi(x) = \lg(x + 2)$ , 则迭代三次可得  $x_3 = 0.38$ , 这已经准确到二位小数.

一般来说, 由迭代公式 (7.7) 产生的序列不一定收敛, 也就是说, 迭代函数  $\varphi(x)$  要具备一定的条件, 迭代法才能收敛.

**定理 1** 设  $\varphi(x)$  为定义在区间  $I = [a, b]$  上的连续函数, 对任何  $x \in I$ , 都有  $\varphi(x) \in I$ , 且  $\varphi(x)$  满足所谓李普希兹 (Lipschitz) 条件, 即存在常数  $0 < L < 1$ , 对区间  $I$  中任意两个值  $x_1$  和  $x_2$ , 恒有

$$|\varphi(x_2) - \varphi(x_1)| \leq L|x_2 - x_1|$$

则对任意初始近似值  $x_0 \in I$ , 由迭代公式 (7.7) 确定的迭代序列  $\{x_k\}$  都收敛于方程 (7.6) 的唯一解  $x^*$ .

**证** 先证方程 (7.6) 有唯一解. 由于  $\varphi(x)$  在  $I$  上连续, 又对任何  $x \in I$  都有  $\varphi(x) \in I$ , 因此  $f(x) = x - \varphi(x)$  在  $I$  上连续, 且有

$$f(a) = a - \varphi(a) \leq 0, \quad f(b) = b - \varphi(b) \geq 0$$

根据连续函数性质, 可知方程  $f(x) = 0$  在  $I$  上至少有一个根  $x^*$ . 现在证明, 方程  $f(x) = 0$  在  $I$  上只能有一个根, 因为若有两个根  $x_1^* \neq x_2^*$ , 则因  $x_1^* = \varphi(x_1^*)$ ,  $x_2^* = \varphi(x_2^*)$  从而有

$$|\varphi(x_2^*) - \varphi(x_1^*)| = |x_2^* - x_1^*|$$

这与李普希兹条件矛盾 (有  $0 < L < 1$ ).

现证由 (7.7) 产生的序列  $\{x_k\}$  收敛于唯一解  $x^*$ . 由

$$x_{k+1} - x^* = \varphi(x_k) - \varphi(x^*), \quad k = 0, 1, 2, \dots$$

有

$$|x_{k+1} - x^*| = |\varphi(x_k) - \varphi(x^*)| \leq L|x_k - x^*|, \quad k = 0, 1, 2, \dots$$

重复使用这个不等式, 最后可得

$$|x_{k+1} - x^*| \leq L^{k+1}|x_0 - x^*|, \quad k = 0, 1, 2, \dots$$



因为  $0 < L < 1$ , 所以

$$\lim_{k \rightarrow \infty} |x_{k+1} - x^*| = 0$$

即

$$\lim_{k \rightarrow \infty} x_{k+1} = x^* \quad (\text{证毕})$$

通常, 定理中的迭代函数  $\varphi(x)$  称为从  $I \rightarrow I$  的压缩映射. 常数  $L$  称为李普希兹常数.

利用微分中值定理可得, 若在  $I$  上  $\varphi'(x)$  存在且满足不等式

$$|\varphi'(x)| \leq L < 1 \quad (7.8)$$

则  $\varphi(x)$  在  $I$  上满足李普希兹条件.

下面, 我们给出迭代收敛阶数的概念. 假设上述定理中的条件以及 (7.8) 均成立, 又  $\varphi(x)$  在  $I$  上  $m+1$  次连续可微,  $x = \varphi(x)$  在  $I$  上的根为  $x^*$ , 记  $e_k = x_k - x^*$ . 根据泰勒公式

$$\begin{aligned} e_{k+1} &= x_{k+1} - x^* = \varphi(x_k) - \varphi(x^*) = \varphi(x^* + e_k) - \varphi(x^*) \\ &= \varphi'(x^*)e_k + \frac{1}{2!}\varphi''(x^*)e_k^2 + \cdots + \frac{1}{m!}\varphi^{(m)}(x^*)e_k^m \\ &\quad + \frac{1}{(m+1)!}\varphi^{(m+1)}(x^* + \theta e_k)e_k^{m+1} \quad (0 \leq \theta \leq 1) \end{aligned}$$

若再假定

$$\varphi'(x^*) = \varphi''(x^*) = \cdots = \varphi^{(m-1)}(x^*) = 0; \quad \varphi^{(m)}(x^*) \neq 0 \quad (7.9)$$

则得

$$e_{k+1} = x_{k+1} - x^* = \frac{1}{m!}\varphi^{(m)}(x^*)e_k^m + \frac{1}{(m+1)!}\varphi^{(m+1)}(x^* + \theta e_k)e_k^{m+1}$$

从而有

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^p} = \lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \begin{cases} 0, & \text{对 } p = 1, 2, \dots, m-1, \\ \frac{1}{m!}|\varphi^{(m)}(x^*)|, & \text{对 } p = m. \end{cases}$$

定义 如果

$$1) \quad e_k = x_k - x^* \neq 0, \quad \text{对 } k = 0, 1, 2, \dots;$$

$$2) \quad \lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^p} = 0, \quad \text{对 } p = 0, 1, \dots, m-1;$$

$$3) \quad \lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^m} = K > 0,$$

则称序列  $\{x_k\}$  是  $m$  阶收敛的, 或称序列  $\{x_k\}$  的收敛阶数为  $m$ . 常数  $K$  称为渐近误差常数.

收敛阶数反映了迭代序列的收敛速度,  $m$  越大收敛越快.

特别地, 当  $m = 1$  时, 称为线性收敛; 当  $m = 2$  时, 称为二阶收敛. 若条件 (7.9) 成立, 则迭代为  $m$  阶收敛.

线性收敛的误差估计如下: 设对一切  $x \in I$ ,  $|\varphi'(x)| \leq L < 1$ , 则由  $|e_n| = |x_n - x^*| = |\varphi'(x^* + \theta e_{n-1})e_{n-1}|$ , 可得

$$|e_n| \leq L|e_{n-1}| \leq \cdots \leq L^n|e_0|$$

二阶收敛的误差估计如下: 设对一切  $x \in I$ ,  $|\varphi''(x)| \leq 2M$ . 因为  $\varphi'(x^*) = 0$ , 所以

$$e_n = x_n - x^* = \frac{1}{2!} \varphi''(x^* + \theta e_{n-1}) e_{n-1}^2,$$

从而可得

$$|e_n| \leq M |e_{n-1}|^2 \leq \dots \leq (M |e_0|)^{2^{n-1}} |e_0|$$

当  $M |e_0| < 1$  时, 上式便可用来估计二阶收敛迭代法的  $n$  次迭代误差.

若要求计算结果达到给定的精度

$$|e_n| \leq 10^{-m}$$

则线性收敛方法所需迭代次数为

$$n \geq \frac{m + \lg |e_0|}{|\lg L|}$$

而二阶收敛方法所需迭代次数为

$$n \geq \log_2 \left( \frac{m + \lg |e_0|}{|\lg M + \lg |e_0||} + 1 \right)$$

## 7-2 方程求根的几个常用方法

### 1. 对分法及其 FORTRAN 程序

设实函数方程为

$$f(x) = 0 \quad x \in [a, b]$$

由数学分析知道, 若  $f(x)$  在某区间  $[a_0, b_0] \subset [a, b]$  上单调连续, 又  $f(x)$  在  $[a_0, b_0]$  两端点处的值异号, 即  $f(a_0) \cdot f(b_0) < 0$ , 则方程在  $[a_0, b_0]$  内有且只有一根.

根据这个原理的对分法属于区间套方法. 其基本思想是: 将方程的有根区间  $[a_0, b_0]$  对分为两个小区间, 并计算  $f\left(\frac{a_0 + b_0}{2}\right)$ , 然后通过判断小区间端点的函数值是否异号, 可以找出新的有根区间  $[a_1, b_1]$ , 如此继续, 就可得到一个有根区间套

$$[a_0, b_0] \supset [a_1, b_1] \supset [a_2, b_2] \supset \dots$$

直至小区间端点处函数  $f(x)$  的绝对值小于给定的精度  $\varepsilon_1$ , 或区间的长度小于给定的精度  $\varepsilon_2$ , 便可得一满足精度的近似根.

利用这种对分法的一种计算机算法及其 FORTRAN 程序可以用来求方程在定义区间  $[a, b]$  上的所有单重实根. 其具体做法是: 自  $a$  开始, 以一个基本步长  $H$ , 向右分割出宽度为  $H$  的小区间  $[a_0, b_0]$ , 若某一步前后两个函数值  $f(a_0), f(b_0)$  异号, 则用上述对分法找出其中包含的一个实根; 若同号, 则继续向右分割, 直至分割的小区间已超过  $[a, b]$ .

显然,  $H$  应尽可能选得使每个小区间至多不超过一个根.  $H$  太大容易失根;  $H$  太小则浪费机时.

下面我们给出一个对分法子程序 ROOT.

程序中哑元设置如下:

A, B 分别为根的下界和上界; 实型变量, 输入参数.

H 步长; 实型变量, 输入参数.

- N 欲求的根的个数;整型变量,输入参数.
- K 实际求出的根的个数;整型变量,输出参数.
- X 存放所求出的根的N个元素的一维实型数组;输出参数.
- EPS1 实型变量,输入参数;函数值精度控制量,当  $|f(x)| < \text{EPS1}$  时, $x$  即为  $f(x) = 0$  的根.
- EPS2 实型变量,输入参数;根的精度控制量,当对分过程中子区间的长度小于 EPS2 时,认为子区间的一端点便是方程  $f(x) = 0$  的根.
- 程序还需调用一个计算方程左端函数  $f(x)$  的函数子程序

FUNCTION F(T)

其中哑元取 T 是为了避免与上述哑元 X 相同. 本函数子程序由使用者按方程形状自行编写.

子程序:

```

SUBROUTINE ROOT (A, B, H, N, K, X, EPS1, EPS2)
  DIMENSION X(N)
  K = 0
  X1 = A
  F1 = F(X1)
  IF (F1) 10, 35, 10
10  F1 = F(X1)
  X2 = X1 + H
  IF (X2 .GT. B) GO TO 100
  F2 = F(X2)
  IF (F1 * F2) 11, 45, 25
11  S2 = X2
22  X3 = 0.5 * (X1 + S2)
  F3 = F(X3)
  IF (ABS(F3) .LT. EPS1) GO TO 50
  D = ABS (S2 - X3)
  IF (D .LT. EPS2) GO TO 50
  FS = F(S2)
  IF (F3 * FS .GT. 0.0) GO TO 23
  X1 = X3
  GO TO 22
23  S2 = X3
  GO TO 22
25  X1 = X2
  GO TO 10
35  X0 = X1

```

```

        X1 = X2
        GO TO 60
45  X0 = X2
        X1 = X2 + H
        GO TO 60
50  X0 = X3
        X1 = X2
60  K = K + 1
        X(K) = X0
        GO TO 10
100 RETURN
        END

```

**算例** 求方程  $f(x) = 0$  在  $[-1, 2]$  上的根, 其中

$$f(x) = \begin{cases} 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 & x \leq 0 \\ e^x + 2^{-x} + 2\cos x - 5 & x > 0 \end{cases}$$

由题知  $A = -1$ ,  $B = 2$ . 今取  $H = 0.1$ ,  $N = 8$ ,  $EPS1 = EPS2 = 10^{-6}$ ; 解题程序(这里使用 FORTRAN 77 语句)及计算结果如下:

```

C      MAIN PROGRAM
        DIMENSION X(8)
        CALL ROOT (-1.0, 2.0, 0.1, 8, K, X, 1E-06, 1E-06)
        IF (K .NE. 0) THEN
            DO 200 I = 1, K
                WRITE (6, 210) I, X(I)
210      FORMAT (1X, 'X(', I2, ') = ', F12.8)
                WRITE (6, 220) F(X(I))
220      FORMAT (1X, 'F(X) = ', F13.8/)
200      CONTINUE
            ELSE
                PRINT *, 'NO ROOT'
            ENDIF
        END

```

```

C      SUBROUTINE ROOT (A, B, H, N, K, X, EPS1, EPS2)
        :
        :      (本子程序段体)
        :
        END

```

```

C      FUNCTION F(T)
      IF(T) 33, 33, 44
33     T2 = T*T
      F = (((512*T2 - 1280)*T2 + 1120)*T2 - 400)*T2 + 50)*
1      T2 - 1
      GO TO 55
44     F = EXP(T) + 2*(-T) + 2*cos(T) - 5
55     RETURN
      END

```

计算结果:

```

X(1) = -0.98768837
F(X) = 0.00000188
X(2) = -0.89100648
F(X) = 0.00000087
X(3) = -0.70710679
F(X) = 0.00000014
X(4) = -0.45399057
F(X) = -0.00000076
X(5) = -0.15643445
F(X) = -0.00000017
X(6) = 1.51027985
F(X) = -0.00000002

```

计算结果除输出求得的根外, 这里还输出该根对应的函数值  $f(x)$ , 从而可比较出该根满足精度控制参数的情况, 还可比较出所求出的根是由哪一个参数控制的。可以看出,  $X(1)$  满足的应该是控制参数 EPS2。事实上,  $f(x)$  在  $x = -1$  附近是一个 10 次多项式, 且  $f(-1) = 1$ , 故从计算结果可看出  $f(x)$  在  $x = -1$  附近变化较大, 因此若要求它满足控制参数 EPS1, 则需要较多的计算量。

## 2. 牛顿迭代法及其 FORTRAN 程序

非线性方程  $f(x) = 0$  求根的牛顿 (Newton) 迭代法是解非线性方程的线性化方法。设  $f(x)$  在点  $x_0$  附近有直至二阶连续导数, 按照泰勒公式

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(\xi)}{2!} (x - x_0)^2$$

其中  $\xi$  在  $x$  与  $x_0$  之间。上式线性化可得

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

因此, 非线性方程  $f(x) = 0$  在  $x_0$  附近的近似方程是线性方程

$$f(x_0) + f'(x_0)(x - x_0) = 0$$

现设  $f'(x_0) \neq 0$ , 则可得解

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

如果  $x_0$  是方程  $f(x)=0$  的某个解  $x^*$  的比较好的近似值, 则我们可得求这个解的迭代公式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad n = 0, 1, 2, \dots \quad (7.10)$$

这就是著名的牛顿迭代公式, 或称牛顿迭代程序. 对应的迭代函数为

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

注意到

$$\varphi'(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}$$

当  $f'(x^*) \neq 0, f(x^*) = 0$  时, 有  $\varphi'(x^*) = 0$ . 因此当  $x$  充分靠近  $x^*$  时可使

$$|\varphi'(x)| \leq L < 1$$

这就保证了由 (7.10) 产生的序列  $\{x_n\}$  的收敛性.

可以证明, 迭代公式 (7.10) 在一定条件下具有二阶收敛速度. 事实上, 由

$$\varphi'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}, \quad \varphi''(x^*) = \frac{f''(x^*)}{f'(x^*)}$$

故只要  $f'(x^*) \neq 0, f''(x^*) \neq 0$ , 便可得  $\varphi'(x^*) = 0, \varphi''(x^*) \neq 0$ , 根据 (7.9) 式这就说明牛顿迭代公式是二阶的.

注意, 以上要求  $f'(x^*) \neq 0$ , 因此是指  $x^*$  为单根的情况. 若  $f'(x^*) = 0$ , 则  $x^*$  为重根. 设为  $q$  重根, 此时  $f(x)$  可以写成

$$f(x) = (x - x^*)^q g(x) \quad (g(x^*) \neq 0)$$

代入已经算出的  $\varphi'(x)$  的表达式可得

$$\varphi'(x) = \frac{\left(1 - \frac{1}{q}\right) + (x - x^*) \frac{2g'(x)}{q \cdot g(x)} + (x - x^*)^2 \frac{q''(x)}{q^2 g(x)}}{\left[1 + (x - x^*) \frac{g'(x)}{q \cdot g(x)}\right]^2}$$

从而得

$$\lim_{x \rightarrow x^*} \varphi'(x) = \varphi'(x^*) = 1 - \frac{1}{q}$$

所以若  $x^*$  为  $f(x) = 0$  的多重根, 则牛顿迭代公式仅为线性收敛.

实际应用中还要考虑初始值  $x_0$  的选取问题. 由上述可知, 若  $f'(x_0) = 0, x_0$  就不能作初始值; 若  $f'(x_0)$  很小,  $x_0$  也不宜作初始值. 从迭代公式可得

$$x_1 - x^* = (x_0 - x^*) - \frac{f(x_0)}{f'(x_0)}$$

记  $e_1 = x_1 - x^*, e_0 = x_0 - x^*$ , 则有

$$\frac{e_1}{e_0} = 1 - \frac{f(x_0)}{f'(x_0)(x_0 - x^*)} = \frac{f(x_0) + f'(x_0)(x^* - x_0)}{f'(x_0)(x^* - x_0)}$$

利用零阶和一阶泰勒公式

$$0 = f(x^*) = f(x_0) + f'(\xi)(x^* - x_0)$$

$$0 = f(x^*) = f(x_0) + f'(x_0)(x^* - x_0) + \frac{f''(\eta)}{2}(x^* - x_0)^2$$

其中  $\xi, \eta$  在  $x^*$  与  $x_0$  之间, 我们有

$$\frac{e_1}{e_0} = \frac{-\frac{f''(\eta)}{2}(x^* - x_0)^2}{f'(x_0)(x^* - x_0)} = -\frac{f''(\eta)(x^* - x_0)}{2f'(x_0)} = \frac{f''(\eta)f(x_0)}{2f'(x_0)f'(\xi)}$$

由于无法计算  $f'(\xi)$  和  $f''(\eta)$ , 我们假定  $f'(x), f''(x)$  在  $x_0$  附近相对变化不大, 即必须设  $f''(x_0) \neq 0$ , 则有近似公式

$$\frac{e_1}{e_0} \approx \frac{f''(x_0)f(x_0)}{2[f'(x_0)]^2}$$

因此, 要求  $|e_1| < |e_0|$ , 这就要求

$$\left| \frac{f''(x_0)}{2} \right| |f(x_0)| < |f'(x_0)|^2 \quad (7.11)$$

如果在  $x_0, f(x)$  满足上述条件 (7.11) 且  $f'(x_0) \neq 0$ , 则可用  $x_0$  作为牛顿迭代公式的初始值。这种作法在大多数情况下还是可以保证收敛性的。

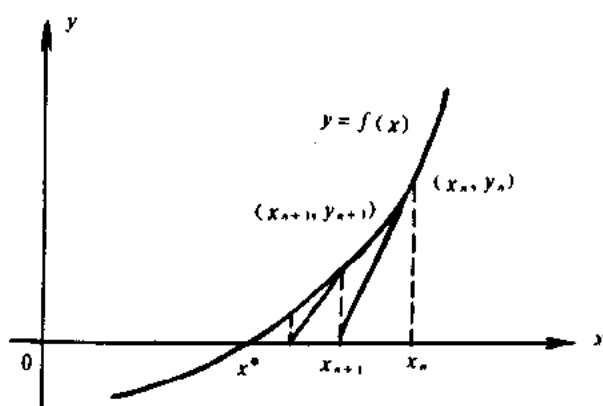


图 7-1

牛顿迭代法有明显的几何意义。当  $f(x)$  为实函数,  $x$  为实变量时, 过曲线  $y=f(x)$  上以  $(x_n, y_n)$  ( $y_n=f(x_n)$ ) 为坐标的点  $P_n$  作切线, 它和  $x$  轴的交点就是  $x_{n+1}$ 。如此继续,  $x_{n+1}, x_{n+2}, \dots$  将趋于解  $x^*$ , 如图 7-1 所示。因此, 牛顿迭代法也称切线法。

下面我们设计牛顿迭代法的计算机算法。给定初始值  $x_0$ ; 给定最大迭

代次数  $N_0$ ; 单元  $X_0$  开始存放初始值  $x_0$ , 后存放近似解  $x_n$ ; 单元  $X$  存放近似解  $x_{n+1}$ ; 给定容许误差  $\varepsilon_1, \varepsilon_2$  分别控制  $|x - x_0| < \varepsilon_1, |f(x)| < \varepsilon_2$ ; 变量  $I_0$  用作标志,  $I_0 = 1$  表示求得满足精度的近似值,  $I_0 = 0$  表示  $f'(x_k) = 0$ , 计算中断;  $I_0 = -1$  表示迭代  $N_0$  次后精度仍不满足。除此, 本算法还需另编计算  $f(x_n)$  和  $f'(x_n)$  的函数子程序

FUNCTION F(X)  
FUNCTION F1(X)

算法如下:

- [步 1] 对于  $K = 1, 2, \dots, N_0$  做至 [步 6];
- [步 2]  $f'(x_0) \rightarrow D$ ;
- [步 3] 如果  $D = 0$ , 则  $0 \rightarrow I_0$ , 返回主调程序;
- [步 4] 计算  $x_{k+1}$ , 即  $x_0 - f(x_0)/f'(x_0) \rightarrow x$ ;
- [步 5] 如果  $|x - x_0| < \varepsilon_1$  或  $|f(x)| < \varepsilon_2$ , 则  $1 \rightarrow I_0$ , 返回主调程序;
- [步 6]  $x \rightarrow x_0$ ;
- [步 7]  $-1 \rightarrow I_0$ , 返回主调程序。

其中控制条件  $|x - x_0| < \varepsilon_1$  也可改为  $|\delta| < \varepsilon_1$ , 而

$$\delta = \begin{cases} |x - x_0|, & |x| < 1; \\ \frac{|x - x_0|}{|x|}, & |x| \geq 1. \end{cases}$$

不过本程序没有采用这种做法。

子程序:

```

SUBROUTINE NWTN (X0, X, N0, EP1, EP2, I0, F, F1)
  I0 = 1
  DO 10 K = 1, N0
    D = F1 (X0)
    IF (D .NE. 0.0) GO TO 50
    I0 = 0
    RETURN
50  X = X0 - F(X0)/D
    IF((ABS(X - X0) .LT. EP1) .OR. (ABS(F(X)) .LT. EP2)) RETURN
10  X0 = X
    I0 = - 1
    RETURN
  END

```

**算例 1** 用牛顿迭代法求解下列方程

$$f(x) = e^x - x - 1 = 0$$

显然,  $x^* = 0$  为方程的根。作为示例, 试取  $x_0 = 1$ , 由  $f'(x) = e^x - 1$ ,  $f''(x) = e^x$ , 我们有

$$\left| \frac{f''(x_0)}{2} \right| |f(x_0)| = \frac{e}{2} (e - 2), \quad |f'(x_0)|^2 = (e - 1)^2$$

即条件 (7.11) 满足, 故  $x_0 = 1$  可作初始值。再取

$$N0 = 50, \quad \epsilon_1 = 10^{-6}, \quad \epsilon_2 = 10^{-6}$$

整个解题程序(包括主程序、NWTN 子程序和计算  $f(x)$  与  $f'(x)$  的子程序)如下:

```

EXTERNAL F, F1
CALL NWTN (1.0, X, 50, 1E - 06, 1E - 15, I0, F, F1)
WRITE (6, 100) I0, X
100 FORMAT (1X, 3H I0 = , I2, 10X, 2HX = , E15.8)
STOP
END

```

C

```

SUBROUTINE NWTN (X0, X, N0, EP1, EP2, I0, F, F1)
..... (本子程序段体)

```



RETURN

END

FUNCTION F(X)

F = EXP(X) - X - 1

RETURN

END

C

FUNCTION F1(X)

F1 = EXP(X) - 1

RETURN

END

计算结果:

10 = 1    X = 0.89837580E - 05

**算例 2** 用牛顿迭代法求方程

$$f(x) = x^{41} + x^3 + 1 = 0$$

在  $x_0 = -1$  附近的实根。

这里,  $f'(x) = 41x^{40} + 3x^2$ ,  $\frac{f''(x)}{2} = 82x^{39} + 6x$ , 有  $f(-1) = -1$ ,  $f'(-1) = 44$ ,  $\frac{f''(-1)}{2} = -823$ , 从而

$$823 \times 1 < 44^2 = 1936$$

可知  $x_0 = -1$  可选作初始值。如再取  $N0 = 10$ ,  $\varepsilon_1 = \varepsilon_2 = 10^{-9}$ , 并仿照算例 1 编写程序, 上机计算结果是

10 = 1    X = -0.95248388

牛顿迭代法也可用来计算方程的分离的复根, 这时, 我们把迭代公式表示为

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}, \quad n = 0, 1, 2, \dots$$

例如, 用牛顿迭代法求方程

$$f(z) = z^3 - z - 1 = 0$$

在  $z_0 = -0.5 + 0.5i$  附近的复根。

迭代计算的结果如下:

| $n$ | $z_n$               | $f(z_n)$           | $f'(z_n)$         |
|-----|---------------------|--------------------|-------------------|
| 0   | $-0.5 + 0.5i$       | $-0.25 - 0.25i$    | $-1.0 - 1.5i$     |
| 1   | $-0.6923 + 0.5385i$ | $-0.037 + 0.0795i$ | $-0.432 - 2.237i$ |
| 2   | $-0.6611 + 0.5611i$ | $-0.003 - 0.0022i$ | $-0.633 - 2.225i$ |
| 3   | $-0.6624 + 0.5621i$ |                    |                   |

这时,已有  $|f(z_1)| \approx 3.7 \times 10^{-3}$ , 如果已满足精度,便可取  $z^* = -0.6624 + 0.5621i$  为所求的根。

由例可见,不难为牛顿迭代法求方程复根编写 FORTRAN 程序。做法是,在程序中  $X0, X, F, F1$  均定义成复型变量,相应的算术运算均作复运算,绝对值则指复数的模(如复数  $z = a + bi$ , 则模  $|z| = \sqrt{a^2 + b^2}$ )。

**算例 3** 用牛顿迭代法求算例 2 中同一方程

$$f(z) = z^4 + z^3 + 1 = 0$$

在  $z_0 = e^{\frac{\pi}{4}i}$  附近的复根。程序设计由读者来做,并上机计算看能否得出如下结果:

$$z^* = 1.0143046 + 0.0809230i$$

最后我们指出,牛顿迭代公式中要用到导数;如果导数不好计算,可以用差商  $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$  代替导数  $f'(x_n)$ 。即得迭代公式

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \quad (n = 1, 2, \dots)$$

这称为**弦截法迭代公式**。弦截法也可从几

何直观导出,如图 7-2 弦的方程

$$\frac{f(x) - f(x_n)}{x - x_n} = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

弦与  $x$  轴的交点  $x = x_{n+1}$ ,  $f(x) = 0$ , 故可得

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

类似于牛顿迭代法,我们也不难为弦截法编写 FORTRAN 程序。

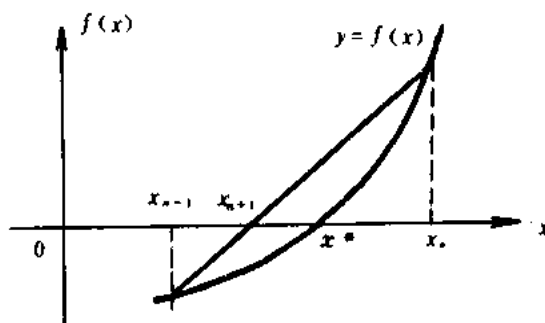


图 7-2

### 3. 劈因子法及其 FORTRAN 程序

劈因子法用于求如下形状的实系数高次代数方程

$$f(x) = x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n = 0 \quad (7.12)$$

的全部根。在计算机上常用的一种劈因子法是所谓林士谔-赵访熊法。这一小节我们介绍这种方法及其 FORTRAN 程序<sup>[C1]</sup>。

我们知道,二次方程是容易求解的,因此,如果能找出  $n$  次多项式  $f(x)$  的一个二次因子,就等于找到了方程 (7.12) 的一对复根。现在用二次式

$$\varphi(x) = x^2 + px + q$$

除  $f(x)$ , 设所得商式  $Q(x)$  和余式  $R(x)$  分别为

$$Q(x) = x^{n-2} + b_1x^{n-3} + \dots + b_{n-3}x + b_{n-2},$$

$$R(x) = Sx + t$$

即有

$$f(x) = Q(x) \cdot \varphi(x) + R(x) \quad (7.13)$$

比较 (7.12) 与 (7.13) 的同次幂系数, 可得

$$\begin{cases} b_1 = a_1 - p \\ b_2 = a_2 - pb_1 - q \\ b_3 = a_3 - pb_2 - qb_1 \\ \dots \dots \dots \\ b_k = a_k - pb_{k-1} - qb_{k-2} \\ \dots \dots \dots \\ b_{n-2} = a_{n-2} - pb_{n-3} - qb_{n-4} \\ S = b_{n-1} = a_{n-1} - pb_{n-2} - qb_{n-3} \\ t = a_n - qb_{n-2} = b_n + pb_{n-1} \end{cases} \quad (7.14)$$

或用递推公式表示:

$$\begin{cases} b_1 = a_1 - p \\ b_2 = a_2 - pb_1 - q \\ b_i = a_i - pb_{i-1} - qb_{i-2} \quad (i = 3, 4, \dots, n) \\ S = b_{n-1} \\ t = b_n + pb_{n-1} \end{cases}$$

如果  $\varphi(x)$  确是  $f(x)$  的二次因式, 则通过求解  $\varphi(x) = 0$  我们便可获得方程 (7.12) 的一对根. 如果  $\varphi(x)$  不是  $f(x)$  的二次因式, 而仅是近似二次因式, 即存在小量  $\Delta p$  和  $\Delta q$ , 使

$$\bar{\varphi}(x) = x^2 + (p + \Delta p)x + (q + \Delta q) \quad (7.15)$$

才是  $f(x)$  的二次因式, 则这时用  $\bar{\varphi}(x)$  除  $f(x)$  所得的余式  $R(x)$  为 0. 注意到  $s$  和  $t$  都是  $p$  和  $q$  的函数, 当  $\Delta p, \Delta q$  很小以致其二阶量可以忽略时, 可得

$$\begin{cases} s(p + \Delta p, q + \Delta q) \approx s + \frac{\partial s}{\partial p} \Delta p + \frac{\partial s}{\partial q} \Delta q = 0 \\ t(p + \Delta p, q + \Delta q) \approx t + \frac{\partial t}{\partial p} \Delta p + \frac{\partial t}{\partial q} \Delta q = 0 \end{cases}$$

再注意到 (7.14) 式的最后二式, 又可得

$$\begin{cases} b_{n-1} + \frac{\partial b_{n-1}}{\partial p} \Delta p + \frac{\partial b_{n-1}}{\partial q} \Delta q = 0 \\ b_n + pb_{n-1} + \left( \frac{\partial b_n}{\partial p} + p \frac{\partial b_{n-1}}{\partial p} + b_{n-1} \right) \Delta p + \left( \frac{\partial b_n}{\partial q} + p \frac{\partial b_{n-1}}{\partial q} \right) \Delta q = 0 \end{cases}$$

整理后可得方程组

$$\begin{cases} \frac{\partial b_{n-1}}{\partial p} \Delta p + \frac{\partial b_{n-1}}{\partial q} \Delta q = -b_{n-1} \\ \left( \frac{\partial b_n}{\partial p} + b_{n-1} \right) \Delta p + \frac{\partial b_n}{\partial q} \Delta q = -b_n \end{cases} \quad (7.16)$$

现在问题就在于解出 (7.16) 中的  $\Delta p, \Delta q$ , 以便构造出二次因式 (7.15). 为此, 先直接对 (7.14) 求微商. :

$$\begin{cases}
\frac{\partial b_1}{\partial p} = -1 \\
\frac{\partial b_2}{\partial p} = -b_1 - p \frac{\partial b_1}{\partial p} \\
\frac{\partial b_3}{\partial p} = -b_2 - p \frac{\partial b_2}{\partial p} - q \frac{\partial b_1}{\partial p} \\
\cdots \cdots \cdots \cdots \cdots \cdots \\
\frac{\partial b_k}{\partial p} = -b_{k-1} - p \frac{\partial b_{k-1}}{\partial p} - q \frac{\partial b_{k-2}}{\partial p} \\
\cdots \cdots \cdots \cdots \cdots \cdots \\
\frac{\partial b_{n-2}}{\partial p} = -b_{n-3} - p \frac{\partial b_{n-3}}{\partial p} - q \frac{\partial b_{n-4}}{\partial p} \\
\frac{\partial b_{n-1}}{\partial p} = -b_{n-2} - p \frac{\partial b_{n-2}}{\partial p} - q \frac{\partial b_{n-3}}{\partial p} \\
\frac{\partial b_n}{\partial p} = -b_{n-1} - p \frac{\partial b_{n-1}}{\partial p} - q \frac{\partial b_{n-2}}{\partial p}
\end{cases}$$

$$\begin{cases}
\frac{\partial b_1}{\partial q} = 0 \\
\frac{\partial b_2}{\partial q} = -1 \\
\frac{\partial b_3}{\partial q} = -b_1 - p \frac{\partial b_2}{\partial q} \\
\cdots \cdots \cdots \cdots \cdots \cdots \\
\frac{\partial b_{n-2}}{\partial q} = -b_{n-3} - p \frac{\partial b_{n-3}}{\partial q} - q \frac{\partial b_{n-4}}{\partial q} \\
\frac{\partial b_{n-1}}{\partial q} = -b_{n-2} - p \frac{\partial b_{n-2}}{\partial q} - q \frac{\partial b_{n-3}}{\partial q} \\
\frac{\partial b_n}{\partial q} = -b_{n-1} - p \frac{\partial b_{n-1}}{\partial q} - q \frac{\partial b_{n-2}}{\partial q}
\end{cases}$$

如果令  $\frac{\partial b_i}{\partial p} = -c_i$ , 则有

$$\begin{cases}
c_1 = 1 \\
c_2 = b_1 - pc_1 \\
c_3 = b_2 - pc_2 - qc_1 \\
\cdots \cdots \cdots \cdots \cdots \cdots \\
c_k = b_{k-1} - pc_{k-1} - qc_{k-2} \\
\cdots \cdots \cdots \cdots \cdots \cdots \\
c_n = b_{n-1} - pc_{n-1} - qc_{n-2}
\end{cases}$$

或引用递推公式

$$\begin{cases}
c_1 = 1 \\
c_2 = b_1 - pc_1 \\
c_i = b_{i-1} - pc_{i-1} - qc_{i-2} \quad (i = 3, 4, \cdots, n)
\end{cases}$$

由于有

$$\begin{cases} \frac{\partial b_{n-1}}{\partial p} = -c_{n-1}, & \frac{\partial b_{n-1}}{\partial q} = -c_{n-2} \\ \frac{\partial b_n}{\partial p} = -c_n, & \frac{\partial b_n}{\partial q} = -c_{n-1} \end{cases}$$

所以方程组 (7.16) 可化为

$$\begin{cases} c_{n-1}\Delta p + c_{n-2}\Delta q = b_{n-1} \\ (c_n - b_{n-1})\Delta p + c_{n-1}\Delta q = b_n \end{cases} \quad (7.16)'$$

用行列式解法解之, 记

$$\begin{aligned} D &= \begin{vmatrix} c_{n-1} & c_{n-2} \\ c_n - b_{n-1} & c_{n-1} \end{vmatrix} = c_{n-1}^2 - (c_n - b_{n-1})c_{n-2}, \\ D_p &= \begin{vmatrix} b_{n-1} & c_{n-2} \\ b_n & c_{n-1} \end{vmatrix} = b_{n-1}c_{n-1} - b_nc_{n-2}, \\ D_q &= \begin{vmatrix} c_{n-1} & b_{n-1} \\ c_n - b_{n-1} & b_n \end{vmatrix} = -b_{n-1}(c_n - b_{n-1}) + b_nc_{n-1} \end{aligned}$$

当  $D \neq 0$  时, 可得方程组 (7.16)' 的解为

$$\begin{cases} \Delta p = D_p/D \\ \Delta q = D_q/D \end{cases}$$

由  $\Delta p, \Delta q$  就可得  $p, q$  的第二次近似值

$$\begin{cases} p^* = p + \Delta p \\ q^* = q + \Delta q \end{cases}$$

用同样方法可得  $p, q$  的第三次、第四次……近似值, 直至  $p, q$  满足所需要的精度 (即  $s, t$  充分小) 为止。

下面设计劈因子法的 FORTRAN 程序。

设置哑元如下:

N 整型变量, 输入参数; 方程的次数。

A N 个元素的一维实型数组, 按降幂存放方程 (7.12) 的系数  $a_i, i = 1, 2, \dots, n$ 。

B, C 分别为 N 个元素的一维实型数组, 工作单元。

ES 实型变量, 用于控制精度 (控制  $s, t$ )。

ESD 实型变量, 控制常数, 当  $|D|$  小于此数时运算不能进行。

KT 整型变量, 允许最大迭代次数。

X, Y 分别为 N 个元素的一维实型数组, 存放根的实部和虚部。

子程序:

```
SUBROUTINE SRPE (N, A, B, C, ES, ESD, KT, X, Y)
  DIMENSION A(N), B(N), C(N), X(N), Y(N)
  NI = N
  C(1) = 1.0
  M1 = 1
  DO 10 I = 1, N
```

```

10  Y(I) = 0.0
11  IF(N1 .LT. 1) GO TO 15
    IF(N1 .EQ. 1) GO TO 16
    IF(N1 .EQ. 2) GO TO 17
    P = 0.0
    Q = 0.0
    K = 0
12  B(1) = A(1) - P
    B(2) = A(2) - P * B(1) - Q
    DO 1 I = 3, N1
1   B(I) = A(I) - P * B(I - 1) - Q * B(I - 2)
    S = B(N1 - 1)
    T = B(N1) + P * B(N1 - 1)
    IF(ABS(S) .GT. ES) GO TO 3
    IF(ABS(T) .LT. ES) GO TO 14
3   C(2) = B(1) - P * C(1)
    DO 2 I = 3, N1
2   C(I) = B(I - 1) - P * C(I - 1) - Q * C(I - 2)
    D = C(N1 - 1) * * 2 - (C(N1) - B(N1 - 1)) * C(N1 - 2)
    DP = B(N1 - 1) * C(N1 - 1) - B(N1) * C(N1 - 2)
    DQ = -B(N1 - 1) * (C(N1) - B(N1 - 1)) + B(N1) * C(N1 - 1)
    IF(ABS(D) .GT. ESD) GO TO 13
    P = P + 1.0
    Q = Q + 1.0
    GO TO 12
13  K = K + 1
    P = P + DP/D
    Q = Q + DQ/D
    IF(K .LE. KT) GO TO 12
    GO TO 15
14  X(M1) = -P/2.0
    X(M1 + 1) = -P/2.0
    FX = -P/2.0
    FX = FX * * 2 - Q
    IF (ABS(FX) .LT. 1.0E - 10) GO TO 4
    IF (FX .LT. 0.0) GO TO 5
    X(M1) = X(M1) + SQRT(FX)
    X(M1 + 1) = X(M1 + 1) - SQRT(FX)
    GO TO 4

