目　录

第 0 章　绪论 …… (**1**)
0.1 数值计算方法与算法 …… (1)
0.2 误差与有效数字 …… (1)
0.3 约束误差 …… (3)
0.4 范数 …… (3)
0.4.1　向量范数 …… (3)
0.4.2　矩阵范数 …… (5)
第 1 章　插值 …… (**9**)
1.1 插值 …… (9)
1.2 多项式插值的拉格朗日(Lagrange)型式 …… (9)
1.2.1 线性插值 …… (10)
1.2.2 二次插值 …… (11)
1.2.3 n 次拉格朗日插值多项式 …… (13)
1.3 多项式插值的牛顿(Newton)型式 …… (16)
1.3.1 差商及其计算 …… (17)
1.3.2 牛顿插值 …… (18)
1.4* Hermite 插值 …… (21)
1.5 分段插值 …… (24)
1.5.1 龙格(Runge)现象 …… (24)
1.5.2 分段线性插值 …… (25)
1.6 三次样条函数 …… (27)
1.6.1 三次样条插值的 M 关系式 …… (27)
1.6.2 三次样条插值的 m 关系式 …… (29)
1.7 程序示例 …… (30)
习题 1 …… (34)
第 2 章　数值微分和数值积分 …… (**36**)
2.1 数值微分 …… (36)
2.1.1 差商与数值微分 …… (36)
2.1.2 插值型数值微分 …… (38)
2.1.3 样条插值数值微分 …… (39)
2.2 数值积分 …… (39)
2.2.1 插值型数值积分 …… (40)
2.2.2 牛顿-柯特斯(Newton-Cote's)积分 …… (41)
2.3 复化数值积分 …… (44)
2.3.1 复化梯形积分 …… (44)

2.3.2 复化辛普森积分 …… (46)
2.3.3 复化积分的自适应算法 …… (47)
2.3.4 龙贝格(Romberg)积分 …… (50)
2.4 重积分计算 …… (51)
2.5* 高斯型积分公式介绍 …… (52)
2.6 程序示例 …… (54)
习题 2 …… (57)
第 3 章 曲线拟合的最小二乘法 …… (59)
3.1 拟合曲线 …… (59)
3.2 线性拟合和二次拟合函数 …… (60)
3.3 解矛盾方程组 …… (63)
3.4 程序示例 …… (69)
习题 3 …… (72)
第 4 章 非线性方程求根 …… (74)
4.1 实根的对分法 …… (74)
4.2 迭代法 …… (75)
4.3 牛顿迭代法 …… (77)
4.4 弦截法 …… (80)
4.5 非线性方程组的牛顿方法 …… (81)
4.6 程序示例 …… (83)
习题 4 …… (85)
第 5 章 解线性方程组的直接法 …… (86)
5.1 消元法 …… (87)
5.1.1 三角形方程组的解 …… (87)
5.1.2 高斯(Gauss)消元法与列主元消元法 …… (88)
5.1.3 Guass-Jordan 消元法 …… (93)
5.2 直接分解法 …… (94)
5.2.1 Doolittle 分解 …… (96)
5.2.2 Courant 分解 …… (100)
5.2.3 追赶法 …… (102)
5.2.4 对称矩阵的 LDL^T 分解 …… (103)
5.3* 矩阵的条件数 …… (105)
5.4 程序示例 …… (106)
习题 5 …… (114)
第 6 章 解线性方程组的迭代法 …… (116)
6.1 简单(Jacobi)迭代 …… (117)
6.1.1 Jacobi 迭代格式 …… (117)
6.1.2 Jacobi 迭代收敛条件 …… (119)
6.2 Gauss-Seidel 迭代 …… (120)

6.3 松弛迭代 …… (123)
6.4 逆矩阵计算 …… (124)
6.5 程序示例 …… (125)
习题 6 …… (130)
第 7 章　计算矩阵的特征值和特征向量 …… (132)
7.1 幂法 …… (132)
7.1.1 幂法运算 …… (132)
7.1.2 幂法的规范运算 …… (135)
7.1.3* 关于幂法的初始值 …… (137)
7.2 反幂法 …… (137)
7.3 实对称矩阵的 Jacobi 方法 …… (138)
7.4 程序示例 …… (144)
习题 7 …… (147)
第 8 章　常微分方程数值解 …… (148)
8.1 欧拉(Euler)公式 …… (148)
8.1.1 基于数值微商的欧拉公式 …… (148)
8.1.2* 欧拉公式的收敛性 …… (151)
8.1.3 基于数值积分的近似公式 …… (152)
8.2 龙格-库塔方法 …… (153)
8.2.1 二阶龙格-库塔方法 …… (153)
8.2.2 四阶龙格-库塔公式 …… (155)
8.2.3 步长的自适应 …… (157)
8.3 线性多步法 …… (157)
8.4 常微分方程组的数值解法 …… (160)
8.4.1 一阶常微分方程组的数值解法 …… (160)
8.4.2 高阶常微分方程数值方法 …… (162)
8.5* 常微分方程的稳定性 …… (163)
8.6 程序示例 …… (165)
习题 8 …… (168)
第 9 章* 在 Mathematica 中做题 …… (169)
9.1 符号计算系统 Mathematica 基本操作 …… (169)
9.2 插值 …… (172)
9.3 数值积分 …… (173)
9.4 曲线拟合 …… (174)
9.5 非线性方程 …… (175)
9.6 方程组求解 …… (176)
9.7 计算特征值和特征向量 …… (177)
9.8 常微分方程数值解 …… (178)
参考文献 …… (181)

第0章 绪 论

0.1 数值计算方法与算法

数值计算方法，是一种研究并解决数学问题的数值近似解方法，是在计算机上使用的解数学问题的方法，简称计算方法。它的计算对象是那些在理论上有解而又无法用手工计算的数学问题。例如，解含有300个未知量的线性方程组；计算6阶矩阵的全部特征值。

在科学研究和工程技术中都要用到各种计算方法。例如，在地质勘探、汽车制造、桥梁设计、天气预报和汉字字样设计中都有计算方法的踪影。在70年代，大多数学校仅在数学系的计算数学专业和计算机系开设计算方法这门课程。随着计算机技术的迅速发展和普及，现在计算方法几乎已成为所有理工科学生的必修课程。

计算方法是一门理论性和实践性都很强的学科，计算方法既有数学类课程中理论上的抽象性和严谨性，又有实用性和实验性的技术特征。计算方法的前提课程是微积分、线性代数、常微分方程和一门计算机语言。

学习计算方法的目的是为了使用方法，在学习计算方法的过程中，在套用计算公式、修改计算公式和创建计算公式中，都需要一定的专业知识和数学基础。在学习计算方法时，要注重逼近和迭代等数学思想和常用手法，获取近似计算的能力，并能触类旁通地应用到各个领域中。一些有创造力的工程师不仅擅长使用某些计算方法，并能创建出简便有效的计算方法。例如，样条函数、快速傅里叶变换和有限元方法都是先由工程师们创建的，然后再由数学家们完善这些方法的理论基础，并从理论上进行提高和推广。

从方法的计算公式到在计算机上实际运行，两者之间还有距离，这是数学能力与计算机应用技术能力之间的距离，也与计算机的运行环境和编程工具有关。为了缩小两者之间的距离，本书将给出部分计算公式的算法描述。用算法容易准确而简便地描述计算公式，在算法中能简洁地表达计算公式中的“循环”和“迭代”等操作。有了方法的算法，将它转化成C或PASCAL等语言的程序，上机运行也就容易了。

在学习计算方法的过程中，建议读者能用某种语言编制该方法的程序并运行通过，这将有利于准确而深刻地掌握该方法的计算步骤和过程。书中每章后面都附有几个C语言程序，这些程序基于数值计算公式，没有进行优化处理，其目的是通过编程上机，加深对方法运行过程的理解，训练和提高计算机应用技术的能力和水平。

0.2 误差与有效数字

绝对误差与绝对误差界

近似计算必然产生误差，误差表示精确值与近似值的距离。

定义：设 x^* 为精确值(或准确值)，x 是 x^* 的一个近似值，称 $e = x^* - x$ 为近似值 x 的

绝对误差或误差。

$$\text{绝对误差} = \text{精确值} - \text{近似值}$$

绝对误差 e 的值可正可负，如果得不到精确值 x^*，也就算不出绝对误差 e 的值。常用限制误差绝对值的范围 ε 来描述和控制误差的范围。

定义：如果精确值 x^* 与近似值 x 的误差的绝对值不超过某正数 ε，即

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

则称 ε 为绝对误差限或误差限。

精确值 x^* 也可表示为：$x^* = x \pm \varepsilon$。通常，在误差允许的范围内的近似值 x 即认为是精确值，这也是计算中控制循环中止的常用手段。