```

```

5  Y(M1) = SQRT (ABS(FX));
   Y(M1 + 1) = -Y(M1)
4  M1 = M1 + 2
   N1 = N1 - 2
   DO 6 I = 1, N1
6  A(I) = B(I)
   GO TO 11
17 P = B(N1 - 1)
   Q = B(N1)
   GO TO 14
16 X(M1) = -B(N1)
15 RETURN
   END

```

**算例** 求实系数高次代数方程

$$f(x) = x^8 + 101x^7 + 208.01x^6 + 10891.01x^5 + 9802.08x^4 \\ + 78108.9x^3 - 99902x^2 + 790x - 1000 = 0$$

的全部根。

取  $ES = 10^{-7}$ ,  $ESD = 10^{-10}$ ,  $KT = 1000$ 。

解题程序如下:

```

C      MAIN PROGRAM
        DIMENSION A(8), B(8), C(8), X(8), Y(8)
        DATA A(1), A(2), A(3), A(4)/101, 208.01, 10891.01, 9802.08/
        DATA A(5), A(6), A(7), A(8)/79108.9, -99902, 790, -1000/
        CALL SRPE (8, A, B, C, 1.0E - 7, 1.0E - 10, 1000, X, Y)
        DO 100 I = 1, 8
100    WRITE (6, 200) X(I), Y(I)
200    FORMAT (1X, 2E20.8)
        STOP
        END

C
        SUBROUTINE SRPE (N, A, B, C, ES, ESD, KT, X, Y)
          : (本子程序段体)
        END

```

计算结果(包括 8 个复根的实部和虚部):

```

0.14513377E - 12      0.10000000E + 00
0.14513377E - 12      -0.10000000E + 00

```

|                   |                   |
|-------------------|-------------------|
| -0.10000000E + 01 | 0.30000000E + 01  |
| -0.10000000E + 01 | -0.30000000E + 01 |
| -0.32165559E - 09 | 0.10000000E + 02  |
| -0.32165559E - 09 | -0.10000000E + 02 |
| 0.10000000E + 01  | 0.00000000E + 00  |
| -0.10000000E + 03 | 0.00000000E + 00  |

原方程的精确解是:

$$\begin{aligned} x_1 &= 0.1i & x_2 &= -0.1i \\ x_3 &= -1 + 3i & x_4 &= -1 - 3i \\ x_5 &= 10i & x_6 &= -10i \\ x_7 &= 1 & x_8 &= -100 \end{aligned}$$

可见实部和虚部最大的误差是  $10^{-9}$ 。计算相当成功。

但必须指出,本程序用于解有重根或密集根的方程可能会遭到失败。另外,本算法的收敛速度比较慢,特别在次数较高的情况。

### 7-3 非线性方程组数值解法

我们考虑非线性方程组

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \dots \dots \dots \dots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases} \quad (7.17)$$

其中  $f_i (i = 1, 2, \dots, n)$  为已知的非线性实函数;  $x_i (i = 1, 2, \dots, n)$  为实变量。

若采用向量记号

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix}$$

则 (7.17) 可以写成与一元方程形式相仿的

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (7.18)$$

解方程组 (7.17) 或 (7.18) 就是求出满足方程组的一组  $x_1^*, x_2^*, \dots, x_n^*$ , 或称解向量  $\mathbf{x}^* = [x_1^*, x_2^*, \dots, x_n^*]^T$ 。

与一个未知数的超越方程一样,并没有简单的定理能告诉我们,方程组 (7.17) 的解是否存在或唯一。

解非线性方程组通常有两类方法:一类属于线性化方法,即将非线性方程组以一线性方程组来近似,由此构造一种迭代格式,用以逐次逼近所求的解案,这类方法有牛顿法及其各种改进;另一类属于求函数极小值的方法,即由这些非线性函数  $f_1, f_2, \dots, f_n$  构造一个模函数,例如构造



$$F(x_1, x_2, \dots, x_n) = \sum_{i=1}^n [f_i(x_1, x_2, \dots, x_n)]^2$$

或写成

$$F(\mathbf{x}) = \|\mathbf{f}(\mathbf{x})\|^2$$

记号  $\|\cdot\|$  指欧几里德范数。于是解方程组 (7.17) 归结为求  $F$  的极小值点, 此极小值点即非线性方程组的一组解, 这类方法有最速下降法等。

### 1. 牛顿-拉夫逊方法

先以一个二阶非线性方程组为例:

$$\begin{cases} f_1(x, y) = 0 \\ f_2(x, y) = 0 \end{cases} \quad (7.19)$$

设已知方程组 (7.19) 的解的一组初始近似值  $x_0, y_0$ , 则有

$$\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases}$$

假设  $f_1, f_2$  对  $x, y$  的二阶偏导数存在且连续, 利用泰勒公式, 我们在近似值邻近将非线性函数  $f_1, f_2$  用如下线性函数来近似:

$$\begin{cases} f_1(x, y) \approx f_1(x_0, y_0) + \frac{\partial f_1(x_0, y_0)}{\partial x} (x - x_0) + \frac{\partial f_1(x_0, y_0)}{\partial y} (y - y_0) \\ f_2(x, y) \approx f_2(x_0, y_0) + \frac{\partial f_2(x_0, y_0)}{\partial x} (x - x_0) + \frac{\partial f_2(x_0, y_0)}{\partial y} (y - y_0) \end{cases}$$

其中  $\frac{\partial f_1(x_0, y_0)}{\partial x}$  表示  $f_1(x, y)$  对  $x$  的一阶偏导数  $\frac{\partial f_1}{\partial x}$  在  $x_0, y_0$  处的取值, 等等。于是得到一线性方程组

$$\begin{cases} \frac{\partial f_1(x_0, y_0)}{\partial x} \Delta x + \frac{\partial f_1(x_0, y_0)}{\partial y} \Delta y = -f_1(x_0, y_0) \\ \frac{\partial f_2(x_0, y_0)}{\partial x} \Delta x + \frac{\partial f_2(x_0, y_0)}{\partial y} \Delta y = -f_2(x_0, y_0) \end{cases} \quad (7.20)$$

这时, 如果方程组的系数矩阵

$$\begin{bmatrix} \frac{\partial f_1(x_0, y_0)}{\partial x} & \frac{\partial f_1(x_0, y_0)}{\partial y} \\ \frac{\partial f_2(x_0, y_0)}{\partial x} & \frac{\partial f_2(x_0, y_0)}{\partial y} \end{bmatrix}$$

非奇异, 则可解出  $\Delta x, \Delta y$ , 即得

$$\begin{cases} x_1 = x_0 + \Delta x \\ y_1 = y_0 + \Delta y \end{cases}$$

以此作为非线性方程组解的一个新近似值, 并以  $x_1, y_1$  代替  $x_0, y_0$ , 重复上述过程, 那么当矩阵

$$\begin{bmatrix} \frac{\partial f_1(x, y)}{\partial x} & \frac{\partial f_1(x, y)}{\partial y} \\ \frac{\partial f_2(x, y)}{\partial x} & \frac{\partial f_2(x, y)}{\partial y} \end{bmatrix}$$

在解的邻近非奇异时,我们可得一系列的近似值  $x_n, y_n, n = 1, 2, \dots$ . 这时,若相邻两次近似值  $x_n, y_n$  和  $x_{n+1}, y_{n+1}$  满足条件:

$$\max \{ \delta_x, \delta_y \} < \varepsilon_1 \quad \text{或} \quad \max \{ |f_1|, |f_2| \} < \varepsilon_2$$

其中

$$\delta_x = \begin{cases} |x_{n+1} - x_n|, & |x_{n+1}| < C \\ \frac{|x_{n+1} - x_n|}{|x_{n+1}|}, & |x_{n+1}| \geq C \end{cases}$$

$$\delta_y = \begin{cases} |y_{n+1} - y_n|, & |y_{n+1}| < C \\ \frac{|y_{n+1} - y_n|}{|y_{n+1}|}, & |y_{n+1}| \geq C \end{cases}$$

我们便将最后得到的近似值  $x_{n+1}, y_{n+1}$  作为要求的答案. 此处,  $\varepsilon_1$  是允许误差,  $C$  是取绝对或相对误差的控制常数,随具体问题的要求而定,通常取 1;  $\varepsilon_2$  是一个接近于零的小数,视计算机的字长和数值范围而定.

上述方法就是解非线性方程组的牛顿迭代法. 它不难推广到  $n$  阶方程组的情形,而且也不难为之编写计算机程序,其中求解形如 (7.20) 的线性方程组可调用下一章中解线性方程组的通用子程序.

现在我们来把上述迭代法就  $n$  阶方程组 (7.18) 的情形表示成矩阵-向量形式. 记

$$\mathbf{x}_k = [x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}]^T, \quad k = 0, 1, 2, \dots$$

设  $\mathbf{x}^*$  为 (7.18) 的解,  $\mathbf{x}_0$  为其近似值,  $f(\mathbf{x})$  在解的邻近二阶可微,则利用泰勒公式可得

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

或线性化得

$$Df(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) = -f(\mathbf{x}_0)$$

这时,若矩阵

$$J = Df(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad \mathbf{x} = \mathbf{x}_0$$

$J$  称为雅可比 (Jacobi) 矩阵,非奇异,则可得唯一解

$$\mathbf{x}_1 = \mathbf{x}_0 - (Df(\mathbf{x}_0))^{-1}f(\mathbf{x}_0)$$

以  $\mathbf{x}_1$  代替  $\mathbf{x}_0$ , 重复这个过程,那么当雅可比矩阵

$$J = Df(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

在解的邻近非奇异时,我们可得

$$x_{n+1} = x_n - (Df(x_n))^{-1}f(x_n), \quad n = 0, 1, \dots \quad (7.21)$$

这就是解非线性方程组的牛顿迭代公式,其中  $(Df(x_n))^{-1}$  称为迭代矩阵. 这种以雅可比矩阵的逆阵为迭代矩阵的牛顿法又称为牛顿-拉夫逊 (Newton-Raphson) 法. 显然,它是一元方程的 (7.10) 公式的直接推广.

应用 (7.21) 型的迭代公式时,我们需要建立某种判别方法,以确定何时达到所研究的方程的解或达到满足精度的解. 这通常用建立所谓“误差函数”来实现. 有各种不同的误差函数,常用的一种是以欧几里德范数  $\|\cdot\|$  定义误差函数

$$e_k = \|f\| = \sqrt{\sum [f_i(x_k)]^2}$$

显然当  $e_k$  趋于零时,各  $f_i(x_k)$  也应该趋于零. 所以  $e_k$  是一个估量误差的良好尺度.

与一元方程的牛顿迭代法相仿,也可以证明,当初始近似值充分接近于解时,牛顿迭代过程按平方收敛速度收敛,即有

$$\lim_{n \rightarrow \infty} \frac{\|x_{n+1} - x^*\|}{\|x_n - x^*\|^2} = K \quad (K \text{ 为常数})$$

计算实践表明,牛顿迭代过程确需要有较好的初值,否则很有可能发散.

## 2. 布罗登方法(拟牛顿法)及其 FORTRAN 程序

牛顿-拉夫逊方法具有良好的收敛性,缺点是要在每一步中计算雅可比矩阵的逆阵  $J^{-1}$ . 要不是这样,它确是一种简单而又完善的方法. 因此,人们对于寻求具有同样收敛速度而对  $J$  和  $J^{-1}$  都不必进行显式计算的方法作了许多努力. 这样的方法一直在发展中,要评价哪一种方法最好恐怕还为期过早. 但不妨说,在经济地使用计算机已成为重要因素的场合,已很少使用牛顿-拉夫逊法了.

从牛顿-拉夫逊法发展起来的一种比较适用的方法是布罗登 (Broyden) 法,也称拟牛顿法. 它与牛顿-拉夫逊法比较主要有两个差别:

- 1) 不用计算雅可比矩阵  $J$  及其逆阵  $J^{-1}$ , 只使用一个常数矩阵去修正  $J^{-1}$  的估算值.
- 2) 引入一个阻尼因子,以保证每一步都能得到更好的估算值. 这是一个合理的想法,因为牛顿法不能保证其收敛性.

布罗登法的主要计算步骤如下:

- 1) 给出  $f(x) = 0$  的解的初始值  $x_0$ .
- 2) 求初始迭代矩阵  $H_0$ .
- 3) 计算  $f_i = f(x_i)$ ,  $i = 0, 1, 2, \dots$
- 4) 计算搜索方向  $p_i = -H_i f_i$ .
- 5) 选择阻尼因子  $t_i$ , 计算  $x_{i+1} = x_i + t_i p_i$ , 使得  $\|f_{i+1}\| < \|f_i\|$ , 这里  $\|f\|$  是  $f$  的欧氏范数

$$\|f\| = \sqrt{f_1^2 + f_2^2 + \dots + f_n^2}$$

- 6) 对给定的收敛控制常数  $\epsilon$ , 检验是否  $\|f_{i+1}\| < \epsilon$ ? 如果是,便停止计算.

- 7) 计算  $y_i = f_{i+1} - f_i$ .

- 8) 计算  $H_{i+1} = H_i - \frac{(H_i y_i + t_i p_i) p_i^T H_i}{p_i^T H_i y_i}$ .

9)  $j+1 \Rightarrow j$ , 转步 3).

上述各步,大多数是容易理解的. 需要说明的有两个方面:首先,阻尼因子  $t$  的选择并不是唯一的,任何可使范数  $\|f\|$  减少的  $t$  都可用. 但计算实践表明,在已知解的良好近似的情况下,可置  $t=1$ . 下面程序就是这样做的. 其次,第 8) 步表示对  $H$  矩阵的修正. 这里,将从  $H_0 = I$  (单位矩阵)开始,并且用  $\delta_i e_i$  代替  $p_i$ , 其中  $e_i$  是单位矩阵的第  $i$  列,  $\delta_i$  是非零的任意数,在本程序中取  $\delta_i = 0.001$ . 通过第 8) 步计算到  $H_{n+1}$ , 则  $H_{n+1}$  就是  $f$  的逆雅可比矩阵的近似. 然而,为了能看出这一点,必须进行较多的分析(参见 [C61]), 故这里我们从略.

以下给出布罗登法的 FORTRAN 子程序,子程序标识符为 BROYD (其中还将包含另一个子程序 STEPP),

哑元设置如下:

N        整型变量,方程的个数.  
X        N 个元素的一维实型数组,调用前存放初值,调用后存放计算结果(如解求列的话).  
TOL      实型变量,控制收敛常数,当  $\|f\| < \text{TOL}$  时过程收敛.  
MAXF    整型变量,计算  $f$  的最大次数.  
IY       整型变量,输出信息的标志,其意义如下:  
         IY = 0 求解成功.  
         IY = 1 计算次数已超过 MAXF.  
         IY = 2 在开始计算 H 时用零作除数.  
         IY = 3 在开始计算 H 时调用 FUNCT 失败.  
         IY = 4 修正 H 时用零作除数.  
         IY = 5 修正 H 时在 FUNCT 失败.  
F        N 个元素的一维实型数组,用于计算  $f(x)$  的函数值.  
Y        N 个元素的一维实型数据,用于存放  $y = f_{i+1} - f_i$   
PQ       N 个元素的一维实型数组,用于存放  $p_i = \delta_i e_i$  (本程序取  $\delta_i = 0.001$ ).  
V        N 个元素的实型数组,存放  $f_i$  的值.  
H        NXN 的二维实型数组,迭代矩阵,开始时  $H_0 = I$  (单位矩阵).

此外,本子程序还需调用计算对应于任一组自变量的一组函数值的子程序

SUBROUTINE FUNCT (N, X, F)

其中哑元的意义同上. 本子程序由使用者自编.

还要指出的是,使用下面程序可能出现  $IY = 1, 2, \dots, 5$  等情况,这时可根据 IY 的值采取适当的措施,如改变方程的次序;改变初始值;放宽控制精度要求;增大计算各  $F(i)$  的最大次数 MAXF 等.

子程序:

```
SUBROUTINE BROYD (N, X, TOL, MAXF, IY,  
1  F, Y, PQ, V, H)  
  DIMENSION X(N), F(N), Y(N), PQ(N), V(N), H(N, N)
```

```

      IY = 0
      CALL FUNCT (N, X, F)
      IF (IY .EQ. 5) GO TO 10
      NCOUNT = 1
      DO 2 I = 1, N
      PQ (I) = 0
      DO 3 J = 1, N
3    H(I, J) = 0.
2    CONTINUE
      DO 4 I = 1, N
4    H(I, I) = 1
      DO 5 K = 1, N
      PQ (K) = 0.001
      CALL STEPP (IY, N, X, PQ, V, F, NCOUNT, Y, H)
      IF (IY .GT. 0) GO TO 10
5    PQ (K) = 0.
6    DO 7 I = 1, N
      SA = 0.
      DO 12 J = 1, N
      SA = SA - H(I, J) * F(I)
12   PQ(I) = SA
7    CONTINUE
      CALL STEPP (IY, N, X, PQ, V, F, NCOUNT, Y, H)
      IF (IY .GT. 0) IY = IY + 2
      IF (IY .GT. 0) GO TO 10
      SA = 0.
      DO 9 I = 1, N
9    SA = SA + F(I) * F(I)
      SA = SQRT (SA)
      IF (SA .LT. TOL) GO TO 10
      IF (NCOUNT .GT. MAXF) GO TO 11
      GO TO 6
11   IY = 1
10   RETURN
      END

      SUBROUTINE STEPP (IY, N, X, PQ, V, F, NCOUNT, Y, H)
      DIMENSION X(N), V(N), F(N), PQ(N), Y(N), H(N, N)
      DO 2 I = 1, N

```

```

      X(I) = X(I) + PQ(I)
2   V(I) = F(I)
      CALL FUNCT (N, X, F)
      IF (IY .EQ. 1) GO TO 9
      NCOUNT = NCOUNT + 1
      DO 3 I = 1, N
3   Y(I) = F(I) - V(I)
      SA = 0.
      DO 4 I = 1, N
      SB = 0.
      DO 5 J = 1, N
5   SB = SB + H(I, J) * Y(J)
      V(I) = SB - PQ(I)
4   SA = SA + SB * PQ(I)
      IF (SA .EQ. 0) IY = 2
      IF (SA .EQ. 0) GO TO 9
      DO 6 J = 1, N
      SB = 0.
      DO 7 I = 1, N
7   SB = SB + PQ(I) * H(I, J)
      SB = SB/SA
      DO 8 I = 1, N
8   H(I, J) = H(I, J) - SB * V(I)
6   CONTINUE
9   RETURN
      END

```

**算例** 求解下列非线性方程组

$$\begin{cases} x_1 - 14x_2 + x_1^2 + x_2^3 - 29 = 0 \\ x_1 - 2x_2 + 5x_1^3 - x_2^3 - 13 = 0 \end{cases}$$

我们取初值  $x_1^{(0)} = 8$ ,  $x_2^{(0)} = -1$ . 又取控制收敛常数  $TOL = 10^{-4}$ , 计算  $f$  的最大次数  $MAXF = 600$

解题程序如下:

```

C      THE MAIN PROGRAM
      DIMENSION X1(2), F1(2), Y1(2), PQ1(2), V1(2), H1(2, 2)
      X1(1) = 8.
      X1(2) = -1.
      CALL BROYD (2, X1, 0.00001, 600, IY, F1, Y1, PQ1, V1, H1)

```

```

        WRITE (6, 100) IY
100  FORMAT (1X, 3HIY = , I2/)
        IF (IY. GT. 0) GO TO 300
        WRITE (6, 200) X1(1), X1(2)
200  FORMAT (1X, 6HX1(1) = , F16.7/6HX1(2) = , F17.6)
300  STOP
      END

```

C

```

      SUBROUTINE BROYD (N, X, TOL, MAXF, IY,
1  F, Y, PQ, V, H)
      :
      :   (本子程序段体)
      :
      END

```

C

```

      SUBROUTINE STEPP (IY, N, X, PQ, V, F, NCOUNT, Y, H)
      :
      :   (本子程序段体)
      :
      END

```

C

```

      SUBROUTINE FUNCT (N, X, F)
      DIMENSION X(N), F(N)
      F(1) = X(1) + ((X(2) + 1) * X(2) - 14.) * X(2) - 29.
      F(2) = X(1) + ((-X(2) + 5.) * X(2) - 2.) * X(2) - 13.
      RETURN
      END

```

计算结果:

```

      IY = 0
      X1(1) =          5.0000045
      X1(2) =          3.9999999

```

本题的精确结果是:

$$x_1 = 5, \quad x_2 = 4$$

可见, 计算机计算结果相当好.

计算实践表明, 对于超越方程组, 布罗登方法不一定能得到满意的计算结果.

### 3. 最速下降法及其 FORTRAN 程序

**最速下降法**也称**梯度法**. 我们仍先以求解两个方程的非线性方程组

$$\begin{cases} f_1(x, y) = 0 \\ f_2(x, y) = 0 \end{cases} \quad (7.23)$$

为例来说明方法的原理.

构造模函数(也称目标函数)

$$F(x, y) = [f_1(x, y)]^2 + [f_2(x, y)]^2 \quad (7.24)$$

于是,方程组(7.23)之解就是  $F(x, y)$  的零极小值点,反之亦然.

函数  $F(x, y)$  在几何上对应于一空间曲面  $F = F(x, y)$ . 如果用一系列平行于  $x$ - $y$  坐标平面的平面  $F = \text{常数}$  来截曲面  $F = F(x, y)$ , 则得一族平面曲线. 把这些平面曲线投影在  $x$ - $y$  面上, 即得所谓等高线族. 对于处在函数  $F(x, y)$  定义域  $D$  内的任何点, 总存在族中的一条等高线通过它. 如果从  $D$  内的某个点  $(x_0, y_0)$  出发沿着使  $F$  值下降的方向逐步地下降  $F$  值, 一直降到它的零极小值, 那么就得到所求方程的解.

由微积分学知道, 在一点处等高线的法向, 也即函数  $F(x, y)$  在该点处的梯度方向

$$\mathbf{g} = \left[ \frac{\partial F}{\partial x}, \frac{\partial F}{\partial y} \right]^T$$

是使  $F$  值上升最快的方向, 因此其反方向

$$-\mathbf{g} = \left[ -\frac{\partial F}{\partial x}, -\frac{\partial F}{\partial y} \right]^T$$

就是使  $F$  值下降最快的方向. 最速下降法就是沿着这个方向来逐步下降  $F$  值的. 具体做法是:

设  $(x_0, y_0)$  是解的一个近似值, 计算  $F$  在此点的梯度

$$\begin{aligned} \mathbf{g}_0 &= [g_{x0}, g_{y0}]^T \\ &= \left[ \left( \frac{\partial F}{\partial x} \right)_0, \left( \frac{\partial F}{\partial y} \right)_0 \right]^T \end{aligned}$$

其中下标 0 表示括号中的函数在  $(x_0, y_0)$  处取值.

从  $(x_0, y_0)$  点出发沿负梯度方向  $-\mathbf{g}_0$  跨一适当的步长, 可得新的点

$$\begin{cases} x_1 = x_0 - \lambda_0 g_{x0} = x_0 - \lambda_0 \left( \frac{\partial F}{\partial x} \right)_0 \\ y_1 = y_0 - \lambda_0 g_{y0} = y_0 - \lambda_0 \left( \frac{\partial F}{\partial y} \right)_0 \end{cases}$$

其中因子  $\lambda_0$  可以有各种不同的取法. 例如, 一种简单的取法是, 取  $\lambda_0 = \bar{\lambda}$  ( $\bar{\lambda}$  为给定值), 然后检验不等式

$$F(x_0 - \bar{\lambda} g_{x0}, y_0 - \bar{\lambda} g_{y0}) < F(x_0, y_0)$$

是否成立, 如果成立, 则继续下一步迭代; 否则, 可缩小  $\lambda_0$ , 比如让  $\lambda_0 = \alpha \bar{\lambda}$  ( $0 < \alpha < 1$ ), 再检验上述不等式, 如此调整  $\alpha$ , 直至上述不等式满足为止. 事实上, 由于负梯度的性质, 满足上述不等式的  $\lambda_0$  总是存在的. 用这种方法选取  $\lambda_0$ , 称为梯度法的可接受点形式.

$\lambda_0$  的另一种取法是使得新点  $(x_1, y_1)$  是  $F(x, y)$  在  $-\mathbf{g}_0$  方向上的相对极小值点, 即取  $\lambda_0 = \bar{\lambda}$ , 使得

$$F(x_0 - \bar{\lambda} g_{x0}, y_0 - \bar{\lambda} g_{y0}) = \min$$

用这种方法选取  $\lambda_0$ , 称为梯度法的完备形式.

在计算机语言程序中, 常采用如下取法:



$$\lambda_0 = \frac{F(x_0, y_0)}{\left(\frac{\partial F}{\partial x}\right)_0^2 + \left(\frac{\partial F}{\partial y}\right)_0^2}$$

其中偏导数在比较难计算的情况下常改用差商来近似代替,即取

$$\begin{aligned}\left(\frac{\partial F}{\partial x}\right)_0 &= \frac{F(x_0 + \Delta x_0, y_0) - F(x_0, y_0)}{\Delta x_0} \\ \left(\frac{\partial F}{\partial y}\right)_0 &= \frac{F(x_0, y_0 + \Delta y_0) - F(x_0, y_0)}{\Delta y_0}\end{aligned}$$

而其中

$$\begin{aligned}\Delta x_0 &= \begin{cases} Cx_0 & x_0 \neq 0, \\ C & x_0 = 0; \end{cases} \\ \Delta y_0 &= \begin{cases} Cy_0 & y_0 \neq 0, \\ C & y_0 = 0, \end{cases}\end{aligned}$$

$C$  为控制常数,一般取  $10^{-5}$  或  $10^{-6}$ .

梯度法的优点是程序简单,每次迭代所需的计算量少,所要求的存储量也少,而且从一个不太好的初始近似出发,还是能收敛于局部极小值点.这个方法的最大缺点是收敛速度缓慢(线性速度——证略),尤其是在解的邻近,常常为了提高一点精度而需要付出很大的代价.因此,实际应用中,常与牛顿法相结合,使两个方法相互取长补短,以达到既能保证收敛性又能加快收敛速度.结合的方式通常是先采用最速下降法,而当  $F$  值降到一定程度以后,便改用牛顿法.

综上所述,下面我们就  $n$  阶方程组的情形来设计计算机算法及其 FORTRAN 程序.

设  $n$  阶非线性方程组为

$$\begin{cases} f_i(x_1, x_2, \dots, x_n) = 0 \\ i = 1, 2, \dots, n \end{cases} \quad (7.25)$$

作目标函数

$$F(x_1, x_2, \dots, x_n) = \sum_{i=1}^n f_i^2(x_1, x_2, \dots, x_n)$$

当  $F(x_1^*, x_2^*, \dots, x_n^*) < \varepsilon$  时,便认为  $x_1^*, x_2^*, \dots, x_n^*$  是方程组 (7.25) 满足精度  $\varepsilon$  的一组解.

计算步骤:

- 1) 从给定的不全为零的初值  $x_1^0, x_2^0, \dots, x_n^0$  出发,设已经算到第  $k$  步,得  $x_1^k, x_2^k, \dots, x_n^k$ .
- 2) 计算目标函数  $F(x_1^k, x_2^k, \dots, x_n^k)$ .
- 3) 若  $F(x_1^k, x_2^k, \dots, x_n^k) < \varepsilon$ , 则认为  $x_1^k, x_2^k, \dots, x_n^k$  是所求的一组解;否则做下一步.
- 4) 计算

$$\frac{\partial F}{\partial x_1} = \frac{F(x_1^k + \Delta x_1, x_2^k, \dots, x_n^k) - F(x_1^k, x_2^k, \dots, x_n^k)}{\Delta x_1}$$

$$\frac{\partial F}{\partial x_1} = \frac{F(x_1^k, x_2^k + \Delta x_2, \dots, x_n^k) - F(x_1^k, x_2^k, \dots, x_n^k)}{\Delta x_2}$$

$$\vdots$$

$$\frac{\partial F}{\partial x_n} = \frac{F(x_1^k, x_2^k, \dots, x_n^k + \Delta x_n) - F(x_1^k, x_2^k, \dots, x_n^k)}{\Delta x_n}$$

其中

$$\Delta x_i = \begin{cases} Cx_i^k & x_i^k \neq 0, \\ C & x_i^k = 0, \end{cases}$$

$C$  为控制常数,一般取  $10^{-3}$  或  $10^{-6}$ .

5) 计算

$$x_i^{k+1} = x_i^k - \lambda_k \frac{\partial F}{\partial x_i}, \quad i = 1, 2, \dots, n$$

其中

$$\lambda_k = \frac{F(x_1^k, x_2^k, \dots, x_n^k)}{\sum_{i=1}^n \left( \frac{\partial F}{\partial x_i} \right)^2}$$

然后重复第 2) 步开始的计算过程,直至算出满足精度  $\varepsilon$  的结果.

子程序中的哑元:

$N$  方程组的阶数;整型变量,输入参数.

$X$   $N$  个元素的一维实型数组,开始存放初值,最后存放计算结果;输入兼输出参数.

$C$  控制常数;实型变量,输入参数;通常取  $10^{-3}$  或  $10^{-6}$ .

$FNC$  计算目标函数的哑过程名,形式为

FUNCTION FNC (N, X)

其中  $N$  与  $X$  的意义如上.

$CX, DF$   $N$  个元素的一维实型数组,工作单元;分别表示自变量增量和模函数的偏导数值.

$EPS$  实型变量;目标函数的精度.

$M$  整型变量;为了解迭代次数而增设的输出参数.

子程序:

```
SUBROUTINE NLGRAD (N, X, C, FNC, CX, DF, EPS, M)
  DIMENSION X(N), CX(N), DF(N)
  3 DO 12 I = 1, N
    IF (X(I)) 6, 9, 6
  9 CX(I) = C
    GO TO 12
  6 CH(I) = C * X(I)
  12 CONTINUE
```

```

F = FNC(N, X)
IF (F .LT. EPS) RETURN
SUM = 0.0
DO 16 I = 1, N
X(I) = X(I) + CX(I)
FH = FNC(N, X)
DF(I) = (FH - F)/CX(I)
SUM = SUM + DF(I) * * 2
16 X(I) = X(I) - CX(I)
RLTA = F/SUM
DO 19 I = 1, N
19 X(I) = X(I) - RLTA * DF(I)
M = M + 1
GO TO 3
END

```

**算例** 求解下列非线性方程组

$$\begin{cases} 2x_1 - x_2^2 + 5x_3 = 0 \\ x_1^2 + x_2^2 - 10x_3 - 60 = 0 \\ x_1 + 2x_2 - x_3^2 - 4 = 0 \end{cases}$$

取初值  $x_1^0 = x_2^0 = x_3^0 = 6$ ,  $\text{EPS} = 10^{-4}$ .

解题程序:

```

EXTERNAL FNC
DIMENSION X(3), CX(3), DF(3)
DATA X(1), X(2), X(3)/6.0, 6.0, 6.0/
N = 3
C = 1E - 5
EPS = 1E - 6
M = 0
CALL NLGRAD (N, X, C, FNC, CX, DF, EPS, M)
WRITE (6,100) X(1), X(2), X(3)
100 FORMAT (1X, 3HX1=, F8.4, 4X, 3HX2=, F8.4, 4X, 3HX3=, F8.4)
WRITE (6,200) M
200 FORMAT (1X, 2HM=, I6)
STOP
END

```

C

SUBROUTINE NLGRAD (N, X, C, FNC, CX, DF, EPS, M)

⋮ (本子程序段体)

END

C

FUNCTION FNC (N, X)

DIMENSION X(N)

FNC = (2.\*X(1) - X(2)\*\*2 + 5.\*X(3))\*\*2

1 + (X(1)\*\*2 + X(2)\*\*2 - 10.\*X(3) - 60.)\* \*\*2

2 + (X(1) + 2.\*X(2) - X(3))\*\*2 - 4.)\* \*\*2

RETURN

END

计算结果如下:

X1 = 8.0000 X2 = 6.0001 X3 = 4.0001 M = 43

计算实践表明,随着精度要求的增高,需用的计算时间往往有较明显的增多.

## 习 题 七

1. 用一般迭代法求方程

$$\lg x + 7 - 2x = 0$$

的最大值(精确到  $10^{-4}$ ).

2. 用一般迭代法求方程

$$x^3 - 2x - 5 = 0$$

的最小正根(精确到  $10^{-3}$ ).

3. 已知四次方程

$$f(x) = x^4 - 12x^3 + 46x^2 - 60x + 25 = 0$$

在  $x = 1$  上具有重根. 由于在重根上有  $f'(x) = 0$ , 因此当重根存在时, 求出重根的简便方法是首先解  $f'(x) = 0$ , 然后检查这些根是否也是原方程  $f(x) = 0$  的根. 试用此方法解上述方程.

4. 为求方程  $x^3 - x^2 - 1 = 0$  在  $x_0 = 1.5$  附近的一个根, 将方程改写成下列等价形式, 并建立相应的迭代公式:

(1)  $x = 1 + \frac{1}{x^2}$ , 迭代公式  $x_{n+1} = 1 + \frac{1}{x_n^2}$ ;

(2)  $x^3 = 1 + x^2$ , 迭代公式  $x_{n+1} = \sqrt[3]{1 + x_n^2}$ ;

(3)  $x^2 = \frac{1}{x-1}$ , 迭代公式  $x_{n+1} = \frac{1}{\sqrt{x_n-1}}$ .

试分析每一种迭代公式的收敛性.

5. 用施多姆定理求出方程

$$x^3 + x^2 - 5x + 3 = 0$$

的重根及其它根.

6. 用施多姆定理证明

$$x^4 + x + 1 = 0$$

没有实根,用笛卡儿法则证明这个方程没有正实根.利用不等式证明它也没有负实根.

7. 利用施多姆序列判断下列方程

$$f(x) = x^4 - 6x^3 + 5x^2 + 12x + 4 = 0$$

有没有重根.如果有,把它求出来.

8. 已知函数  $f(x) = x^2 - 0.9x - 8.5$  在区间  $2 \leq x \leq 3$  内有一个实数根.要使这个根精确到  $\varepsilon = 10^{-6}$ ,问需进行多少次对分?

9. 利用对分法的 FORTRAN 程序,上机求解下列方程

$$f(x) = x^4 - 7.223x^3 + 13.447x^2 - 0.672x - 10.223 = 0$$

的所有实根.

10. 设  $f(x) = 0$  在  $[a, b]$  内有根  $x^*$ ,  $f'(x)$  在  $[a, b]$  上连续且不变号,试确定使迭代公式

$$x_{n+1} = x_n - \alpha f(x_n), \quad \alpha = 0, 1, 2, \dots$$

收敛的  $\alpha$  取值范围.

11. 已知由连续函数  $f(x)$  构成的方程

$$f(x) = 0$$

在  $[a, b]$  内只有一个根.如果用逐次三等分区间的方法来求根,试问可得出什么结论?并与对分区间方法比较其优劣.

12. 用区间对分法确定方程

$$f(x) = x^3 - 3x + 1 = 0$$

的最小正根所在的区间  $[a, b]$ ,使之满足

$$K = \frac{M_1}{2m_1} < 1$$

其中

$$M_1 = \max_{a \leq x \leq b} |f''(x)|, \quad m_1 = \min_{a \leq x \leq b} |f'(x)|$$

13. 证明计算  $\sqrt[n]{a}$  的牛顿迭代公式为

$$x_{n+1} = \frac{1}{3} \left( 2x_n + \frac{a}{x_n^2} \right)$$

并用它来计算  $\sqrt[3]{411.7910}$  (精确到  $10^{-6}$ ).

14. 试用求根的方法求  $\sqrt{7}$ .

15. 设  $N > 0$ , 试导出

(1) 计算  $\frac{1}{N}$  的牛顿迭代公式.

(2) 计算  $\sqrt{N}$  的牛顿迭代公式.

(3) 证明  $\sqrt{N} \approx \frac{A+B}{4} + \frac{N}{A+B}$  ( $N = A+B$ ).

16. 试导出  $\frac{1}{\sqrt{a}}$  ( $a > 0$ ) 的牛顿迭代程序,要求公式中既无开方,又无除法运算.

17. 用牛顿迭代法求下列方程的最小正根:

(1)  $x^3 - 2x - 5 = 0$ .

(2)  $2^x - 4x = 0$ .

(3)  $x^6 - 40x^3 + 70x^2 - 15 = 0$

要求精确到  $10^{-4}$ .