例 0.1：若经四舍五入得到 $x = 123.456$，

对于数 123.4559，123.4555，123.4561，123.4564 的近似值都是 $x = 123.456$，即第四位小数大于 5 时，必然进位到第三位小数；第四位小数小于 5 时，必然舍去。

它的误差限是：

$$|e| = |x^* - x| < 10^{-4} \cdot 5 = \frac{1}{2} \cdot 10^{-3}$$

若 $x^* = 0.0123456$，则它的误差限是：

$$|e| = |x^* - x| < 10^{-8} \cdot 5 = \frac{1}{2} \cdot 10^{-7}$$

相对误差与相对误差限

在很多情况下，绝对误差并不能全面地反映近似程度。例如，某电器公司两次进货的某型号电风扇分别为 1000 台和 2000 台，其中开箱不合格电风扇分别为 8 和 12（绝对误差的值）。不合格率分别为 8/1000 = 0.8% 和 12/2000 = 0.6 %（相对误差的值），这说明该电风扇的质量有所提高。我们把绝对误差与准确值的比值定义为相对误差。

定义：设 x^* 为精确值，x 是 x^* 的一个近似值，称

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

为近似值 x 的相对误差。

在实际计算中，有时得不到精确值 x^*，当 e_r 较小时 x^* 可用近似值 x 代替，即

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

$$\text{相对误差} = \frac{\text{绝对误差}}{\text{精确值}} \quad \text{或} \quad \text{相对误差} = \frac{\text{绝对误差}}{\text{近似值}}$$

相对误差 e_r 的值也可正可负，与绝对误差一样不易计算，常用相对误差限控制相对误差的范围。

定义：如果有正数 ε_r 使得 $e_r = \left|\frac{e}{x^*}\right| < \varepsilon_r$，则称 ε_r 为 x^* 的相对误差限。

产生误差的因素很多，主要有：

(1)原始误差

由客观存在的模型抽象到物理模型产生的误差。包括模型误差和原始数据误差。

(2)截断误差

用有限项近似无限项时，由截取函数的部分项产生的误差，称为截断误差。

例如：$\text{arctg}x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}x^{2n-1}}{2n-1}$，在计算中用 $\text{arctg}x \approx \sum_{n=1}^{N} \frac{(-1)^{n-1}x^{2n-1}}{2n-1}$ 作近似计算。

(3)舍入误差

在数值计算中，通常都按有限位进行运算。例如，按照四舍五入的原则，2 / 3 = 0.666667 或 2 / 3 = 0.667，由舍入产生的误差，称为舍入误差。

在实际计算中的数据通常是近似值，它们由观察、估计或一些计算而得到，这些数在计算机上表示和计算后也会带来进一步误差，即误差的积累和传播。关于误差的传播似乎没有多少统一的理论，通常积累误差的界是以通例分析为基础而建立的。

有效位数

定义：当 x 的误差限为某一位的半个单位，则这一位到第一个非零位的位数称为 x 的有效位数。

例如，$x = 12.34$，$y = 0.004067$ 均有 4 位有效数字，而 3.00 与 3.0000 分别有 3 位和 5 位有效位数。

有效位的多少直接影响到近似值的绝对误差和相对误差，因此，在计算中也应注意保持一定的有效位数。

0.3 约束误差

选择收敛的稳定的方法

对同一问题选择不同的数值计算方法，可能得到不同的计算结果。在计算方法中，除了给出方法的数值计算公式，还要讨论计算公式的收敛性、稳定性和截断误差的特性。选择收敛性要求低、稳定性好的方法是约束误差扩张的最重要的措施。例如，样条插值函数比高次多项式的效果好得多，是构造插值函数的首选方法。

提高数值计算精度

数值在计算机中存放的位数称为字长。有限位的字长是带来舍入误差和抑制数值计算精度的根源。对同一种方法，字长大的计算机上的计算效果要优于字长小的计算机。

在计算机上，对同一种数值计算方法而言，对数据选用不同的数值类型有时会直接影响到计算效果。例如，对病态的线性方程组使用消元方法，采用单精度数据时其数值解大大失真，而用双精度数据却可得到满意的数值解。

0.4 范　　数

0.4.1 向量范数

在一维空间中，实轴上任意两点 a, b 的距离用两点差的绝对值 $|a-b|$ 表示。绝对值是一种度量形式的定义。

范数是对函数、向量和矩阵定义的一种度量形式。任何对象的范数值都是一个非负

实数。使用范数可以测量两个函数、向量或矩阵之间的距离。向量范数是度量向量长度的一种定义形式。范数有多种定义形式，只要满足下面的三个条件即可定义为一个范数。同一向量，采用不同的范数定义，可得到不同的范数。

对任一向量 $X \in R^n$，按照一个规则确定一个实数与它对应，记该实数为 $\| X \|$，若 $\| X \|$ 满足下面三个性质：

$$\left.\begin{array}{ll}\text{(i) } \forall X \in R^n, \text{有 } \| X \| \geq 0, \text{当且仅当 } X=0 \text{ 时, } \| X \| = 0 & \text{(非负性)} \\ \text{(ii) } \forall X \in R^n, \text{有 } \| \alpha X \| = |\alpha| \| X \| & \text{(齐次性)} \\ \text{(iii) } \forall X, Y \in R^n, \text{有 } \| X+Y \| \leq \| X \| + \| Y \| & \text{(三角不等式)}\end{array}\right\} \quad (0.1)$$

那么称该实数 $\| X \|$ 为向量 X 的范数。

向量范数定义

向量 $X=(x_1, x_2, \cdots, x_n)^T$ 的 L_p 范数定义为

$$\| X \|_p = \left(\sum_{i=1}^{n} | x_i |^p \right)^{1/p}, \qquad 1 \leq p \leq +\infty \tag{0.2}$$

其中，经常使用的三种向量范数是 $p = 1, 2, \infty$。

$$\| X \|_1 = | x_1 | + | x_2 | + \cdots + | x_n | = \sum_{i=1}^{n} | x_i | \tag{0.3}$$

$$\| X \|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{0.4}$$

或写成 $\| X \|_2 = \sqrt{(X, X)}$

$$\| X \|_\infty = \max\{ | x_1 |, | x_2 |, \cdots, | x_n | \} = \max_{1 \leq i \leq n} \{ | x_i | \} \tag{0.5}$$

例 0.2：计算向量 $X=(1, 3, -5)^T$，$p = 1, 2, \infty$ 的三种向量范数。

$$\| X \|_1 = 1 + 3 + 5 = 9$$

$$\| X \|_2 = (1^2 + 3^2 + 5^2)^{1/2} = \sqrt{35}$$

$$\| X \|_\infty = \max\{1, 3, | -5 | \} = 5$$

不同向量范数的关系

有限维线性空间 R^n 中任意向量范数的定义都是等价的。若 $R_1(X)$，$R_2(X)$ 是 R^n 上两种不同的范数定义，则必存在 $0 < m < M < \infty$，使 $\forall X \in R^n$，均有

$$mR_2(X) \leq R_1(X) \leq MR_2(X)$$

或

$$m \leq \frac{R_1(X)}{R_2(X)} \leq M \qquad (X \neq 0) \tag{0.6}$$

（证明略。）

向量的极限

有了向量范数的定义，也就有了度量向量距离的标准，即可定义向量的极限和收敛概念了。

设 $X^{(1)}, \cdots, X^{(m)} \cdots$ 为 R^n 上向量序列，若存在向量 $\alpha \in R^n$ 使 $\lim\limits_{m \to \infty} \| X^{(m)} - \alpha \| = 0$，则称向量列 $X^{(1)}, X^{(2)}, \cdots$ 是收敛的，α 称为该向量序列的极限。

由向量范数的等价性知，向量列是否收敛与选取哪种范数无关。

向量序列 $X^{(m)}=(x_1^{(m)},x_2^{(m)},\cdots,x_n^{(m)})^T;m=1,2,\cdots$,收敛的充分必要条件为其序列的每个分量收敛,即 $\lim\limits_{m\to\infty}x_i^{(m)}$ 存在。

若 $\lim\limits_{m\to\infty}x_i^{(m)}=x_i$,则 $X=(x_1,x_2,\cdots,x_n)^T$ 就是向量序列 $\{X^{(m)}\}=\{(x_1^{(m)},x_2^{(m)},\cdots,x_n^{(m)})^T\}$ 的极限。

例 0.3:求向量序列 $\{X^{(k)}\}=\left\{\begin{pmatrix}\dfrac{1}{k+1}\\ \left(1+\dfrac{1}{k}\right)^k\\ \dfrac{2k-1}{k+11}\end{pmatrix}\right\}$ 的极限向量。

解:算出每个向量分量的极限后得

$$\lim_{k\to\infty}X^{(k)}=\lim_{k\to\infty}\begin{pmatrix}\dfrac{1}{k+1}\\ \left(1+\dfrac{1}{k}\right)^k\\ \dfrac{2k-1}{k+11}\end{pmatrix}=\begin{pmatrix}\lim\limits_{k\to\infty}\dfrac{1}{k+1}\\ \lim\limits_{k\to\infty}\left(1+\dfrac{1}{k}\right)^k\\ \lim\limits_{k\to\infty}\dfrac{2k-1}{k+11}\end{pmatrix}=\begin{pmatrix}0\\ e\\ 2\end{pmatrix}$$

在计算方法中,计算的向量序列都是数据序列,当 $\|X^{(k+1)}-X^{(k)}\|$ 小于给定精度时,取 $X^{(k+1)}$ 为极限向量 X^*。

0.4.2 矩阵范数

矩阵范数定义

矩阵范数可用向量范数定义。设 $A\in R^{n\times n}$,记方阵 A 的范数为 $\|A\|$,那么

$$\|A\|=\sup_{\substack{X\in R^n\\ X\neq 0}}\frac{\|AX\|}{\|X\|}\quad\text{或}\quad\|A\|=\sup_{\substack{X\in R^n\\ \|X\|=1}}\|AX\| \tag{0.7}$$

称为矩阵的范数。这样定义的矩阵范数满足下列性质:

$$\left.\begin{aligned}&\text{(i)}\ \|A\|\geq 0,\text{当且仅当 } A=0\text{ 时},\|A\|=0 && \text{(非负性)}\\ &\text{(ii)}\ \|\lambda A\|=|\lambda|\,\|A\|,\text{其中 } \lambda \text{ 为复数} && \text{(齐次性)}\\ &\text{(iii) 对于任意两个阶数相同的矩阵 } A,B \text{ 有}\\ &\|A+B\|\leq\|A\|+\|B\| && \text{(三角不等式)}\\ &\text{(iv) } A,B \text{ 为同阶矩阵}\|AB\|\leq\|A\|\,\|B\|\\ &\text{(v) } A \text{ 为 } n \text{ 阶矩阵,对}\forall X\in R^n,\text{恒有}\ \|AX\|\leq\|A\|\cdot\|X\| && \text{(相容性)}\end{aligned}\right\}\quad(0.8)$$

常用矩阵范数

向量有三种常用范数,相对应的矩阵范数的三种形式为:

$$\|A\|_1=\max_{1\leq j\leq n}\sum_{i=1}^{n}|a_{ij}| \tag{0.9}$$

$$\|A\|_\infty=\max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}| \tag{0.10}$$

$$\|A\|_2=\sqrt{\lambda_1}(\lambda_1\text{ 是 }A^TA\text{ 的最大特征值}) \tag{0.11}$$

证明*：既然矩阵的范数是 R^n 上满足 $\| X \| = 1$ 向量范数 $\| AX \|$ 的上确界，那么，找到这个上确界也就找到了矩阵的范数。

(1) 任取 $X \in R^n$，$\| X \|_1 = 1$，则

$$\| AX \|_1 = \sum_{i=1}^{n} |(AX)_i| = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} \alpha_{ij} x_j \right| \leq \sum_{i=1}^{n} \sum_{j=1}^{n} |\alpha_{ij}| |x_j|$$

$$= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} |\alpha_{ij}| \right) |x_j| \leq \left(\max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}| \right) \sum_{j=1}^{n} |x_j| = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}|$$

即

$$\| A \|_1 \leq \max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}|$$

另一方面设极大值在 k 列达到，即 $\max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}| = \sum_{i=1}^{n} |\alpha_{ik}|$，取 $X = e = (0,0,\cdots,0,1,0,\cdots,0)^T$，$e$ 除第 k 个分量为 1 外，其余分量均为 0，于是有

$$\| Ae \|_1 = \| (\alpha_{1k}, \alpha_{2k}, \cdots, \alpha_{nk})^T \|_1 = \sum_{i=1}^{n} |\alpha_{ik}| = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}|$$

由于 $\| e \|_1 = 1$，故 $\| A \|_1 \geq \sum_{i=1}^{n} |\alpha_{ik}| = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}|$，因此有

$$\| A \|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |\alpha_{ij}|$$

(2) 任取 X，$\| X \|_\infty = 1$，则

$$\| AX \|_\infty = \max_{1 \leq i \leq n} |(AX)_i| = \max_{1 \leq i \leq n} \left| \sum_{j=1}^{n} \alpha_{ij} x_j \right|$$

$$\leq \max_{1 \leq i \leq n} \sum_{j=1}^{n} |\alpha_{ij}| |x_j| \leq \max_{1 \leq i \leq n} \sum_{j=1}^{n} |\alpha_{ij}|$$

即

$$\| A \|_\infty \leq \max_{1 \leq i \leq n} \sum_{j=1}^{n} |\alpha_{ij}|$$

另一方面设极大值在 k 行达到，取 $X = e = (\mathrm{sign}\alpha_{k1}, \mathrm{sign}\alpha_{k2}, \cdots, \mathrm{sign}\alpha_{kn})^T$，这里

$$\mathrm{sign}\alpha = \begin{cases} 1, & \alpha \geq 0 \\ -1, & \alpha < 0 \end{cases}$$

于是

$$\| Ae \|_\infty = \sum_{j=1}^{n} |\alpha_{kj}|$$

即

$$\| A \|_\infty \geq \sum_{j=1}^{n} |\alpha_{kj}|$$

故

$$\| A \|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |\alpha_{ij}|$$

(3) A^TA 为对称非负矩阵，具有非负特征值，并具有 n 个相互正交的单位特征向量。

设 A^TA 的特征值为 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$，相应的特征向量为 u_i，其中 u_i 为相互正交的单位向量。设 $X = x_1 u_1 + x_2 u_2 + \cdots + x_n u_n$，并且 $\| X \|_2 = 1$，即 $\sum_{i=1}^{n} x_i^2 = 1$，则

$$A^TAX = \lambda_1 x_1 u_1 + \lambda_2 x_2 u_2 + \cdots + \lambda_n x_n u_n$$

$$\| AX \|_2^2 = (AX, AX) = (A^TAX, X) = \sum_{i=1}^{n} \lambda_i x_i^2 \leq \lambda_1 \sum_{i=1}^{n} x_i^2 = \lambda_1$$

即对任意 $\| X \|_2 = 1$ 均有 $\| AX \|_2 \leq \sqrt{\lambda_1}$，故

$$\| A \|_2 \leq \sqrt{\lambda_1}$$

取 $X = u_1$，则有

$$\| Au_1 \|_2^2 = \lambda_1$$

故 $$\| A \|_2 \geq \sqrt{\lambda_1}$$

于是 $$\| A \|_2 = \sqrt{\lambda_1}$$

如果 A 是对称矩阵，那么 $A^TA = AA$，设 A 的特征值是 t_i，则有

$$\| A \|_2 = \max \sqrt{\lambda_i} = \max | t_i |$$

按式(0.7)定义的矩阵范数满足矩阵范数的各种条件，并称它为从属于该向量范数的矩阵范数。由式(0.7)对一切非零向量 X，有

$$\| AX \| \leq \| A \| \| X \| \tag{0.12}$$

使得式(0.12)成立的矩阵与向量范数称为相容性。根据定义，对任一种从属范数有 $\| I \| = 1$，即单位矩阵的范数是 1。

还有一种与向量范数 $\| X \|_2$ 相容的矩阵范数，称为 Euclid 或 Schur 范数，用 $\| A \|_E$ 表示，其定义为

$$\| A \|_E = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} | a_{ij} |^2 \right)^{\frac{1}{2}} \tag{0.13}$$

因为 Euclid 范数易于计算，在实用中是一种十分有用的范数。但它不能从属于任何一种范数，因为 $\| I \|_E = n^{\frac{1}{2}}$。

与向量范数的等价性质类似，矩阵范数之间也是等价的。

例 0.4：$A = \begin{pmatrix} -1 & 2 \\ 3 & 7 \end{pmatrix}$，分别求 $\| A \|_1$，$\| A \|_\infty$，$\| A \|_2$，$\| A \|_E$。

解：$\| A \|_1 = \max\{ | -1 | + 3, 2 + 7 \} = 9$

$\| A \|_\infty = \max\{ | -1 | + 2, 3 + 7 \} = 10$

$$A^TA = \begin{pmatrix} -1 & 3 \\ 2 & 7 \end{pmatrix} \begin{pmatrix} -1 & 2 \\ 3 & 7 \end{pmatrix} = \begin{pmatrix} 10 & 19 \\ 19 & 53 \end{pmatrix}$$

A^TA 的特征值 $\lambda_1 = 60.19$，$\lambda_2 = 2.81$

$\| A \|_2 = \sqrt{60.19}$

$\| A \|_E = (1 + 4 + 9 + 49)^{1/2} = \sqrt{63}$

矩阵特征值与范数

若 λ 是矩阵 A 的特征值，$AX = \lambda X$，对任一对相容的矩阵与向量范数有

$$\| \lambda \| \| A \| = \| \lambda X \| = \| AX \| \leq \| A \| \| X \|$$

$$\| \lambda \| \leq \| A \| \tag{0.14}$$

即矩阵特征值的模不大于矩阵的任一范数。

谱半径

若 A 有特征值 $\lambda_1, \lambda_2, \cdots, \lambda_n$，记 $\rho(A) = \max|\lambda_r|$，则称 $\rho(A)$ 为 A 的谱半径。