18. 应用复数运算的牛顿迭代程序,在计算机上求解下列方程

(1)  $z^4 - 3z^2 + 20z^2 + 44z + 54 = 0^*$

在  $z_0 = 2.5 + 4.5i$  附近的复根.

$$(2) z^4 - z^2 - 4z + 34z - 120 = 0$$

在  $z_0 = 4 + 4i$  附近的复根.

$$19. \text{ 设 } m_1 = \min_{a \leq x \leq b} |f'(x)|, \quad M_1 = \max_{a \leq x \leq b} |f''(x)|$$

试证明牛顿迭代法有误差估计式

$$\begin{aligned} |x_n - x^*| &\approx \left| \frac{f'(x_n)}{2f''(x_n)} \right| |x_n - x_{n-1}|^2 \\ &\leq \frac{M_1}{2m_1} |x_n - x_{n-1}|^2 \end{aligned}$$

20. 证明: 设  $f(x)$  在  $[a, b]$  上满足条件:

$$(1) f(a)f(b) < 0$$

$$(2) f'(x), f''(x) \text{ 连续, 不变号;}$$

$$(3) \text{ 取 } x_0, \text{ 使得 } f(x_0)f''(x_0) > 0,$$

则方程  $f(x) = 0$  在  $[a, b]$  内有唯一根  $x^*$ , 且由牛顿迭代程序

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

确定的序列  $\{x_n\}$  收敛于  $x^*$ , 且具有二阶收敛速度.

21. 解方程  $f(x) = 0$  的迭代程序

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

称为简化的牛顿程序. 试推导简化牛顿程序的收敛条件, 并证明

$$|x_{n+1} - x^*| \leq \frac{|f'(x_0)|}{m_1} |x_{n+1} - x_n|$$

其中  $m_1 = \min_{a \leq x \leq b} |f'(x)|$ ,  $f(x)$  在  $[a, b]$  内有唯一解  $x^*$ .

22. 用劈因子法求多项式方程

$$x^4 + 7x^3 + 24x^2 + 25x - 15 = 0$$

的二次因式(精确到  $10^{-4}$ ).

23. 利用劈因子法的 FORTRAN 程序求实系数高次代数方程

$$f(x) = x^6 - 2x^5 + 2x^4 + x^3 + 6x^2 - 6x + 8 = 0$$

的全部根.

24. 利用劈因子法的 FORTRAN 程序, 在计算机上求解多项式方程

$$x^4 + 39x^3 + 958x^2 + 1080x - 2000 = 0$$

的根.

$$25. \text{ 设 } f(x) = \begin{bmatrix} e^{(x_1^2 + x_2^2)} - 3 \\ x_1 + x_2 - \sin(3(x_1 + x_2)) \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ 试计算 Jacobi 矩阵 } Df(x), \text{ 并求出使}$$

$Df(x)$  奇异的  $x$  值.

26. 试用牛顿-拉夫逊法解下列非线性方程组

$$\begin{cases} x_1 + 3e^{x_2} - 2 = 0 \\ -x_1^2 - 3x_1e^{x_2} + 1 = 0 \end{cases}$$

27. 选择合适的方法, 求解下列非线性方程组:

$$\begin{aligned} (1) \begin{cases} x_1^2 + x_2^2 - x_1 = 0 \\ x_1^2 - x_1^2 - x_2 = 0 \end{cases} & \quad (3) \begin{cases} x_1 + 2x_2 + x_3 + 4x_4 = 20.700 \\ x_1^2 + 2x_1x_2 + x_3^2 = 15.880 \\ x_1^3 + x_1^3 + x_4 = 21.218 \\ 3x_2 + x_3x_4 = 7.900 \end{cases} \\ (2) \begin{cases} x_1^2 - 10x_1 + x_2^2 + 8 = 0 \\ x_1x_2^2 + x_1 - 10x_2 + 8 = 0 \end{cases} & \end{aligned}$$

## 第8章 线性代数方程组数值解法

## 8-1 引言

线性代数方程组的数值解法还包括求矩阵的逆阵和行列式之值。

有人估计,工程实践中提出的计算问题,几乎有一半以上涉及求解线性代数方程组。这些线性代数方程组可能以完整的问题独立地出现,也可能作为更复杂的过程的组成部分而存在。

### 4. 阶线性代数方程组的一般形式为

[illegible]

或写成矩阵-向量形式

$$Ax = b \quad (8.2)$$

其中  $A$  称为系数矩阵,  $x$  称为解向量,  $b$  称为右端常向量, 分别为

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

我们知道,若矩阵  $A$  非奇异,即  $A$  的行列式  $\det A \neq 0$ ,根据克兰姆(Cramer)法则,方程组(8.1)有唯一解

$$x_i = \frac{D_i}{D}, \quad i = 1, 2, \dots, n \quad (8.3)$$

其中  $D$  表示  $\det A$ ,  $D_i$  表示  $D$  中第  $i$  列换成  $b$  后所得的行列式。显然, 对于较高阶的情形, 用公式 (8.3) 计算方程组 (8.1) 的解是不现实的。一个  $n$  阶行列式有  $n!$  项, 每一项又是  $n$  个数的乘积。就算不计舍入误差对计算结果的影响, 对较大的  $n$ , 其运算量之大, 也是计算机在一般情况下难以容许的。因此, 线性代数方程组的计算机解法使用另外两类方法: 直接法和迭代法。

使用直接法,经有限次运算可求出(假定没有舍入误差的话)方程组的精确解;使用迭代法,则将求解方程组的问题化为构造一个无穷序列,用这个无穷序列去逐步逼近精确解。两类方法各有优缺点。前者由于所需的计算机存储随着方程组阶数的增高而增加,故适宜于系数矩阵阶数不太高的问题,其计算量较小,精度较高,但程序较复杂。后者适宜于高阶问题,一般说计算量较大,但程序较简单。

解线性代数方程组直接法的实质, 在于对方程组系数矩阵(行列式)进行某种简化变换, 因此, 我们可利用解线性代数方程组的直接法兼求矩阵的逆阵和求行列式之值.

解线性代数方程组的方法很多,本章我们只介绍在计算机上最常使用的、简单有效的少数几个算法.

## 8-2 解线性方程组(包括求逆矩阵及行列式值)的高斯消去法

### 1. 高斯消去法的基本思想

高斯消去法属于直接法,一般由“消元过程”和“回代过程”两部分组成.下面先以一个简单实例,然后再对一般的 $n$ 阶方程组,说明高斯消去法的基本思想.

考虑三阶方程组

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ x_1 - x_2 + 5x_3 = 0 \\ 4x_1 + x_2 - 2x_3 = 2 \end{cases} \quad (8.4)$$

及其相应的增广矩阵

$$\left[ \begin{array}{ccc|c} 2 & 4 & -2 & 6 \\ 1 & -1 & 5 & 0 \\ 4 & 1 & -2 & 2 \end{array} \right] \quad (8.5)$$

(1) 消元过程:

第一步,从(8.4)的第二、三个方程消去 $x_1$ 的系数,即用 $-\frac{1}{2}$ 乘第一个方程,加到第二个方程;用 $-\frac{4}{2} = -2$ 乘第一个方程,加到第三个方程,于是方程组(8.4)化为等价方程组

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ -3x_2 + 6x_3 = -3 \\ -7x_2 + 2x_3 = -10 \end{cases} \quad (8.6)$$

相应地,增广矩阵(8.5)化为

$$\left[ \begin{array}{ccc|c} 2 & 4 & -2 & 6 \\ -3 & 6 & -3 & -3 \\ -7 & 2 & -10 & -10 \end{array} \right] \quad (8.7)$$

第二步,从(8.6)的第三个方程消去 $x_2$ 的系数,即用 $-\frac{7}{3}$ 乘第二个方程,加到第三个方程,于是方程组(8.4)化为等价方程组

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ -3x_2 + 6x_3 = -3 \\ -12x_3 = -3 \end{cases} \quad (8.8)$$

相应地,增广矩阵(8.5)化为

$$\left[ \begin{array}{ccc|c} 2 & 4 & -2 & 6 \\ -3 & 6 & -3 & -3 \\ & & -12 & -3 \end{array} \right] \quad (8.9)$$



这就是说,消元过程把原方程组化为上三角形方程组(8.8),系数矩阵化为上三角形矩阵(8.9).

(2) 回代过程:

由上三角形方程组(8.8)的最后一个方程往回依次解出

$$x_3 = (-3)/(-12) = 0.25$$

$$x_2 = (-3 - 6 \times 0.25)/(-3) = 1.5$$

$$x_1 = (6 + 2 \times 0.25 - 4 \times 1.5)/2 = 0.25$$

即得解向量

$$\mathbf{x} = [0.25, 1.5, 0.25]^T$$

现在考虑一般的  $n$  阶方程组(8.2)

$$A\mathbf{x} = \mathbf{b}$$

把它记为

$$A^{(0)}\mathbf{x} = \mathbf{b}^{(0)}$$

即

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} \end{bmatrix} \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_n^{(0)} \end{bmatrix} \quad (8.10)$$

(1) 消元过程:

第1次消元, 设  $a_{11}^{(0)} \neq 0$ , 计算乘数

$$m_{i1} = a_{i1}^{(0)} / a_{11}^{(0)} \quad (i = 2, 3, \cdots, n)$$

用  $-m_{i1}$  乘第1个方程, 加到第  $i$  个方程 ( $i = 2, 3, \cdots, n$ ), 即计算

$$\begin{cases} a_{ij}^{(1)} = a_{ij}^{(0)} - m_{i1}a_{1j}^{(0)} & (i, j = 2, 3, \cdots, n) \\ b_i^{(1)} = b_i^{(0)} - m_{i1}b_1^{(0)} & (i = 2, 3, \cdots, n) \end{cases}$$

于是方程组(8.10)化为等价方程组

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & \vdots & \cdots & \vdots \\ & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \quad (8.11)$$

第2次消元, 设  $a_{22}^{(1)} \neq 0$ , 计算乘积

$$m_{i2} = a_{i2}^{(1)} / a_{22}^{(1)} \quad (i = 3, 4, \cdots, n)$$

用  $-m_{i2}$  乘第2个方程, 加到第  $i$  个方程 ( $i = 3, 4, \cdots, n$ ), 即计算

$$\begin{cases} a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i2}a_{2j}^{(1)} & (i, j = 3, 4, \cdots, n) \\ b_i^{(2)} = b_i^{(1)} - m_{i2}b_2^{(1)} & (i = 3, 4, \cdots, n) \end{cases}$$

于是方程组(8.11)又化为等价方程组

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ & & \vdots & \cdots & \vdots \\ & & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix} \quad (8.12)$$

继续此过程, 经过  $n-1$  次消元, 最后得等价的上三角形方程组

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{bmatrix} \quad (8.13)$$

(2) 回代过程:

对方程组 (8.13), 记

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix}$$

$$f = [f_1, f_2, \cdots, f_n]^T = [b_1^{(0)}, b_2^{(1)}, \cdots, b_n^{(n-1)}]^T$$

计算

$$\begin{cases} x_n = f_n / u_{nn} \\ x_k = \left( f_k - \sum_{i=k+1}^n u_{ki} x_i \right) / u_{kk} \quad (k = n-1, \cdots, 2, 1) \end{cases}$$

于是可得解向量

$$x = [x_1, x_2, \cdots, x_n]^T$$

以上就是高斯消去法的基本思想及其具体做法.

元素  $a_{ii}^{(i-1)}$  ( $i = 1, 2, \cdots, n$ ) 称为**主元素**.

在消元过程中, 我们总是假定  $a_{11}^{(0)} \neq 0, a_{22}^{(1)} \neq 0, \cdots$ . 事实上, 如果  $A$  是非奇异矩阵, 则  $A^{(0)}$  的第 1 列至少有一个非零元素. 因此, 如果  $a_{11}^{(0)} = 0$ , 则只须先把非零元素所在的行与第 1 行交换, 即可保证  $a_{11}^{(0)} \neq 0$ . 同理, 第 1 次消元以后, 右下角的  $n-1$  阶矩阵也必非奇异, 因此, 如果  $a_{22}^{(1)} = 0$ , 也只须交换两行, 即可保证  $a_{22}^{(1)} \neq 0$ . 如此等等. 消元过程可以进行到底.

从数值计算的角度来看, 主元素要作为除数, 其绝对值越小, 引起的舍入误差就可能越大. 因此, 我们应该尽量选取绝对值大的元素作主元素. 许多文献都有具体例子说明这种选主元素的必要性, 这里我们从略. 但必须指出, 正是针对选主元素的不同技巧, 相应地提出了高斯消去法的各种不同方案. Wilkinson (1965) 就曾推荐了下述技巧, 他的推荐基于这样的事实, 即在这种情况下能得到误差界, 且可以证明计算是稳定的.

第一个技巧叫做选全主元, 即在消元过程的第  $k$  步上, 从  $A^{(k-1)}$  的后  $n-(k-1)$  行和列中, 选取绝对值最大的元素作主元素, 也即选取

$$|a_{ii}^{(k-1)}| = \max_{k \leq i, i \leq n} |a_{ii}^{(k-1)}|$$

然后将第  $k$  行与  $i$  行交换, 将第  $k$  列与第  $i$  列交换, 并且保留下交换信息, 以供后面调整解向量中分量的次序时使用.

第二个技巧叫做选列主元, 即在消元过程的第  $k$  步上, 从  $A^{(k-1)}$  的第  $k$  行  $k$  列以及第  $k$  行以下的元素中选取绝对值最大的元素, 也即

$$|a_{ik}^{(k-1)}| = \max_{k \leq i \leq n} |a_{ik}^{(k-1)}|$$

然后将第  $k$  行与第  $i$  行交换. 选列主元比选全主元的运算量小, 但一般可以满足精度要求, 所以选列主元更常被采用.

综上所述,我们可得下列定理.

化成三角形方程组以后,采用自后向前的回代过程,便很容易求得方程组的解.

这里,我们介绍采用选列主元技巧的高斯消去法解线性方程组的 FORTRAN 程序.

为程序设计和计算方便,我们记方程组(8.1)的形式为

$$\left\{ \begin{aligned} a_{11}^{(0)} x_1 + a_{12}^{(0)} x_2 + \dots + a_{1n}^{(0)} x_n &= a_{1, n+1}^{(0)} \\ a_{21}^{(0)} x_1 + a_{22}^{(0)} x_2 + \dots + a_{2n}^{(0)} x_n &= a_{2, n+1}^{(0)} \\ &\dots \\ a_{n1}^{(0)} x_1 + a_{n2}^{(0)} x_2 + \dots + a_{nn}^{(0)} x_n &= a_{n, n+1}^{(0)} \end{aligned} \right.$$

[illegible]

$$\begin{array}{ccccccc}
 1 & a_{12}^{(1)} & a_{13}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\
 & 1 & a_{23}^{(2)} & \dots & \dots & a_{2n}^{(2)} \\
 & & \ddots & \ddots & \ddots & \vdots \\
 & & & \ddots & \ddots & \vdots \\
 & & & & \ddots & \vdots \\
 & & & & & a_{n-1, n}^{(n-1)} \\
 & & & & & 1
 \end{array}$$

设消元过程已经进行到第  $k$  步 ( $k = 1, 2, \dots, n$ ), 在系数矩阵第  $k$  列的元素  $a_{k+1}^{(k-1)}, a_{k+2}^{(k-1)}, \dots, a_{nn}^{(k-1)}$  中选出绝对值最大者, 称为第  $k$  步主元素, 即

$$|a_{ik}^{(k-1)}| = \max_{k \leq i \leq n} |a_{ik}^{(k-1)}|$$

如果  $l \neq k$ , 我们将第  $l$  个方程与第  $k$  个方程互换位置, 即使新的  $a_{kk}^{(l-1)}$  变为主元, 然后再着手消元.

$$\begin{cases} a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)} & (j = k+1, k+2, \dots, n+1) \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)} & \begin{pmatrix} i = k+1, k+2, \dots, n \\ j = k+1, k+2, \dots, n, n+1 \end{pmatrix} \end{cases}$$

经过  $n$  次消元,即可把方程组化为上三角形方程组,然后进行回代求解.

回代过程计算公式是

$$\begin{cases} x_n = a_{n, n+1}^{(n)} \\ x_k = a_{k, k+1}^{(k)} - \sum_{j=k+1}^n a_{k, j}^{(k)} x_j \quad (k = n-1, n-2, \dots, 2, 1) \end{cases}$$

下面给出列主元高斯消去法的 FORTRAN 子程序<sup>[6]</sup>

GAUSS (N, N1, A, EPS)

其中哑元说明如下:

N 整型变量,输入参数;方程组的阶数  $n$ .

N1 整型变量,输入参数; N1 为方程组阶数加 1.

A  $N \times N1$  个元素的二维实型数组,输入输出参数;开始存放方程组的系数与右端项组成的增广矩阵,最后在第 N1 列存放方程组的解

EPS 实型变量,输入参数;主元素绝对值的最小允许值,如果主元素绝对值小于此值,则停机,打印 7777 作标志.

子程序如下:

```

SUBROUTINE GAUSS (N, N1, A, EPS)
  DIMENSION A (N, N1)
  DO 50 K = 1, N
    BMAX = 0.0
    DO 20 I = K, N
      IF (BMAX - ABS (A (I, K))) 10, 20, 20
10    BMAX = ABS (A (I, K))
      L = I
20    CONTINUE
    IF (BMAX .LT. EPS) STOP 7777
    IF (L .EQ. K) GO TO 30
    DO 25 J = K, N1
      T = A(L, J)
      A(L, J) = A(K, J)
25    A(K, J) = T
30    T = 1.0/A(K, K)
      K1 = K + 1
      DO 40 J = K1, N1
        A(K, J) = A(K, J)*T
      DO 40 I = K1, N
40    A(I, J) = A(I, J) - A(I, K)*A(K, J)
50    CONTINUE
      DO 60 IK = 2, N

```

```

I = N1 - IK
II = I + 1
DO 60 J = II, N
60 A(I, N1) = A(I, N1) - A(I, J) * A(J, N1)
RETURN
END

```

**算例** 用列主元高斯消去法解下列 6 阶线性代数方程组

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 1 & 1 & 0 \\ 1 & -0.1 & -1 & 0 & 0 & -1 \\ 0.1 & 0 & 0 & 0 & 0 & -0.4 \\ 0 & -0.85 & 0 & 0.75 & 1 & 0 \\ -0.6 & 0.085 & 0.7 & 0 & 0 & 0.6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 73 \\ 0 \\ 27 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

取  $\varepsilon = 10^{-5}$ ，初始数据分 6 行，每行 7 个（每行最后一个为右端项），由终端键盘输入，并在调用子程序之前打印输出，以供校对。

解题程序：

```

C      THE MAIN PROGRAM
      DIMENSION A(6,7)
      READ (5, 100) ((A(I, J), J = 1, 7), I = 1, 6)
100    FORMAT (7F10. 3)
      WRITE (6, 100) ((A(I, J), J = 1, 7), I = 1, 6)
      CALL GAUSS (6, 7, A, 1.0E - 5)
      WRITE (6, 200) (A(I, 7), I = 1, 6)
200    FORMAT (10X, E13.6)
      STOP
      END

      SUBROUTINE GAUSS (N, N1, A, EPS)
      :   (本子程序段体)
      END

```

打印输出的结果为(初始数据和解):

|        |        |        |       |       |        |        |
|--------|--------|--------|-------|-------|--------|--------|
| 0.000  | 0.000  | 0.000  | 0.000 | 1.000 | 1.000  | 73.000 |
| 0.000  | -1.000 | 0.000  | 1.000 | 1.000 | 0.000  | 0.000  |
| 1.000  | -0.100 | -1.000 | 0.000 | 0.000 | -1.000 | 27.000 |
| 0.100  | 0.000  | 0.000  | 0.000 | 0.000 | -0.400 | 0.000  |
| 0.000  | -0.850 | 0.000  | 0.750 | 1.000 | 0.000  | 0.000  |
| -0.600 | 0.085  | 0.700  | 0.000 | 0.000 | 0.600  | 0.000  |

$$0.246286E + 03$$

$$0.285714E + 02$$

$$0.154857E + 03$$

$$0.171429E + 02$$

$$0.114286E + 02$$

$$0.615714E + 02$$

我们指出,列主元高斯消去法程序可以改为更一般的形式,即求系数矩阵相同,右端常向量有 $m$ 个的方程组

$$AX = B$$

的解,其中 $A \in R^{n \times n}$  矩阵, $X, B \in R^{n \times m}$  矩阵, $m$ 为右端列向量的个数.

### 3. 高斯-约当消去法

上面所讲的高斯消去法,是先把方程组化为上三角形方程组,然后用回代过程求解.事实上,只要用消元过程多做些工作,就可以把方程组化为对角形,这时,求解就不需要回代过程了.特别是,还可以把方程组的系数矩阵化为单位矩阵,这时,方程组的解就在右端向量了.

例如,设选主元和作必要的行列交换如同上面所说的消去法一样,对消元过程来说,在第 $k$ 次消元时,不仅把第 $(k, k)$ 位置上的元素化为1,以及该列中 $(k, k)$ 位置以下的元素均化为零,而且同时把 $(k, k)$ 位置以上的元素也化为零.即设方程组的增广矩阵为

$$\left[ \begin{array}{cccc|c} * & * & \cdots & * & * \\ * & * & \cdots & * & * \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ * & * & \cdots & * & * \end{array} \right]$$

经第1次消元后化为

$$\left[ \begin{array}{cccc|c} 1 & * & \cdots & * & * \\ * & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * & * \end{array} \right]$$

经第2次消元后化为

$$\left[ \begin{array}{cccc|c} 1 & * & \cdots & * & * \\ & 1 & * & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ & * & \cdots & * & * \end{array} \right]$$

依此类推,经 $n$ 次消元后增广矩阵化为

$$\left[ \begin{array}{cccc|c} 1 & & & & * \\ & 1 & & & * \\ & & \ddots & & \vdots \\ & & & 1 & * \end{array} \right]$$

这时,显然可以直接写出方程组的解.这种无回代过程的消元法称为高斯-约当(Jordan)

消去法.

设方程组为

$$Ax = b$$

用高斯-约当消去法求解方程组的整个计算过程仍由  $n$  步组成. 令  $A \equiv A^{(0)}$ ,  $A^{(k-1)}$  为第  $k$  步 ( $k = 1, 2, \dots, n$ ) 开始时的矩阵,  $a_{ij}^{(k-1)}$  为  $A^{(k-1)}$  的第  $i$  行第  $j$  列的元素. 假定我们已经进行了  $k-1$  步, 得到矩阵  $A^{(k-1)}$ , 它的形式为

$$A^{(k-1)} = \begin{bmatrix} 1 & 0 & \cdots & 0 & a_{1k}^{(k-1)} & \cdots & a_{1n}^{(k-1)} \\ & 1 & \cdots & 0 & a_{2k}^{(k-1)} & \cdots & a_{2n}^{(k-1)} \\ & & \ddots & \vdots & \vdots & \ddots & \vdots \\ & & & 1 & a_{k-1,k}^{(k-1)} & \cdots & a_{k-1,n}^{(k-1)} \\ & & & & a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ & & & & \vdots & \ddots & \vdots \\ & & & & a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{bmatrix}$$

其右端向量为

$$b^{(k-1)} = \begin{bmatrix} b_1^{(k-1)} \\ b_2^{(k-1)} \\ \vdots \\ b_n^{(k-1)} \end{bmatrix}$$

进行第  $k$  步消元得到的矩阵  $A^{(k)}$  和右端向量  $b^{(k)}$  的元素由下列公式计算

$$\begin{cases} a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)} & (j = 1, 2, \dots, n) \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k-1)} / a_{kk}^{(k-1)} & (i \neq k, i, j = 1, 2, \dots, n) \\ b_k^{(k)} = b_k^{(k-1)} / a_{kk}^{(k-1)} \\ b_i^{(k)} = b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k-1)} / a_{kk}^{(k-1)} & (i \neq k, i = 1, 2, \dots, n) \end{cases} \quad (8.15)$$

按这步骤进行到  $k = n$ , 则矩阵  $A^{(n)} = I$  为单位矩阵, 而这时  $b^{(n)}$  即为线性方程组的解.

解线性代数方程组的高斯-约当消去法, 很适于用来求满秩矩阵的逆阵和行列式之值. 其原理是, 设  $A$  非奇异(满秩), 则  $A^{-1}$  存在. 记  $A^{-1}$  为

$$A^{-1} = X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

则求  $A$  的逆阵  $A^{-1}$  等价于求  $n$  阶矩阵  $X$  使

$$AX = I_n$$

其中  $I_n$  为单位矩阵. 把  $X$  和  $I_n$  分块

$$X = [x_1, x_2, \dots, x_n], \quad I_n = [e_1, e_2, \dots, e_n]$$

则求解  $AX = I_n$  又等价于求解系数矩阵相同的  $n$  个方程组

$$Ax_j = e_j \quad (j = 1, 2, \dots, n)$$

即

$$Ax_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad Ax_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots, \quad Ax_s = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

我们可用上述高斯-约当消去法(或再加上选主元技巧)求解这方程组.

我们注意到,对于第  $k$  组方程

$$Ax_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

用上述高斯-约当消去法求解时,在第  $k$  步之前,其右端项是不变的,而在第  $k$  步其右端项变成

$$b_k^{(k)} = \begin{bmatrix} -a_{1k}^{(k-1)} / a_{kk}^{(k-1)} \\ \vdots \\ -a_{k-1,k}^{(k-1)} / a_{kk}^{(k-1)} \\ 1 / a_{kk}^{(k-1)} \\ -a_{k+1,k}^{(k-1)} / a_{kk}^{(k-1)} \\ \vdots \\ -a_{nk}^{(k-1)} / a_{kk}^{(k-1)} \end{bmatrix}$$

并且在以后的各步参加高斯-约当消去法的运算. 另外,我们还可注意到,从第  $k+1$  步开始,在进行消去时,  $A^{(k)}$  的前  $k$  列是不变的,而且对后面的运算不起作用. 因此,我们可以将列向量  $b_k^{(k)}$  放在方阵  $A^{(k)}$  的第  $k$  列的位置上,使求逆运算仅在方阵  $A$  的位置上进行(叫原地踏步). 为此,将计算公式(8.15)改为

$$\begin{cases} a_{kk}^{(k)} = 1 / a_{kk}^{(k-1)} \\ a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)} \quad (j \neq k) \\ a_{ik}^{(k)} = -a_{ik}^{(k-1)} / a_{kk}^{(k-1)} \quad (i \neq k) \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k-1)} / a_{kk}^{(k-1)} \quad (i, j \neq k) \end{cases} \quad (8.16)$$

则可得

$$A^{(n)} = A^{-1}$$

至于  $A$  的行列式之值  $\det A$ , 可由化为三角形矩阵的主对角线元素的乘积得到.

#### 4. 行主元高斯-约当消去法求逆矩阵与行列式值的 FORTRAN 程序

行主元高斯-约当消去法求  $n$  阶非奇异矩阵  $A$  的逆矩阵  $A^{-1}$  及其行列式值  $\det A$ , 这是直接法求逆中最基本的一种方法. 下面介绍这种方法, 并从另一途径导出计算公式(8.16). 这种方法的运算是在增广矩阵



$$[A, I] = \left[ \begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1n} & 1 & & & \\ a_{21} & a_{22} & \cdots & a_{2n} & & 1 & & \\ \cdots & \cdots & \cdots & \cdots & & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} & & & & 1 \end{array} \right]$$

上进行的,但其中  $n$  阶单位矩阵  $I$  只是形式上的表示,计算机程序实现时,并不实际占用存储单元。

具体的运算过程是对  $[A, I]$  逐次作变换,使得  $[A, I]$  最后变成  $[I, A^{-1}]$ 。为此,假设作如下所示的变换:

$$\begin{aligned} D &= [A, I] \\ &= I[A, I] & (I = V^{(0)}, [A, I] = W^{(0)}) \\ &= V^{(0)}W^{(0)} \\ &= V^{(1)}W^{(1)} \\ &= \cdots \cdots \\ &= V^{(n-1)}W^{(n-1)} \\ &= V^{(n)}W^{(n)} & (V^{(n)} = A, W^{(n)} = [I, A^{-1}]) \\ &= A[I, A^{-1}] \end{aligned}$$

其中变换的每一步,就是将  $V^{(k)}$  的单位列向量换为  $A$  相应的一列,即设经过  $k-1$  次变换已有

$$\begin{aligned} D &= V^{(k-1)}W^{(k-1)} \\ &= [A_1, A_2, \cdots, A_{k-1}, I_k, I_{k+1}, \cdots, I_n]W^{(k-1)} \\ &= [A_1, A_2, \cdots, A_{k-1}, I_k, I_{k+1}, \cdots, I_n] \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,2n} \\ W_{21} & W_{22} & \cdots & W_{2,2n} \\ \cdots & \cdots & \cdots & \cdots \\ W_{n1} & W_{n2} & \cdots & W_{n,2n} \end{bmatrix} \end{aligned}$$

其中  $A_1, A_2, \cdots, A_{k-1}$  分别表示  $A$  的第  $1, 2, \cdots, k-1$  列;  $I_k, I_{k+1}, \cdots, I_n$  分别表示单位矩阵  $I$  的第  $k, k+1, \cdots, n$  列。对比上式两边可得  $D$  的第  $i$  列元素为

$$\begin{aligned} D_i &= \sum_{j=1}^{k-1} A_j W_{ji} + \sum_{j=k}^n I_j W_{ji} \\ &= \sum_{j=1}^{k-1} A_j W_{ji} + I_k W_{ki} + \sum_{j=k+1}^n I_j W_{ji} \end{aligned} \quad (8.17)$$

特别地,我们注意到由  $D = [A, I]$  有  $D_k = A_k$ , 因而

$$A_k = \sum_{j=1}^{k-1} A_j W_{jk} + I_k W_{kk} + \sum_{j=k+1}^n I_j W_{jk}$$

故当  $W_{kk} \neq 0$  时,可求出

$$I_k = \left( A_k - \sum_{j=1}^{k-1} A_j W_{jk} - \sum_{j=k+1}^n I_j W_{jk} \right) / W_{kk}$$

代入(8.17)有

$$D_j = \sum_{i=1}^{k-1} A_i(W_{ij} - W_{ik}W_{ki}/W_{kk}) + A_k(W_{ki}/W_{kk}) + \sum_{i=k+1}^n I_i(W_{ij} - W_{ik}W_{ki}/W_{kk}) \quad (8.18)$$

对比(8.17)与(8.18), 注意  $I_k$  被换为  $A_k$ , 于是可得第  $k$  步将  $W^{(k-1)}$  变换为  $W^{(k)}$  时, 对所有的  $j$ , 相应元素的变换公式为

$$\begin{cases} \tilde{W}_{ji} = W_{ji} - W_{ik} \cdot W_{ki}/W_{kk} & (i \neq k) \\ \tilde{W}_{ki} = W_{ki}/W_{kk} \end{cases} \quad (8.19)$$

这样, 利用上述变换公式, 即可逐次将  $W^{(0)} = [A, I]$  变换到  $W^{(n)} = [I, A^{-1}]$ , 从而求得  $A^{-1}$ .

为了处理  $W_{kk} = 0$  的情况, 我们在变换过程中选取行主元素  $W_{k, i_0}$  代替  $W_{kk}$ , 即

$$|W_{k, i_0}| = \max_{k \leq i \leq n} \{|W_{ki}|\}$$

并用一个整型数组 ME 来记录主元的列号:

$$ME(K) = j_0$$

以便在最后对逆矩阵作列交换, 恢复原列次序.

按公式(8.19), 对  $[A, I]$  的具体运算如下图所示

$$[A, I] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \boxed{1} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & \boxed{1/a_{11}} & 0 & \cdots & 0 \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & -a_{21}/a_{11} & 1 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & -a_{n1}/a_{11} & 0 & \cdots & 1 \end{bmatrix} \Rightarrow \cdots$$

其中把处于  $I_k$  列位置上的元素放到  $A_k$  列位置上去(如框框和箭头所示), 并在下一步变换中参与矩阵  $A$  的下一次消去运算, 因而整个运算可以仅在方阵  $A$  的位置上进行. 在计算过程中, 若记

$$a_{ij} = a_{ij}^{(0)} \quad (i, j = 1, 2, \cdots, n)$$

则由变换公式(8.19)可得对于  $k = 1, 2, \cdots, n$  的递推计算公式为(对照(8.16)式)

$$\begin{cases} a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} \cdot a_{kj}^{(k-1)} / a_{kk}^{(k-1)} & (i, j \neq k) \\ a_{kk}^{(k)} = 1 / a_{kk}^{(k-1)} \\ a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)} & (j \neq k) \\ a_{ik}^{(k)} = -a_{ik}^{(k-1)} / a_{kk}^{(k-1)} & (i \neq k) \end{cases}$$

在程序设计中, 用实型数组 B(J) 和 C(I) 分别记  $a_{kj}^{(k)} (j \neq k)$  和  $a_{ik}^{(k)} (i \neq k)$ .

另外, 由高斯-约当消去法可知,  $A$  的行列式值, 可由作为主元素的  $a_{kk}^{(k-1)}$  (对应于

$W_{kk}$ ) 的连乘积得到, 即

$$\det A = \prod_{k=1}^n a_{kk}^{(k-1)}$$

下面给出用主元素高斯-约旦消去法求  $n$  阶矩阵  $A$  的逆矩阵和行列式值的 FORTRAN 程序<sup>[3]</sup>。子程序语句为

```
SUBROUTINE GJIVDE (N, A, DET, EPS, ME, B, C)
```

哑元说明:

N 整型变量, 矩阵  $A$  的阶数  $n$ 。

A  $N \times N$  个元素的二维实型数组, 开始存放矩阵  $A$ , 最后存放逆矩阵  $A^{-1}$ 。

DET 实型变量, 存放行列式  $\det A$  之值。

EPS 实型变量, 行主元的控制常数, 若行主元的绝对值小于此值, 认为  $A$  奇异, 停机, 打印 STOP 7777。

ME, B, C 分别为  $N$  个元素的一维整型 (ME) 和实型 (B, C) 工作数组。

FORTRAN 子程序:

```
SUBROUTINE GJIVDE (N, A, DET, EPS, ME, B, C)
  DIMENSION A(N, N), ME(N), B(N), C(N)
  DET = 1.0
  DO 10 J = 1, N
10  ME(J) = J
    DO 20 I = 1, N
      Y = 0.0
      DO 30 J = 1, N
        IF (ABS(A(I, J)) .LE. ABS(Y)) GO TO 30
        K = J
        Y = A(I, J)
30  CONTINUE
      DET = DET * Y
      IF (ABS(Y) .LT. EPS) STOP 7777
      Y = 1.0/Y
      DO 40 J = 1, N
        C(J) = A(J, K)
        A(J, K) = A(J, I)
        A(J, I) = -C(J) * Y
        B(J) = A(I, J) * Y
40  A(I, J) = A(I, J) * Y
      A(I, J) = Y
      J = ME(I)
      ME(I) = ME(K)
```

```

      ME(K) = J
      DO 11 K = 1, N
      IF (K .EQ. I) GO TO 11
      DO 12 J = 1, N
      IF (J .EQ. I) GO TO 12
      A(K, J) = A(K, J) - B(J)*C(K)
12  CONTINUE
11  CONTINUE
20  CONTINUE
      DO 33 I = 1, N
      DO 44 K = 1, N
      IF (ME(K) .EQ. I) GO TO 55
44  CONTINUE
55  IF(K .EQ. I) GO TO 33
      DO 66 J = 1, N
      W = A(I, J)
      A(I, J) = A(K, J)
66  A(K, J) = W
      IW = ME(I)
      ME(I) = ME(K)
      ME(K) = IW
      DET = - DET
33  CONTINUE
      RETURN
      END

```

**算例** 求下列矩阵

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

的逆矩阵  $A^{-1}$  及其行列式值  $\det A$ 。

解题程序及计算结果如下:

```

C      THE MAIN PROGRAM
      DIMENSION A(5,5), ME(5), B(5), C(5)
      READ (5,*) A
      WRITE (6, 100)

```

```

100 FORMAT (12X, 'MATRIX A')
    WRITE (6, '(5 (2X, F10.3))') A
    CALL GJIVDE (5, A, DE, 1.0E-5, ME, B, C)
    WRITE (6, 110)
110 FORMAT (12X, 'INVERT OF MATRIX A')
    WRITE (6, '(5 (2X, F10.3))') A
    WRITE (6, 120) DET
120 FORMAT (12X, 'DET=', F10.3)
    STOP
    END

```

C

```

SUBROUTINE GJIVDE (N, A, DET, EPS, ME, B, C)
:   (本子程序段体)
END

```

#### MATRIX A

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| 5.000 | 1.000 | 2.000 | 3.000 | 4.000 |
| 1.000 | 5.000 | 3.000 | 4.000 | 1.000 |
| 2.000 | 2.000 | 5.000 | 1.000 | 2.000 |
| 3.000 | 3.000 | 4.000 | 5.000 | 3.000 |
| 4.000 | 4.000 | 1.000 | 2.000 | 5.000 |

#### INVSU OF MATRIX A

|        |        |        |        |        |
|--------|--------|--------|--------|--------|
| 0.733  | 0.483  | 0.055  | -0.731 | -0.267 |
| 0.067  | 0.317  | 0.031  | -0.326 | 0.067  |
| -0.067 | -0.067 | 0.219  | 0.076  | -0.067 |
| -0.067 | -0.067 | -0.210 | 0.362  | -0.067 |
| -0.600 | -0.600 | -0.029 | 0.686  | 0.400  |

DET = 420.00

## 8-3 解线性方程组的三角分解法

### 1. 矩阵三角分解原理

我们从矩阵运算的角度来考察高斯消去法的消元过程。事实上，如果假定  $a_{11}^{(0)} \neq 0$ ，即消元过程无需作行交换，那么，高斯消去法的每一步消元，相当于对系数矩阵  $A$  作一次初等变换。例如，第一步，从第 2 至第  $n$  个方程消去  $x_1$  的系数，就相当于用初等变换矩阵

$$L_1 = \begin{bmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ \vdots & & \ddots & & \\ -m_{n1} & & & \ddots & 1 \end{bmatrix}$$

左乘矩阵  $A = A^{(0)}$ ，即

$$\begin{bmatrix} 1 & & & \\ -m_{21} & 1 & & \\ \vdots & & \ddots & \\ -m_{n1} & & & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(1)} \end{bmatrix}$$

相仿地,第二步,从第3至第 $n$ 个方程消去 $x_2$ 的系数,就相当于用初等变换矩阵

$$L_2 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -m_{32} & 1 & \\ & \vdots & & \ddots \\ & -m_{n2} & & & 1 \end{bmatrix}$$

左乘前一次的结果。如此等等,即在主对角线元素 $a_{kk}^{(k-1)} \neq 0$  ( $k = 1, 2, \dots, n-1$ )的情况下,经过 $n-1$ 步消元, $A$ 可化为上三角矩阵 $U$

$$L_{n-1}L_{n-2} \cdots L_2L_1A = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix} = U \quad (8.20)$$

一般地,形如

$$L_k = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & -m_{k+1,k} & 1 \\ & & \vdots & & \ddots \\ & & -m_{n,k} & & & 1 \end{bmatrix} \quad (\text{第 } k \text{ 列})$$

的单位下三角矩阵通常称为 Frobenius 矩阵,可以验证,它具有下列两个简单性质:

1)  $L_k$  的逆矩阵存在,且

$$L_k^{-1} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & m_{k+1,k} & 1 \\ & & \vdots & & \ddots \\ & & m_{n,k} & & & 1 \end{bmatrix} \quad (8.21)$$

2)

$$L_1^{-1}L_2^{-1} = \begin{bmatrix} 1 & & & \\ m_{21} & 1 & & \\ m_{31} & m_{32} & 1 & \\ \vdots & \vdots & & \ddots \\ m_{n1} & m_{n2} & & & 1 \end{bmatrix}$$

以及

$$L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{n, n-1} & 1 \end{bmatrix} \quad (8.22)$$

由此, 对 (8.20) 式两端依次左乘  $L_{n-1}^{-1}, L_{n-2}^{-1}, \cdots, L_2^{-1}, L_1^{-1}$ , 即可得

$$A = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} U = LU \quad (8.23)$$

其中

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$$

为 (8.22) 所示的单位下三角矩阵.