有了谱半径的定义，矩阵的 2 范数可记为：$\| A \|_2 = \sqrt{\rho(A^T A)}$。

谱半径与矩阵范数

由矩阵谱半径定义，可得到矩阵范数的另一重要性质，$\rho(A) \leq \| A \|$。

本节内容的要点是学会计算向量和矩阵的范数，其中向量范数部分可放在第 3 章中，矩阵范数部分可放在第 6 章中。

第1章 插 值

1.1 插 值

什么是插值？简单地说，用给定的未知函数 $f(x)$ 的若干函数值的点构造 $f(x)$ 近似函数 $\varphi(x)$，称函数 $\varphi(x)$ 为插值函数。例如：在服装店订做风衣时，选择好风衣的样式后，服装师量出并记下你的胸围、衣长和袖长等几个尺寸，这几个尺寸就是风衣函数的插值点数值，在衣料上画出的裁剪线就是服装师构造的插值函数 $\varphi(x)$，裁剪水平的差别就在于量准插值点和构造合乎身材的插值函数。

在实际问题中，有时只能给出函数 $f(x)$ 在平面上的一些离散点的值 $\{(x_i, f(x_i))\}$，$i=0,1,\cdots,n$，而不能给出 $f(x)$ 的具体解析表达式，或者 $f(x)$ 的表达式过于复杂而难于运算。这时我们需要用近似函数 $\varphi(x)$ 来逼近函数 $f(x)$，在数学上常用的函数逼近的方法有：

·插值。

·一致逼近。

·均方逼近或称最小二乘法。

本章讨论用插值逼近函数的方法。

定义：$f(x)$ 为定义在区间 $[a,b]$ 上的函数，$x_0, x_1, \cdots, x_n$ 为 $[a,b]$ 上 $n+1$ 个互不相同的点，Φ 为给定的某一函数类。若 Φ 上有函数 $\varphi(x)$，满足

$$\varphi(x_i) = f(x_i), i = 0,1,\cdots,n$$

则称 $\varphi(x)$ 为 $f(x)$ 关于节点 $x_0, x_1, \cdots, x_n$ 在 Φ 上的插值函数；称点 $x_0, x_1, \cdots, x_n$ 为插值节点；称 $\{(x_i, f(x_i))\}$，$i=0,1,\cdots,n$ 为插值型值点，简称型值点或插值点；$f(x)$ 称为被插函数。

这样，对函数 $f(x)$ 在区间 $[a,b]$ 上的各种计算，就用对插值函数 $\varphi(x)$ 的计算取而代之。

构造插值函数需要关心下列问题：

·插值函数是否存在？

·插值函数是否唯一？

·如何表示插值函数？

·如何估计被插函数与插值函数的误差？

1.2 多项式插值的拉格朗日(Lagrange)型式

可对插值函数 $\varphi(x)$ 的类型作多种不同函数的选择，由于代数多项式具有简单和一些良好的特性，例如，多项式是无穷光滑的，容易计算它的导数和积分，故常选用代数多项

式作为插值函数。

1.2.1 线性插值

问题 1.1：给定两个插值点(x_0, y_0),(x_1, y_1),其中 $x_0 \neq x_1$,怎样做通过这两点的一次插值函数?

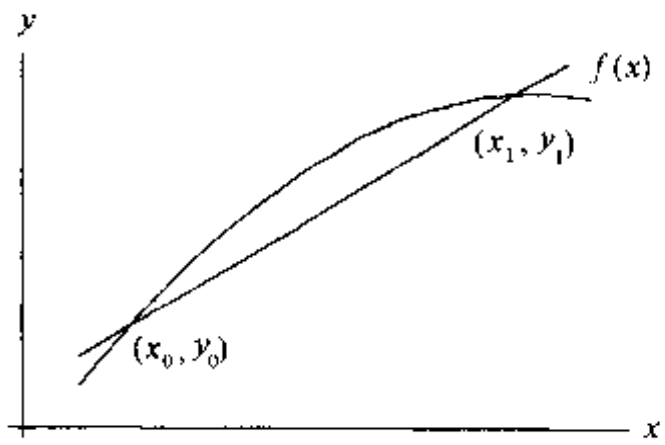


图 1.1 线性插值函数 $L_1(x)$

过两点作一条直线,这条直线就是通过这两点的一次多项式插值函数,简称线性插值。如图 1.1 所示。

在初等数学中,可用两点式、点斜式或截距式构造通过两点的一条直线。

下面先用待定系数法构造插值直线。

设直线方程为 $L_1(x) = ax + b$,将(x_0, y_0),(x_1, y_1)分别代入直线方程 $L_1(x)$得:

$$\begin{cases} ax_0 + b = y_0 \\ ax_1 + b = y_1 \end{cases}$$

当 $x_0 \neq x_1$ 时,因 $\begin{vmatrix} x_0 & 1 \\ x_1 & 1 \end{vmatrix} \neq 0$,所以方程组有解,而且解是唯一的。这也表明,平面上两个点,有且仅有一条直线通过。用待定系数法构造插值多项式的方法简单直观,容易看到解的存在性和唯一性,但要解一个方程组才能得到插值函数的系数,因工作量较大和不便向高阶推广,故这种构造方法通常不宜采用。

当 $x_0 \neq x_1$ 时,若用两点式表示这条直线,则有:

$$L_1(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{1.1}$$

这种型式称为拉格朗日插值多项式。

记 $l_0(x) = \frac{x - x_1}{x_0 - x_1}$,$l_1(x) = \frac{x - x_0}{x_1 - x_0}$,$l_0(x)$,$l_1(x)$称为插值基函数,计算 $l_0(x)$,$l_1(x)$的值,易见

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{1.2}$$

在拉格朗日插值多项式中可将 $L_1(x)$看做两条直线 $\frac{x - x_1}{x_0 - x_1} y_0$,$\frac{x - x_0}{x_1 - x_0} y_1$ 的迭加,并可看到两个插值点的作用和地位都是平等的。

拉格朗日插值多项式型式免除了解方程组的计算,易于向高次插值多项式型式推广。

线性插值误差

定理 1:记 $L_1(x)$为以(x_0, y_0),(x_1, y_1)为插值点的插值函数,这里 $y_0 = f(x_0)$,$y_1 = f(x_1)$,设 $f(x)$一阶连续可导,$f''(x)$在(a, b)上存在,则对任意给定的 $x \in [a, b]$,总存在一点 $\zeta \in [a, b]$,使

$$R(x) = f(x) - L_1(x) = \frac{f''(\zeta)}{2!}(x - x_0)(x - x_1), \zeta \in [x_0, x_1] \tag{1.3}$$

证明：令 $R(x)=f(x)-L_1(x)$，因 $R(x_0)=R(x_1)=0$，x_0,x_1 是 $R(x)$的根，所以可设 $R(x)=k(x)(x-x_0)(x-x_1)$。

对任何一个固定的点 x，引进辅助函数 $\Psi(t)$：

$$\Psi(t)=f(t)-L_1(t)-k(x)(t-x_0)(t-x_1)$$

则 $\Psi(x_i)=0,i=0,1$。

由定义可得 $\Psi(x)=0$，这样 $\Psi(t)$至少有 3 个零点，不失一般性，假定 $x_0<x<x_1$，分别在$[x_0,x]$和$[x,x_1]$上应用洛尔定理，可知 $\Psi'(t)$在每个区间至少存在一个零点，不妨记为 ζ_1 和 ζ_2，即 $\Psi'(\zeta_1)=0$ 和 $\Psi'(\zeta_2)=0$，对 $\Psi'(t)$在$[\zeta_1,\zeta_2]$上应用洛尔定理，得到 $\Psi''(t)$在$[\zeta_1,\zeta_2]$上至少有一个零点 ζ，$\Psi''(\zeta)=0$。

现在对 $\Psi(t)$求二次导数，其中 $L''_1(t)=0$($L_1(t)$为 t 的线性函数)，故有

$$\Psi''(t)=f''(t)-2!k(x)$$

代入 ζ，得 $f''(\zeta)-2!k(x)=0$，

$$\therefore k(x)=f''(\zeta)/2!$$

即

$$R(x)=\frac{f''(\zeta)}{2!}(x-x_0)(x-x_1),\zeta\in[x_0,x_1]。$$

1.2.2 二次插值

问题 1.2：给定三个插值点$(x_i,f(x_i))$，$i=0,1,2$，其中 x_i 互不相等，怎样构造函数 $f(x)$的二次(抛物线)插值多项式？

平面上的三个点能确定一条二次曲线，如图 1.2 所示。

仿造线性插值的拉格朗日插值型式，即用插值基函数的方法构造插值多项式。设 $L_2(x)=l_0(x)f(x_0)+l_1(x)f(x_1)+l_2(x)f(x_2)$，每个基函数 $l_i(x)$是一个二次函数，对 $l_0(x)$来说，要求 x_1,x_2 是它的零点，因此可设

$$l_0(x)=A(x-x_1)(x-x_2)$$

同理 $l_1(x),l_2(x)$也有相对应的形式，得

$$L_2(x)=A(x-x_1)(x-x_2)f(x_0)+B(x-x_0)(x-x_2)f(x_1)+C(x-x_0)(x-x_1)f(x_2)$$

将 $x=x_0$ 代入 $L_2(x)$，得

$$L_2(x_0)=A(x_0-x_1)(x_0-x_2)f(x_0)=f(x_0)$$

$$\therefore A=\frac{1}{(x_0-x_1)(x_0-x_2)}$$

$$l_0(x)=A(x-x_1)(x-x_2)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}$$

同理将 $x=x_1,x=x_2$ 代入 $L_2(x)$得到 B 和 C 的值，以及 $l_1(x)$和 $l_2(x)$的表达式。

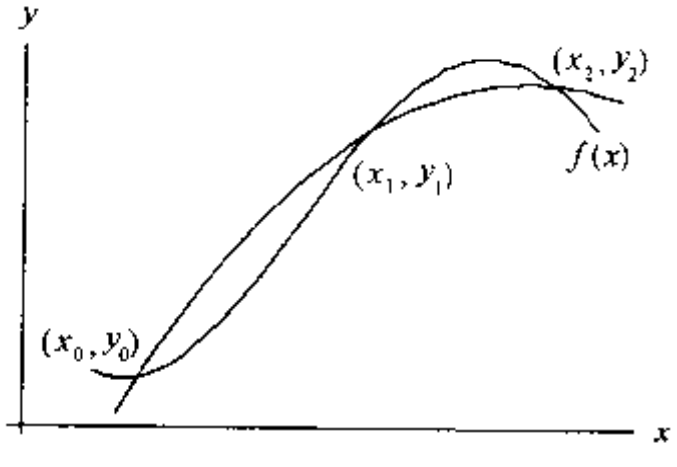


图 1.2 二次(抛物线)插值函数 $l_2(x)$

$$B = \frac{1}{(x_1 - x_0)(x_1 - x_2)}$$

$$C = \frac{1}{(x_2 - x_0)(x_2 - x_1)}$$

$$l_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$l_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

$$\therefore L_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2)$$

$$= \sum_{i=0}^{2} l_i(x) f(x_i) \tag{1.4}$$

也容易验证：

$$l_0(x_0) = 1, \quad l_0(x_1) = 0, \quad l_0(x_2) = 0,$$
$$l_1(x_0) = 0, \quad l_1(x_1) = 1, \quad l_1(x_2) = 0,$$
$$l_2(x_0) = 0, \quad l_2(x_1) = 0, \quad l_2(x_2) = 1。$$

插值基函数仍然满足：

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

二次插值函数误差

$$R_2(x) = \frac{f^{(3)}(\zeta)}{3!}(x - x_0)(x - x_1)(x - x_2), \zeta \in [\min\{x_0, x_1, x_2\}, \max\{x_0, x_1, x_2\}] \tag{1.5}$$

上式证明完全类似于线性插值误差的证明，故省略。

插值作为函数逼近方法，常用来作函数的近似计算。当计算点落在插值点区间之内时叫做内插，否则叫做外插。内插的效果一般优于外插。

例 1.1：给定 $\sin 11° = 0.190809$，$\sin 12° = 0.207912$。构造线性插值函数并用插值函数计算 $\sin 11°30'$ 和 $\sin 10°30'$。

解：构造线性插值函数：

$$L_1(x) = \frac{(x - 12)}{11 - 12} 0.190809 + \frac{(x - 11)}{12 - 11} 0.207912$$

分别将 $x = 11.5, x = 10.5$ 代入上式，得

$L_1(11.5) = 0.199361$，准确值 $\sin 11°30' = 0.199368$

$L_1(10.5) = 0.182258$，准确值 $\sin 10°30' = 0.182236$

$$R(x) = \frac{f''(\zeta)}{2!}(x - x_0)(x - x_1) = \frac{-\sin(\zeta)}{2}(x - 11)(x - 12)$$

$$|R(x)| \leq \frac{1}{2}|(11.5 - 11)(11.5 - 12)| = 0.125$$

例 1.2:给定 $\sin 11° = 0.190809, \sin 12° = 0.207912, \sin 13° = 0.224951$。构造二次插值函数并计算 $\sin 11°30'$。

解:$L_2(x) = \dfrac{(x-12)(x-13)}{(11-12)(11-13)}0.190809 + \dfrac{(x-11)(x-13)}{(12-11)(12-13)}0.207912$

$$+\frac{(x-11)(x-12)}{(13-11)(13-12)}0.224951$$

$L_2(11.5) = 0.199369$,准确值 $\sin 11°30' = 0.199368$

例 1.3:要制做三角函数 $\sin x$ 的函数值表,已知表值有四位小数,要求用线性插值引起的截断误差不超过表值的舍入误差,试决定其最大允许步长。

解:设最大允许步长 $h \approx h_i = x_i - x_{i-1}$

$$|R(x)| = \left|\frac{f''(\zeta)}{2!}(x-x_{i-1})(x-x_i)\right| = \left|\frac{\sin\zeta}{2}(x-x_{i-1})(x-x_i)\right|$$

$$\leq \frac{1}{2}|(x-x_{i-1})(x-x_i)| \leq \frac{1}{2}\left|\left(\frac{x_{i-1}+x_i}{2}-x_{i-1}\right)\left(\frac{x_{i-1}+x_i}{2}-x_i\right)\right|$$

$$= \frac{1}{8}|(x_{i-1}-x_i)(x_i-x_{i-1})| = \frac{h^2}{8} < \frac{1}{2}\cdot 10^{-4}$$

$$h \leq 0.02$$

1.2.3 n 次拉格朗日插值多项式

问题 1.3:给定平面上两个互不相同的插值点$(x_i, f(x_i))$,$i=0,1$,有且仅有一条通过这两点的直线;给定平面上三个互不相同的插值点$(x_i, f(x_i))$,$i=0,1,2$,有而且仅有通过这三个点的二次曲线;给定平面上 $n+1$ 个互不相同的插值点$(x_i, f(x_i))$,$i=0,1,2,\cdots,n$,互不相同是指 x_i 互不相等,是否有而且仅有一条不高于 n 次的插值多项式,如果曲线存在,那么如何简单地作出这条 n 次插值多项式曲线?

分析:n 次多项式 $P_n(x) = a_0 + a_1 x + \cdots + a_n x^n$,它完全由 $n+1$ 个系数 $a_0, a_1, \cdots, a_n$ 决定。若曲线 $P_n(x)$通过给定平面上 $n+1$ 个互不相同的插值点$(x_i, f(x_i))$,$i=0,1,2,\cdots,n$,则 $P_n(x)$应满足 $P_n(x_i) = f(x_i)$,$i=0,1,2,\cdots,n$,事实上一个插值点就是一个插值条件。

将$(x_i, f(x_i))$,$i=0,1,2,\cdots,n$ 依次代入 $P_n(x)$中得到线性方程组:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = f(x_0) \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = f(x_1) \\ \cdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = f(x_n) \end{cases} \tag{1.6}$$

方程组的系数行列式是范德蒙(Vandermonde)行列式:

$$V(x_0, x_1, \cdots, x_n) = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ & & \cdots & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix} = \prod_{0 \leq i \leq j \leq n} (x_i - x_j)$$

当 x_i 互异时，$\prod\limits_{0\le i\le j\le n}(x_i-x_j)\neq 0$，所以方程组(1.6)的解存在且唯一。即问题 1.3 的解存在而且唯一。

通过求解(1.6)得到插值多项式 $P_n(x)$，因其计算量太大而不可取，仿照线性以及二次插值多项式的拉格朗日型式，我们可构造 n 次拉格朗日插值多项式。

对于 $n+1$ 个互不相同的插值节点 $x_i, i=0,1,2,\cdots,n$，由 n 次插值多项式的唯一性，可对每个插值节点 x_i 作出相应的 n 次插值基函数 $l_i(x), i=0,1,2,\cdots,n$。

要求 $x_0, x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n$ 是 $l_i(x)$ 的零点，因此可设

$$l_i(x) = a_i(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)$$

由 $l_i(x_i)=1$，将 $x=x_i$ 代入 $l_i(x)$，得到

$$l_i(x_i) = a_i(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n) = 1$$

$$l_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \prod_{\substack{0\le j\le n\\ j\neq i}}\frac{x-x_j}{x_i-x_j} \tag{1.7}$$