一般地, 把一个矩阵  $A$  分解成一个下三角矩阵  $L$  和一个上三角矩阵  $U$  的乘积

$$A = LU \quad (8.24)$$

称为矩阵  $A$  的三角分解或  $LU$  分解.

$LU$  分解的更标准表示是

$$A = LDU \quad (8.25)$$

其中  $D$  为一非奇异对角矩阵,  $L$  和  $U$  分别为单位下三角矩阵和单位上三角矩阵.

如式 (8.23) 所示把  $A$  分解为一个单位下三角矩阵  $L$  和一个上三角矩阵  $U$  (相当于 (8.25) 中的  $DU$ ) 的三角分解通常称为杜利特尔 (Doolittle) 分解. 如果把  $A$  分解为下三角矩阵  $L$  (相当于 (8.25) 中的  $LD$ ) 和单位上三角矩阵  $U$  的乘积, 这时称为克劳特 (Crout) 分解.

由上所述, 可知对于任何非奇异矩阵  $A \in R^{n \times n}$  (包括经过必要的行交换), 都存在有三角分解  $A = LU$ ; 而且因为  $A$  非奇异, 有  $\det A = \det L \cdot \det U$ , 所以  $L$  和  $U$  也都为非奇异.

现在的问题是, 虽然任何非奇异矩阵 (包括经过必要的行交换) 都存在  $LU$  分解, 但是, 按自然顺序, 有时就作不出上述的  $LU$  分解. 例如, 矩阵  $A = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$  就是这样. 因此, 我们要研究, 在什么条件下, 可以保证非奇异矩阵  $A$  不经行交换而按自然顺序就可以作出  $LU$  分解. 因为如果我们知道  $A$  有这种分解, 就可以不必象高斯消去法那样, 先构造一系列中间矩阵  $L_k$ , 而可以设法直接把这种分解求出来, 下述定理解决了这个问题.

**定理** 设  $A \in R^{n \times n}$ , 令

$$A_1 = [a_{11}], \quad A_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad \cdots, \quad A_k = \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \cdots & \cdots & \cdots \\ a_{k1} & \cdots & a_{kk} \end{bmatrix}, \quad \cdots, \quad A_n = A$$

$A_k (k = 1, 2, \cdots, n)$  称为矩阵  $A$  的各阶主子矩阵, 则矩阵  $A$  有唯一的  $LDU$  分解的充分必要条件是:  $A_k (k = 1, 2, \cdots, n)$  均为非奇异矩阵.

**证** 先证若  $A$  有  $LDU$  分解, 则此分解必为唯一. 设  $A$  有两种  $LDU$  分解

$$A = LDU = \tilde{L}\tilde{D}\tilde{U}$$

因为  $A$  非奇异, 所以  $LDU$  和  $L, D, U$  和  $\tilde{L}, \tilde{D}, \tilde{U}$  均为非奇异. 从而

$$\tilde{L}^{-1}L = \tilde{D}\tilde{U}U^{-1}D^{-1}$$

左端是一个单位下三角矩阵,而右端是一个上三角矩阵,因此,它们只能是  $n \times n$  阶单位矩阵  $I$ ,故由  $\tilde{L}^{-1}L = I$  得  $L = \tilde{L}$ . 同理可证  $U = \tilde{U}$ , 以及最后由于  $L, \tilde{L}, U, \tilde{U}$  都是非奇异,从而由  $LDU = \tilde{L}\tilde{D}\tilde{U}$  可知  $D = \tilde{D}$ .

现在证明充分性: 设  $A$  的顺序主子矩阵  $A_k (k = 1, 2, \dots, n)$  都非奇异, 则  $A$  必存在  $LDU$  分解. 显然,  $A_1$  有  $L_1 D_1 U_1$  分解. 今设  $A_k$  有  $L_k D_k U_k$  分解, 令

$$\begin{aligned} A_{k+1} = \begin{bmatrix} A_k & s \\ r^T & a \end{bmatrix} &= \begin{bmatrix} L_k & 0 \\ u^T & 1 \end{bmatrix} \begin{bmatrix} D_k & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} U_k & v \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} L_k D_k U_k & L_k D_k v \\ u^T D_k U_k & u^T D_k v + d \end{bmatrix} \end{aligned}$$

其中  $r, s, a$  均为已知. 故得

$$u^T D_k U_k = r^T, \quad L_k D_k v = s, \quad u^T D_k v + d = a$$

从而可求出唯一的  $u, v, d$ :

$$u = D_k^{-1} (U_k^T)^{-1} r, \quad v = D_k^{-1} L_k^{-1} s, \quad d = a - u^T D_k v$$

可知,  $A_{k+1}$  也必存在  $LDU$  分解. 于是, 按归纳法原理,  $A_{k+2}, A_{k+3}, \dots, A_n = A$  均有  $LDU$  分解存在.

最后证明必要性: 设  $A$  有  $LDU$  分解, 则  $L, D, U$  均为非奇异. 由于  $A_k = L_k D_k U_k$ ,  $L_k, D_k, U_k$  分别表示  $L, D, U$  的  $k$  阶主子矩阵, 而  $L, U$  为单位三角矩阵,  $D$  为非奇异对角矩阵, 所以  $L_k, D_k$  和  $U_k$  都非奇异, 从而可知  $A_k$  必为非奇异.

定理至此证毕.

下面我们讨论三角分解在解线性方程组中的应用.

## 2. 解线性方程组的三角分解法及其 FORTRAN 程序

矩阵  $A$  的  $LU$  分解的实用意义在于, 如果我们能直接从  $A$  的元素计算出  $L, U$  的元素, 而不需要任何中间步骤, 那么, 求解方程组

$$Ax = b$$

即变成求解方程组

$$LUx = b$$

从而, 也就等价于求解两个三角形方程组

$$Ly = b$$

和

$$Ux = y$$

于是我们可

1) 由  $Ly = b$ , 求出  $y$ ;

2) 由  $Ux = y$ , 求出  $x$ .

我们知道, 求解三角形方程组是很容易的.

下面, 我们设  $A$  的各阶主子矩阵均非奇异, 即存在三角分解  $A = LU$ , 我们来推导由  $A$  的元素经  $n$  步直接计算出  $L$  和  $U$  的公式, 从而建立解线性方程组  $Ax = b$  的计算方法.

先考虑杜利特尔分解, 设  $A = LU$  为



$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \\ & & & u_{nn} \end{bmatrix} \quad (8.26)$$

显然,由矩阵乘法规则,对比(8.26)式两边即有

$$a_{1i} = u_{1i} \quad (i = 1, 2, \cdots, n)$$

$$a_{i1} = l_{i1}u_{11} \quad (i = 2, 3, \cdots, n)$$

于是可得计算 $U$ 的第1行元素和 $L$ 的第1列元素的公式

$$\begin{cases} u_{1i} = a_{1i} & (i = 1, 2, \cdots, n) \\ l_{i1} = a_{i1}/u_{11} & (i = 2, 3, \cdots, n) \end{cases} \quad (8.27)$$

设已算出 $U$ 的前 $r-1$ 行元素和 $L$ 的前 $r-1$ 列元素,则对于 $i = r, r+1, \cdots, n$ ,

由

$$\begin{aligned} a_{ri} &= \sum_{k=1}^r l_{rk}u_{ki} = \sum_{k=1}^{r-1} l_{rk}u_{ki} + u_{ri} \\ a_{ir} &= \sum_{k=1}^r l_{ik}u_{kr} = \sum_{k=1}^{r-1} l_{ik}u_{kr} + l_{ir}u_{rr} \end{aligned}$$

又可得计算 $U$ 的第 $r$ 行元素和 $L$ 的第 $r$ 列元素的公式

$$\begin{cases} u_{ri} = a_{ri} - \sum_{k=1}^{r-1} l_{rk}u_{ki} & (i = r, r+1, \cdots, n) \\ l_{ir} = (a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr})/u_{rr} & (i = r+1, r+2, \cdots, n) \end{cases} \quad (8.28)$$

利用这些计算公式,我们便可得用直接三角分解法解线性方程组 $Ax = b$ 的计算方法如下:

- 1) 按公式(8.27)计算 $u_{1i}(i = 1, 2, \cdots, n)$ 和 $l_{i1}(i = 2, 3, \cdots, n)$ ;
- 2) 对于 $r = 2, 3, \cdots, n$ ,按公式(8.28)计算 $u_{ri}(i = r, r+1, \cdots, n)$ 和 $l_{ir}(i = r+1, r+2, \cdots, n)$ ;
- 3) 求解 $Ly = b$ ,即计算

$$\begin{cases} y_1 = b_1 \\ y_i = b_i - \sum_{k=1}^{i-1} l_{ik}y_k & (i = 2, 3, \cdots, n) \end{cases}$$

- 4) 求解 $Ux = y$ ,即计算

$$\begin{cases} x_n = y_n/u_{nn} \\ x_i = (y_i - \sum_{k=i+1}^n u_{ik}x_k)/u_{ii} & (i = n-1, \cdots, 2, 1) \end{cases}$$

注意到 $L, U$ 的结构及其元素的计算次序,可知当计算出 $u_{ri}$ 以后 $a_{ri}$ 就不用了;同理,当计算出 $l_{ir}$ 以后 $a_{ir}$ 也不用了.因此,设计计算机算法时,可以在 $u_{ri}$ 计算出来后就以 $u_{ri}$ 冲掉 $a_{ri}$ ,当 $l_{ir}$ 计算出来后就以 $l_{ir}$ 冲掉 $a_{ir}$ ,即 $L, U$ 的元素仍放在 $A$ 的相应位置,如下所示(以四阶为例):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{bmatrix}$$

此外,在计算公式中,需要计算形如  $\sum a_i b_i$  的和式,这时可采用“双精度累加”运算,以提高精度。

还有,如果已经实现了  $A = LU$  的分解计算,且  $L, U$  保存在  $A$  的相应位置,则用直接三角分解法解具有相同系数矩阵的方程组

$$Ax = b_k \quad (k = 1, 2, \dots, m)$$

是很方便的。每多解一个方程组只需多解两个三角形矩阵方程组。

现在考虑克劳特分解。

设  $A = LU$  为

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ & 1 & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

类似于上述的杜利特尔分解的推导,这里可得

$$\begin{cases} l_{i1} = a_{i1} & (i = 1, 2, \dots, n) \\ u_{iv} = a_{iv}/l_{i1} & (i = 2, 3, \dots, n) \end{cases} \quad (8.27)'$$

并且当  $L$  的前  $r-1$  列元素和  $U$  的前  $r-1$  行元素已经算出以后,对于  $i = r, r+1, \dots, n$  有

$$\begin{cases} l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik} u_{kr} & (i = r, r+1, \dots, n) \\ u_{ri} = \left( a_{ri} - \sum_{k=1}^{r-1} l_{rk} u_{ki} \right) / l_{rr} & (i = r+1, r+2, \dots, n) \end{cases} \quad (8.28)'$$

于是解线性方程组  $Ax = b$  的计算方法如下:

- 1) 按公式 (8.27)' 计算  $l_{i1} (i = 1, 2, \dots, n)$  和  $u_{iv} (i = 2, 3, \dots, n)$ ;
- 2) 对于  $r = 2, 3, \dots, n$ , 按公式 (8.28)' 计算  $l_{ir} (i = r, r+1, \dots, n)$  和  $u_{ri} (i = r+1, r+2, \dots, n)$ ;
- 3) 求解  $Ly = b$ , 即计算

$$\begin{cases} y_1 = b_1/l_{11} \\ y_i = \left( b_i - \sum_{k=1}^{i-1} l_{ik} y_k \right) / l_{ii} & (i = 2, 3, \dots, n) \end{cases}$$

- 4) 求解  $Ux = y$ , 即计算

$$\begin{cases} x_n = y_n \\ x_i = y_i - \sum_{k=i+1}^n u_{ik} x_k & (i = n-1, n-2, \dots, 2, 1) \end{cases}$$

下面, 我们给出用上述计算方法求解系数矩阵相同, 而右端常向量不同的  $k$  个线性方程组的 FORTRAN 程序。为了程序编码方便, 我们把公式 (8.28)' 的下标稍作改动如下:

$$\begin{cases} l_{ik} = a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk} & (i = k, k+1, \dots, n) \\ u_{ki} = \left( a_{ki} - \sum_{m=1}^{k-1} l_{km} u_{mi} \right) / l_{kk} & (i = k+1, k+2, \dots, 2, 1) \end{cases}$$

子程序语句为

SUBROUTINE CROUT (N, M, A)

其中哑元:

- N 整型变量, 方程组的阶数  $n$ 。
- M 整型变量,  $M = N + k$ ,  $k$  为系数矩阵相同、右端常向量不同的方程组个数。
- A  $N * M$  个元素的二维实型数组。开始存放按列排列的由系数矩阵及  $k$  个右端项所组成的增广矩阵; 程序结束时, A 的前  $N$  列为  $L$  和  $U$  矩阵 ( $U$  缺对角线), 后  $k$  列为解向量。

FORTRAN 子程序:

```

SUBROUTINE CROUT (N, M, A)
  DIMENSION A(N, M)
  DO 11 I = 2, N
11  A(1, I) = A(1, I)/A(1, 1)
    DO 3 K = 2, N
      DO 2 IK = K, N
        DO 2 MI = 2, K
          2  A(IK, K) = A(IK, K) - A(IK, MI - 1)*A(MI - 1, K)
          JI = K + 1
          IF (JI - N) 6, 6, 9
        6  DO 3 J = JI, N
          DO 4 MJ = 2, K
            4  A(K, J) = A(K, J) - A(K, MJ - 1)*A(MJ - 1, J)
          3  A(K, J) = A(K, J)/A(K, K)
        9  II = N + 1
          DO 5 I = II, M
            5  A(1, I) = A(1, I)/A(1, 1)
            DO 7 JJ = II, M
              DO 7 L = 2, N
                DO 8 KL = 2, L
                  8  A(L, JJ) = A(L, JJ) - A(L, KL - 1)*A(KL - 1, JJ)
                7  A(L, JJ) = A(L, JJ)/A(L, L)

```

```

DO 10 J1 = 11, M
DO 10 K1 = 2, N
K2 = N - K1 + 2
K4 = N - K1 + 1
DO 10 K3 = K2, N
10 A(K4, J1) = A(K4, J1) - A(K4, K3)*A(K3, J1)
RETURN
END

```

**算例** 求解线性方程组

$$\begin{bmatrix} 1 & -1 & 1 \\ 5 & -4 & 3 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} = \begin{bmatrix} -4 & 15 \\ -12 & 56 \\ 11 & 13 \end{bmatrix}$$

解题程序与计算结果:

```

DIMENSION A(3, 5)
READ (5, 100) ((A(I, J), J = 1, 5), I = 1, 3)
100 FORMAT (5F0.0)
CALL CROUT (3, 5, A)
WRITE (6, 200) ((A(I, J), I = 1, 3), J = 4, 5)
200 FORMAT (1X, 6HX1(I) = , 3F10.5/6HX2(I) = , 3F10.5)
STOP
END

```

C

```

SUBROUTINE CROUT (N, M, A)
: <本子程序段体>
END

```

```

X1(I) = 3.00000    6.00000   -1.00000
X2(I) = 4.00000   -3.00000    8.00000

```

### 3. 解对称正定矩阵方程组的平方根法

实际计算中的一些线性方程组, 其系数矩阵具有对称正定的性质. 直接三角分解法用于解这类方程组, 可以呈现出更简单的形式, 这就是所谓平方根法.

先复习一下对称正定矩阵的一些有关概念.

一个对称矩阵  $A \in R^{n \times n}$ , 如果对任意非零向量  $x \in R^n$  都有

$$x^T A x > 0$$

则矩阵  $A$  称为正定矩阵.

由矩阵代数可知

$$\mathbf{x}^T A \mathbf{x} = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j$$

其中  $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ ,  $A = [a_{ij}]$ , 这就是说,  $\mathbf{x}^T A \mathbf{x}$  是变量  $x_1, x_2, \dots, x_n$  的二次函数.

对称正定矩阵具有下列一些基本性质:

- 1) 如果  $A$  为  $n$  阶对称正定矩阵, 则  $A$  为非奇异矩阵, 且  $A^{-1}$  也是对称正定矩阵.
- 2) 如果  $A$  为  $n$  阶对称正定矩阵, 记  $A_k (k = 1, 2, \dots, n)$  为  $A$  的顺序主子矩阵, 则  $A_k$  也是正定矩阵, 且  $\det(A_k) > 0, k = 1, 2, \dots, n$ .
- 3) 如果  $A$  为对称正定矩阵, 则  $A$  的特征值  $\lambda_i > 0, i = 1, 2, \dots, n$ .

这些性质的证明可从线性代数中找到, 这里从略.

现在我们来讨论对称正定矩阵的三角分解.

设  $A \in R^{n \times n}$  是一个对称正定矩阵, 根据性质 2) 有  $\det(A_k) > 0 (k = 1, 2, \dots, n)$ , 即  $A_k$  非奇异. 因此, 由定理知  $A$  有唯一的  $LDU$  分解  $A = LDU$ . 又因  $A$  是对称的, 有

$$A = A^T = (LDU)^T = U^T D L^T$$

即有

$$LDU = U^T D L^T$$

可知这种分解的形式必然为

$$A = LDL^T$$

另一方面, 由于  $A$  正定, 故对非零向量  $\mathbf{x}$  有

$$\mathbf{x}^T A \mathbf{x} = \mathbf{x}^T LDL^T \mathbf{x} = (L^T \mathbf{x})^T D (L^T \mathbf{x}) > 0$$

只要取  $\mathbf{x}$  为方程组  $L^T \mathbf{x} = \mathbf{e}_i$  的解, 其中

$$\mathbf{e}_i = [0, \dots, 0, \underset{(x_i \text{ 列})}{1}, 0, \dots, 0]^T, \quad i = 1, 2, \dots, n$$

则上式有

$$\mathbf{x}^T A \mathbf{x} = \mathbf{e}_i^T D \mathbf{e}_i = d_i > 0, \quad i = 1, 2, \dots, n$$

于是可把对角矩阵  $D$  表示成

$$D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & & & \\ & \sqrt{d_2} & & \\ & & \ddots & \\ & & & \sqrt{d_n} \end{bmatrix} \begin{bmatrix} \sqrt{d_1} & & & \\ & \sqrt{d_2} & & \\ & & \ddots & \\ & & & \sqrt{d_n} \end{bmatrix} = D^{\frac{1}{2}} D^{\frac{1}{2}}$$

因而, 对称正定矩阵  $A$  可以分解为

$$A = LDL^T = LD^{\frac{1}{2}} D^{\frac{1}{2}} L^T = (LD^{\frac{1}{2}})(LD^{\frac{1}{2}})^T = L_1 L_1^T$$

或者仍记成

$$A = LL^T$$

其中  $L$  为下三角矩阵. 这种分解通常称为乔累斯基 (Cholesky) 分解.

现在我们用直接分解方法来确定下三角矩阵  $L$  的元素. 令

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ & l_{22} & \cdots & l_{2n} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{bmatrix}$$

其中  $l_{ii} > 0 (i = 1, 2, \dots, n)$ . 对比上式两边可得求  $L$  第 1 列元素的计算公式

$$l_{11} = \sqrt{a_{11}}, \quad l_{i1} = a_{i1}/l_{11} \quad (i = 2, 3, \dots, n) \quad (8.29)$$

同样,由

$$a_{ii} = \sum_{k=1}^n l_{ik}l_{ik} = \sum_{k=1}^{j-1} l_{ik}l_{ik} + l_{ij}l_{ij}$$

可得求  $L$  第  $j$  列元素 ( $j = 2, 3, \dots, n$ ) 的计算公式

$$\begin{cases} l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} & (j = 2, 3, \dots, n) \\ l_{ij} = \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk} \right) / l_{jj} & (j = 2, 3, \dots, n; i = j+1, j+2, \dots, n) \end{cases} \quad (8.30)$$

利用这些计算公式,我们便可得用直接三角分解法解对称正定线性方程组  $A\mathbf{x} = \mathbf{b}$  的计算方法,即平方根法:

- 1) 按公式 (8.29) 计算  $l_{11}$  和  $l_{i1}$  ( $i = 2, 3, \dots, n$ ).
- 2) 按公式 (8.30) 计算  $l_{ji}$  ( $j = 2, 3, \dots, n$ ) 和  $l_{ji}$  ( $j = 2, 3, n; i = j+1, j+2, \dots, n$ ).
- 3) 求解  $L\mathbf{y} = \mathbf{b}$ , 即计算

$$\begin{cases} y_1 = b_1 / l_{11} \\ y_i = \left( b_i - \sum_{k=1}^{i-1} l_{ik}y_k \right) / l_{ii} & (i = 2, 3, \dots, n) \end{cases}$$

- 4) 求解  $L^T\mathbf{x} = \mathbf{y}$ , 即计算

$$\begin{cases} x_n = y_n / l_{nn} \\ x_i = \left( y_i - \sum_{k=i+1}^n l_{ki}x_k \right) / l_{ii} & (i = n-1, n-2, \dots, 2, 1) \end{cases}$$

平方根法的一个重要特点是,由

$$a_{jj} = \sum_{k=1}^j l_{jk}^2 \quad (j = 1, 2, \dots, n)$$

有

$$|l_{jk}| \leq \sqrt{a_{jj}} \leq \max_{1 \leq j \leq n} \sqrt{a_{jj}} \quad (j, k = 1, 2, \dots, n)$$

可知在平方根法中,矩阵  $L$  的元素总是有界的,且  $l_{jj} > 0$  ( $j = 1, 2, \dots, n$ ). 这就可以说明,平方根法将是数值稳定的. 计算实践表明,平方根法解对称正定线性方程组具有较高的精度.

由于平方根法依据的是对称正定矩阵的分解  $A = LL^T$ , 所以只需计算出  $L$  的元素,这就比一般的直接  $LU$  分解减少了近一半的计算量. 而且由于  $A$  的对称性,因此在利用电子计算机计算时,只需用一维数组  $A[1:M]$  ( $M = \frac{n(n+1)}{2}$ ) 按行存储  $A$  的下三角部分的元素,且  $L$  的元素也只需存储在这个一维数组  $A[1:M]$  的相应位置.

#### 4. 改进平方根法及其 FORTRAN 程序

平方根法不理想的地方是在计算元素  $l_{ij}$  时要用到开方运算. 为了避免开方运算,人们改进了平方根法. 采用对称正定矩阵的  $A = LDL^T = TL^T$  分解,即令

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} 1 & l_{21} & \cdots & l_{n1} \\ & 1 & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

于是有  $a_{ii} = d_i$  和下列关系式

$$a_{ij} = \sum_{k=1}^n (LD)_{ik} (L^T)_{kj} = \sum_{k=1}^{j-1} l_{ik} d_k l_{jk} + l_{ij} d_j$$

由此可得计算  $L$  第  $i$  行元素和  $D$  主对角元素的计算公式

$$\begin{cases} d_1 = a_{11} \\ l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}) / d_j \quad (i = 2, 3, \dots, n) \\ d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 d_k \quad (i = 2, 3, \dots, n) \end{cases} \quad (8.31)$$

于是,利用这组公式,我们可得用直接三角分解法解对称正定线性方程组  $Ax = b$  的改进平方根法(或称  $LDL^T$  法):

1) 按公式(8.31)计算  $d_i, l_{ij}$  ( $i = 2, 3, \dots, n; j = 1, 2, \dots, i-1$ ),  $d_i$  ( $i = 2, 3, \dots, n$ );

2) 解方程  $Lz = b$ , 即计算

$$\begin{cases} z_1 = b_1 \\ z_i = b_i - \sum_{k=1}^{i-1} l_{ik} z_k \quad (i = 2, 3, \dots, n) \end{cases}$$

3) 解方程  $Dy = z$ , 即计算

$$y_i = z_i / d_i \quad (i = 1, 2, \dots, n)$$

4) 解方程  $L^T x = y$ , 即计算

$$\begin{cases} x_n = y_n \\ x_i = y_i - \sum_{k=i+1}^n l_{ki} x_k \quad (i = n-1, n-2, \dots, 2, 1) \end{cases}$$

下面给出改进平方根法的 FORTRAN 程序<sup>[C3]</sup>. 子程序语句为

SUBROUTINE LDLT (N, L, A, B, W)

其中哑元说明如下:

N 整型变量,输入参数;方程组的阶数  $n$ .

L 整型变量,输入参数;系数矩阵下三角形部分元素的个数,共有  $L = \frac{(1+n)n}{2}$ .

A L 个元素的一维实型数组,存放按行排列的系数矩阵的三角形部分元素,即排列顺序为  $a_{11}, a_{21}, a_{22}, a_{31}, a_{32}, a_{33}, \dots, a_{n1}, a_{n2}, \dots, a_{nn}$ .

B N 个元素的一维实型数组,开始存放方程组右端常向量,最后存放方程组的解向量.

W N 个元素的一维实型工作数组.

当系数矩阵非正定时,停机并打印出 STOP 7777 作标志.  
FORTRAN 子程序:

```
SUBROUTINE LDLT (N, L, A, B, W)
  DIMENSION A(L), B(N), W(N)
  DO 5 I = 2, N
    L1 = I*(I - 1)/2
    I1 = I - 1
    DO 3 J = 1, I1
      S = 0.0
      IF (J .EQ. 1) GO TO 2
      J1 = J - 1
      DO 1 K = 1, J1
        K1 = K + J*(J - 1)/2
1      S = S + W(K)*A(K1)
2      J2 = L1 + J
        J3 = J*(J + 1)/2
        W(J) = A(J2) - S
3      A(J2) = W(J)/A(J3)
      S = 0.0
      M = I*(I + 1)/2
      DO 4 J = 1, I1
        J1 = L1 + J
4      S = S + A(J1)*W(J)
        A(M) = A(M) - S
        IF (ABS (A(M)) .LT. 1E - 15) STOP 7777
5      CONTINUE
      DO 7 I = 2, N
        S = 0.0
        I1 = I - 1
        DO 6 J = 1, I1
          I2 = I*(I - 1)/2 + J
6      S = S + A(I2)*B(J)
7      B(I) = B(I) - S
        DO 8 I = 1, N
          I1 = I*(I + 1)/2
8      B(I) = B(I)/A(I1)
        DO 10 K = 2, N
          I = N + 1 - K
```



```

      S = 0.0
      I1 = I + 1
      DO 9 J = I1, N
        J1 = J*(J-1)/2 + 1
      9  S = S + A(J1)*B(J)
    10  B(I) = B(I) - S
      RETURN
    END

```

**算例** 求解下列线性方程组

$$\begin{bmatrix} 7 & 5 & 6 & 5 & 1 \\ 5 & 10 & 8 & 7 & 0 \\ 6 & 8 & 10 & 9 & 1 \\ 5 & 7 & 9 & 10 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 20 \\ 25 \\ 36 \\ 29 \\ 2 \end{bmatrix}$$

显然,系数矩阵是一个对称矩阵。我们采用改进平方根法的 LDLT 程序求解;如果系数矩阵非正定,则程序运行中将打印出 STOP 7777 而停机。

注意到,给程序中数组 A(L),  $L = \frac{(1+5) \times 5}{2} = 15$  输入的初始数据为系数矩阵的下三角阵,即依次为

7, 5, 10, 6, 8, 10, 5, 7, 9, 10, 1, 0, 1, 0, 1

解题程序:

```

C      THE MAIN PROGRAM
      DIMENSION A(15), B(5), W(5)
      READ*, (A(I), I = 1, 15)
      READ*, (B(I), I = 1, 5)
      CALL LDLT (5, 15, A, B, W)
      DO 100 I = 1, 5
        WRITE (6, 200) I, B(I)
    200 FORMAT (11X, 'X(', I2, ')=' , F12.6)
    100 CONTINUE
      STOP
    END

      SUBROUTINE LDLT (N, L, A, B, W)
      :   (本子程序段体)
    END

```

计算结果

$$X(1) = 0.692308$$

$$X(2) = -5.615385$$

$$X(3) = 19.000001$$

$$X(4) = -10.615386$$

$$X(5) = -17.692309$$

## 8-4 解三对角方程组的追赶法及其 FORTRAN 程序

形式如下的线性方程组

$$\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & \ddots & \ddots & \ddots \\ & & a_{n-1, n-2} & a_{n-1, n-1} & a_{n-1, n} \\ & & & a_{n, n-1} & a_{n, n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \quad (8.33)$$

称为三对角方程组。三对角方程组常出现在诸如三次样条插值和解二阶常微分方程边值问题的差分方法等场合。三对角方程组的系数矩阵是半带宽为 1 的带状矩阵。显然，当  $|i - j| > 1$  时， $a_{ij} = 0$ 。

我们用直接三角分解法来研究三对角方程组的求解。把 (8.33) 记成如下形式

$$\begin{bmatrix} b_1 & c_1 & & \\ a_1 & b_2 & c_2 & \\ & \ddots & \ddots & \ddots \\ & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} \quad (8.34)$$

我们的作法是，先假定 (8.34) 的系数矩阵存在一定形式的  $LU$  分解，从而导出  $L, U$  的计算公式以及求解三对角方程组的计算方法；然后证明在一定条件下，所给的分解的确存在，且所导出的计算公式是稳定的。

设方程组 (8.34) 的系数矩阵可分解如下：

$$\begin{bmatrix} b_1 & c_1 & & \\ a_1 & b_2 & c_2 & \\ & \ddots & \ddots & \ddots \\ & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & a_{n-1} & b_n \end{bmatrix} = \begin{bmatrix} \beta_1 & & & \\ a_1 & \beta_2 & & \\ & \ddots & \ddots & \\ & & a_{n-2} & \beta_{n-1} \\ & & & a_{n-1} & \beta_n \end{bmatrix} \begin{bmatrix} 1 & \delta_1 & & \\ & 1 & \delta_2 & \\ & & \ddots & \ddots \\ & & & 1 & \delta_{n-1} \\ & & & & 1 \end{bmatrix}$$

对比上述等式两边，易得计算  $\beta_i, \delta_i$  的递归公式为

$$\begin{cases} \beta_1 = b_1 \\ \delta_i = c_i / \beta_i \quad (i = 1, 2, \dots, n-1) \\ \beta_{i+1} = b_{i+1} - a_i \delta_i \quad (i = 1, 2, \dots, n-1) \end{cases} \quad (8.35)$$

由此我们可得求解三对角方程组 (8.34) 的计算方法如下：

1) 按公式 (8.35) 计算  $L, U$  的元素  $\beta_i (i = 1, 2, \dots, n)$  和  $\delta_i (i = 1, 2, \dots,$

$n-1$ ).

2) 解方程组  $Ly = f$ , 即计算

$$\begin{cases} y_1 = f_1/b_1 \\ y_i = (f_i - a_{i-1}y_{i-1})/\beta_i \quad (i = 2, 3, \dots, n) \end{cases}$$

3) 解方程组  $Ux = y$ , 即计算

$$\begin{cases} x_n = y_n \\ x_i = y_i - \delta_i x_{i+1} \quad (i = n-1, \dots, 2, 1) \end{cases}$$

通常,第2)步的计算称为“追”,第3)步的计算称为“赶”.上述解三对角方程组的方法称为**追赶法**.

由于这种系数矩阵的形状特殊,因而导出的计算比较简单.计算量只有  $5n-4$  次乘法运算.编制计算机程序时,只需用三个一维数组(而不是一个二维数组)来存储系数矩阵.

现在,我们来证明上述所作的分解的确存在,且所导出的计算公式是稳定的.

**定理** 如果三对角方程组(8.34)的系数矩阵满足下列所谓对角线占优条件:

- 1)  $|b_i| > |c_i| > 0$ ;
  - 2)  $|b_i| \geq |a_{i-1}| + |c_i|$ , 且  $a_i c_{i-1} \neq 0$ ,  $i = 2, 3, \dots, n-1$ ;
  - 3)  $|b_n| > |a_{n-1}| > 0$ ,
- (8.36)

则对所有的  $i$ , 由(8.35)算出  $\beta_i, \delta_i$  有  $0 < |\delta_i| < 1$ ,  $|\beta_i| > |c_i| > 0$ .

**证** 用数学归纳法.由已知条件(8.36)和公式(8.35),显然有

$$0 < |\delta_i| = \left| \frac{c_i}{\beta_i} \right| = \frac{|c_i|}{|b_i|} < 1$$

今设  $|\delta_{i-1}| < 1$ , 我们来证明有  $|\delta_i| < 1$ . 事实上,注意到归纳法假设可知

$$\begin{aligned} |\beta_i| &\geq |b_i| - |a_{i-1}||\delta_{i-1}| \\ &> |b_i| - |a_{i-1}| \geq |c_i| > 0 \end{aligned}$$

所以有

$$\begin{aligned} |\delta_i| &= \left| \frac{c_i}{\beta_i} \right| = \frac{|c_i|}{|b_i - a_{i-1}\delta_{i-1}|} \\ &\leq \frac{|c_i|}{|b_i| - |a_{i-1}||\delta_{i-1}|} < \frac{|c_i|}{|b_i| - |a_{i-1}|} \leq 1 \end{aligned}$$

从而,对所有  $i$  有  $0 < |\delta_i| < 1$ , 也有

$$|\beta_i| > |b_i| - |a_{i-1}| \geq |c_i| > 0$$

定理证毕.

根据定理,由于  $|\beta_i| > 0$ , 即  $\beta_i \neq 0$ , 因此追赶法的计算总是可以进行下去,即所作  $LU$  分解存在. 又由

$$0 < |\delta_i| < 1, \quad |\beta_i| > |b_i| - |a_{i-1}|$$

且有

$$\begin{aligned} |\beta_i| &= |b_i - a_{i-1}\delta_{i-1}| \leq |b_i| + |a_{i-1}||\delta_{i-1}| \\ &< |b_i| + |a_{i-1}| \end{aligned}$$

可知在追赶法的计算过程中不会出现数量级的巨大增长和舍入误差的严重积累,所以,追赶法是数值稳定的.

下面我们给解三对角方程组的追赶法设计一个通用的 FORTRAN 子程序。子程序语句为

SUBROUTINE TRID (N, A, B, C, F)

哑元说明:

- N 整型变量, 输入参数; 方程组的阶数  $n$ 。
- A 存放  $a_1, a_2, \dots, a_{n-1}$  的一维实型数组; 输入参数。
- B 存放  $b_1, b_2, \dots, b_n$  的一维实型数组; 输入参数。
- C 存放  $c_1, c_2, \dots, c_{n-1}$  的一维实型数组; 输入参数。
- F N 个元素的一维实型数组, 开始存放方程组右端向量  $f$ , 最后存放方程组的解向量  $x$ ; 输入/输出参数。

FORTRAN 子程序

```

SUBROUTINE TRID (N, A, B, C, F)
  DIMENSION A(N), B(N), C(N), F(N)
  F(1) = F(1)/B(1)
  W = B(1)
  N1 = N - 1
  DO 10 I = 1, N1
    B(I) = C(I)/W
    W = B(I + 1) - A(I)*B(I)
10  F(I + 1) = (F(I + 1) - A(I)*F(I))/W
    DO 20 J = 1, N1
      K = N - J
20  F(K) = F(K) - B(K)*F(K + 1)
  RETURN
END

```

注: 为了程序简单, 子程序中的数组 A 和 C 定义为 A(N) 和 C(N), 实际上只用到 A(N - 1) 和 C(N - 1)。在具体编译程序允许时, 可改定义为 A(N - 1) 和 C(N - 1)。

**算例** 用追赶法程序求解三对角方程组

$$\begin{bmatrix} 1 & 1 & & & \\ 1 & 2 & 1 & & \\ & 1 & 3 & 1 & \\ & & 1 & 4 & 1 \\ & & & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \\ 15 \\ 24 \\ 29 \end{bmatrix}$$

解题程序:

C THE MAIN PROGRAM

```

        DIMENSION A(5), B(5) C(5), F(5)
        READ *, (A(I), I = 1, 4)
        READ *, (B(I), I = 1, 5)
        READ *, (C(I), I = 1, 4)
        READ *, (F(I), I = 1, 5)
        CALL TRID (5, A, B, C, F)
        WRITE (6, 100) F
100    FORMAT (1X, 'X(I) = ', 5F10.5)
        STOP
        END

```

```

SUBROUTINE TRID (N, A, B, C, F)
:   (本子程序段体)
END

```

计算结果:

X(I) =        1.00000        2.00000        3.00000        4.0000        5.00000

## 8-5 解线性方程组的迭代法

求解线性代数方程组  $Ax = b$  的迭代法, 是对于任意给定的初始近似向量

$$x^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]^T$$

按一定的规则, 逐次构造一个向量序列  $\{x^{(k)}\}$

$$x^{(0)}, x^{(1)}, x^{(2)}, \dots, x^{(k)}, \dots$$

使其极限  $x$  (如果存在的话)

$$\lim_{k \rightarrow \infty} x^{(k)} = x$$

是方程组  $Ax = b$  的解. 因此, 迭代法的基本问题是:

- 1) 如何构造向量序列  $\{x^{(k)}\}$ ;
  - 2) 在什么条件下,  $\{x^{(k)}\}$  收敛, 且收敛于方程组的解.
- 这一节我们介绍解线性代数方程组的迭代法中最基本的两种.

### 1. 雅可比迭代法

假设  $n$  阶线性方程组

$$Ax = b \quad (8.37)$$

的系数矩阵  $A$  非奇异, 且对角线元素  $a_{11}, a_{22}, \dots, a_{nn}$  均不为零, 分别从第一个方程解出  $x_1$ , 从第二个方程解出  $x_2$ , 等等, 即将 (8.37) 改写成

$$\left\{ \begin{aligned} x_1 &= \frac{1}{a_{11}} (-a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n) + \frac{b_1}{a_{11}} \\ x_2 &= \frac{1}{a_{22}} (-a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n) + \frac{b_2}{a_{22}} \\ &\dots\dots\dots \\ x_n &= \frac{1}{a_{nn}} (-a_{n1}x_1 - a_{n2}x_2 - \dots - a_{nn-1}x_{n-1}) + \frac{b_n}{a_{nn}} \end{aligned} \right. \quad (8.38)$$

用矩阵向量记为

$$\dot{x} = Mx + d' \quad (8.38)'$$

$$(I - M)x = d \quad (8.38)''$$

其中

$$M = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix} \quad d = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix}$$

由于  $A$  非奇异, 可知  $I - M$  也非奇异. 因此, (8.37) 和 (8.38) 都有唯一解, 且解相同.

$$\mathbf{x}^{(0)} = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]^T$$

并构造迭代格式

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{11}} (-a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \dots - a_{1n}x_n^{(k)}) + \frac{b_1}{a_{11}} \\ x_2^{(k+1)} = \frac{1}{a_{22}} (-a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \dots - a_{2n}x_n^{(k)}) + \frac{b_2}{a_{22}} \\ \vdots \\ x_n^{(k+1)} = \frac{1}{a_{nn}} (-a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - \dots - a_{nn-1}x_{n-1}^{(k)}) + \frac{b_n}{a_{nn}} \end{cases} \quad (8.39)$$

或写成矩阵向量形式

$$\mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{d} \quad (8.39)'$$

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(k)}, \mathbf{x}^{(k+1)}, \dots \quad (8.40)$$

其中上标指迭代次数.

下面,我们证明这样作出的向量序列(8.40)在一定条件下,收敛于方程组(8.37)的解向量 $x$ .

**定理 1** 如果  $\mu = \max_{1 \leq i \leq n} \sum_{j=1, j \neq i}^n \left| \frac{a_{ij}}{a_{ii}} \right| < 1$ , 则雅可比迭代法收敛。

**证** 由 (8.39) 的第  $i$  个方程减去 (8.38) 的第  $i$  个方程, 可得

$$x_i^{(k+1)} - x_i = \sum_{j=1, j \neq i}^n \frac{a_{ij}}{a_{ii}} (x_j^{(k)} - x_j) \quad (i = 1, 2, \dots, n)$$

令  $e_k = \max_{1 \leq i \leq n} |x_i^{(k)} - x_i|$  ( $k = 0, 1, 2, \dots$ ) 并根据定理条件有

$$\begin{aligned} |x_i^{(k+1)} - x_i| &\leq \sum_{j=1, j \neq i}^n \left| \frac{a_{ij}}{a_{ii}} \right| |x_j^{(k)} - x_j| \\ &\leq \left( \sum_{j=1, j \neq i}^n \left| \frac{a_{ij}}{a_{ii}} \right| \right) \max_{1 \leq i \leq n} |x_i^{(k)} - x_i| \\ &= \left( \sum_{j=1, j \neq i}^n \left| \frac{a_{ij}}{a_{ii}} \right| \right) e_k \\ &= \mu e_k \end{aligned}$$

即

$$e_{k+1} = \mu e_k$$

于是由递归关系可得

$$e_{k+1} \leq \mu e_k \leq \mu^2 e_{k-1} \leq \dots \leq \mu^{k+1} e_0$$

因此, 对于  $\mu < 1$  有

$$\lim_{k \rightarrow \infty} e_k = 0$$

即

$$\lim_{k \rightarrow \infty} x_i^{(k)} = x_i \quad (i = 1, 2, \dots, n)$$

这说明雅可比迭代序列  $\{x^{(k)}\}$  收敛于方程组 (8.38) 的解。证毕。

请注意, 定理所述条件只是一个充分条件, 而并非必要条件。关于迭代法收敛性的更一般讨论, 见本节第 3 小节。

由上可见, 雅可比方法很简单, 每迭代一次只需计算一次矩阵与向量的乘积。程序设计时只需用两个一维数组存放  $x^{(k)}$  和  $x^{(k+1)}$ 。

当  $A$  非奇异时, 可通过作必要的行交换保证  $a_{ii} \neq 0$  ( $i = 1, 2, \dots, n$ ) 且其绝对值尽可能地大。

## 2. 高斯-赛德尔迭代法及其 FORTRAN 程序

雅可比方法可加以改进。

当用迭代公式 (8.39) 计算  $x_i^{(k+1)}$  时, 实际上  $x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_{i-1}^{(k+1)}$  已经算出来了, 但计算  $x_i^{(k+1)}$  的公式中仍用  $x_1^{(k)}, x_2^{(k)}, \dots, x_{i-1}^{(k)}$ 。雅可比方法的改进在于, 当计算  $x_i^{(k+1)}$  时, 改用  $x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_{i-1}^{(k+1)}$  代替  $x_1^{(k)}, x_2^{(k)}, \dots, x_{i-1}^{(k)}$ , 即用改进的迭代格式

$$(8.41)$$

$$x^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + d \quad (8.41)'$$
$$L = \begin{bmatrix} 0 & & & & \\ -\frac{a_{21}}{a_{22}} & 0 & & & \\ \vdots & \vdots & \ddots & & \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & \dots & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ & \vdots & \ddots & \vdots \\ & & & 0 \end{bmatrix}$$

还可把(8.41)写成另一种矩阵形式

或

(其中  $L$  和  $U$  的意义已与 (8.41)' 不同) 由假设知  $(D - L)$  非奇异, 因此

于是高斯-赛德尔迭代公式也可表示为

由于在计算  $x_i^{(k+1)}$  时, 及时利用已经算出的  $x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ , 因此, 高斯-赛德尔迭代法有可能比雅可比迭代法收敛得快, 即要达到同样的精度, 所需迭代次数较少. 但这个结论并不是在所有的情况下都能成立. 可以找到有这样的方程组, 用雅可比迭代法收敛, 用高斯-赛德尔迭代法反而发散.

下面,我们研究高斯-赛德尔迭代法的计算机程序设计.

$$x_i^{(k)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} \right) / a_{ii} \quad i = 1, 2, \dots, n \quad (8.41)''$$
$$\max |x_i^{(k)} - x_i^{(k-1)}| < \varepsilon, \quad i = 1, 2, \dots, n$$



便迭代结束。 $x^{(k)}$  为所求之解,  $\varepsilon$  为允许的误差。

另外, 算法中将给定一个最大允许迭代次数, 当迭代次数超过这个控制常数仍未能收敛时程序将自动停止执行。

设置子程序哑元如下:

- N 整型变量, 输入参数; 方程组阶数  $n$ 。
- A  $N \times N$  个元素的二维实型数组, 输入参数; 方程组的系数矩阵。
- B  $N$  个元素的一维实型数组, 输入参数; 方程组的右端向量。
- X  $N$  个元素的一维实型数组, 开始时由程序置全为零的初值, 结束时存放解向量。
- KMAX 整型变量, 允许最大迭代次数。
- EPS 实型变量, 允许误差。
- K 整型变量, 输出参数; 调用子程序返回时给出实际的迭代次数。

程序中还使用中间变量 E, E1, Y 等。其中 E, E1 分别用来计算各点两次迭代的最大误差。Y 用于计算  $x_i$  的中间迭代公式

$$Y = Y - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)}$$

FORTRAN 子程序如下:

```
SUBROUTINE GSSD (N, A, B, X, KMAX, EPS, K)
  DIMENSION A(N, N), B(N), X(N)
  DO 1 I = 1, N
1    X(I) = 0.0
    K = 0
2    E = 0.0
    DO 4 I = 1, N
      Y = B(I)
      DO 3 J = 1, N
        IF (I .NE. J) Y = Y - A(I, J) * X(J)
3      CONTINUE
      Y = Y/A(I, I)
      E1 = ABS(Y - X(I))
      X(I) = Y
      IF (E .LT. E1) E = E1
4    CONTINUE
    IF (E .LT. EPS) RETURN
    K = K + 1
    IF (K - KMAX) 2, 2, 5
5  RETURN
  END
```

**算例** 我们在这一章的开头就说过,解线性代数方程组的迭代法适宜于高阶问题,且有程序简单的优点。这里,我们考虑求解结构分析中一个比前面举过的例子较为高阶的线性方程组

$$[K]\{\delta\} = \{P\}$$

其中  $[K]$  为  $9 \times 9$  的已知矩阵,在结构分析中称刚度矩阵;  $\{\delta\}$  是 9 个元素的未知列向量,点的位移;  $\{P\}$  是 9 个元素的已知列向量,称荷载向量。

系数矩阵  $[K]$  和右端向量  $\{P\}$  的数据用自由格式输入,并用打印机印出。因数据太多,这里题目不列出,详见下述输出结果。

解题程序如下:

```
C      THE MAIN PROGRAM
      DIMENSION A(9, 9), B(9), X(9)
      READ (5, *) ((A(I, J), J = 1, 9), B(I), I = 1, 9)
      WRITE (6, 100) A
100  FORMAT (1X, 'COEFFICIENT MATRIX' 119 (1X, F7.3))
      WRITE (6, 200) B
200  FORMAT (1X//1X, 'RIGHT-SIDE VECTOR'//9 (1X, F7.3))
      CALL GSSD (9, A, B, X, 1000, 1E - 8, K)
      WRITE (6, 300) X
300  FORMAT (1X//1X, 'SOLUTION'//9 (1X, F7.3))
      WRTEE (6, 400) K
400  FORMAT (1X//1X, 'NUMBER OF ITERATIONS', 14)
      STOP
      END

C      SUBROUTINE GSSD (N, A, B, X, KMAX, EPS, K)
      : (本子程序段体)
      END
```

输出结果:

```
COEFFICIENT MATRIX
189.360    0.000   11.835   94.680    0.000   -11.835    0.000    0.000    0.000
   0.000   22.500    0.000    0.000  -22.500    0.000    0.000    0.000    0.000
  11.835    0.000    2.186   11.835    0.000   -0.986    0.000    0.000    0.000
  94.680    0.000   11.835  340.848    0.000   -4.261   75.744    0.000  -7.574
   0.000  -22.500    0.000    0.000   41.948    0.000    0.000  -18.000    0.000
 -11.835    0.000   -0.986   -4.261    0.000   64.491    7.574    0.000  -0.505
   0.000    0.000    0.000   75.744    0.000    7.574  151.488    0.000  -7.574
   0.000    0.000    0.000    0.000  -18.000    0.000    0.000   18.000    0.000
   0.000    0.000    0.000  -7.574    0.000   -0.505  -7.574    0.000    1.705

RIGHT-SIDE VECTOR
   0.000    0.000    0.000  -56.250    1.562   -7.500   56.250    0.000  -7.500

SOLUTION
   0.099    1.079    1.787   -0.444    1.079   -0.180    0.362    1.079  -4.820

NUMBER OF ITERATIONS 429
```

### 3. 迭代法收敛性讨论

对于写成 (8.38)' 形式

$$\mathbf{x} = M\mathbf{x} + \mathbf{d}$$

的方程组, 用公式 (8.39)'

$$\mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{d}$$

逐步代入求出近似解的方法称为迭代法, 或称一阶定常迭代法. 如果存在极限

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

则称此迭代法收敛, 且  $\mathbf{x}^*$  就是方程组的解; 否则, 称此迭代法不收敛.

对迭代过程 (8.39)' 需要研究  $\{\mathbf{x}^{(k)}\}$  的收敛性.

设  $\mathbf{x} = M\mathbf{x} + \mathbf{d}$  有唯一解  $\mathbf{x}^*$ , 即

$$\mathbf{x}^* = M\mathbf{x}^* + \mathbf{d} \quad (8.43)$$

引进误差向量

$$\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$$

将 (8.43) 式减去 (8.39)' 式, 可得递推关系

$$\mathbf{e}^{(k+1)} = M\mathbf{e}^{(k)} \quad (k = 0, 1, 2, \dots)$$

于是可得

$$\mathbf{e}^{(k)} = M\mathbf{e}^{(k-1)} = \dots = M^k\mathbf{e}^{(0)}$$

这就表明, 要研究  $\{\mathbf{x}^{(k)}\}$  的收敛性, 就是要研究  $M$  在什么条件下有  $\mathbf{e}^{(k)} \rightarrow 0$  ( $k \rightarrow \infty$ ), 这又相当要研究当  $k \rightarrow \infty$  时,  $M$  满足什么条件有  $M^k \rightarrow 0$  (零矩阵).

**定理 1** (迭代法收敛的充分条件) 若迭代公式 (8.39)' 中的迭代矩阵  $M$  的某一种范数

$$\|M\| = q < 1$$

则有

1) 迭代法收敛 (即  $\{\mathbf{x}^{(k)}\}$  收敛于方程组的解);

$$2) \|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \frac{q}{1-q} \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|;$$

$$3) \|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \frac{q^k}{1-q} \|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|.$$

**证**

1) 由误差向量的递推公式

$$\mathbf{e}^{(k+1)} = M\mathbf{e}^{(k)} \quad (k = 0, 1, 2, \dots)$$

可得

$$\|\mathbf{e}^{(k+1)}\| = \|M\mathbf{e}^{(k)}\| \leq \|M\|\|\mathbf{e}^{(k)}\|$$

反复利用此递推关系, 又可得

$$\|\mathbf{e}^{(k)}\| \leq \|M\|^k \|\mathbf{e}^{(0)}\| = q^k \|\mathbf{e}^{(0)}\|$$

于是当  $\|M\| = q < 1$  时, 有

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\| = \|\mathbf{e}^{(k)}\| \rightarrow 0 \quad (k \rightarrow \infty)$$

即  $\{\mathbf{x}^{(k)}\}$  收敛于  $\mathbf{x}^*$

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*$$

2) 首先,由迭代公式对于  $k = 1, 2, \dots$  有

$$\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = M(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$$

即有

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \|M\| \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| = q \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|$$

注意到

$$\begin{aligned} \|\mathbf{x}^* - \mathbf{x}^{(k)}\| &= \|\mathbf{x}^* - \mathbf{x}^{(k+1)} + \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \\ &\leq \|\mathbf{x}^* - \mathbf{x}^{(k+1)}\| + \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \end{aligned}$$

因此有

$$\begin{aligned} \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| &\geq \|\mathbf{x}^* - \mathbf{x}^{(k)}\| - \|\mathbf{x}^* - \mathbf{x}^{(k+1)}\| \\ &\geq \|\mathbf{x}^* - \mathbf{x}^{(k)}\| - \|M\| \|\mathbf{x}^* - \mathbf{x}^{(k)}\| \\ &= (1 - q) \|\mathbf{x}^* - \mathbf{x}^{(k)}\| \end{aligned}$$

于是可得

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \frac{q}{1 - q} \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|$$

3) 反复利用递推关系,又可得

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \frac{q^k}{1 - q} \|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|$$

定理证毕.

由定理可知,当  $\|M\| = q < 1$  愈小时,迭代收敛愈快;当  $M$  的某一种范数  $\|M\| < 1$  时,如果相邻两次迭代  $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \varepsilon$  ( $\varepsilon$  为给定的精度要求),则  $\|\mathbf{x}^* - \mathbf{x}^{(k)}\| < \frac{q}{1 - q} \varepsilon$ . 所以在程序设计中通常用  $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \varepsilon$  来作为控制迭代结束的条件.

**例 1** 试考察用雅可比迭代法解方程组

$$\begin{cases} 8x_1 - 3x_2 + 2x_3 = 20 \\ 4x_1 + 11x_2 - x_3 = 33 \\ 6x_1 + 3x_2 + 12x_3 = 36 \end{cases}$$

时是否收敛.

**解** 由方程组可知迭代矩阵

$$M = \begin{bmatrix} 0 & \frac{3}{8} & -\frac{2}{8} \\ -\frac{4}{11} & 0 & \frac{1}{11} \\ -\frac{6}{12} & -\frac{3}{12} & 0 \end{bmatrix}$$

它的  $\infty$  范数

$$\|M\|_{\infty} = \max \left\{ \frac{5}{8}, \frac{5}{11}, \frac{9}{12} \right\} = \frac{9}{12} < 1$$

所以用雅可比迭代法收敛.

**例 2** 仍以例 1 中的方程组为例,考察采用高斯-赛德尔迭代法时是否收敛.

**解** 分解

$$A = D - L - U = \begin{bmatrix} 8 & & \\ & 11 & \\ & & 12 \end{bmatrix} - \begin{bmatrix} 0 & & \\ -4 & 0 & \\ -6 & -3 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ & 0 & 1 \\ & & 0 \end{bmatrix}$$

于是迭代矩阵

$$\begin{aligned} G = (D - L)^{-1}U &= \begin{bmatrix} \frac{1}{8} & & \\ -\frac{1}{22} & \frac{1}{11} & \\ -\frac{9}{176} & -\frac{1}{44} & \frac{1}{12} \end{bmatrix} \begin{bmatrix} 0 & 3 & -2 \\ & 0 & 1 \\ & & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \frac{3}{8} & -\frac{2}{8} \\ 0 & -\frac{3}{22} & \frac{2}{11} \\ 0 & -\frac{27}{176} & \frac{7}{88} \end{bmatrix} \end{aligned}$$

由此有

$$\|G\|_{\infty} = \max \left\{ \frac{5}{6}, \frac{7}{22}, \frac{41}{176} \right\} = \frac{5}{8} < 1$$

可知应用高斯-赛德尔迭代法时收敛。

下面再给出迭代法收敛性的充要条件,但证明从略。

**定理 2** (迭代法基本定理) 对任意初始向量解方程组的迭代法收敛的充分必要条件是迭代矩阵  $M$  的谱半径

$$\rho(M) < 1$$

且当  $\rho(M) < 1$  时,迭代矩阵  $M$  的谱半径愈小收敛愈快。

对于方程组系数矩阵具有某种特征的情况,迭代法收敛性条件还有更便于使用的形式。

**定义** 设矩阵  $A \in R^{n \times n}$  满足条件

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \quad (i = 1, 2, \dots, n)$$

即  $A$  的每一行对角元素的绝对值都严格大于同行其它元素绝对值之和,则称  $A$  为严格对角占优矩阵。

可以证明(例如用反证法),严格对角占优的矩阵必为非奇异矩阵。

利用严格对角占优的概念,我们有

**定理 3** 设方程组为  $Ax = b$ ,  $A \in R^{n \times n}$ 。如果  $A$  为严格对角占优矩阵,则解此方程的雅可比迭代法和高斯-赛德尔迭代法都收敛。

这个定理的证明也从略。

## 8-6 方程组的条件问题

实际问题中提出的线性方程组

$$Ax = b$$

由于数据  $A$  和  $b$  往往是从观测或实验得到的, 因而带有一定的误差, 这些误差对计算结果的精确度自然有一定的影响. 问题是这种影响的大小程度究竟如何?

看一个简单例子.

**例 1** 设有线性方程组

$$\begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$$

或记成

$$Ax = b$$

其中

$$A = \begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$$

容易验证, 方程组的精确解为  $x^* = [0, 0.1]^T$ .

今再设方程组的右端向量  $b$  有微小变化  $\delta b$ , 即

$$b + \delta b = [0.7 + (-0.01), 1 + (0.01)]^T = [0.69, 1.01]^T$$

相应地, 解  $x$  也有改变  $\delta x$ , 即  $\tilde{x} = x + \delta x$ . 于是有方程组

$$A(x + \delta x) = b + \delta b$$

或

$$\begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} 0.69 \\ 1.01 \end{bmatrix}$$

可以验证, 这时方程组的解  $\tilde{x}^* = [-0.17, 0.22]^T$ , 其中  $\delta x = [-0.17, 0.12]^T$ , 可见, 上述方程组当右端向量的两个分量只有  $\frac{1}{100}$  的微小变化时, 方程组的解却有相对较大的变化. 这说明此方程组的解对原始数据  $b$  是很灵敏的. 对这样的方程组显然不易求得较精确的解. 这样的方程组就称为病态方程组.

方程组原始数据误差对方程组解的影响问题, 就是所谓方程组的条件问题. 方程组的条件问题是方程组本身的“好”、“坏”问题.

分两个方面来讨论.

1. 设方程组系数矩阵  $A$  非奇异, 而且是准确的. 我们讨论右端常向量  $b$  的误差对方程组的影响.

设  $b$  有误差  $\delta b$ , 相应地解  $x$  有误差  $\delta x$ , 则有

$$A(x + \delta x) = b + \delta b$$

于是有

$$\delta x = A^{-1}\delta b$$

$$\|\delta x\| \leq \|A^{-1}\| \|\delta b\|$$

但由原方程组  $Ax = b$  有

$$\|b\| \leq \|A\|\|x\|$$

因此,有

$$\|\delta x\|\|b\| \leq \|A\|\|A^{-1}\|\|x\|\|\delta b\|$$

且当  $x \neq 0, b \neq 0$  时,有

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\| \frac{\|\delta b\|}{\|b\|}$$

这就是说,解  $x$  的“相对误差”是初始数据  $b$  的“相对误差”的  $\|A\|\|A^{-1}\|$  倍.

2. 设方程组右端常向量  $b$  是准确的,系数矩阵  $A$  非奇异且有误差. 我们讨论  $A$  的误差对方程组解的影响.

设矩阵  $A$  有误差  $\delta A$ , 相应地解有误差  $\delta x$ , 则有

$$(A + \delta A)(x + \delta x) = b$$

设  $A + \delta A$  也非奇异(容易验证,只需当  $\|A^{-1}\|\|\delta A\| < 1$  时即可),则

$$Ax + (\delta A)x + A\delta x + \delta A\delta x = b$$

$$A\delta x = -(\delta A)x - \delta A\delta x$$

$$\delta x = -A^{-1}(\delta A)x - A^{-1}\delta A\delta x$$

根据范数的性质,有

$$\|\delta x\| \leq \|A^{-1}\|\|\delta A\|\|x\| + \|A^{-1}\|\|\delta A\|\|\delta x\|$$

$$(1 - \|A^{-1}\|\|\delta A\|)\|\delta x\| \leq \|A^{-1}\|\|\delta A\|\|x\|$$

于是可得

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\delta A\|}{1 - \|A^{-1}\|\|\delta A\|} = \frac{\|A^{-1}\|\|A\| \frac{\|\delta A\|}{\|A\|}}{1 - \|A^{-1}\|\|A\| \frac{\|\delta A\|}{\|A\|}}$$

这就是说,若  $\|A^{-1}\|\|A\| \frac{\|\delta A\|}{\|A\|}$  很小,解  $x$  的“相对误差”也是初始数据  $A$  的“相对误差”的  $\|A^{-1}\|\|A\|$  倍,或者说,  $\|A^{-1}\|\|A\|$  表示相对误差的近似放大率.

由上述讨论,可知方程组的初始数据  $A$  或  $b$  有微小改变时,数  $\|A\|\|A^{-1}\|$  标志着解  $x$  的敏感程度,解  $x$  的相对误差可能随  $\|A\|\|A^{-1}\|$  的增大而增大. 这就是说,系数矩阵  $A$  刻画了线性方程组的性态. 为此,我们引入如下概念.

**定义 1** 设  $A$  为  $n$  阶非奇异矩阵,数

$$\|A\|\|A^{-1}\|$$

称为矩阵  $A$  的**条件数**,记为  $\text{Cond}(A)$ .

由定义可知,矩阵条件数与矩阵的范数有关. 常使用的条件数是

$$\text{Cond}(A) = \|A\|_{\infty}\|A^{-1}\|_{\infty}$$

或谱条件数

$$\text{Cond}(A)_2 = \|A\|_2\|A^{-1}\|_2 = \sqrt{\frac{\lambda_{\max}(A^{-1}A)}{\lambda_{\min}(A^{-1}A)}}$$

其中  $\lambda_{\max}(A^{-1}A)$  与  $\lambda_{\min}(A^{-1}A)$  分别为  $A^{-1}A$  的最大、最小特征值.

条件数具有下列简单性质:

- 1)  $\text{Cond}(A) \geq 1$ ;
- 2)  $\text{Cond}(kA) = \text{Cond}(A)$ ,  $k$  为不等零的常数;
- 3) 若  $\|A\| = 1$ , 则  $\text{Cond}(A) = \|A^{-1}\|$ .

这些性质容易直接验证.

**定义 2** 设方程组  $Ax = b$ ,  $A$  为非奇异矩阵. 如果  $\text{Cond}(A)$  相对地大, 如  $\text{Cond}(A) \gg 1$ , 便称方程组  $Ax = b$  是病态的, 或是坏条件的; 如果  $\text{Cond}(A)$  相对地小, 便称方程组  $Ax = b$  是良态的, 或是好条件的.

**例 2** 重新检查上述例 1 的方程组. 因为

$$A = \begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} 10 & 7 \\ 7 & 5 \end{bmatrix}$$

所以  $\text{Cond}(A)_{\infty} = \|A\|_{\infty} \|A^{-1}\|_{\infty} = 17 \times 17 = 289$ , 可知该方程组是病态的.

**例 3** 考察方程组

$$\begin{bmatrix} 1.001 & 0.25 \\ 0.25 & 0.0625 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.501 \\ 0.375 \end{bmatrix}$$

先看  $A$  的条件数. 由

$$A = \begin{bmatrix} 1.001 & 0.25 \\ 0.25 & 0.0625 \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} 1000 & -4000 \\ -4000 & 16016 \end{bmatrix}$$

可得

$$\|A\|_{\infty} = 1.251, \quad \|A^{-1}\|_{\infty} = 20016$$

于是有

$$\text{Cond}(A) = 25040$$

这表明所给方程组是病态的. 可以具体算一算. 直接验算知原方程组的解为  $x = [1, 2]^T$ . 现在把系数矩阵及右端取成近似值, 比如取为方程组

$$\begin{bmatrix} 1 & 0.25 \\ 0.25 & 0.063 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 0.37 \end{bmatrix}$$

则可得其解  $\tilde{x} = [4, -10]^T$ . 可见当系数矩阵及右端常向量中的元素的绝对误差最大变化为  $\frac{1}{2} \times 10^{-2}$  时, 方程组解的变化却相当地大. 可知方程组确实是病态的.

再一次强调, 方程组(也即系数矩阵)“病态”的性质是方程组本身的固有特性, 或者说提供的线性方程组本身的条件较“坏”. 对于病态方程组, 即使采用稳定的算法, 由于舍入误差的存在, 也可能算不出令人满意的结果. 至于条件数多大才算属于病态范围, 一般来说没有具体的标准, 只是相对而言.

由于计算条件数涉及到计算逆矩阵, 因此是很麻烦的. 实际计算中一般通过可能产生病态的现象来进行判断. 这些现象大致有:

- 1) 矩阵元素间的数量级相差很大, 即大的很大, 小的很小, 而且无一定规律.
- 2) 在消元过程中出现有效数字严重损失, 或者出现小主元.
- 3) 矩阵的行列式值相对来说很小, 或某些行(列)近似线性相关.

对于病态方程组, 通常采用如下一些处理方法:

- 1) 采用双精度运算, 以便改善和减轻矩阵病态的影响.
- 2) 对方程组进行某些预处理, 如适当选择非奇异对角矩阵  $D, C$ , 把解方程组  $Ax = b$  转化为解等价方程组  $D^{-1}AC^{-1}(Cx) = D^{-1}b$ , 且使  $D^{-1}AC^{-1}$  的条件数得到改善.



3) 采用所谓迭代改善的方法求解。迭代改善的方法简单说明如下:

设用选主元消去法解得  $Ax = b$  的一个近似解  $x^{(1)}$ , 又  $x^{(1)}$  的剩余向量为

$$r^{(1)} = b - Ax^{(1)}$$

再由方程组  $Ad^{(1)} = r^{(1)}$  解出  $d^{(1)}$ , 并计算  $x^{(2)} = x^{(1)} + d^{(1)}$ , 则有

$$\begin{aligned} x^{(2)} &= x^{(1)} + d^{(1)} \\ &= x^{(1)} + A^{-1}b - x^{(1)} = A^{-1}b = x^* \end{aligned}$$

即  $x^{(2)}$  就是方程组的精确解  $x^*$ 。不过, 在实际计算中, 由于存在舍入误差, 因而  $x^{(2)}$  不可能是精确解。为此, 我们重复上述过程, 即依次由  $r^{(2)} = b - Ax^{(2)}$ ,  $Ad^{(2)} = r^{(2)}$ ,  $x^{(3)} = x^{(2)} + d^{(2)}$  得近似解  $x^{(3)}$ , 并且继续此过程可得近似解序列  $\{x^{(k)}\}$ 。只要矩阵  $A$  不是过份病态, 我们所得的近似序列  $\{x^{(k)}\}$  收敛于精确解, 当  $A$  过份病态时,  $\{x^{(k)}\}$  可能不收敛于精确解。这就是迭代改善方法的大意。

## 习 题 八

1. 假设在四位数字浮点计算机上解(用手算代替)方程组

$$\begin{cases} x + 592y = 437 \\ 592x + 4308y = 2255 \end{cases}$$

(1) 用不选主元的高斯消去法;

(2) 用列主元高斯消去法;

(3) 用全主元高斯消去法。

试比较计算结果。

2. 试分别用高斯消去法、列主元高斯消去法和全主元高斯消去法(手算)解下列方程组

$$\begin{cases} 2x_1 + 3x_2 + 5x_3 = 5 \\ 3x_1 + 4x_2 + 7x_3 = 6 \\ x_1 + 3x_2 + 3x_3 = 5 \end{cases}$$

3. 利用列主元高斯消去法的 FORTRAN 程序, 解下列方程组

$$(1) \begin{bmatrix} 2 & -4 & 2 & -4 \\ -4 & 10 & 2 & 5 \\ 2 & 2 & 12 & -5 \\ 4 & 5 & -5 & 12 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ -5 \\ -2 \\ -4 \end{bmatrix}$$

$$(2) \begin{bmatrix} 3.3330 & 15.920 & -10.333 \\ 2.2220 & 16.710 & 9.6120 \\ 1.5611 & 5.1791 & 1.6852 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 15.913 \\ 28.544 \\ 8.4252 \end{bmatrix}$$

4. 用高斯-约当消去法求逆矩阵和行列式值的 FORTRAN 程序, 求下列矩阵  $A$  之逆阵  $A^{-1}$  及  $A$  的行列式值  $\det A$

$$A \equiv \begin{bmatrix} 2 & 1 & -3 & -1 \\ 3 & 1 & 0 & 7 \\ -1 & 2 & 4 & -2 \\ 1 & 0 & -1 & 5 \end{bmatrix}$$

5. 试作出下列矩阵  $A$  的  $LU$  分解和  $LDU$  分解:

$$A = \begin{bmatrix} 2 & -1 & -1 \\ 1 & 2 & 0 \\ 1 & 0 & 3 \end{bmatrix}.$$

6. 设  $A \in R^{n \times n}$  为对称正定矩阵, 则

(1) 对任意  $i \neq j$ , 都有  $a_{ii}^2 < a_{ii}a_{jj}$ .

(2)  $A$  的主元素必在对角线上.

7. 证明: 单位上(下)三角矩阵的逆阵和乘积仍为单位上(下)三角矩阵.

8. 设  $A$  为对称矩阵, 且  $a_{ii} \neq 0$ , 经高斯消去法一步后  $A$  约化为

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_1^T \\ 0 & A_2 \end{bmatrix}$$

试证明  $A_2$  也是对称矩阵.

9. 用改进平方根法解(手算)线性方程组

$$\begin{bmatrix} 5 & 10 & 30 \\ 10 & 30 & 100 \\ 30 & 100 & 354 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 20 \\ 70 \end{bmatrix}$$

10. 用改进平方根法的 FORTRAN 程序, 解下列方程组

$$\begin{bmatrix} 4 & 1 & -1 & 0 & x_1 \\ 1 & 3 & -1 & 0 & x_2 \\ -1 & -1 & 5 & 2 & x_3 \\ 0 & 0 & 2 & 4 & x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \\ -4 \\ 6 \end{bmatrix}$$

11. 用追赶法解(手算)三对角方程组

$$\begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & 1 & -4 & 1 \\ & & 1 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

12. 用追赶法的 FORTRAN 程序, 上机解算下列三对角方程组

$$\begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & -1 & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 100 \\ 200 \\ 200 \\ 200 \\ 100 \end{bmatrix}$$

13. 设线性方程组的形式为

$$\begin{bmatrix} a_1 & c_1 & & & b_1 \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_{n-1} \\ c_n & & & b_n & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

试提出解这种方程组的一种计算方法.

14. 用雅可比迭代法解(手算)下列方程组

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$

15. 用高斯-赛德尔迭代法解(手算)下列方程组

$$\begin{bmatrix} 7 & 1 & 2 \\ 1 & 8 & 2 \\ 2 & 2 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

16. 用高斯-赛德尔迭代法的 FORTRAN 程序上机解算下列方程组 (要求  $\max_i |x_i^{(k)} - x_i^{(k-1)}| < 0.001$  时迭代终止):

$$(1) \begin{bmatrix} 5.738 & -2.432 & 1 \\ -3.417 & 8.357 & -4.913 \\ 7.562 & 3.721 & -12.432 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 17.538 \\ 12.249 \\ 14.826 \end{bmatrix}$$

$$(2) \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 1 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 6 \\ -2 \\ 6 \end{bmatrix}$$

17. 说明高斯-赛德尔迭代法对下列方程组不收敛:

$$\begin{bmatrix} 1 & -2 & -3 \\ -2 & 1 & -3 \\ -2 & -3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

18. 设有方程组

$$\begin{cases} x = f(x, y) \\ y = g(x, y) \end{cases}$$

用迭代公式

$$\begin{cases} x_{n+1} = f(x_n, y_n) \\ y_{n+1} = g(x_n, y_n) \end{cases} \quad n = 0, 1, 2, \dots$$

求解. 证明: 若

$$\left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial f}{\partial y} \right| \leq L < 1, \quad \left| \frac{\partial g}{\partial x} \right| + \left| \frac{\partial g}{\partial y} \right| \leq L < 1$$

$L$  是一常数, 则迭代收敛.

19. 下列方程组

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 18 \\ 26 \\ 34 \\ 82 \end{bmatrix}$$

能用高斯-赛德尔迭代法求解吗? 如果能, 试解之.