作其组合：

$$L_n(x) = \sum_{i=0}^{n} l_i(x)f(x_i) \tag{1.8}$$

那么 $L_n(x)$ 不高于 n 次且满足 $L_n(x_i)=f(x_i), i=0,1,2,\cdots,n$，故 $L_n(x)$ 就是关于插值点 $x_0, x_1, \cdots, x_n$ 的插值多项式，这种插值型式称为拉格朗日插值多项式。$\{l_i(x)\}$ 称为关于节点 $\{x_i\}$ 的拉格朗日基函数。由插值多项式的存在唯一性，对于函数 $f(x)=x^k, k=0,1,\cdots,n$，关于节点 $x_0, x_1, \cdots, x_n$ 的拉格朗日插值多项式就是其本身，故拉格朗日基函数 $\{l_i(x)\}$ 满足

$$L_n(x) = \sum_{i=0}^{n} l_i(x)x_i^k = x^k, k = 0,1,\cdots,n$$

令 $k=0$，得到 $\sum\limits_{i=0}^{n} l_i(x) \equiv 1$。

例 1.4：给出下列插值节点数据，做三次拉格朗日插值多项式，并计算 $f(0.6)$。

x_i	-2.00	0.00	1.00	2.00
$f(x_i)$	17.00	1.00	2.00	17.00

解：基函数为：

$$l_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = \frac{(x-0)(x-1.00)(x-2.00)}{(-2.00-0)(-2.00-1.00)(-2.00-2.00)}$$

$$= -\frac{1}{24}x(x-1)(x-2)$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = \frac{1}{4}(x+2)(x-1)(x-2)$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = -\frac{1}{3}(x+2)x(x-2)$$

$$l_3(x)=\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}=\frac{1}{8}(x+2)x(x-1)$$

三次拉格朗日插值多项式：

$$L_3(x)=-\frac{17}{24}x(x-1)(x-2)+\frac{1}{4}(x+2)(x-1)(x-2)-\frac{2}{3}(x+2)x(x-2)$$
$$+\frac{17}{8}(x+2)x(x-1)$$

$$f(0.6)\approx L_3(x)=-0.472$$

n 次插值多项式的误差

定理 2：设 $L_n(x)$是$[a,b]$上过$(x_i,f(x_i))$，$i=0,1,\cdots,n$ 的 n 次插值多项式，当 $f\in C^{n+1}[a,b]$ 时，则插值多项式的误差：

$$R_n(x)=\frac{f^{(n+1)}(\zeta)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n),\text{其中 }\zeta\in[a,b] \tag{1.9}$$

证明*：记 $R_n(x)=f(x)-L_n(x)$。由于 $L_n(x_i)=f(x_i)$，$i=0,1,\cdots,n$，因而 $x_0,x_1,\cdots,x_n$ 是 $R_n(x)$的根，于是可设

$$R_n(x)=K(x)(x-x_0)(x-x_1)\cdots(x-x_n)$$

下面的目标是算出 $K(x)$，为此引入变量为 t 的函数 $\Psi(t)$：

$$\Psi(t)=f(t)-L_n(t)-K(x)(t-x_0)(t-x_1)\cdots(t-x_n) \tag{1.10}$$

令 $t=x_i$，得 $\Psi(x_i)=0,i=0,1,\cdots,n$

令 $t=x$，由定义 $\Psi(x)=f(x)-L_n(x)-K(x)(x-x_0)(x-x_1)\cdots(x-x_n)=0$

即 $\Psi(t)$至少有 $n+2$ 个零点，由于 $f\in C^{n+1}[a,b]$，由洛尔定理，$\Psi'(t)$在 $\Psi(t)$相邻的两个零点之间至少有一个零点，即 $\Psi'(t)$至少有 $n+1$ 个零点。同理再对 $\Psi''(t)$应用洛尔定理，即 $\Psi''(t)$至少有 n 个零点，反复应用洛尔定理得到 $\Psi^{(n+1)}(t)$至少有一个零点 ζ。

另一方面，对 $\Psi(t)$求 $n+1$ 阶导数，有

$$\Psi^{(n+1)}(t)=f^{(n-1)}(t)-K(x)(n+1)!$$

令 $t=\zeta$，有

$$0=\Psi^{(n+1)}(\zeta)=f^{(n+1)}(\zeta)-K(x)(n+1)!$$

得到 $K(x)=\dfrac{f^{(n+1)}(\zeta)}{(n+1)!}$ (1.11)

$$R_n(x)=\frac{f^{(n+1)}(\zeta)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n),\zeta\in[a,b]$$

由于 $\Psi^{(n+1)}(t)$的零点 ζ 与 $\Psi(t)$的零点 $x,x_0,\cdots,x_n$ 有关，因而 ζ 为 x 的函数。

若$|f^{(n+1)}(x)|\le M,x\in[a,b]$，则 $R_n(x)$可表示为

$$|R_n(x)|\le\frac{M}{(n+1)!}\prod_{i=0}^{n}|x-x_i| \tag{1.12}$$

定理 2 给出了当被插函数充分光滑时的插值误差或称插值余项表达式，但是，在实际计算中，并不知道 $f(x)$的具体表示，难以得到 $f^{(n+1)}(x)$的形式或较精确的界限 M，因此也难以得到界$|R_n(x)|$。在实际计算中，可对误差运用下面的事后估计方法。

给出 $n+2$ 个插值节点 $x_0,x_1,\cdots,x_{n+1}$，任选其中的 $n+1$ 个插值节点，不妨取 x_i，$i=0,1,\cdots,n$，构造一个 n 次插值多项式，记为 $L_n(x)$。在 $n+2$ 个插值节点中另选 $n+1$ 个

插值点，不妨取 $x_i, i=1,2,\cdots,n+1$，构造一个 n 次插值多项式，记为 $\tilde{L}_n(x)$。由定理 2，可得到

$$f(x)-L_n(x)=\frac{f^{(n+1)}(\zeta_1)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n) \tag{1.13}$$

$$f(x)-\tilde{L}_n(x)=\frac{f^{(n+1)}(\zeta_2)}{(n+1)!}(x-x_1)(x-x_2)\cdots(x-x_{n+1}) \tag{1.14}$$

设 $f^{(n+1)}(x)$ 在插值区间内连续而且变化不大，有 $f^{(n+1)}(\zeta_1)\approx f^{(n+1)}(\zeta_2)$，则

$$\frac{f(x)-L_n(x)}{f(x)-\tilde{L}_n(x)}\approx\frac{x-x_0}{x-x_{n+1}}$$

从而可得到

$$f(x)\approx\frac{x-x_{n+1}}{x_0-x_{n+1}}L_n(x)+\frac{x-x_0}{x_{n-1}-x_0}\tilde{L}_n(x) \tag{1.15}$$

$$f(x)-L_n(x)\approx\frac{x-x_0}{x_0-x_{n+1}}(L_n(x)-\tilde{L}_n(x)) \tag{1.16}$$

拉格朗日插值多项式的算法

下面用伪码描述拉格朗日插值多项式的算法。详细程序和算例请看 1.7 节。

1：输入：插值节点控制数 n，插值点序列 $(x_i,y_i), i=0,1,\cdots,n$，要计算的函数点 x。

2:FOR i := 0,1,…,n　　！i 控制拉格朗日基函数序列

{tmp: = 1;

2.1　FOR j: = 0,1,…,n

！对于给定 x，计算拉格朗日基函数 $l_i(x)=\prod\limits_{\substack{0\le j\le n\\ j\ne i}}\frac{x-x_j}{x_i-x_j}$

if (j< >i)then

tmp: = tmp * $(x-x_j)/(x_i-x_j)$;

}　　！tmp 表示拉格朗日基函数 $l_i(x)$

2.2　fx: = fx + tmp * y_i

}　　！计算 $L_n(X)=\sum\limits_{i=0}^{n}l_i(x)y_i=fx$

3：输出 $L_n(x)$ 的计算结果 fx。

伪码又称过程设计语言，主要用于描述算法。它是某种高级语言和自然语言的混杂语言，它取某种高级语言中的一些关键字，用于描述算法的结构化构造和数据说明等。伪码的语句中嵌有自然语言的叙述，伪码易于理解和修改，也易于转化为程序代码，是一种使用频率较高的写算法的语言。