20. 试用高斯-赛德尔迭代法并(编写计算机程序)解非线性方程组

$$\begin{cases} 4x = y^2 = z = 11 \\ x + 4y + z^2 = 18 \\ x^2 + y + 4z = 15 \end{cases}$$

21. 选择合适的方法及现成的标准子程序, 求解下列方程组:

$$(1) \begin{bmatrix} 5 & -1 & 0 & -1 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 6 & -1 \\ -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

$$(2) \begin{bmatrix} 8 & -3 & 0 & -2 & 0 \\ -3 & 8 & -2 & 0 & 0 \\ 0 & -2 & 4 & -1 & -1 \\ -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ -2 \end{bmatrix}$$

22. 分别在范数  $\|\cdot\|_1$ ,  $\|\cdot\|_\infty$  的意义下求下列矩阵  $A$  的条件数  $\text{Cond}(A)$

$$A = \begin{bmatrix} 1 & 2 \\ 1.001 & 2.001 \end{bmatrix}$$

23. 用迭代改善的方法解下列病态方程组

$$\begin{bmatrix} 9 & 9 & 8 \\ 9 & 8 & 7 \\ 8 & 7 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 26 \\ 24 \\ 21 \end{bmatrix}$$

## 附录 矩阵指数 $e^{At}$ 的数值计算方法及其 FORTRAN 程序

在现代控制系统的状态空间分析中,会遇到计算矩阵指数  $e^{At}$  的问题。所谓矩阵指数是指

$$e^{At} = I + At + \frac{1}{2!} A^2 t^2 + \cdots + \frac{1}{k!} A^k t^k + \cdots = \sum_{k=0}^{\infty} \frac{A^k t^k}{k!} \quad (\text{A-1})$$

其中  $A$  为  $n \times n$  矩阵,  $I$  为  $n \times n$  单位矩阵,  $t$  在状态空间分析中是时间增量(采样周期)。可以证明,矩阵指数  $e^{At}$  对于所有有限时间是绝对收敛的。 $e^{At}$  本身是一个矩阵。

关于矩阵指数  $e^{At}$  的计算有解析计算方法和数值计算方法两大类<sup>[14,15]</sup>。前者包括利用约当标准形法、拉普拉斯变换法和有限级数表示法等。后者包括截取  $e^{At}$  级数表示的有限项方法、巧妙地利用凯莱-哈密顿(Cayley-Hamilton)定理和 Leverrier-Faddeeva 算法等矩阵固有性质的方法等。

这里我们介绍  $e^{At}$  的一种数值计算方法,即截取  $e^{At}$  的级数表示的有限项的方法及其 FORTRAN 程序<sup>[17,18]</sup>。从原理说,它是最简单的。具体做法如下:

把  $e^{At}$  的无穷级数分成  $M$  和  $R$  两部分,即

$$e^{At} = I + At + \frac{A^2 t^2}{2!} + \cdots + \frac{A^K t^K}{K!} + \frac{A^{K+1} t^{K+1}}{(K+1)!} + \cdots = M + R \quad (\text{A-2})$$

其中

$$M = I + At + \frac{A^2 t^2}{2!} + \cdots + \frac{A^K t^K}{K!} = \sum_{k=0}^K \frac{A^k t^k}{k!} \quad (\text{A-3})$$

$$R = \frac{A^{K+1} t^{K+1}}{(K+1)!} + \frac{A^{K+2} t^{K+2}}{(K+2)!} + \cdots = \sum_{k=K+1}^{\infty} \frac{A^k t^k}{k!} \quad (\text{A-4})$$

$M$  称为  $e^{At}$  的近似矩阵,  $R$  称为  $e^{At}$  的剩余矩阵。由于  $M$  只有有限项,所以经过反复迭代,它总是可以求值的。问题是  $K$  需取多大,才能符合精确度要求。

现在假定给出精度  $b$ 。如果能够使得剩余矩阵  $R$  的每一个元素  $r_{ij}$  比起近似矩阵  $M$  的相应元素  $m_{ij}$  是这样的小,比如满足条件

$$|r_{ij}| \leq 10^{-b} |m_{ij}| \quad (\text{A-5})$$

则自然就可认为  $e^{At} \approx M$ 。因此,问题又转化为如何求出  $R$  的上限元素  $|r_{ij}|_{\max}$ 。

我们利用矩阵范数

$$\|A\| = \sum_{i=1}^n \sum_{j=1}^n |a_{ij}| \quad (\text{A-6})$$

及其性质

$$\|A^k\| \leq \|A\|^k, \quad k = 1, 2, \dots$$

对于矩阵  $R$ , 即有

$$\begin{aligned} |r_{ij}|_{\max} &\leq \|R\| = \left\| \sum_{k=K+1}^{\infty} \frac{A^k t^k}{k!} \right\| = \sum_{k=K+1}^{\infty} \frac{\|A^k t^k\|}{k!} \\ &\leq \sum_{k=K+1}^{\infty} \frac{\|A\|^k}{k!} \\ &= \frac{\|A\|^{K+1}}{(K+1)!} + \frac{\|A\|^{K+2}}{(K+2)!} + \dots \\ &= \frac{\|A\|^{K+1}}{(K+1)!} \left( 1 + \frac{\|A\|}{K+2} + \frac{\|A\|^2}{(K+2)(K+3)} + \dots \right) \\ &\leq \frac{\|A\|^{K+1}}{(K+1)!} \left( 1 + \frac{\|A\|}{K+2} + \frac{\|A\|^2}{(K+2)^2} + \dots \right) \end{aligned}$$

今令

$$\epsilon = \frac{\|A\|}{K+2} \quad (\text{A-7})$$

则上不等式右边括号是公比为  $\epsilon$  的等比级数, 即有

$$\begin{aligned} |r_{ij}|_{\max} &\leq \frac{\|A\|^{K+1}}{(K+1)!} (1 + \epsilon + \epsilon^2 + \dots) \\ &= \frac{\|A\|^{K+1}}{(K+1)!} \frac{1}{1 - \epsilon} \end{aligned} \quad (\text{A-8})$$

用它代入 (A-5) 式, 如果满足不等式, 迭代计算即可结束。

可以总结迭代计算  $e^{At}$  的步骤如下:

- 1) 按 (A-6) 式计算  $A$  的范数  $\|A\|$ 。
- 2) 选择  $K$  的一个任意初始值, 比如取  $K = 1$ 。
- 3) 按 (A-7) 式计算  $\epsilon$ 。
- 4) 按 (A-8) 计算  $r_{ij}$ 。
- 5) 按 (A-3) 计算  $m_{ij}$ 。
- 6) 按 (A-5) 式对  $|r_{ij}|$  与  $|m_{ij}|$  进行比较。
- 7) 如果不等式 (A-5) 不满足, 则把  $K$  值加 1, 重复步骤 3) 至 6); 否则, 迭代结束。

这里需要说明的是, 时间增量  $t$  必须选择适当。  $t$  太大会使级数  $M$  过早收敛, 以致达不到精度;  $t$  太小, 则整个计算过程时间过长。通常取

$$t = \frac{0.9}{n |a_{ij}|_{\max}}$$

其中  $n$  为矩阵  $A$  的阶数,  $|a_{ij}|_{\max}$  为矩阵  $A$  中绝对值最大的元素。

下面设计计算  $e^{At}$  的通用子程序。子程序语句为

SUBROUTINE EAT (A, N, T, H, ANO, ACC, MAX, I I, AP, TT)

其中哑元说明如下:

- A       $N \times N$  个元素的二维实型数组, 输入参数; 表示已知矩阵  $A$ .
- N      整型变量, 输入参数; 矩阵  $A$  的阶数  $n$ .
- T      实型变量, 输出参数;  $e^{At}$  中的时间增量  $t$ , 由程序算出.
- H       $N \times N$  个元素的二维实型数组, 输出参数;  $e^{At}$  的近似矩阵  $M$ .
- ANO    实型变量, 输出参数; 矩阵  $A$  的范数  $\|A\|$ .
- ACC    实型变量, 输入参数; 计算精度  $10^{-6}$ .
- MAK    整型变量, 输入参数; 收敛的最大项数.
- I I    整型变量, 输出参数; 作信息标志: 当输出 I I = 0 时, 表示结果已满足精度, 迭代未达到收敛的最大项数; 当输出 I I = 1 时, 则表示相反情况.
- AP     $N \times N$  个元素的二维实型数组; 工作数组.
- TT     $N$  个元素的一维实型数组; 工作数组.

此外, 程序还将用到的另外几个主要工作单元是:

- AIJM    表示矩阵  $A$  中绝对值最大的元素.
- RIJM    表示剩余矩阵  $R$  中元素的上限  $|r_{ij}|_{\max}$ .
- EPS    表示算法中的  $\epsilon$ , 即  $\epsilon = \frac{\|A\|t}{K+2}$ .

FORTTRAN 子程序:

```

SUBROUTINE EAT (A, N, T, H, ANO, ACC, MAK, I I, AP, TT)
  DIMENSION A(N, N), H(N, N), AP(N, N), TT(N)
  II = 0
  AIJM = A(1, 1)
  DO 10 I = 1, N
    DO 10 J = 1, N
      IF (ABS (A(I, J)) .GT. ABS (AIJM)) AIJM = A(I, J)
10  CONTINUE
  T = 0.9/(N * ABS (AIJM))
  ANO = 0.0
  AMX = 0.0
  DO 30 I = 1, N
    DO 20 J = 1, N
20  AMX = AMX + ABS (A(I, J))
  ANO = AMX
30  CONTINUE
  KK = 1
  DO 40 I = 1, N

```

```

DO 35 J = 1, N
H(I, J) = A(I, J) * T
35 AP(I, J) = H(I, J)
H(I, I) = H(I, I) + 1.0
40 CONTINUE
RIJN = ANO * T
50 EPS = (ANO * T) / (KK + 2)
RIJN = RIJN * (ANO * T) / (KK + 1)
RIJM = ABS (RIJN / (1 - EPS))
SIZE = RIJM / ACC
DO 60 I = 1, N
DO 60 J = 1, N
IF (ABS (H(I, J))) .GE. SIZE) GO TO 120
IF (KK .GE. MAK) GO TO 110
60 CONTINUE
KK = KK + 1
DO 80 J = 1, N
DO 70 I = 1, N
TT(I) = 0.0
DO 70 K = 1, N
70 TT(I) = TT(I) + AP(J, K) * A(K, I) * T / KK
DO 80 K = 1, N
80 AP(J, K) = TT(K)
DO 100 I = 1, N
DO 100 J = 1, N
90 H(I, J) = H(I, J) + AP(I, J)
100 CONTINUE
GO TO 50
110 II = I
120 RETRUN
END

```

**算例** 计算矩阵指数  $e^{At}$ , 其中

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

我们取  $ACC = 10^{-7}$ ,  $MAK = 20$ .

解题程序:

```

C      THE MAIN PROGRAM
      DIMENSION A(5, 5), H(5, 5), AP(5, 5), TT(5)
      READ *, N, ACC, MAK
      READ *, ((A(I, J), J=1, N), I=1, N)
      WRITE (6, 200) N, ACC, MAK
200    FORMAT (// 2HN=, I3, 5X, 4HACC=, F12.5, 5X, 4HMAK=, I3)
      WRITE (6, 210)
210    FORMAT (1X, 8HA-MATRIX, 2X, 2HIS/)
      DO 220 I=1, N
220    WRITE (6, 230) (A(I, J), J=1, N)
230    FORMAT (5E16.7/)
      CALL EAT (A, N, T, H, ANO, ACC, MAK, II, AP, TT)
      WRITE (6, 240) T, ANO, II
240    FORMAT (//2HT=, F12.5, 5X, 6HANORM=, F12.5, 5X, 3HII=, I3)
      WRITE (6, 250)
250    FORMAT (1X, 8HH-MATRIX, 2X, 2HIS/)
      DO 260 I=1, N
260    WRITE (6, 220) (H(I, J), J=1, N)
      STOP
      END

      SUBROUTINE EAT (A, N, T, H, ANO, ACC, MAK, II, PC, TT)
      : (本子程序段体)
      END
  
```

计算输出结果:

```

N = 5          ACC = 0.10000E-06          MAK = 20

A-MATRIX      IS
- 0.2000000E+01  0.1000000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+00
  0.2000000E+01 -0.4000000E+01  0.2000000E+01  0.0000000E+00  0.0000000E+00
  0.0000000E+01  0.3000000E+01 -0.6000000E+01  0.3000000E+01  0.0000000E+00
  0.0000000E+01  0.0000000E+00  0.4000000E+01 -0.8000000E+01  0.4000000E+01
  0.0000000E+01  0.0000000E+00  0.0000000E+00  0.5000000E+01 -0.5000000E+01

T = 0.22500E-01      ANORN = 0.49000E+02      II = 0

H-MATRIX      IS
  0.9564744E+00  0.2104711E-01  0.4631394E-03  0.1019142E-01  0.2292112E-06
  0.4209422E-01  0.9157696E+00  0.4028243E-01  0.1329415E-02  0.4007806E-04
  0.1389418E-02  0.6042365E-01  0.8776828E+00  0.5785482E-01  0.2638563E-02
  0.4076569E-04  0.2658831E-02  0.7713977E-01  0.8421811E+00  0.7797921E-01
  0.1146056E-05  0.1001951E-03  0.4397605E-02  0.9747401E-01  0.8980270E+00
  
```



## 第9章 常微分方程数值解法

### 9-1 引言: 基本概念

以常微分方程形式表达的数学模型是很常见的一种数学模型, 特别是在描述系统的动态过程时, 便常常归结为以时间为自变量的常微分方程或方程组. 例如, 生长和蜕变等物理过程, 其最简单的数学模型就是一阶常系数微分方程

$$\frac{du}{dt} = \alpha u$$

其中系数  $\alpha$  为蜕变概率. 又如, 简谐振子的运动规律可描述为二阶常微分方程

$$a \frac{d^2 s}{dt^2} = -cs$$

其中,  $s$  为振子位移,  $a > 0$  为质量,  $c > 0$  为弹性常数,  $-cs$  为弹性恢复力, 负号表示恢复力与位移反向. 在实际问题中, 增衰和谐振两种典型往往是耦合并存的, 因此, 又可提出阻尼谐振方程

$$a \frac{d^2 v}{dt^2} + b \frac{dv}{dt} + cv = 0$$

或具有外界驱动函数  $f(t)$  的方程

$$a \frac{d^2 v}{dt^2} + b \frac{dv}{dt} + cv = f(t)$$

其中, 阻尼项  $b \frac{dv}{dt}$ ,  $b \geq 0$ , 在机械振动中是阻力, 正比而反向于速度; 在电路振荡中  $a = L$  (电感),  $b = R$  (电阻),  $c = \frac{1}{C}$  ( $C$  为电容).

对于常微分方程, 实际中需要求解的是所谓定解问题. 定解问题分“初值问题”和“边值问题”两种提法. 前者是指给定方程和解的初始条件, 如

$$\begin{cases} \mu''(t) = f(t, \mu, \mu'), & t \in [t_0, T] \\ \mu(t_0) = u_0, & \mu'(t_0) = u'_0 \end{cases}$$

求满足方程和初始条件的解; 后者是指给定方程和解的两端点条件, 如

$$\begin{cases} u''(t) = f(t, u, u''), & t \in [a, b] \\ u(a) = \alpha, & u(b) = \beta \end{cases}$$

求满足方程和边界条件的解.

从微分方程基础教程我们知道, 已就最高阶导数解出的高阶微分方程 (包括其方程组), 原则上总可以化为一阶方程组来求解. 如  $m$  阶方程构成的初值问题:

$$\begin{cases} u^{(m)} = f(t, u, u', \dots, u^{(m-1)}) \\ u(t_0) = u_0, \quad u'(t_0) = u'_0, \dots, \quad u^{(m-1)}(t_0) = u_0^{(m-1)} \end{cases} \quad (9.1)$$

引进新变量

$$u_1 = u, \quad u_2 = u', \quad u_3 = u'', \quad \dots, \quad u_m = u^{(m-1)}$$

则(9.1)可化为下列一阶方程组的初值问题:

$$\begin{cases} u_1' = u_2 \\ u_2' = u_3 \\ \dots\dots\dots \\ u_{m-1}' = u_m \\ u_m' = f(t, u_1, u_2, \dots, u_m) \\ u_1(t_0) = u_1^0 \\ u_2(t_0) = u_2^0 \\ \dots\dots\dots \\ u_m(t_0) = u_m^{(m-1)} \end{cases} \quad (9.2)$$

可知,不失一般性,我们可以只讨论一阶方程组的情形.

设一阶方程组的初值问题为

$$\begin{cases} u_i' = f_i(t, u_1, u_2, \dots, u_n) \\ u_i(t_0) = u_i^0 \end{cases} \quad (i = 1, 2, \dots, n) \quad (9.3)$$

记成向量形式,令

$$\begin{aligned} \mathbf{u} &= [u_1, u_2, \dots, u_n]^T \\ \mathbf{u}^0 &= [u_1^0, u_2^0, \dots, u_n^0]^T \\ \mathbf{f} &= [f_1, f_2, \dots, f_n]^T \end{aligned}$$

则一阶方程组的初值问题(9.3)可表为

$$\begin{cases} \mathbf{u}' = \mathbf{f}(t, \mathbf{u}) \\ \mathbf{u}(t_0) = \mathbf{u}^0 \end{cases} \quad (9.4)$$

由此可见,为了叙述简单,我们又可以只讨论一个一阶方程的初值问题

$$\begin{cases} u' = f(t, u), \quad t \in [t_0, T] \\ u(t_0) = u_0 \end{cases} \quad (9.5)$$

只要把  $u, u_0, f$  理解成向量,则所得结果容易推广到方程组的情形.

下面,我们综述微分方程基础教程中证明了的结果.

**定理 1** (解存在唯一性定理) 对初值问题(9.5), 若  $f(t, u)$  在区域  $G = \{t_0 \leq t \leq T, |u| < +\infty\}$  内连续且对  $u$  满足李普希兹条件, 即存在常数  $L$ , 使得当  $(t, u_1)$  和  $(t, u_2) \in G$  时有

$$|f(t, u_1) - f(t, u_2)| \leq L|u_1 - u_2|$$

则初值问题(9.5)的解存在唯一且连续可微.

在实际问题中, 方程右端  $f(t, u)$  和初值  $u_0$  通常是通过测量获得的, 有一定的误差, 因此, 我们还需要研究当它们有微小变化时对解的影响情况. 为此, 引入了刻划这种连续相依关系的概念.

**定义** 如果初值问题(9.5)存在唯一解  $u(t)$ , 又有常数  $K$ , 使得对任意  $\varepsilon > 0$ , 当

$$\begin{aligned} |f(t, u) - \tilde{f}(t, u)| &< \varepsilon, \quad t_0 \leq t \leq T, \quad |u| < +\infty \\ |u_0 - \tilde{u}_0| &< \varepsilon \end{aligned}$$

时, 初值问题

$$\begin{cases} v' = \tilde{f}(t, v), \quad t \in [t_0, T] \\ v(t_0) = \tilde{u}_0 \end{cases}$$

的解  $v(t)$  唯一存在, 且满足

$$|u(t) - v(t)| \leq K\varepsilon$$

则称初值问题 (9.5) 是适定的.

**定理 2** 如果  $f(t, u)$  在  $G$  上连续, 且关于  $u$  满足李普希兹条件, 则初值问题 (9.5) 是适定的.

现在给出关于常微分方程数值解的概念. 如所知道, 常微分方程只在一些特殊类型才能获得解析解, 大部分问题则只能求数值解.

数值解就是寻求解  $u(t)$  在一系列离散节点

$$t_1 < t_2 < \cdots < t_n < \cdots$$

上的解  $u(t_i)$  的近似值  $u_1, u_2, \cdots, u_n, \cdots$ . 注意这里是  $u_i \approx u(t_i)$ . 相邻两个节点的距离  $h = t_{n+1} - t_n$  称为步长, 通常假定  $h$  为定长, 这时有

$$t_n = t_0 + nh, \quad n = 0, 1, \cdots$$

常微分方程初值问题的数值解法通常具有“步进式”特点, 即由  $(t_0, u_0)$  出发, 沿着节点排列的次序一步步向前推进. 其算法总是由已知信息  $u_n, u_{n-1}, \cdots, u_1, u_0$ , 给出计算  $u_{n+1}$  的递推公式.

## 9-2 尤拉法与预测-校正法

### 1. 尤拉法及其精度分析

现在考虑初值问题

$$\begin{cases} u' = f(t, u), & t \in [t_0, T] \end{cases} \quad (9.6)$$

$$u(t_0) = u_0 \quad (9.7)$$

通常, 可以通过不同途径来导出尤拉法. 下面我们由几何直观入手. 我们知道, 微分方程 (9.6) 的解  $u = u(t)$  也称为积分曲线, 积分曲线上每一点  $(t, u)$  的切线斜率等于

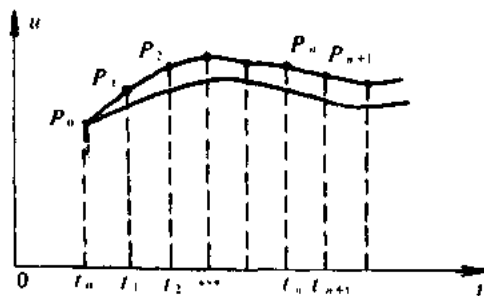


图 9-1

$f(t, u)$  的值. 因此, 若以  $f(t, u)$  在  $t-u$  平面上的值形成一个方向场, 则积分曲线上每一点的切线方向与所作方向场在该点的方向一致.

如图 9-1 所示, 从初始点  $P_0(t_0, u_0)$  出发, 按方向场在该点的方向推进到  $t = t_1$  上的一点  $P_1(t_1, u_1)$ , 又从  $P_1(t_1, u_1)$  出发, 按方向场在该点的方向推进到  $t = t_2$  上的一点  $P_2(t_2, u_2)$ , 循此前进, 可作出折线  $\overline{P_0 P_1 P_2 \cdots}$ . 假设我们已经推进到点  $P_n(t_n, u_n)$  过  $P_n(t_n, u_n)$ , 按方向场的方向又推进到点  $P_{n+1}(t_{n+1}, u_{n+1})$ , 则顶点  $P_n$  与  $P_{n+1}$  的坐标满足关系

$$\frac{u_{n+1} - u_n}{t_{n+1} - t_n} = f(t_n, u_n)$$

注意  $t_{n+1} - t_n = h$ , 从而可得

$$u_{n+1} = u_n + hf(t_n, u_n) \quad (9.8)$$

公式 (9.8) 称为尤拉 (Euler) 公式, 应用这个公式, 由已知的初值  $u_0$  出发, 便可逐一求出初值问题的数值解

$$\begin{aligned} u_1 &= u_0 + hf(t_0, u_0) \\ u_2 &= u_1 + hf(t_1, u_1) \\ &\dots\dots\dots \\ u_{n+1} &= u_n + hf(t_n, u_n) \\ &\dots\dots\dots \end{aligned}$$

这就是尤拉法或称折线法。像尤拉公式 (9.8) 这样的方程, 有时也称为差分格式或差分方程。

**例 1** 设初值问题为

$$\begin{cases} u' = -\frac{0.9}{1+2t}u, & t \in [0, 0.1] \\ u(0) = 1 \end{cases}$$

试用尤拉法求当  $h = 0.02$  时的数值解。

**解** 根据尤拉公式 (9.8) 有

$$\begin{aligned} u_{n+1} &= u_n - h \frac{0.9}{1+2t_n} u_n \\ &= \left(1 - \frac{0.018}{1+2t_n}\right) u_n, \quad n = 0, 1, 2, 3, 4 \end{aligned}$$

因此, 可具体计算结果如下:

| $n$   | 0      | 1      | 2      | 3      | 4      | 5      |
|-------|--------|--------|--------|--------|--------|--------|
| $t_n$ | 0.00   | 0.02   | 0.04   | 0.06   | 0.08   | 0.10   |
| $u_n$ | 1.0000 | 0.9820 | 0.9655 | 0.9489 | 0.9336 | 0.9192 |

如果对比初值问题的真解

$$u(t) = (1+2t)^{-0.45}$$

在  $t_n$  处的值  $u(t_n)$  以及近似值  $u_n$  的误差  $\varepsilon_n = u(t_n) - u_n$ , 我们还有

| $n$             | 0      | 1      | 2      | 3      | 4      | 5      |
|-----------------|--------|--------|--------|--------|--------|--------|
| $u(t_n)$        | 1.0000 | 0.9825 | 0.9660 | 0.9503 | 0.9354 | 0.9213 |
| $\varepsilon_n$ | 0      | 0.0005 | 0.0005 | 0.0014 | 0.0018 | 0.0021 |

从计算结果可见, 随着  $n$  的增大, 误差也在增大, 从图 9-1 也可看出, 折线“越偏越离”, 自然不可能有高的精度。

下面我们简单作尤拉法的精度分析。

假定  $u(t_n) = u_n$ , 则有

$$u'(t_n) = f(t_n, u(t_n)) = f(t_n, u_n)$$

因此,利用泰勒公式和尤拉公式并在  $u''(t)$  有界的情况下,可得

$$\begin{aligned} u(t_{n+1}) - u_{n+1} &= \left[ u(t_n) + u'(t_n)h + \frac{u''(t_n)}{2!}h^2 + \dots \right] \\ &\quad - [u_n + hf(t_n, u_n)] \\ &= \left[ u_n + hf(t_n, u_n) + \frac{u''(t_n)}{2!}h^2 + \dots \right] \\ &\quad - [u_n + hf(t_n, u_n)] \\ &= \frac{u''(t_n)}{2}h^2 + \dots \\ &= O(h^2) \end{aligned}$$

通常,在微分方程数值解法中引入如下概念.

**定义 1** 假定  $u(t_{n+1})$  是初值问题的准确解,又假定  $u(t_n) = u_n$ ,而  $u_{n+1}$  是通过  $u_n$  从数值公式(差分格式)求出的近似值,则  $u(t_{n+1})$  与  $u_{n+1}$  之差

$$u(t_{n+1}) - u_{n+1}$$

称为该数值方法(差分格式)的**局部截断误差**. (局部截断误差表示在利用差分格式计算  $u_{n+1}$  时方法本身的误差)

**定义 2** 如果一种数值方法的局部截断误差为  $O(h^{p+1})$ ,  $h$  为步长,就称该数值方法(差分格式)具有  $p$  阶精度.

由上可知,尤拉法的局部截断误差为  $O(h^2)$ ,精度为 1 阶.也就是说,当  $h \rightarrow 0$  时,局部截断误差与  $h^2$  是同阶的无穷小量.

关于误差分析的更一般讨论,我们将在 9-5 节给出.

## 2. 改进的尤拉法与预测-校正法

尤拉法可加以改进.

我们将方程离散化,设法消除导数项.离散化的一种基本方法是直接用差商(均差)代替导数.

分三种情形来考虑:

(1) 考虑用向前差商的情形:在点  $t_n$  列出方程

$$u'(t_n) = f(t_n, u(t_n))$$

用向前差商  $\frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n}$  近似代替导数  $u'(t_n)$ ,于是得

$$u(t_{n+1}) \approx u(t_n) + hf(t_n, u(t_n))$$

这时若以  $u_n$  代替  $u(t_n)$  代入上式右端算出  $u_{n+1}$ ,并以它作为  $u(t_{n+1})$  的近似值,即

$$u_{n+1} = u_n + hf(t_n, u_n)$$

这是我们上面已经导出的尤拉公式(9.8).

(2) 考虑用向后差商的情形:在点  $t_{n+1}$  列出方程

$$u'(t_{n+1}) = f(t_{n+1}, u(t_{n+1}))$$

用向后差商  $\frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n}$  近似代替导数  $u'(t_{n+1})$ ,于是得

$$u(t_{n+1}) \approx u(t_n) + hf(t_{n+1}, u(t_{n+1}))$$

类似于上一种情形,我们可得

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}) \quad (9.9)$$

这称为**后退尤拉公式**。然而,我们注意到,尤拉公式(9.8)是关于 $u_{n+1}$ 的一个直接计算公式,称显式公式;后退尤拉公式(9.9)右端还含有 $u_{n+1}$ ,它实际是关于 $u_{n+1}$ 的一个函数方程,称隐式公式。隐式公式通常用迭代法求解。例如可用下面迭代程序:

1) 用尤拉公式算出迭代初值

$$u_{n+1}^{(0)} = u_n + hf(t_n, u_n)$$

2) 反复代入后退尤拉公式右端算出

$$\begin{aligned} u_{n+1}^{(1)} &= u_n + hf(t_{n+1}, u_{n+1}^{(0)}) \\ u_{n+1}^{(2)} &= u_n + hf(t_{n+1}, u_{n+1}^{(1)}) \\ &\dots\dots\dots \\ u_{n+1}^{(k+1)} &= u_n + hf(t_{n+1}, u_{n+1}^{(k)}) \\ &\dots\dots\dots \end{aligned}$$

容易证明,这个迭代过程是收敛的。事实上,由后退尤拉公式及上述迭代公式有

$$u_{n+1} - u_{n+1}^{(k+1)} = h[f(t_{n+1}, u_{n+1}) - f(t_{n+1}, u_{n+1}^{(k)})]$$

根据李普希兹条件可得

$$|u_{n+1} - u_{n+1}^{(k+1)}| \leq hL |u_{n+1} - u_{n+1}^{(k)}|$$

可知只要选取 $h$ ,使得 $hL < 1$ ,则当 $k \rightarrow \infty$ 时, $u_{n+1}^{(k)} \rightarrow u_{n+1}$ ,即迭代过程收敛。

对于局部截断误差 $u(t_{n+1}) - u_{n+1}$ ,可利用泰勒公式来进行分析。设 $u(t_n) = u_n$ ,因为

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}) = u(t_n) + hf(t_{n+1}, u_{n+1})$$

其中  $f(t_{n+1}, u_{n+1}) = f(t_{n+1}, u(t_{n+1})) + f_u(t_{n+1}, \eta)[u_{n+1} - u(t_{n+1})]$

又  $f(t_{n+1}, u(t_{n+1})) = u'(t_{n+1}) = u'(t_n) + hu''(t_n) + \dots$

所以有

$$u_{n+1} = u(t_n) + h[u'(t_n) + hu''(t_n) + \dots] + hf_u(t_{n+1}, \eta)[u_{n+1} - u(t_{n+1})]$$

比较泰勒展式

$$u(t_{n+1}) = u(t_n) + hu'(t_n) + \frac{h^2}{2} u''(t_n) + \dots$$

相减可得

$$u(t_{n+1}) - u_{n+1} = hf_u(t_{n+1}, \eta)[u(t_{n+1}) - u_{n+1}] - \frac{h^2}{2} u''(t_n) + \dots$$

移项整理又得

$$[u(t_{n+1}) - u_{n+1}][1 - hf_u(t_{n+1}, \eta)] = -\frac{h^2}{2} u''(t_n) + \dots$$

$$\begin{aligned} u(t_{n+1}) - u_{n+1} &= \frac{1}{1 - hf_u(t_{n+1}, \eta)} \left[ -\frac{h^2}{2} u''(t_n) + \dots \right] \\ &= [1 + hf_u(t_{n+1}, \eta) + \dots] \left[ -\frac{h^2}{2} u''(t_n) + \dots \right] \\ &= -\frac{h^2}{2} u''(t_n) + \dots \\ &= O(h^2) \end{aligned}$$

可见,后退尤拉公式并没有比尤拉公式提高精度。但是,由对比局部截断误差式可见,如果将尤拉公式和后退尤拉公式作算术平均,则可消去局部截断误差的主要部分  $\pm \frac{h^2}{2} u''(t_n)$ , 使精度提高到  $O(h^3)$ 。为此,由 (9.8) 和 (9.9) 取算术平均可得

$$u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, u_{n+1})] \quad (9.10)$$

这称为**梯形公式**。它也是隐式公式,故通常也用迭代法求解。例如可用下列迭代程序:

$$\begin{cases} u_{n+1}^{(0)} = u_n + hf(t_n, u_n) \\ u_{n+1}^{(k+1)} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, u_{n+1}^{(k)})], \quad k = 0, 1, 2, \dots \end{cases}$$

类似地,也容易证明,当选取  $h$  使得  $\frac{h}{2} L < 1$  时,这个迭代过程是收敛的。

梯形公式每次迭代计算都要重新计算函数值,工作量大。为了简化,联合尤拉公式和梯形迭代公式,人们提出了**改进的尤拉公式**:

1) 算出  $\bar{u}_{n+1} = u_n + hf(t_n, u_n)$ ;

2) 迭代一次  $u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, \bar{u}_{n+1})]$  前者称预测值,后者称校正值。它们也可合并为

$$u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_n + h, u_n + hf(t_n, u_n))] \quad (9.11)$$

然而,改进的尤拉公式却存在预测公式精度低,校正公式精度高的不匹配现象。为此,我们考虑离散化的第三种情况:

(3) 用中心差商代替导数的情形: 在点  $t_n$  列出方程

$$u'(t_n) = f(t_n, u(t_n))$$

用中心差商  $\frac{u(t_{n+1}) - u(t_{n-1}))}{2h}$  近似代替导数  $u'(t_n)$ , 于是可导出**尤拉两步公式**

$$u_{n+1} = u_{n-1} + 2hf(t_n, u_n) \quad (9.12)$$

所谓两步,就是计算  $u_{n+1}$  时,要调用前两步的信息  $u_{n-1}, u_n$ 。在这之前,我们导出的方法都是单步法,即计算  $u_{n+1}$  时,只用到前一步的信息  $u_n$ 。对单步法,只须给出初值  $u_0$ , 按公式就可依次计算出  $u_1, u_2, \dots$ 。对两步法,除了给出  $u_0$ , 还要设法提供  $u_1$ , 才能启动公式算出  $u_2, u_3, \dots$ 。

联合尤拉两步公式和梯形公式,可得新的**预测-校正公式**:

$$\begin{cases} \text{预测} & \bar{u}_{n+1} = u_{n-1} + 2hf(t_n, u_n) \\ \text{校正} & u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, \bar{u}_{n+1})] \end{cases} \quad (9.13)$$

关于截断误差分析,同样利用泰勒公式可以推出: 在预测公式中有

$$u(t_{n+1}) - \bar{u}_{n+1} = \frac{h^3}{3} u'''(t_n) + \dots$$

在校正公式中有

$$u(t_{n+1}) - u_{n+1} = -\frac{h^3}{12} u'''(t_n) + \dots$$

这就是说,校正误差与预测误差之比大约是  $-\frac{1}{4}$ ,即

$$\frac{u(t_{n+1}) - u_{n+1}}{u(t_{n+1}) - \bar{u}_{n+1}} \approx -\frac{1}{4}$$

因而变换上式可得所谓事后估计式:

$$\begin{cases} u(t_{n+1}) - \bar{u}_{n+1} \approx -\frac{4}{5}(\bar{u}_{n+1} - u_{n+1}) \\ u(t_{n+1}) - u_{n+1} \approx \frac{1}{5}(\bar{u}_{n+1} - u_{n+1}) \end{cases} \quad (9.14)$$

可见,利用估计出的误差(9.14)作为计算结果的一种补偿,又有可能使精度得到改善. 设  $a_n$  和  $b_n$  分别表示第  $n$  步的预测值和校正值,我们就以  $a_{n+1} - \frac{4}{5}(a_{n+1} - b_{n+1})$  和  $b_{n+1} + \frac{1}{5}(a_{n+1} - b_{n+1})$  分别作  $a_{n+1}$  和  $b_{n+1}$  的改进值. 不过,在校正值  $b_{n+1}$  未算出之前,我们可用上一次的偏差值  $a_n - b_n$  代替  $a_{n+1} - b_{n+1}$ . 这样,可设计带有补偿的预测-校正公式如下:

$$\begin{cases} \text{预测} & a_{n+1} = u_{n+1} + 2hf(t_n, u_n) \\ \text{补偿} & \bar{a}_{n+1} = a_{n+1} - \frac{4}{5}(a_n - b_n) \\ \text{校正} & b_{n+1} = u_n + \frac{h}{2}[f(t_n, u_n) + f(t_{n+1}, \bar{a}_{n+1})] \\ \text{补偿} & u_{n+1} = b_{n+1} + \frac{1}{5}(a_{n+1} - b_{n+1}) \end{cases}$$

采用这组公式计算  $u_{n+1}$  时,要用到前一步的  $u_n$ ,  $a_n - b_n$  和更前一步的  $u_{n-1}$ , 因此在启动计算之前,必须给出开始值  $u_1$  和  $a_1 - b_1$ . 其中  $u_1$  可用其它单步法计算,  $a_1 - b_1$  一般可取为 0. 计算实践表明,这种简单的处理方法通常已经可获得满意的结果.

最后,我们将给出用尤拉法和改进的尤拉法分别求解初值问题的一个实例. 用改进的尤拉法求解初值问题

$$\begin{cases} u' = f(t, u), & t \in [t_0, T] \\ u(t_0) = u_0 \end{cases}$$

的算法可以设计如下:

- 1) 输入端点  $t_0, T$ , 等分数  $n$  和初值  $u_0$ .
- 2) 计算  $h = (T - t_0)/n$ , 令  $t = t_0$ ,  $u = u_0$ , 输出  $(t, u)$ .
- 3) 计算  $\underline{u}_p = u + hf(t, u)$ ,  
 $\underline{u}_c = u + hf(t + h, \underline{u}_p)$ ,  
 $\underline{u} = \frac{1}{2}(\underline{u}_p + \underline{u}_c)$ .
- 4) 令  $t = t + h$ , 输出  $(t, u)$ .
- 5) 如果  $t \leq T$ , 则转 3), 否则, 计算结束.