在伪码中，惊叹号“!”表示注释语句。

1.3　多项式插值的牛顿(Newton)型式

拉格朗日插值多项式的优点是格式整齐和规范，它的缺点是没有承袭性质，当需要增

加插值节点时,不得不重新计算所有插值基函数 $l_i(x)$。本节给出具有承袭性质的牛顿插值多项式,并首先介绍在牛顿插值中需要用到的差商计算。

1.3.1 差商及其计算

一阶差商

称函数值的差 $f(x_1)-f(x_0)$ 与自变量的差 x_1-x_0 之比值为 $f(x)$ 关于点 x_0,x_1 的一阶差商,并记为 $f[x_0,x_1]$,即

$$f[x_0,x_1]=\frac{f(x_1)-f(x_0)}{x_1-x_0}$$

而称 $f[x_0,x_1,x_2]=\dfrac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$ 为 $f(x)$ 关于点 x_0,x_1,x_2 的二阶差商。

k 阶差商

设点 $x_0,x_1,\cdots,x_k$ 互不相同,$f(x)$ 关于点 $x_0,x_1,\cdots,x_k$ 的 k 阶差商为

$$f[x_0,x_1,\cdots,x_k]=\frac{f[x_1,x_2,\cdots,x_k]-f[x_0,x_1,\cdots,x_{k-1}]}{x_k-x_0}$$

差商有很多性质,我们仅列举其中的两条。

性质 1:k 阶差商 $f[x_0,x_1,\cdots,x_k]$ 是由函数值 $f(x_0),f(x_1),\cdots,f(x_k)$ 的线性组合而成。

$$f[x_0,x_1,\cdots,x_k]=\sum_{i=0}^{k}\frac{1}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_k)}f(x_i)$$

用归纳法可以证明这一性质。

例如:

$$f[x_0,x_1,x_2]=\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$$

$$=\frac{f(x_0)}{(x_0-x_1)(x_0-x_2)}+\frac{f(x_1)}{(x_1-x_0)(x_1-x_2)}+\frac{f(x_2)}{(x_2-x_0)(x_2-x_1)}$$

性质 2:若 $i_0,i_1,\cdots,i_k$ 为 $0,1,\cdots,k$ 的任一排列,则

$$f[x_0,x_1,\cdots,x_k]=f[x_{i_0},x_{i_1},\cdots,x_{i_k}]$$

该性质表明差商的值只与节点有关而与节点的顺序无关,即差商对节点具有对称性,这一性质由性质 1 可直接推出。

差商的计算

按照差商定义,用两个 $k-1$ 阶差商的值计算 k 阶差商,通常用差商表的形式计算和存放(见表 1.1)。

由于差商对节点具有对称性,可以任意选择两个 $k-1$ 阶差商的值计算 k 阶差商。例如:

$$f[x_0,x_1,x_2]=\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}=\frac{f[x_0,x_2]-f[x_0,x_1]}{x_2-x_1}$$

表 1.1　差商表

i	x_i	$f(x_i)$	一阶差商	二阶差商	三阶差商	n 阶差商
0	x_0	$f(x_0)$				
1	x_1	$f(x_1)$	$f[x_0,x_1]$			
2	x_2	$f(x_2)$	$f[x_1,x_2]$	$f[x_0,x_1,x_2]$		
3	x_3	$f(x_3)$	$f[x_2,x_3]$	$f[x_1,x_2,x_3]$	$f[x_0,x_1,x_2,x_3]$	
⋮	⋮	⋮	⋮	⋮	⋮	
n	x_n	$f(x_n)$	$f[x_{n-1},x_n]$	$f[x_{n-2},x_{n-1},x_n]$	$f[x_{n-3},\cdots,x_n]$	$f[x_0,x_1,\cdots,x_n]$

例 1.5:计算 $(-2,17)$, $(0,1)$, $(1,2)$, $(2,19)$的一至三阶差商。

i	x_i	$f(x_i)$	$f[x_{i-1},x_i]$	$f[x_{i-2},x_{i-1},x_i]$	$f[x_{i-3},x_{i-2},x_{i-1},x_i]$
0	-2	17			
1	0	1	-8		
2	1	2	1	3	
3	2	19	17	8	5/4

解:$f[-2,0]=(1-17)/(0-(-2))=-8$

$f[0,1]=(2-1)/(1-0)=1$

$f[1,2]=(19-2)/(2-1)=17$

$f[-2,0,1]=(f[0,1]-f[-2,0])/(1-(-2))=3$

$f[0,1,2]=(f[1,2]-f[0,1])/(2-0)=8$

$f[-2,0,1,2]=(f[0,1,2]-f[-2,0,1])/(2-(-2))=5/4$

1.3.2　牛顿插值

线性插值

问题 1.4:给定互不相同的插值点$(x_0,f(x_0))$,$(x_1,f(x_1))$,怎样构造线性插值函数的牛顿型式?

用点斜式构造线性插值函数,设 $N_1(x)=a+b(x-x_0)$,将 $x=x_0$,$x=x_1$ 代入得

$$N_1(x_0)=a=f(x_0)$$

$$N_1(x_1)=f(x_0)+b(x_1-x_0)=f(x_1)$$

所以
$$b=\frac{f(x_1)-f(x_0)}{x_1-x_0}=f[x_0,x_1]$$

得到线性插值的牛顿型式:

$$N_1(x)=f(x_0)+f[x_0,x_1](x-x_0) \tag{1.17}$$

由插值惟一性,线性插值的牛顿型式 $N_1(x)$与拉格朗日型式 $L_1(x)$为同一个多项式,仅是表达型式不同而已。

二次插值多项式

问题 1.5: 给定$(x_i, f(x_i))$, $i=0,1,2$,怎样构造二次牛顿插值多项式?

将二次多项式按幂次逐步递增的方式设置:

$$N_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

由 $N_2(x_0) = f(x_0)$, $N_2(x_1) = f(x_1)$,知 $a_0 + a_1(x - x_0)$就是 $f(x)$关于 x_0, x_1 的线性牛顿插值 $N_1(x)$,即

$$N_2(x) = N_1(x) + a_2(x - x_0)(x - x_1)$$

在构造 $N_1(x)$过程中已经计算出 a_0, a_1。现将 $x = x_2$ 代入上式得

$$N_2(x_2) = f(x_0) + f[x_0, x_1](x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = f(x_2)$$

整理得

$$a_2 = \left[\frac{f(x_2) - f(x_0)}{x_2 - x_0} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}\right] \Big/ (x_2 - x_1)$$

$$= \frac{f[x_0, x_2] - f[x_0, x_1]}{x_2 - x_1} = f[x_0, x_1, x_2]$$

故二次插值的牛顿型式:

$$N_2(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \quad (1.18)$$

n 次牛顿插值函数

问题 1.6: 给定$(x_i, f(x_i))$, $i=0,1,\cdots,n$,其中 x_i 互不相同,怎样构造 n 次牛顿插值多项式?

由一阶差商的定义 $f[x, x_0] = \dfrac{f(x) - f(x_0)}{x - x_0}$,得到 $f(x) = f(x_0) + (x - x_0)f[x, x_0]$;

类似地,由二阶差商至 n 阶差商的定义得到下列方程:

$$\begin{cases} f(x) = f(x_0) + (x - x_0)f[x, x_0] & (1) \\ f[x, x_0] = f[x_0, x_1] + (x - x_1)f[x, x_0, x_1] & (2) \\ f[x, x_0, x_1] = f[x_0, x_1, x_2] + (x - x_2)f[x, x_0, x_1, x_2] & (3) \\ \cdots\cdots \\ f[x, x_0, x_1, \cdots, x_{n-1}] = f[x_0, x_1, \cdots, x_n] + (x - x_n)f[x, x_0, x_1, \cdots, x_n] & (n+1) \end{cases}$$

用$(x - x_0)$乘(2)式,用$(x - x_0)(x - x_1)$乘(3)式,$\cdots$,$(x - x_0)\cdots(x - x_{n-1})$乘$(n+1)$式,所有等式相加,得到

$$\begin{aligned} f(x) = {} & f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \cdots \\ & + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f[x_0, x_1, \cdots, x_n] \\ & + (x - x_0)(x - x_1)\cdots(x - x_n)f[x, x_0, x_1, \cdots, x_n] \\ = {} & N(x) + R(x) \end{aligned}$$

其中

$$\begin{aligned} N(x) = {} & f(x_0) + (x - x_0)f[x_0, x_1] + \cdots \\ & + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f[x_0, x_1, \cdots, x_n] \end{aligned}$$

为不高于 n 次的多项式,可以验证 $N(x_i) = f(x_i)$,称 $N(x)$是过 $n+1$ 个插值点的 n 阶牛顿插值多项式。

$$R(x) = f[x, x_0, x_1, \cdots, x_n] \prod_{i=0}^{n} (x - x_i)$$

为插值多项式的误差。由插值多项式的唯一性得到拉格朗日插值多项式 $L(x)$与牛顿插值多项式 $N(x)$是完全相同的，它们仅是同一插值多项式的不同表达型式而已，因此得到拉格朗日插值多项式的误差与牛顿插值多项式的误差也完全相等。故当 $f \in C^{n+1}[a, b]$ 时，有

$$R(x) = \frac{f^{(n+1)}(\zeta)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) = f[x, x_0, \cdots, x_n] \prod_{i=0}^{n} (x - x_i)$$

$$\frac{f^{(n-1)}(\zeta)}{(n+1)!} = f[x, x_0, x_1, \cdots, x_n] \tag{1.19}$$

例 1.6：设 $f(x) = 10x^3 - 100x + 1$，计算 $f[x_0, x_1, x_2, x_3]$，$f[x_0, x_1, x_2, x_3, x_4]$。