**例 2** 分别用尤拉法和改进的尤拉法解初值问题

$$\begin{cases} u' = 2u + 2t^2, & t \in [0, 1] \\ u(0) = 1 \end{cases}$$



取  $n = 10$ , 并将计算结果与准确解

$$u = \frac{3}{2} e^{2t} - t^2 - t - \frac{1}{2}$$

比较。

**解** 具体计算结果如下:

| $t_i$ | 用尤拉法求出的 $u_i$ | 用改进的尤拉法求出的 $u_i$ | 精确解    |
|-------|---------------|------------------|--------|
| 0     | 1             | 1                | 1      |
| 0.1   | 1.2000        | 1.2210           | 1.2221 |
| 0.2   | 1.4420        | 1.4923           | 1.4977 |
| 0.3   | 1.7384        | 1.8284           | 1.8432 |
| 0.4   | 2.1041        | 2.2466           | 2.2983 |
| 0.5   | 2.5569        | 2.7680           | 2.8274 |
| 0.6   | 3.1183        | 3.4176           | 3.5202 |
| 0.7   | 3.8139        | 4.2257           | 4.3928 |
| 0.8   | 4.6747        | 5.2288           | 5.4895 |
| 0.9   | 5.7376        | 6.4704           | 6.8645 |
| 1.0   | 7.0472        | 8.0032           | 8.5836 |

## 9-3 龙格-库塔法

### 1. 龙格-库塔法及其 FORTRAN 程序

继续考虑初值问题

$$\begin{cases} u' = f(t, u), & t \in [t_0, T] \\ u(t_0) = u_0 \end{cases}$$

由所给方程和微分中值定理, 我们有

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + hu'(t_n + \theta_n h) \\ &= u(t_n) + hf(t_n + \theta_n h, u(t_n + \theta_n h)), \quad 0 < \theta_n < 1 \end{aligned} \quad (9.16)$$

记  $\bar{K} = u'(t_n + \theta_n h)$ , 并称  $\bar{K}$  为  $[t_n, t_{n+1}]$  上的平均斜率。由于  $\theta_n$  无法具体确定, 因而平均斜率也无法具体计算。因此, 若能对平均斜率提供一种可行的数值逼近, 这就等于对初值问题提供了一种可行的数值算法。

回忆尤拉公式 (9.8), 可知它相当于 (9.16) 中取  $\theta_n = 0$ , 即简单地用斜率  $u'(t_n)$  代替平均斜率  $\bar{K}$ 。

回忆梯形公式 (9.10), 可知它相当于 (9.16) 中用两点处的斜率  $u'(t_n)$  与  $u'(t_{n+1})$  的算术平均代替平均斜率  $\bar{K}$ 。

再回忆改进的尤拉公式 (9.11), 或把它写成平均化形式

$$\begin{cases} u_{n+1} = u_n + \frac{h}{2} (K_1 + K_2) \\ K_1 = f(t_n, u_n) \\ K_2 = f(t_{n+1}, u_{n+1}) \end{cases}$$

可见, 它也是用两点的斜率  $K_1$  与  $K_2$  的算术平均来逼近  $\bar{K}$ , 只是其中  $K_1$  是斜率  $u'(t_n)$ ,

$K_2$  则是通过预测  $u_{n+1}$  而产生的  $t_{n+1}$  处的斜率。

尤格-库塔 (Runge-Kutta) 法的基本思想就是: 在  $[t_n, t_{n+1}]$  内多预测几个点处的斜率, 然后用其加权平均来逼近平均斜率  $\bar{K}$ , 以期获得更高精度的计算公式。下面分别讨论几种简单情形。

### (1) 二阶龙格-库塔法

用两点  $t_n$  和  $t_{n+p}$  (其中  $t_{n+p} = t_n + ph$ ,  $0 < p \leq 1$ ) 的斜率值  $K_1$  和  $K_2$  的线性组合作平均斜率  $\bar{K}$ , 并要求具有二阶精度。为此, 我们构造

$$\begin{cases} u_{n+1} = u_n + h(\lambda_1 K_1 + \lambda_2 K_2) \\ K_1 = f(t_n, u_n) \\ K_2 = f(t_{n+p}, u_n + phK_1) \end{cases} \quad (9.17)$$

其中  $\lambda_1, \lambda_2, p$  为待定参数; 并且通过确定这些参数来使上述公式具有二阶精度。

将 (9.17) 中的  $K_1, K_2$  代入第一式, 并将  $f(t_{n+p}, u_n + phK_1)$  在  $(t_n, u_n)$  处泰勒展开, 我们有

$$\begin{aligned} u_{n+1} &= u_n + h[\lambda_1 f(t_n, u_n) + \lambda_2 f(t_{n+p}, u_n + phf(t_n, u_n))] \\ &= u_n + h[\lambda_1 f_n + \lambda_2 (f_n + ph(f_t + f_u \cdot f)_n + O(h^2))] \\ &= u_n + (\lambda_1 + \lambda_2)hf_n + \lambda_2 ph^2(f_t + f_u \cdot f)_n + O(h^3) \end{aligned}$$

其中  $f_n$  和  $(f_t + f_u \cdot f)_n$  的下标  $n$  表示在  $(t_n, u_n)$  处取值,  $f_t$  表示对  $t$  求偏导数,  $f_u$  表示对  $u$  求偏导数。再与  $u(t_{n+1})$  在  $t_n$  处的泰勒展开式比较

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + hu'(t_n) + \frac{h^2}{2} u''(t_n) + O(h^3) \\ &= u(t_n) + hf_n + \frac{1}{2} h^2 (f_t + f_u \cdot f)_n + O(h^3) \end{aligned}$$

可见, 在假定  $u(t_n) = u_n$  情况下, 只需取

$$\begin{cases} \lambda_1 + \lambda_2 = 1 \\ \lambda_2 p = \frac{1}{2} \end{cases} \quad (9.18)$$

则 (9.17) 的局部截断误差  $u(t_{n+1}) - u_{n+1} = O(h^3)$ , 即它具有二阶精度。(9.17) 和 (9.18) 联立称为二阶龙格-库塔公式, 这是以  $\lambda_1, \lambda_2, p$  为参数满足条件 (9.18) 的一族公式。

特别地, 取  $\lambda_1 = 0, \lambda_2 = 1, p = \frac{1}{2}$ , 则二阶龙格-库塔公式成为

$$\begin{cases} u_{n+1} = u_n + hK_2 \\ K_1 = f(t_n, u_n) \\ K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1\right) \end{cases} \quad (9.19)$$

它可以看作变形的尤拉公式。

### (2) 三阶龙格-库塔法

用三点  $t_n, t_{n+p}, t_{n+q}$  ( $t_{n+p} = t_n + ph, t_{n+q} = t_n + qh, 0 < p \leq q \leq 1$ ) 的斜率  $K_1, K_2, K_3$  的线性组合作平均斜率  $\bar{K}$ , 并要求具有三阶精度。为此, 构造

$$\begin{cases} u_{n+1} = u_n + h(\lambda_1 K_1 + \lambda_2 K_2 + \lambda_3 K_3) \\ K_1 = f(t_n, u_n) \\ K_2 = f(t_n + ph, u_n + phK_1) \\ K_3 = f(t_n + qh, u_n + qh(rK_1 + sK_2)) \end{cases} \quad (9.20)$$

其中  $\lambda_1, \lambda_2, \lambda_3, p, q, r, s$  为待定参数。我们同样通过确定这些参数使上述公式具有三阶精度。

与推导二阶龙格-库塔公式类似,利用泰勒展式,经过较复杂的运算,我们可得,只需取

$$\begin{cases} r + s = 1 \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \\ \lambda_2 p + \lambda_3 q = \frac{1}{2} \\ \lambda_2 p^2 + \lambda_3 q^2 = \frac{1}{3} \\ \lambda_3 p q s = \frac{1}{6} \end{cases} \quad (9.21)$$

则公式(9.20)便具有三阶精度。(9.20)和(9.21)联立称为**三阶龙格-库塔公式**。它也是带多个参数的一族公式。

常用的一种具体形式是

$$\begin{cases} u_{n+1} = u_n + \frac{h}{6} (K_1 + 4K_2 + K_3) \\ K_1 = f(t_n, u_n) \\ K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1\right) \\ K_3 = f(t_n + h, u_n - hK_1 + 2hK_2) \end{cases} \quad (9.22)$$

### (3) 四阶龙格-库塔法

类似地,我们可以导出具有四阶精度的四阶龙格-库塔公式。最常用的一种**四阶龙格-库塔公式**是:

$$\begin{cases} u_{n+1} = u_n + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(t_n, u_n) \\ K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1\right) \\ K_3 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_2\right) \\ K_4 = f(t_n + h, u_n + hK_3) \end{cases} \quad (9.23)$$

也称为古典公式。它的向量形式是:

$$\begin{cases} u_{n+1} = u_n + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(t_n, u_n) \\ K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1\right) \\ K_3 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_2\right) \\ K_4 = f(t_n + h, u_n + hK_3) \end{cases} \quad (9.24)$$

具体写出来是:

$$\begin{cases} u_{i,n+1} = u_{in} + \frac{h}{6} (K_{i1} + 2K_{i2} + 2K_{i3} + K_{i4}) \\ K_{i1} = f_i(t_n, u_{1n}, u_{2n}, \dots, u_{Nn}) \\ K_{i2} = f_i\left(t_n + \frac{h}{2}, u_{1n} + \frac{h}{2} K_{11}, u_{2n} + \frac{h}{2} K_{21}, \dots, u_{Nn} + \frac{h}{2} K_{N1}\right) \\ K_{i3} = f_i\left(t_n + \frac{h}{2}, u_{1n} + \frac{h}{2} K_{12}, u_{2n} + \frac{h}{2} K_{22}, \dots, u_{Nn} + \frac{h}{2} K_{N2}\right) \\ K_{i4} = f_i(t_n + h, u_{1n} + hK_{13}, u_{2n} + hK_{23}, \dots, u_{Nn} + hK_{N3}) \\ i = 1, 2, \dots, N \end{cases}$$

这里  $u_{in}$  是第  $i$  个因变量  $u_i(t)$  在节点  $t_n = t_0 + nh$  的近似值.

**例 1** 用四阶龙格-库塔法解初值问题

$$\begin{cases} u' = t + u & (t \geq 0) \\ u(0) = 1 \end{cases}$$

取  $h = 0.1$ , 又  $t_0 = 0, u_0 = 1$ , 便有

$$K_1 = 0 + 1 = 1$$

$$K_2 = \left(0 + \frac{0.1}{2}\right) + \left(1 + \frac{0.1}{2} \times 1\right) = 1.1$$

$$K_3 = \left(0 + \frac{0.1}{2}\right) + \left(1 + \frac{0.1}{2} \times 1.1\right) = 1.105$$

$$K_4 = (0 + 0.1) + (1 + 0.1 \times 1.105) = 1.2105$$

故得

$$u_1 = 1 + \frac{0.1}{6} \times (1 + 2 \times 1.1 + 2 \times 1.105 + 1.2105) = 1.11034$$

类似可算得

$$u_2 = 1.24280, \dots$$

**例 2** 考虑两个方程的初值问题

$$\begin{cases} u' = f(t, u, v), & t \in [t_0, T] \\ v' = g(t, u, v), & t \in [t_0, T] \\ u(t_0) = u_0 \\ v(t_0) = v_0 \end{cases}$$

这时四阶龙格-库塔公式为

$$\left\{ \begin{array}{l} u_{n+1} = u_n + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \\ v_{n+1} = v_n + \frac{h}{6} (L_1 + 2L_2 + 2L_3 + L_4) \\ K_1 = f(t_n, u_n, v_n) \\ L_1 = g(t_n, u_n, v_n) \\ K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1, v_n + \frac{h}{2} L_1\right) \\ L_2 = g\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_1, v_n + \frac{h}{2} L_1\right) \\ K_3 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_2, v_n + \frac{h}{2} L_2\right) \\ L_3 = g\left(t_n + \frac{h}{2}, u_n + \frac{h}{2} K_2, v_n + \frac{h}{2} L_2\right) \\ K_4 = f(t_n + h, u_n + hK_3, v_n + hL_3) \\ L_4 = g(t_n + h, u_n + hK_3, v_n + hL_3) \end{array} \right.$$

这是一步法,利用节点  $t_n$  上的值  $u_n, v_n$ ,由公式中的后八个式顺序计算出  $K_1, L_1, K_2, L_2, K_3, L_3, K_4, L_4$ ,然后代入前二式,即可求得  $t_{n+1}$  上的近似解  $u_{n+1}$  和  $v_{n+1}$ 。

必须指出的是,龙格-库塔方法的推导基于泰勒展开,这意味着解具有较好的光滑性。如果解的光滑性本来就较差,则使用改进的尤拉公式反而有可能获得更高精度的结果。

四阶龙格-库塔法的标准子程序很简单,可以有不同的写法<sup>[C1,C3,B4]</sup>。下面给出两种形式的子程序。

形式(1)子程序语句为:

```
SUBROUTINE RGKT1 (N, T, H, U, FCT, F, P, W)
```

哑元设置如下:

N 整型变量,输入参数,方程的个数。

T 实型变量,输入参数和输出参数;开始存放自变量的初值,返回时存放初值加步长。

H 实型变量,输入参数;积分步长。

U N个元素的实型数组,开始存放未知函数的初值,返回时存放计算结果。

FCS 计算方程右端函数值的子程序,子程序语句为

```
SUBROUTINE FCS (T, U, F)
```

其中F是N个元素的实型数组,代表方程右端诸函数值。

F, P, W 均为N个元素的实型工作数组。

子程序(1):

```
SUBROUTINE RGKT1 (N, T, H, U, FCS, F, P, W)
```

```
DIMENSION U(N), F(N), P(N), W(N), C(5)
```

```

C(1) = 0.5 * H
C(2) = C(1)
C(3) = H C(1)
C(4) = H
C(5) = C(1) H
TS = T
DO 5 I = 1, N
P(I) = U(I)
5 W(I) = U(I)
DO 10 J = 1, 4
CALL FCS (T, U, F)
T = TS + C(J)
DO 10 I = 1, N
P(I) = C(J) * F(I) + W(I)
10 U(I) = C(J + 1) * F(I) / 3.0 + U(I)
RETURN
END

```

注：调用本子程序一次即对给定的步长和初值向前积分一步。需要继续积分必须再次调用本子程序。如此继续。

**算例** 解下列初值问题

$$\begin{cases} \dot{u}_1 = 1/(u_2 - t), & t \in [0, 1] \\ \dot{u}_2 = 1 - 1/u_1, & t \in [0, 1] \\ u_1(0) = u_2(0) = 1 \end{cases}$$

取步长  $h = 0.1$ 。

解题程序（包括子程序 RGKT1 和子程序 FCS）如下：

```

DIMENSION U(2), F(2), P(2), W(2)
EXTERNAL FCS
T = 0.0
U(1) = 1.0
U(2) = 1.0
WRITE (6, 100) T, U
DO 30 I = 1, 20
CALL RGKT1 (2, T, 0.1, U, FCS, F, P, W)
30 WRITE (6, 100) T, U
100 FORMAT (IX, F4.1, 2F16.8)
STOP
END

```

```

C      SUBROUTINE RGKT1 (N, T, H, U, FCS, F, P, W)
      ; (本子程序段体)
      END

C      SUBROUTINE FCS (T, U, F)
      DIMENSION U(2), F(2)
      F(1) = 1.0/(U(2) - T)
      F(2) = 1.0 - 1.0/U(1)
      RETURN
      END

```

下面列出部分计算结果:

| T   | U1(T)      | U2(T)      |
|-----|------------|------------|
| 0.0 | 1.00000000 | 1.00000000 |
| 0.2 | 1.22140204 | 1.01873122 |
| 0.4 | 1.49182294 | 1.07032081 |
| 0.6 | 1.82211559 | 1.14881258 |
| 0.8 | 2.22553572 | 1.24932999 |
| 1.0 | 2.71827390 | 1.36788049 |

本题的解析解是

$$u_1 = e^t, \quad u_2 = e^{-t} + t$$

相应于数值解的点的值为

| t   | u <sub>1</sub> | u <sub>2</sub> |
|-----|----------------|----------------|
| 0.0 | 1.00000000     | 1.00000000     |
| 0.2 | 1.22140276     | 1.01873075     |
| 0.4 | 1.49182470     | 1.07032004     |
| 0.6 | 1.82211880     | 1.14881163     |
| 0.8 | 2.22554093     | 1.24932896     |
| 1.0 | 2.71828183     | 1.36787944     |

形式(2)子程序语句为:

```
SUBROUTINE RGKT2 (N, H, U, DU, UC, UU)
```

哑元说明如下:

- N 整型变量,输入参数;方程个数.
- H 实型变量,输入参数;积分步长.
- U N个元素的一维实型数组,输入与输出参数;调用本子程序和计算右函数子程序(见下)前需把积分初值存放到本数组;积分一个步长后本数组存放积分结果;需要继续积分必须连续调用本子程序.
- DU N个元素的一维实数组,输出参数;存放右函数值.
- UC, UU 均是N个元素的一维实型工作数组,调用前无需赋值.

本子程序中还需调用的另一子程序是

SUBROUTINE DIF (N, DU, UC)

它是计算右函数子程序段,由使用者自编,其中N为整型变量,方程的个数(但注意这里是  $N = n + 1$ 。这是因为在本子程序中,引入第0个方程  $\frac{dt}{dt} = 1$ , 而把原来积分  $n$  个方程变成积分  $n + 1$  个方程); DU, UC 都是 N 个元素的一维实型数组,前者存放右函数值,而右函数自变量取自后者。

本子程序除了引入第0个方程  $\frac{dt}{dt} = 1$  外,调用本子程序之前,在主程序段中需要调用一次计算右函数子程序,取得自变量与右函数值同步,以后每调用本子程序一次便向前积分一步。

子程序 (2):

```

SUBROUTINE RGKT2 (N, H, U, DU, UC, UU)
DIMENSION U(N), DU(N), UC(N), UU(N), S(4)
S(1) = 0.5 * H
S(2) = S(1)
S(3) = H
S(4) = H
DO 10 I = 1, N
10  UU(I) = U(I)
DO 13 J = 1, 3
DO 12 I = 1, N
UC(I) = UU(I) + S(J) * DU(I)
12  U(I) = U(I) + S(J + 1) * DU(I) / 3.0
13  CALL DIF (N, UC, DU)
DO 14 I = 1, N
14  U(I) = U(I) + S(1) * DU(I) / 3.0
CALL DIF (N, U, DU)
RETRUN
END

```

**算例** 求解某振动运动的初值问题:

$$\begin{cases} m_1 \ddot{x}_1 + K_{11}x_1 + K_{12}x_2 = 0 \\ m_2 \ddot{x}_2 + K_{21}x_1 + K_{22}x_2 = 0 \\ x_1(0) = x_2(0) = 0 \\ \dot{x}_1(0) = \dot{x}_2(0) = 1 \end{cases}$$

其中各系数为

$$m_1 = m_2 = 0.15;$$

$$K_{11} = 278.78, \quad K_{12} = K_{21} = -110.57, \quad K_{22} = 110.57$$

要求对积分区间  $[0, 0.2]$ , 以步长  $h = 0.01$  进行积分。



把原微分方程写成

$$\begin{cases} \ddot{x}_1 = -\frac{1}{m_1}(K_{11}x_1 + K_{12}x_2) \\ \ddot{x}_2 = -\frac{1}{m_2}(K_{21}x_1 + K_{22}x_2) \end{cases}$$

代入各系数,可得

$$\begin{cases} \ddot{x}_1 = -1858.533x_1 + 737.133x_2 \\ \ddot{x}_2 = 737.133x_1 - 737.133x_2 \end{cases}$$

再把原初值问题化为适合 RGKT2 程序求解的形式,即令

$$\dot{x}_1 = \frac{dx_1}{dt} = x_3, \quad \dot{x}_2 = \frac{dx_2}{dt} = x_4, \quad \frac{dt}{dt} = 1$$

于是可得

$$\begin{cases} \frac{dt}{dt} = 1 \\ \frac{dx_1}{dt} = x_3 \\ \frac{dx_2}{dt} = x_4 \\ \frac{dx_3}{dt} = -1858.533x_1 + 737.133x_2 \\ \frac{dx_4}{dt} = 737.133x_1 - 737.133x_2 \\ x_1(0) = x_2(0) = 0 \\ x_3(0) = x_4(0) = 1 \end{cases}$$

程序中以数组 U 的下标变量 U(1), U(2), ..., U(5) 依次代表  $t, x_1, \dots, x_4$ . 输出结果中以 X1, X2, X3, X4 依次代表  $x_1, x_2, x_3, x_4$ .

解题程序

```

DIMENSION U(5), UC(5), DU(5), UU(5)
U(1) = 0.0
U(2) = 0.0
U(3) = 0.0
U(4) = 1.0
U(5) = 1.0
H = 0.01
WRITE (6, 100)
100 FORMAT (10X, 1HT, 13X, 2HX1, 13X, 2HX2, 13X, 2HX3, 13X,
1 2HX4)
CALL DIF (5, U, DU)
WRITE (6, 200) U

```

```

300 CALL RGKT2 (5, H, U, DU, UC, UU)
      WRITE (6, 200) U
      IF (U(1) .LT. 0.19999998) GO TO 300
200  FORMAT (1X, 5F15.8)
      STOP
      END

```

C

```

SUBROUTINE RGKT2 (N, H, U, DU, UC, UU)
:   (本子程序段体)
END

```

C

```

SUBROUTINE DIF (N, U, DU)
DIMENSION U(N), DU(N)
DU(1) = 1.0
DU(2) = U(4)
DU(3) = U(5)
DU(4) = - 1858.533 * U(2) + 737.133 * U(3)
DU(5) = 737.133 * U(2) - 737.133 * U(3)
RETURN
END

```

计算输出结果:

| T         | X1          | X2          | X3          | X4          |
|-----------|-------------|-------------|-------------|-------------|
| 0.0000000 | 0.0000000   | 0.0000000   | 1.0000000   | 1.0000000   |
| 0.0100000 | 0.00981310  | 0.0100000   | 0.94479841  | 0.99965557  |
| 0.0200000 | 0.01855628  | 0.01997963  | 0.78930481  | 0.99463735  |
| 0.0300000 | 0.02535402  | 0.02984257  | 0.56153175  | 0.97409990  |
| 0.0400000 | 0.02967735  | 0.03936075  | 0.30155343  | 0.92312080  |
| 0.0500000 | 0.03142383  | 0.04815286  | 0.05311908  | 0.82667516  |
| 0.0600000 | 0.03090872  | 0.05570475  | -0.14541013 | 0.67388193  |
| 0.0700000 | 0.02877037  | 0.06142960  | -0.26884149 | 0.46155968  |
| 0.0800000 | 0.02581141  | 0.06475663  | -0.30997320 | 0.19627248  |
| 0.0900000 | 0.02281134  | 0.06523052  | -0.28043228 | -0.10563016 |
| 0.1000000 | 0.02035130  | 0.06260066  | -0.20737276 | -0.42004247 |
| 0.1100000 | 0.01869026  | 0.05688132  | -0.12678338 | -0.71881459 |
| 0.1200000 | 0.01771881  | 0.04836917  | -0.07489336 | -0.97439531 |
| 0.1300000 | 0.01700049  | 0.03761383  | -0.07957876 | -1.16444776 |
| 0.1400000 | 0.01589114  | 0.02534637  | -0.15366953 | -1.27548331 |
| 0.1500000 | 0.01370908  | 0.01237968  | -0.29163959 | -1.30476643 |
| 0.1600000 | 0.00991802  | -0.0049984  | -0.47042055 | -1.26011431 |
| 0.1700000 | 0.00428157  | -0.01262813 | -0.65417544 | -1.15766418 |
| 0.1800000 | -0.00304588 | -0.02352882 | -0.80200032 | -1.01811552 |
| 0.1900000 | -0.01151301 | -0.03293671 | -0.87687774 | -0.86227567 |
| 0.2000000 | -0.02025309 | -0.04077739 | -0.85392782 | -0.70687816 |

## 2. 自动选步长的龙格-库塔法

与数值积分一样,我们既要考虑截断误差,也要考虑计算量及舍入误差,因此常微分方程数值解法中也存在自动选合适步长的问题。

设以步长  $h$  计算的近似值  $u_n$  记为  $u_n^{[h]}$ , 对四阶龙格-库塔法来说,由于它是四阶的,故有

$$u(t_{n+1}) - u_{n+1}^{[h]} \approx ch^5$$

其中  $c$  与  $u^{(4)}(t)$  在  $[t_n, t_{n+1}]$  内之值有关。再以步长  $\frac{h}{2}$  从  $t_n$  出发计算两步,又有

$$u(t_{n+1}) - u_{n+1}^{[\frac{h}{2}]} \approx 2c \left(\frac{h}{2}\right)^5$$

于是可得

$$\frac{u(t_{n+1}) - u_{n+1}^{[\frac{h}{2}]} }{u(t_{n+1}) - u_{n+1}^{[h]} } \approx \frac{1}{16}$$

或得事后估计式

$$u(t_{n+1}) - u_{n+1}^{[\frac{h}{2}]} \approx \frac{1}{15} (u_{n+1}^{[\frac{h}{2}]} - u_{n+1}^{[h]})$$

由此,我们可以根据步长折半前后两次计算结果的偏差

$$\delta = |u_{n+1}^{[\frac{h}{2}]} - u_{n+1}^{[h]}|$$

来判定所选步长是否合适。具体做法是,设要求数值解的精度为  $\epsilon$ :

1) 若  $\delta < \epsilon$ , 则反复加倍步长计算,直至  $\delta > \epsilon$ ,再以上一次步长计算所得之值作为  $u_{n+1}$ 。

2) 若  $\delta > \epsilon$ , 则反复折半步长计算,直至  $\delta < \epsilon$ ,其最后一次所得之值作为  $u_{n+1}$ 。

已有包含上述自动选步长技巧的通用标准子程序可供引用<sup>[C1, C2, C3]</sup>。

## 9-4 阿当姆斯公式与预测-校正方法

阿当姆斯(Adams)方法属于多步方法,也有预测-校正方案。多步法的基本思想就是计算  $u_{n+1}$  时,充分利用前面已经求出的若干近似值比如  $u_0, u_1, \dots, u_n$  的信息,以期提高计算结果的精度;预测-校正是指先用显式公式提供解的一个首次估值,然后再用隐式公式校正所求之解。

阿当姆斯公式分显式和隐式两种。

可以通过不同途径,如基于数值积分或基于泰勒展开来构造阿当姆斯公式。下面我们采用前者。

### 1. 阿当姆斯显式与隐式公式

再次考虑初值问题

$$\begin{cases} u' = f(t, u), & t \in [t_0, T] \\ u(t_0) = u_0 \end{cases} \quad (9.25)$$

将方程两端从  $t_n$  到  $t_{n+1}$  求积分, 可得

$$u(t_{n+1}) = u(t_n) + \int_{t_n}^{t_{n+1}} f(t, u(t)) dt \quad (9.26)$$

即微分方程初值问题 (9.25) 与积分方程 (9.26) 等价.

显然, 对应 (9.26) 中求积分的一个数值积分公式, 就相当于对应初值问题的一种数值解法. 如用矩形公式计算

$$\int_{t_n}^{t_{n+1}} f(t, u(t)) dt \approx hf(t_n, u(t_n))$$

并据此离散化, 便可导出尤拉公式

$$u_{n+1} = u_n + hf(t_n, u_n)$$

又如用梯形公式计算

$$\int_{t_n}^{t_{n+1}} f(t, u(t)) dt \approx \frac{h}{2} [f(t_n, u(t_n)) + f(t_{n+1}, u(t_{n+1}))]$$

也类似地可导出梯形公式

$$u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, u_{n+1})]$$

梯形公式实际就是积分号下的被积函数  $f(t, u(t))$  用插值点为  $t_n, t_{n+1}$  的线性函数代替的结果.

对于 (9.26) 中的积分, 我们考虑用插值节点为  $t_{n-2}, t_{n-1}, t_n, t_{n+1}$  的插值多项式  $F(x)$  来代替  $f(t, u(t))$ . 这时

$$\begin{aligned} F(t) = & \frac{(t - t_{n-1})(t - t_n)(t - t_{n+1})}{(t_{n-1} - t_{n-2})(t_{n-2} - t_n)(t_{n-2} - t_{n+1})} f(t_{n-2}, u(t_{n-2})) \\ & + \frac{(t - t_{n-2})(t - t_n)(t - t_{n+1})}{(t_{n-1} - t_{n-2})(t_{n-1} - t_n)(t_{n-1} - t_{n+1})} f(t_{n-1}, u(t_{n-1})) \\ & + \frac{(t - t_{n-2})(t - t_{n-1})(t - t_{n+1})}{(t_n - t_{n-2})(t_n - t_{n-1})(t_n - t_{n+1})} f(t_n, u(t_n)) \\ & + \frac{(t - t_{n-2})(t - t_{n-1})(t - t_n)}{(t_{n+1} - t_{n-2})(t_{n+1} - t_{n-1})(t_{n+1} - t_n)} f(t_{n+1}, u(t_{n+1})) \end{aligned}$$

把  $F(t)$  代入 (9.26), 并作变换  $t = t_n + xh$ , 且注意  $t_{n+1} - t_n = t_n - t_{n-1} = t_{n-1} - t_{n-2} = h$ , 则得

$$\begin{aligned} u(t_{n+1}) = & u(t_n) + \frac{1}{6} hf(t_{n+1}, u(t_{n+1})) \int_0^1 x(x+1)(x+2)dx \\ & - \frac{1}{2} hf(t_n, u(t_n)) \int_0^1 (x-1)(x+1)(x+2)dx \\ & + \frac{1}{2} hf(t_{n-1}, u(t_{n-1})) \int_0^1 (x-1)x(x+2)dx \\ & - \frac{1}{6} hf(t_{n-2}, u(t_{n-2})) \int_0^1 (x-1)x(x+1)dx \\ = & u(t_n) + \frac{h}{24} [9f(t_{n+1}, u(t_{n+1})) + 19f(t_n, u(t_n)) \\ & - 5f(t_{n-1}, u(t_{n-1})) + f(t_{n-2}, u(t_{n-2}))] \end{aligned}$$

同样作离散化,即用  $u_i$  代替  $u(t_i)$ , 便得

$$u_{n+1} = u_n + \frac{h}{24} [9f(t_{n+1}, u_{n+1}) + 19f(t_n, u_n) - 5f(t_{n-1}, u_{n-1}) + f(t_{n-2}, u_{n-2})]$$

这就是阿当姆斯隐式公式,也称内插公式。

如果我们取插值节点为  $t_{n-3}, t_{n-2}, t_{n-1}, t_n$  作成插值多项式  $F(t)$  来代替 (9.26) 中的  $f(t, u(t))$ , 则类似的推导可得

$$u_{n+1} = u_n + \frac{h}{24} [55f(t_n, u_n) - 59f(t_{n-1}, u_{n-1}) + 37f(t_{n-2}, u_{n-2}) - 9f(t_{n-3}, u_{n-3})]$$

这就是阿当姆斯的显式公式,也称外推公式。

关于两个公式的局部截断误差,利用泰勒展式,可以推导出

$$\text{对于隐式有 } u(t_{n+1}) - u_{n+1} = -\frac{19}{720} h^5 u_n^{(5)} + \dots$$

$$\text{对于显式有 } u(t_{n+1}) - u_{n+1} = \frac{251}{720} h^5 u_n^{(5)} + \dots$$

即两者(同有 4 阶精度)与四阶龙格-库塔法同阶。

如果单独采用显式公式,每计算一个  $u_{n+1}$ ,只需计算一次  $f(t, u)$  的值,计算量比龙格-库塔法少。但存在的问题是,  $u_1, u_2, u_3$  需另用其它方法求取。计算过程中要改变步长也很麻烦,往往又需从头做起。这些缺点,对隐式公式也一样存在。隐式公式需用迭代求解,往往迭代过程也费时较多。因此,一个高效率的技巧就是所谓预测-校正方案。

## 2. 预测-校正方法及其 FORTRAN 程序

我们选择阿当姆斯的显式公式作“预测”式,隐式公式作“校正”式,前者提供首次估值,借以使后者收敛迅速,即构造预测-校正系统如下:

$$\begin{aligned} \text{预测} \quad \bar{u}_{n+1} &= u_n + \frac{h}{24} [55f(t_n, u_n) - 59f(t_{n-1}, u_{n-1}) \\ &\quad + 37f(t_{n-2}, u_{n-2}) - 9f(t_{n-3}, u_{n-3})] \end{aligned} \quad (9.27)$$

$$\begin{aligned} \text{校正} \quad u_{n+1} &= u_n + \frac{h}{24} [9f(t_{n+1}, \bar{u}_{n+1}) + 19f(t_n, u_n) \\ &\quad - 5f(t_{n-1}, u_{n-1}) + f(t_{n-2}, u_{n-2})] \end{aligned} \quad (9.28)$$

同时计算过程开始前,可用比如四阶龙格-库塔公式算出头三步的  $u$  及  $f$  值。有了这些开始值,用预测公式就可估算出下一个  $u$  值,而有了首次估值,就可用校正公式进行迭代,直至达到预定的最大  $t$  值。

与龙格-库塔方法一样,上述阿当姆斯预测-校正方法自然可以推广到方程组初值问题的情形

$$\begin{cases} \mathbf{u}' = \mathbf{f}(t, \mathbf{u}) \\ \mathbf{u}(t_0) = \mathbf{u}_0 \end{cases}$$

下面我们讨论其 FORTRAN 程序设计<sup>[61]</sup>。

先要指出的是,象 RGKT2 程序一样,下面给出的 FORTRAN 程序中,我们用自变

量: 作出第 0 个方程  $\frac{dt}{dt} = 1$ , 即右函数  $f_0(t) = 1$ . 这样, 原来积分  $n$  个方程便变为积分

$n + 1$  个方程. 从而使得给出的算法具有自变量与右函数值同步的特点.

将给出的子程序为 ADAMS.

设置哑元如下:

- N 整型变量, 输入参数, 方程个数 ( $N = n + 1$ ).
- H 实型变量, 输入参数, 积分步长.
- L 整型变量, 输入输出参数; 用来控制程序走向, 第一次调用前置  $L = 0$ , 第一次调用后自行将  $L$  改变 (第一次调用本程序时, 仅对右函数求值一次, 不进行积分, 使得自变量与右函数同步; 以后每调用一次便向前积分一步, 包括前三步用龙格-库塔方法求出供本法起步用的结果).
- U  $N$  个元素的一维实数组, 输入、输出参数, 第一次调用前存放积分初值, 调用后存放积分结果.
- DU  $N$  个元素的一维实数组, 输出参数, 存放右函数值.
- P  $N$  个元素的一维实型工作数组, 计算过程中, 存放前一步积分结果 (调用前无需赋值).
- Q  $N$  个元素的一维实型工作数组 (调用前无需赋值)
- WOR  $4 * N$  个元素的二维实型工作数组, 存放四步右函数值, 即
- $$\begin{aligned} \text{WOR}(1, I) &= f'_{n-3}, & \text{WOR}(2, I) &= f'_{n-2} \\ \text{WOR}(3, I) &= f'_{n-1}, & \text{WOR}(4, I) &= f'_n \\ I &= 1, 2, \dots, N, \text{ 调用前无需赋值.} \end{aligned}$$

子程序中还要调用另外两个子程序:

- 1) 用来计算三个始值的龙格-库塔法子程序

SUBROUTINE RGKT2 (N, H, U, DU, UC, UU)

- 2) 计算右函数的子程序

SUBROUTINE DIFF (N, U, DU)

由使用者自编,  $N$  为整型变量, 方程个数,  $U$  是  $N$  个元素的一维实型数组, 右函数自变量值取自本数组,  $DU$  是  $N$  个元素的一维实型数组, 存放右函数值.

子程序:

```
SUBROUTINE ADAMS (N, H, L, U, DU, P, Q, WOR)
  DIMENSION U(N), DU(N), P(N), Q(N), WOR(4, N)
  IF (L - 1) 12, 15, 15
12 CALL DIFF (N, U, DU)
20 L = L + 1
  DO 22 J = 1, N
22 WOR (L, J) = DU(J)
  RETURN
```

```

15 IF (L - 4) 16, 13, 13
16 CALL RGKT2 (N, H, U, DU, P, Q)
   GO TO 20
13 DO 33 J = 1, N
   P(J) = U(J)
   U(J) = U(J) + H * (2.291666666 * WOR (4, J) - 2.458333333 *
1 WOR (3, J) + 1.541666666 * WOR (2, J) - 0.375 * WOR (1, J))
   WOR (1, J) = WOR (2, J)
   WOR (2, J) = WOR (3, J)
33 WOR (3, J) = WOR (4, J)
   CALL DIFF (N, U, DU)
   DO 44 J = 1, N
44 U(J) = P(J) + H * (0.375 * DU(J) + 0.791666666 * WOR (3, J) -
1 0.208333333 * WOR (2, J) + 0.041666666 * WOR (1, J))
   CALL DIFF (N, U, DU)
   DO 55 J = 1, N
55 WOR (4, J) = DU (J)
   RETURN
   END

```

**算例** 解下列一阶初值问题

$$\begin{cases} u_1' = u_2 \\ u_2' = -10^6 u_1 - 200 u_2 \\ u_1(0) = 10 \\ u_2(0) = 0 \end{cases}$$

积分区间  $[0, 0.05]$ , 积分步长  $h = 0.0001$ , 输出步长  $0.001$ .

根据程序使用方法, 先把原问题改为

$$\begin{cases} t' = 1 \\ u_1' = u_2 \\ u_2' = -10^6 u_1 - 200 u_2 \\ t(0) = 1 \\ u_1(0) = 10 \\ u_2(0) = 0 \end{cases}$$

主程序以及需自编的子程序 DIFF 如下:

```

DIMENSION U(3) DU(3), P(3), Q(3), WOR (4, 3)
U(1)=0.0
U(2)=10.0
U(3)=0.0

```

```

H=0.0001
DELTAT=0.001-1.E-10
L=0
CALL ADAMS (3, H, L, U, DU, P, Q, WOR)
WRITE (6, 100) U
100 FORMAT (1X, F10.3, 2E20.8)
111 CALL ADAMS (3, H, L, U, P, Q, WOR)
IF (U(1) .LT. DELTAT) GO TO 111
WRITE (6, 100) U
DELTAT=U(1)+0.001-1.E-10
IF (U(1) .LT. 0.05) GO TO 111
STOP
END

```

C

```

SUBROUTINE DIFF (N, U, DU)
DIMENSION U(N), DU(N)
DU(1)=1.0
DU(2)=U(3)
DU(3)=-1000000.0*U(2)=200.0*U(3)
RETURN
END

```

连同子程序 ADAMS 和子程序 RGKT2 一起, 组成解题程序。上机计算的部分结果抄录如下:

| $t$   | $u_1$           | $u_2$           |
|-------|-----------------|-----------------|
| 0.000 | 0.10000000E 02  | 0.00000000E 00  |
| 0.001 | 0.56897071E 01  | -0.76275880E 04 |
| 0.010 | -0.33688034E 01 | 0.18535536E 04  |
| 0.020 | 0.79109861E 00  | -0.11800193E 04 |
| 0.050 | 0.55267470E-01  | 0.33410958E 02  |

## 9-5 收敛性与稳定性

收敛性与稳定性是微分方程数值解法中的两个基本问题。

在数值解法的差分格式中, 由于存在截断误差, 因此, 如果我们假定  $\tilde{u}_{n+1}$  是在计算过程中不作舍入的情况下由差分格式求出的解(我们称它为差分格式的真解), 那么, 当步长  $h$  取得充分小时,  $\tilde{u}_{n+1}$  是否能足够精确地逼近微分方程的真解  $u(t_{n+1})$ , 也即当  $h \rightarrow 0$  (同时  $n \rightarrow \infty$ ) 时, 是否有  $\tilde{u}_{n+1} \rightarrow u(t_{n+1})$ ? 这就是常微分方程数值解法中所考虑的收敛性问题。

但是, 在实际计算中, 由于计算机字长有限等原因, 因此, 不能没有舍入误差。所以问题是, 在某一步上产生的舍入误差(包括初始值所带的初始误差), 在以后的计算中是否会



无限制扩大,以致于得不到所需的数值解。这就是常微分方程数值解法中所讨论的稳定性问题。

综上所述,可知利用数值方法求解常微分方程的初值问题时,数值解  $u_{n+1}$  与真解  $u(t_{n+1})$  的误差可以归结为两部分:

$$\begin{aligned} u(t_{n+1}) - u_{n+1} &= (u(t_{n+1}) - \tilde{u}_{n+1}) + (\tilde{u}_{n+1} - u_{n+1}) \\ &= \epsilon_{n+1} + \tau_{n+1} \end{aligned}$$

其中

$$\epsilon_{n+1} = u(t_{n+1}) - \tilde{u}_{n+1}$$

称为整体误差,关于它的讨论即收敛性问题。

$$\tau_{n+1} = \tilde{u}_{n+1} - u_{n+1}$$

为舍入误差,关于它的讨论即稳定性问题。

整体误差与局部截断误差的区别在于,局部截断误差是初值问题的真解  $u(t_{n+1})$  与假定前  $u_i (i = 0, 1, \dots, n)$  为方程的精确解的前提下由差分格式求出的  $u_{n+1}$  之差。而整体误差  $\epsilon_{n+1}$  是初值问题的真解  $u(t_{n+1})$  与差分格式的真解  $\tilde{u}_{n+1}$  之差。由此可见,整体误差包含各局部截断误差,也包括了前若干步的局部截断误差在逐步计算中的积累。因此,作为误差估计,最根本的是估计整体误差。但估计整体误差并不容易。好在于整体误差与局部截断误差通常有一定的关系(见下面定理),因而,要构造高精度的计算方法,只需设法提高局部截断误差即可。

为了简化讨论,通常在讨论收敛性时,便不考虑舍入和初值误差 ( $\epsilon_0 = u(t_0) - u_0 = 0$ ); 在讨论稳定性时,则只比较差分格式真解  $\tilde{u}_{n+1}$  与近似解  $u_{n+1}$  的关系。

## 1. 收敛性

**定义 1** 如果在一种数值方法的差分格式中,对于任意固定的  $t_n = t_0 + nh$ ,当  $h \rightarrow 0$  (同时  $n \rightarrow \infty$ ) 时,由差分格式在假定不作舍入的情况下求得的  $\tilde{u}_{n+1} \rightarrow u(t_{n+1})$ ,便称这种数值方法(或差分格式)是收敛的。

为了书写方便,定义中所指  $\tilde{u}_{n+1}$  在下面讨论中仍记为  $u_{n+1}$ 。

现在,以单步法为例来进行讨论。我们知道,显式单步法的共同特征是,差分格式均由  $u_n$  加上某种形式的增量而得出  $u_{n+1}$ ,即形如

$$u_{n+1} = u_n + h\varphi(t_n, u_n, h) \quad (9.29)$$

式中  $\varphi(t, u, h)$  称为增量函数。

例如,对于尤拉公式(9.8)有

$$\varphi = f(t, u)$$

对于改进的尤拉公式(9.11)有

$$\varphi = \frac{1}{2} [f(t, u) + f(t + h, u + hf(t, u))]$$

我们有下述收敛性定理。

**定理** 假定单步法(9.29)具有  $p$  阶精度,且增量函数  $\varphi(t, u, h)$  关于  $u$  满足李普希兹条件

$$|\varphi(t, u, h) - \varphi(t, \bar{u}, h)| \leq L_\varphi |u - \bar{u}|$$

又设初值  $u_0$  是准确的, 即  $\varepsilon_0 = u(t_0) - u_0 = 0$ , 则其整体误差

$$u(t_n) - u_n = O(h^p)$$

这个定理的证明从略。

根据这个定理可知:

1) 整体误差比局部截断误差低一阶。所以, 我们说, 要构造高精度的计算方法, 只需设法提高局部截断误差即可。

2) 按定理, 要判断某个单步法是否收敛, 归结为判断增量函数是否关于  $u$  满足李普希兹条件。

例如, 对尤拉公式, 因为增量函数  $\varphi$  就是  $f$ , 故当  $f(t, u)$  关于  $u$  满足李普希兹条件时, 它是收敛的。

对于改进的尤拉公式, 因为

$$\begin{aligned} |\varphi(t, u, h) - \varphi(t, \bar{u}, h)| &\leq \frac{1}{2} [|f(t, u) - f(t, \bar{u})| \\ &\quad + |f(t+h, u+hf(t, u)) - f(t+h, \bar{u}+hf(t, \bar{u}))|] \\ &\leq \frac{1}{2} [L|u - \bar{u}| + L|u + hf(t, u) - \bar{u} - hf(t, \bar{u})|] \\ &\leq \frac{1}{2} [L|u - \bar{u}| + L|u - \bar{u}| + Lh|f(t, u) - f(t, \bar{u})|] \\ &\leq L|u - \bar{u}| + \frac{h}{2} L|u - \bar{u}| \\ &= L\left(1 + \frac{h}{2} L\right) |u - \bar{u}| \end{aligned}$$

设定  $h \leq h_0$  ( $h_0$  为定数), 取

$$L_\varphi = L\left(1 + \frac{h_0}{2} L\right)$$

于是上式表明  $\varphi$  关于  $u$  满足李普希兹条件。因此, 改进的尤拉法是收敛的。

类似地, 可证明龙格-库塔法也是收敛的。

## 2. 稳定性

在微分方程数值解中, 稳定性的概念分为一般稳定性和绝对稳定性两种, 其中绝对稳定性又有条件稳定和无条件稳定的区别。

由于观测不准确或舍入等原因, 初始值往往带有一定的误差。设  $\tilde{u}_0$  是初始值的真值,  $u_0$  是带有误差的近似值, 初始值的误差记为

$$e_0 = \tilde{u}_0 - u_0$$

又设  $\tilde{u}_{n+1}$  和  $u_{n+1}$  分别是由初始值  $\tilde{u}_0$  和  $u_0$  出发, 在不再考虑以后计算中的舍入误差的情况下, 按某一差分格式求得的真解和近似解, 其误差记为

$$e_{n+1} = \tilde{u}_{n+1} - u_{n+1}$$

我们引入下列定义:

**定义 2** 如果对于解满足李普希兹条件的初值问题的差分格式, 存在常数  $K$  和  $h_0$ , 使得当  $0 < h \leq h_0$  时, 按该差分格式求得任何二解满足不等式

$$|\tilde{u}_{n+1} - u_{n+1}| \leq K |\tilde{u}_0 - u_0| \quad (9.30)$$

则该差分格式称为稳定的。

定义中的不等式 (9.30) 表明, 当  $e_0 \rightarrow 0$  时, 有  $e_{n+1} \rightarrow 0$ 。因此, 当  $h$  充分小时, 差分格式的解将连续依赖于初始值。这就是说, 一个稳定的方法, 其误差是不会被传播和扩散的, 误差的积累是可以控制的。

容易验证, 尤拉法是稳定的。事实上, 由

$$\begin{aligned}\tilde{u}_{n+1} &= \tilde{u}_n + hf(t_n, \tilde{u}_n) \\ u_{n+1} &= u_n + hf(t_n, u_n)\end{aligned}$$

两边相减可得

$$\begin{aligned}|e_{n+1}| &\leq |e_n| + h|f(t_n, \tilde{u}_n) - f(t_n, u_n)| \\ &\leq (1 + hL)|e_n| \leq (1 + hL)^2|e_{n-1}| \\ &\leq \dots \leq (1 + hL)^{n+1}|e_0|\end{aligned}$$

由于  $(1 + hL)^{n+1} \leq (1 + hL)^{\frac{T-t_0}{h}} = (1 + hL)^{\frac{1}{hL}(T-t_0)L} < e^{(T-t_0)L}$   
取  $K = e^{(T-t_0)L}$ , 即得条件 (9.30), 从而可知尤拉法是稳定的。

上述一般稳定性概念是在  $h \rightarrow 0$  的情况下来讨论的, 因此, 也称为渐近稳定性。

实际计算中, 我们只能取固定的有限步长  $h$ , 它不能随意缩小。因此, 我们要考虑的是, 对固定步长  $h$ , 计算过程中在某一节点上所产生的误差 (也称摄动) 是否会在以后的计算中步步增长的问题。

**定义 3** 如果某数值方法的差分格式, 在某节点上  $\tilde{u}_n$  产生摄动  $e_n$ , 在没有舍入误差的条件下, 由此引起以后各节点上  $u_m$  ( $m > n$ ) 产生的摄动都满足

$$|e_m| \leq |e_n|$$

则称这个差分格式是绝对稳定的。

要对各种差分格式的绝对稳定性做全面的考察是困难的。为此, 在讨论绝对稳定性时, 把问题简化, 不论方程的具体形式如何, 均以模型方程

$$\begin{cases} u' = Au & (A \text{ 为复数}) \\ u(t_0) = u_0 \end{cases} \quad (9.31)$$

代替原方程进行讨论。这种简化可以直观地理解为: 如果一个差分格式对如此简单的方程还不是绝对稳定的, 就更难以用它来解一般方程的初值问题。当然, 一个差分格式, 对模型方程是绝对稳定的, 也不一定对一般的方程是绝对稳定的。但用模型方程在一定程度上反映了数值方法的某些特性。

现在我们通过模型方程来考察一些差分格式的绝对稳定性。

**例 1** 用尤拉法解模型方程 (9.31)。由

$$\begin{aligned}\tilde{u}_{n+1} &= \tilde{u}_n + hA\tilde{u}_n \\ u_{n+1} &= u_n + hAu_n\end{aligned}$$

可得

$$\begin{aligned}e_{n+1} &= e_n + hAe_n = (1 + hA)e_n = \dots \\ &= (1 + hA)^{n+1}e_0\end{aligned}$$

故当  $|1 + hA| \leq 1$  时, 有  $|e_{n+1}| \leq |e_n|$ , 即当  $|1 + hA| \leq 1$  时, 尤拉法绝对稳定。满

足不等式

$$|1 + hA| \leq 1$$

的  $hA$  值称为绝对稳定域,它是以  $-1$  为心,以  $1$  为半径的圆.

**例 2** 我们用改进尤拉公式来解模型方程 (9.31), 由

$$\tilde{u}_{n+1} = \tilde{u}_n + \frac{h}{2} (A\tilde{u}_n + A\tilde{u}_{n+1})$$

$$u_{n+1} = u_n + \frac{h}{2} (Au_n + Au_{n+1})$$

相减可得

$$e_{n+1} = e_n + \frac{hA}{2} e_n + \frac{hA}{2} e_{n+1}$$

即有

$$|e_{n+1}| = \left| \frac{1 + \frac{hA}{2}}{1 - \frac{hA}{2}} \right| |e_n|$$

故当  $\operatorname{Re}(A) < 0$  时, 有

$$\left| \frac{1 + \frac{hA}{2}}{1 - \frac{hA}{2}} \right| < 1$$

这时改进的尤拉法是绝对稳定的, 绝对稳定域是  $hA$  复平面的左半平面.

象尤拉法和改进的尤拉法这样, 具有有限的绝对稳定域, 我们称之为条件稳定, 否则, 称为无条件稳定.

是不是所有解初值问题的差分格式都是绝对稳定呢? 不是的.

**例 3** 考虑差分格式

$$u_{n+1} = u_{n-1} + 2hu_n \quad (9.32)$$

用于解模型方程 (9.31), 则可得

$$e_{n+1} = e_{n-1} + 2hAe_n$$

为了简便, 模型方程中我们取  $A$  为小于 0 的实数. 把上式改写成

$$e_{n+1} - 2hAe_n - e_{n-1} = 0$$

这是关于  $n$  的二阶差分方程. 令  $e_n = r^n$ , 代入差分方程可得“特征方程”

$$r^2 - 2hAr - 1 = 0$$

解之, 得两个特征根

$$r_{1,2} = hA \pm \sqrt{h^2 A^2 + 1}$$

从而得通解

$$e_n = \alpha r_1^n + \beta r_2^n$$

其中  $\alpha, \beta$  是与  $n$  无关的常数. 现在因为  $A < 0$ , 故  $|r_1| < 1, |r_2| > 1$ . 可见, 当步长  $h$  足够小时,  $e_n$  趋于无穷大. 这就说明了差分格式 (9.32) 是绝对不稳定的. 但可以证明, 它却是收敛的.

事实上,一般稳定性概念与方程本身形式无关;而绝对稳定性则由两方面决定:一是微分方程形式本身,二是所采用的方法.如果方程本身是不稳定的,那么用什么方法求解都不会稳定.如果方程稳定,方法不稳定也不行.只有方程稳定,方法也稳定才行.实际计算中,可以通过求绝对稳定域,选取适当的  $h$ , 来满足稳定性的要求.

**例 4** 用龙格-库塔法解初值问题

$$\begin{cases} u' = -10u \\ u(0) = u_0 \end{cases}$$

用二阶龙格-库塔法可得

$$u_{n+1} = (1 - 10h + 50h^2)u_n$$

或

$$\begin{aligned} u_{n+1} &= (1 - 10h + 50h^2)^{n+1}u_0 \\ &= \lambda_1^{n+1}u_0 \end{aligned}$$

其中  $\lambda_1 = 1 - 10h + 50h^2$ . 于是,若  $|\lambda_1| > 1$ , 则所得的解无限增大,与原问题的真解  $u = u_0 e^{-10x}$  完全不同.但当  $0 \leq h < 0.2$  时,有  $0 < \lambda < 1$ , 则可得到稳定解.

对于四阶龙格-库塔法,可得

$$\begin{aligned} u_{n+1} &= \left(1 - 10h + 50h^2 - \frac{500}{3}h^3 + \frac{1250}{3}h^4\right)u_n \\ &= \left(1 - 10h + 50h^2 - \frac{500}{3}h^3 + \frac{1250}{3}h^4\right)^{n+1}u_0 \\ &= \lambda_2^{n+1}u_0 \end{aligned}$$

其中  $\lambda_2 = 1 - 10h + 50h^2 - \frac{500}{3}h^3 + \frac{1250}{3}h^4$ . 于是,可以验证,当  $0 < h < 0.27$  时,  $|\lambda_2| < 1$ , 这时四阶龙格-库塔法是稳定的.

总之,只有那些既收敛又稳定的差分格式才是实用的差分格式.

## 9-6 边值问题的数值解法

以二阶微分方程

$$u'' = f(x, u, u'), \quad x \in [a, b] \quad (9.33)$$

为例.其边值条件通常有下列三种提法:

第 I 边值条件

$$u(a) = \alpha, \quad u(b) = \beta \quad (9.33)'$$

第 II 边值条件

$$u'(a) = \alpha, \quad u'(b) = \beta \quad (9.33)''$$

第 III 边值条件

$$\begin{aligned} u'(a) - \alpha_0 u(a) &= \alpha_1, \quad u'(b) + \beta_0 u(b) = \beta_1 \\ \alpha_0 &\geq 0, \quad \beta_0 \geq 0, \quad \alpha_0 + \beta_0 > 0 \end{aligned} \quad (9.33)'''$$

这里我们不详细研究上述边值问题解的存在唯一性问题.下面我们总是假定所考虑的边值问题的解的存在唯一性是成立的.

如果方程(9.33)为线性微分方程,则可写成

$$u'' + p(x)u' + q(x)u = f(x) \quad (9.34)$$

令  $u = cv$ , 代入方程(9.34)有

$$cv'' + (2c' + pc)v' + (c'' + pc' + qc)v = f(x)$$

再令  $2c' + pc = 0$ , 解出  $c = e^{-\frac{1}{2} \int p dx}$ , 故可知只须令  $u = e^{-\frac{1}{2} \int p dx} v$ , 方程(9.34)便化为不带一阶导数项的形式. 为此, 不失一般性, 必要时我们可以只讨论下列形状的二阶线性方程

$$u'' + p(x)u = Q(x) \quad (9.34)'$$

边值问题的数值解法比较常用的有差分方法、打靶法和样条函数法等. 下面介绍前两种, 它们分别适用于不同情形的微分方程.

### 1. 线性边值问题的差分方法

仅考虑如下二阶线性微分方程的边值问题

$$\begin{cases} u'' + p(x)u' + q(x)u = f(x) & x \in [a, b] \\ u(a) = \alpha, \quad u(b) = \beta \end{cases} \quad (9.35)$$

我们的目的就是求在  $[a, b]$  上的  $n+1$  个等距节点

$$x_i = a + ih$$

(其中  $i = 0, 1, \dots, n$ ;  $h = \frac{b-a}{n}$ ,  $x_0 = a$ ,  $x_n = b$ ) 处的近似解  $u_i$ . 将方程离散化, 用中心差分近似代替  $u'$  和  $u''$ :

$$\begin{aligned} u'_i &\approx \frac{u_{i+1} - u_{i-1}}{2h} \\ u''_i &\approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \end{aligned}$$

并简记  $p(x_i) = p_i$ ,  $q(x_i) = q_i$ ,  $f(x_i) = f_i$ , 代入(9.35)中的方程式, 可得近似差分方程组

$$\left( \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \right) + p_i \left( \frac{u_{i+1} - u_{i-1}}{2h} \right) + q_i u_i = f_i, \quad i = 1, 2, \dots, n-1$$

化简可得

$$\begin{aligned} \left( 1 - \frac{h}{2} p_i \right) u_{i-1} + (-2 + h^2 q_i) u_i + \left( 1 + \frac{h}{2} p_i \right) u_{i+1} &= h^2 f_i, \\ i &= 1, 2, \dots, n-1 \end{aligned}$$

这实际是包含  $n-1$  个未知数的  $n-1$  个方程式的三对角代数方程组. 令

$$\begin{cases} a_i = \left( 1 - \frac{h}{2} p_i \right) \\ b_i = (-2 + h^2 q_i) \\ c_i = \left( 1 + \frac{h}{2} p_i \right) \\ d_i = h^2 f_i \end{cases} \quad i = 1, 2, \dots, n-1$$



为了改进此结果,我们可以再把  $h$  取半,即在  $2n+1$  个节点上求出近似解.设前后两组近似解在同一些点上的值为

$$u_i(h) \text{ 和 } u_i\left(\frac{h}{2}\right), \quad i = 1, 2, \dots, n-1$$

取 
$$u_i = \frac{4u_i\left(\frac{h}{2}\right) - u_i(h)}{3}, \quad i = 1, 2, \dots, n-1$$

作为新的近似解,通常有较好的结果.

## 2. 打靶法

打靶法可以用来解线性和非线性边值问题.其基本思想是把边值问题转化为初值问题.为了说明这种方法的原理,我们考察如下边值问题

$$\begin{cases} u''(x) + Qu(x) = R, & x \in [0, l] \\ u(0) = 0, & u(l) = L \end{cases} \quad (9.36)$$

现在我们要把它作为初值问题来处理,这就需要已知  $u'(0)$  的值.令  $u'(0) = \alpha$ ,  $\alpha$  为待定参数,我们这样选择  $\alpha$ ,使得由它构成的初值问题

$$\begin{cases} u''(x) + Qu(x) = R, & x \in [0, l] \\ u(0) = 0, & u'(0) = \alpha \end{cases} \quad (9.37)$$

的近似解在一定精度范围内等于规定的  $u(l) = L$ ,那么,原来的边值问题也就解决了.为此,我们的基本做法是,先猜测斜率  $u'(0)$  的起始值,然后建立一个迭代过程使它收敛于所规定的斜率值.

今令  $\alpha_1$  与  $\alpha_2$  是  $u'(0)$  的两个猜测值,并记  $u(\alpha_1; l)$  和  $u(\alpha_2; l)$  为对微分方程求解所得到的在  $x = l$  的  $u$  值.我们可以选择  $\alpha_1$  为  $u(0)$  和  $u(l)$  之间的斜率,即猜测

$$\alpha_1 = \frac{u(l) - u(0)}{l - 0}$$

然后用龙格-库塔方法或其它方法解此初值问题

$$\begin{cases} u''(x) + Qu(x) = R, & x \in [0, l] \\ u(0) = 0, & u'(0) = \alpha_1 \end{cases}$$

求出  $u_1, u_2, \dots, u_n$ , 其中  $u_n = u(\alpha_1; l)$ .

这时,如果  $u(\alpha_1; l) > u(l)$ ,则可适当减少  $\alpha_1$ ,否则可适当增大  $\alpha_1$ .并取新的  $\alpha_1$  为  $\alpha_2$ ,然后,类似地由  $\alpha_2$  求出  $u_1, u_2, \dots, u_n$ , 其中  $u_n = u(\alpha_2; l)$ .

较好的  $\alpha$  近似值可用线性内插公式求得

$$\alpha_3 = \alpha_1 + (\alpha_2 - \alpha_1) \frac{u(l) - u(\alpha_1; l)}{u(\alpha_2; l) - u(\alpha_1; l)} \quad (9.38)$$

于是,又由  $u'(0) = \alpha_3$  解出  $u(\alpha_3; l)$ .再根据  $\alpha_2$  和  $\alpha_3$  使用形如 (9.38) 的内插公式,得出下一个近似值  $\alpha_4$ .等等.重复此过程直至  $u(\alpha_k; l)$  在所要求的精度内相当于  $u(l) = L$  为止.这里收敛的速度与初始猜测值的好坏有关.

上述做法与弹道问题相似,所以被取名为“打靶法”,即不断寻找  $\alpha$ ,使它“射中” $u(\alpha; l) = u(l) = L$ .

现在对一般的二阶边值问题(线性和非线性)



$$\begin{cases} u'' = f(x, u, u'), & x \in [a, b] \\ u(a) = u_0, & u(b) = u_b \end{cases}$$

给出打靶法的计算步骤:

1) 令  $\alpha_1, \alpha_2$  为  $u'(a)$  的两个近似值, 分  $[a, b]$  为  $N$  等分, 用龙格-库塔法或其它方法解下列两个初值问题:

$$(I) \begin{cases} u'' = f(x, u, u') \\ u(a) = u_0 \\ u'(a) = \alpha_1 \end{cases} \quad (II) \begin{cases} u'' = f(x, u, u') \\ u(a) = u_0 \\ u'(a) = \alpha_2 \end{cases}$$

分别求出  $u_1^{(1)}, u_2^{(1)}, \dots, u_N^{(1)}$  和  $u_1^{(2)}, u_2^{(2)}, \dots, u_N^{(2)}$ , 把在  $x=b$  处的解分别记为  $u(\alpha_1; b)$  和  $u(\alpha_2; b)$ .

2) 检验: 对给定的  $\varepsilon$ , 如果  $|u(\alpha_1; b) - u(b)| < \varepsilon$  或  $|u(\alpha_2; b) - u(b)| < \varepsilon$ , 则所求出的相应的解便为原边值问题之解; 否则, 做下一步.

3) 逐次对  $k = 2, 3, \dots$  计算出

$$\alpha_{k+1} = \alpha_k + (\alpha_k - \alpha_{k-1}) \frac{u(b) - u(\alpha_{k-1}; b)}{u(\alpha_k; b) - u(\alpha_{k-1}; b)}$$

4) 解初值问题.

$$\begin{cases} u'' = f(x, u, u') \\ u(a) = u_0 \\ u'(a) = \alpha_{k+1} \end{cases}$$

求出  $u_1^{(k+1)}, u_2^{(k+1)}, \dots, u_N^{(k+1)} = u(\alpha_{k+1}; b)$ .

5) 重复步骤 3) 和 4) 直至对给定的  $\varepsilon$ , 满足

$$|u(\alpha_k; b) - u(b)| < \varepsilon \quad (9.39)$$

为止, 则  $u_1^{(r)}, u_2^{(r)}, \dots, u_N^{(r)}$  为所求之解.

但必须注意的是, 满足 5) 的近似值并不能保证其中  $u_2^{(r)}, u_3^{(r)}, \dots, u_N^{(r)}$  全都很好, 也就是说, 即使  $u_N^{(r)}$  和  $u_N$  一致, 其它的值也可能相差尚大. 这里的问题主要看子区间的大小  $h$  的选择是否恰当.

如上一小节中提出的, 一种简单实用的处理方法是, 在使用上述算法并得到满足 (9.39) 的解之后, 我们可令  $h_1 = \frac{h}{2}$ , 并重复上述过程. 当用  $h_1$  算法收敛时, 检查是否对每一个  $m = 1, 2, \dots, N-1$  都有

$$|u_m - \bar{u}_m| < \varepsilon_1 \quad (\varepsilon_1 \text{ 给定})$$

其中  $u_m$  和  $\bar{u}_m$  分别是以  $h$  和  $h_1$  作步长得出的共同点上的近似解. 若上述条件并非对每个  $m$  都满足, 则再把子区间缩小一半, 重复上述过程直至满足上述条件.

可以为打靶法编写适用一定问题的 FORTRAN 程序.

## 习 题 九

1. 试将下列二阶方程的初值问题化为一阶方程组的初值问题:

$$(1) \begin{cases} u'' + 3u' + 2u = 0 \\ u(0) = u'(0) = 1 \end{cases}$$

$$(2) \begin{cases} u'' = 0.1(1-u^2)u + u = 0 \\ u(0) = 1, \quad u'(0) = 0 \end{cases}$$

$$(3) \begin{cases} x''(r) = -\frac{x}{r^3} \\ y''(r) = -\frac{y}{r^3} \\ x(0) = 0.4, \quad x'(0) = 0 \\ y(0) = 0, \quad y'(0) = 2 \end{cases} \quad (r = \sqrt{x^2 + y^2})$$

2. 用尤拉方法求下列问题的数值解:

$$(1) \begin{cases} u' = -2tu^2 \\ u(0) = 1 \end{cases}$$

$$(2) \begin{cases} u' = 10t - 10tu \\ u(0) = 0 \end{cases}$$

其中用定步长  $h = 0.1$ , 要求计算 10 步.

3. 试分别用泰勒展开式的方法和数值积分的方法导出解初值问题

$$\begin{cases} u' = f(t, u) \\ u(t_0) = u_0 \end{cases}$$

的尤拉公式

$$u_{n+1} = u_n + hf(t_n, u_n)$$

并估计其局部截断误差.

4. 试用数值积分方法, 分别导出解初值问题

$$\begin{cases} u' = f(t, u) \\ u(t_0) = u_0 \end{cases}$$

的梯形公式

$$u_{n+1} = u_n + \frac{h}{2} [f(t_n, u_n) + f(t_{n+1}, u_{n+1})]$$

和尤拉两步公式(中点折线公式)

$$u_{n+1} = u_{n-1} + 2hf(t_n, u_n)$$

并分别估计其局部截断误差.

5. 证明: 对初值问题

$$\begin{cases} u' + u = 0 \\ u(0) = 1 \end{cases}$$

用梯形公式所求得的近似解为

$$u_n = \left( \frac{2-h}{2+h} \right)^n$$

并且当  $h \rightarrow 0$  时, 它收敛于准确解  $e^{-t}$ .

6. 试分别用尤拉公式和改进尤拉公式求初值问题

$$\begin{cases} u' = u - \frac{2x}{u}, \quad x \in [0, 1.5] \\ u(0) = 1 \end{cases}$$

的数值解,  $h = 0.1$ , 并与真解

$$u = \sqrt{1+2x}$$

进行比较.

7. 设有初值问题

$$\begin{cases} y' = -0.01y \\ y(0) = 100 \end{cases}$$

试分别按下列各方法求  $y(100)$ :

- 1) 用尤拉方法, 取  $h = 5$ ;
- 2) 用尤拉方法, 取  $h = 1$ ;
- 3) 用改进尤拉方法, 取  $h = 20$ ;
- 4) 用改进尤拉方法, 取  $h = 10$ ;
- 5) 用四阶龙格-库塔法, 取  $h = 100$ ;
- 6) 用四阶龙格-库塔法, 取  $h = 50$ .

将上面结果与真解  $y(100) = 100e^{-0.01 \times 100} \approx 36.788$  比较.

8. 设有初值问题

$$\begin{cases} y' = -5y \\ y(0) = 1 \end{cases}$$

- 1) 用 2 阶龙格-库塔法求解,  $h = 0.5$ ;
- 2) 用 4 阶龙格-库塔法求解,  $h = 0.5$ ;
- 3) 将上面结果与真解  $y(t) = e^{-5t}$  作比较(取适当步数).
9. 用四阶龙格-库塔法求解初值问题

$$\begin{cases} u' = u - \frac{2t}{u}, & t \in [0, 1.6] \\ u(0) = 1 \end{cases}$$

的数值解, 步长  $h = 0.2$ .

并与用尤拉公式和改进尤拉公式的计算就精度和计算量两方面进行比较.

10. 利用四阶龙格-库塔法的 FORTRAN 程序, 求解初值问题

$$\begin{cases} u' = t^2 + t^3u, & t \in [1, 3] \\ u(1) = 1 \end{cases}$$

取  $h = 0.1$ .

11. 用四阶龙格-库塔法求出下列初值问题第一个步长的解:

$$\begin{cases} \frac{d^2u}{dt^2} = \frac{u}{e^t + 1} \\ u|_{t=0} = 1 \\ \left. \frac{du}{dt} \right|_{t=0} = 0 \end{cases}$$

取步长  $h = 0.1$ .

12. 利用四阶龙格-库塔法的 FORTRAN 程序, 在计算机上求解下列初值问题:

(1) 一阶线性时变微分方程

$$\begin{cases} \dot{x}(t) = -(1 + 0.5 \sin t)x(t) + \cos t \\ x(0) = 1 \end{cases}$$

在区间  $0 \leq t \leq 10$  秒内的解 (取  $h = 0.1$ ).

(2) 非线性时变微分方程

$$\begin{cases} \dot{x} = -t^2x^2 - 0.5 \sin(2x) \\ x(0) = 2 \end{cases}$$

在区间  $0 \leq t \leq 5$  秒内的解, 并用作图子程序描绘  $x(h)$  曲线.

13. 利用四阶龙格-库塔法的 FORTRAN 程序, 在计算机上求解初值问题:

(1) 线性时变微分方程组

$$\begin{cases} \dot{x}_1 = e^{-t}x_1 - 2x_2 + 3e^{-t}x_3, & x_1(0) = 0 \\ \dot{x}_2 = \frac{1}{1+t}x_1 + 3x_2 - 6x_3, & x_2(0) = 1 \\ \dot{x}_3 = -x_1 + e^{-t}x_2 - x_3 + \sin t, & x_3(0) = -1 \end{cases}$$

在区间  $0 \leq t \leq 10$  的解 (取  $h = 0.1$ ).

(2) 线性时不变微分方程组

$$\begin{cases} \dot{x}_1 = -x_1 + 2x_2 + 6x_3, & x_1(0) = 1 \\ \dot{x}_2 = -x_2 + 3x_3 + 2\sin t, & x_2(0) = -1 \\ \dot{x}_3 = -x_3 + t^2e^{-t} + \cos t, & x_3(0) = 0 \end{cases}$$

在区间  $0 \leq t \leq 5$  的解, 并用作图子程序描绘出  $x_1(t)$ ,  $x_2(t)$  曲线.

(3) 非线性时变微分方程组

$$\begin{cases} \dot{x}_1 = -x_1 + x_2^2 + 4x_3^2, & x_1(0) = 1 \\ \dot{x}_2 = -\sin x_1 - x_2 + 2x_3, & x_2(0) = -1 \\ \dot{x}_3 = -x_1^2 + 2x_2^2 - e^{-t}, & x_3(0) = 0 \end{cases}$$

在区间  $0 \leq t \leq 5$  的解 (取  $h = 0.05$ ).

14. 用阿当姆斯方法的预测-校正系统解初值问题

$$\begin{cases} u' = u - \frac{2t}{u}, & t \in [0, 1.5] \\ u(0) = 1 \end{cases}$$

的数值解, 步长  $h = 0.1$ . 然后讨论这种解法的优缺点.

15. 试用阿当姆斯预测-校正方法求下列初值问题

$$\begin{cases} u_1' = 3u_1 + 2u_2, \\ u_2' = 4u_1 + u_2, & t \in [0, 1] \\ u_1(0) = u_2(0) = 0 \end{cases}$$

的数值解, 取  $h = 0.1$ , 并将数值解与精确解

$$\begin{cases} u_1(t) = \frac{1}{3}(e^{3t} - e^{-t}) \\ u_2(t) = \frac{1}{3}(e^{3t} + 2e^{-t}) \end{cases}$$

比较

16. 已知某系统的运动表示为非线性微分方程

$$m\ddot{x} + c\dot{x} + kx + k^*x^3 = 0$$

所以作为时间  $t$  的函数的位移  $x$  不能用解析法求出, 因此, 这个微分方程需要采用数值解.

如果该系统的物理参量和初始条件如下:

$$\begin{aligned} m &= 0.01, & c &= 0.15, \\ k &= 2.0, & k^* &= 2.0, \\ x(0) &= 10.0, & \dot{x}(0) &= 0. \end{aligned}$$

试选用合适的数值方法和程序在计算机上进行计算, 以模拟该系从  $0 \sim 1.0$  秒的运动.

17. 用差分方法求解下列边值问题:

$$(1) \begin{cases} u'' - 3u = 2x, & x \in [0, 1] \\ u(0) = 0, & u(1) = 1 \end{cases}$$

取  $h = 0.2$ ,

$$(2) \begin{cases} v'' + v = 0, & x \in [0, \pi] \\ v(0) = 1, & v(\pi) = -1 \end{cases}$$

$$= \frac{\pi}{3}$$

18. 用打靶法解下列边值问题

$$(1) \begin{cases} u'' - 3u^2 - 2xu = 2x^2, & x \in [0, 1] \\ u(0) = 0, & u(1) = 1.0 \end{cases}$$

取  $h=0.02$

$$(2) \begin{cases} y' + y = 0, & x \in [0, \pi] \\ y(0) = 1, & y(\pi) = -1 \end{cases}$$

取  $h = \frac{\pi}{4}$ .

19. 选择合适的方法求边值问题

$$\begin{cases} u'' - (1+x^2)u = -1 \\ u(-1) = u(1) = 0 \end{cases}$$

的数值解 (取  $h=0.5$ ).

## 参 考 资 料

### A. 数值计算方法方面

- [A1] 冯康等编,《数值计算方法》,国防工业出版社,1978年.
- [A2] 李庆扬,王能超,易大义,《数值分析》,华中工学院出版社,1982年.
- [A3] 易大义,蒋叔豪,李有法编,《数值方法》,浙江科学技术出版社,1984年.
- [A4] 李岳生,黄友谦编,《数值逼近》,人民教育出版社,1978年.
- [A5] 清华大学北京大学编,《计算方法》(上册),科学出版社,1975年.
- [A6] 东北师范大学白玉山主编,《计算方法》,辽宁人民出版社,1984年.
- [A7] Robert W. Hornbeck 著,刘元久,郭耀煌,荣廷玉译,《数值方法》,中国铁道出版社,1982年.
- [A8] 杨晓引,马正午,孙宇等编,《电子计算机应用数学》,第一册,冶金工业出版社,1979年.
- [A9] 何旭初,苏煜城,包雪松编,《计算数学简明教程》,人民教育出版社,1980年.
- [A10] William S. Dorn and Daniel D. McCracken, «Numerical Methods with Fortran IV Case Studies» 1972.
- [A11] R. T. Burden, J. D. Faires, A. C. Reynolds, «Numerical Analysis», 1981.

### B. FORTRAN 语言方面

- [B1] 国际标准化组织推荐文本《程序设计语言 FORTRAN》,韩淑娟,姚兆炜译,曹德和校,科学出版社,1980年.
- [B2] 郝柏林编著,《FORTRAN 程序设计》,人民邮电出版社,1980年.
- [B3] 谭浩强,田淑清编著,《FORTRAN 语言》,清华大学出版社,1981年.
- [B4] 丘玉圃编著,《FORTRAN 程序设计》,科学出版社,1979年.
- [B5] 华南工学院编,《电子计算机与算法语言》,高等教育出版社,1978年.
- [B6] 邓自立编,《程序设计语言 FORTRAN 77》,1983年.
- [B7] 黄秉刚,谈建中,蒋恩杰编,《FORTRAN 77 程序设计》,上海科学技术文献出版社,1984年.
- [B8] A. Balfour and D. H. Marwick, «Programming in Standard FORTRAN 77», 1979.
- [B9] Samuel L. Marateck, «FORTRAN 77», Second Edition, 1983.

### C. 其它

- [C1] 刘德贵,费景高,于泳江,李广元编,《FORTRAN 算法汇编》(第一分册与第二分册),国防工业出版社,1980年.
- [C2] 中国科学院沈阳计算所,后字四一四部队,北京工业大学编,《电子计算机常用算法》,科学出版社,1983年(增订版).