解：$f[x_0, x_1, x_2, x_3] = \frac{f^{(3)}(\zeta)}{3!} = \frac{60}{6} = 10$

$$f[x_0, x_1, x_2, x_3, x_4] = \frac{f^{(4)}(\zeta)}{4!} = \frac{0}{4!} = 0$$

记 $N_n(x) = \sum_{i=0}^{n} t_i(x) f[x_0, \cdots, x_i]$。

其中 $t_0(x) \equiv 1$

$$t_i(x) = (x - x_{i-1}) t_{i-1}(x) = (x - x_0)(x - x_1) \cdots (x - x_{i-1}), \quad i = 1, 2, \cdots, n$$

也有 $\begin{cases} t_i(x_j) = 0 & j < i \\ t_i(x_j) \neq 0 & j \geq i \end{cases}$

可以看到拉格朗日插值多项式是基函数 $\{l_i(x)\}$ 的线性组合，牛顿插值多项式是基函数 $\{t_i(x)\}$ 的线性组合。

牛顿插值多项式的承袭性质表现在

$$N_k(x) = N_{k-1}(x) + t_k(x) f[x_0, x_1, \cdots, x_k]$$

对 $k-1$ 阶牛顿插值多项式 $N_{k-1}(x)$，只需增加一项 $t_k(x) f[x_0, x_1, \cdots, x_k]$，即可得到 k 阶牛顿插值多项式 $N_k(x)$。

例 1.7：给定下列插值节点的值，构造牛顿型式插值函数，计算 $N_2(0.9)$，$N_3(0.9)$。

x_i	-2	0	1	2
$f(x_i)$	17	1	2	19

解：$N_2(x) = f(x_0) + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2]$

取 $x_0 = -2, x_1 = 0, x_2 = 1$；用例 1.5 中算出的差商代入上式得到二阶牛顿插值多项式：

$$N_2(x) = 17 - 8(x + 2) + 3(x + 2)x$$

$$\therefore N_2(0.9) = 17.0 - 8(0.9 + 2) + 3(0.9 + 2) \cdot 0.9 = 1.63$$

$$N_3(x) = N_2(x) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$$

取 $x_0 = -2, x_1 = 0, x_2 = 1, x_3 = 2$；得到三阶牛顿插值多项式：

$$N_3(x) = 17 - 8(x+2) + 3x(x+2) + 1.25x(x+2)(x-1)$$

$$\therefore N_3(0.9) = N_2(0.9) + 1.25 \cdot 0.9 \cdot (x+0.9)(0.9-1.0) = 1.30375$$

牛顿插值的算法

在计算机上实现牛顿算法时,用递推形式易于实现牛顿插值多项式的承袭性质。下面为牛顿插值的算法。

1: 输入:插值节点数 n,插值点序列$\{x(i), f(i)\}, i = 1,2,\cdots,n$,要计算的插值点 u。

2:形成差商表 $g[k], k = 1,2,\cdots,n$　　! $g[k]$表示 $f[x_0, x_1, \cdots, x_k]$

3:置初始值 $t = 1$;newton $= f(0)$

4:对 $i = 1,2,\cdots,n$

$t = (u - x(i-1)) \cdot t$　　! 由 $t_k = (u - x_{k-1})t_{k-1}$形成$(u - x_0)\cdots(u - x_{i-1})$

newton = newton $+ t \cdot g[i]$

5:输出 $f(u)$的插值数值 $N_n(u)$ = newton。

从表 1.1 的差商计算中可以看到,给定 $n+1$ 个插值点,存放所有差商值需要 n^2 个存储单元,而在牛顿插值表达式中只用到 $f[x_0, x_1], f[x_0, x_1, x_2], \cdots, f[x_0, x_1, \cdots, x_n]$这些差商值,在计算中可用一维数组 $g[i], i = 0,1,2,\cdots,n$ 存放各阶差商值。

下面对算法的第 2 步"形成差商表 $g(k), k = 1,2,\cdots,n$"作进一步展开。

(1)对 $g[i], i = 0,1,2,\cdots,n$ 初始化,即 $g[j] = f(x_j), j = 0,1,2,\cdots,n$;

(2)除了 $g[0](f(x_0))$以外,其余函数值 $g[j], j = n, n-1,\cdots,1$ 在计算一阶差商后不再使用,因此可将一阶差商 $f[x_{j-1}, x_j]$放在 $g[j], j = n, n-1,\cdots,1$ 中,这时仍有 $g[0] = f(x_0)$;

(3)计算二阶差商后也不再调用 $f[x_{j-1}, x_j]$即 $g[j]$,可将 $f[x_{j-2}, x_{j-1}, x_j], j = n, n-1,\cdots,2$ 放在 $g[j]$中,这时有 $g[1] = f[x_0, x_1]$;

……

一直做到 n 阶差商$f[x_0, x_1, \cdots, x_n]$,这时仍有 $g[n] = f[x_0, x_1, \cdots, x_n]$。

详细的 C 语言程序和算例请看 1.7 节。

1.4* Hermite 插值

在构造插值时,如果不仅要求插值多项式节点的函数值与被插函数的函数值相同,还要求在节点处的插值函数与被插函数的一阶导数的值也相同,这样的插值称为 Hermite 插值或称密切插值。

常用 Hermite 插值描述如下:设 $f(x)$具有一阶连续导数,以及插值点 x_i, $i = 0,1,\cdots,n$, 若有至多为 $2n+1$ 次的多项式函数 $H_{2n+1}(x)$满足

$$H_{2n+1}(x_i) = f(x_i)$$

$$H'_{2n+1}(x_i) = f'(x_i), i = 0,1,\cdots,n$$

则称 $H_{2n+1}(x)$为 $f(x)$关于节点$\{x_i\}_{i=0}^n$的 Hermite 插值多项式。

问题 1.7:给定 $f(x_0) = y_0, f(x_1) = y_1, f'(x_0) = m_0, f'(x_1) = m_1$;怎样构造给定两个

节点的函数值和一阶导数值的 Hermite 插值多项式?

分析:用 4 个条件,至多可确定 3 次多项式。设满足插值条件的三次 Hermite 插值多项式为:

$$H_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

将插值条件代入 $H_3(x)$得到线性方程组:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 = y_1 \\ a_0 + 2a_2 x_0 + 3a_3 x_0^2 = m_0 \\ a_1 + 2a_2 x_1 + 3a_3 x_1^2 = m_1 \end{cases}$$

因为方程组的系数行列式

$$\begin{vmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 0 & 1 & 2x_0 & 3x_0^2 \\ 0 & 1 & 2x_1 & 3x_1^2 \end{vmatrix} = -(x_0 - x_1)^4 \neq 0$$

所以方程组有解,可唯一解出 a_0, a_1, a_2, a_3,即关于节点 x_0, x_1 的 Hermite 插值多项式存在唯一。类似于拉格朗日插值多项式的构造手法,也可通过插值基函数作出 $H_3(x)$。

设　$H_3(x) = h_0(x)y_0 + h_1(x)y_1 + g_0(x)m_0 + g_1(x)m_1$

要　$H_3(x_0) = h_0(x_0)y_0 = y_0$

可设　$h_0(x_0) = 1, h_1(x_0) = 0, g_0(x_0) = 0, g_1(x_0) = 0$

同理要　$H_3(x_1) = h_1(x_1)y_1 = y_1$

有　$h_0(x_1) = 0, h_1(x_1) = 1, g_0(x_1) = 0, g_1(x_1) = 0$

要　$H'_3(x_0) = m_0$

有　$h'_0(x_0) = 0, h'_1(x_0) = 0, g'_0(x_0) = 1, g'_0(x_0) = 0$

要　$H'_3(x_1) = m_1$

有　$h'_0(x_1) = 0, h'_1(x_1) = 0, g'_0(x_1) = 0, g'_1(x_1) = 1$

由上述要求,对 $h_0(x)$来说,至多为三次多项式,x_1 是它的二重根,可设

$$h_0(x) = (a_0 + b_0 x)\left(\frac{x - x_1}{x_0 - x_1}\right)^2 = (a_0 + b_0 x)l_0^2(x) \qquad (1.20)$$

利用　$h_0(x_0) = (a_0 + b_0 x_0)l_0^2(x_0) = (a_0 + b_0 x_0) = 1$

$$h'_0(x_0) = b_0 l_0^2(x) + (a_0 + b_0 x_0)2l_0(x_0)l'_0(x_0) = 0$$

解出　$h_0(x) = \left(1 + 2\frac{x - x_0}{x_1 - x_0}\right)\left(\frac{x - x_1}{x_0 - x_1}\right)^2$

同理可得

$$h_1(x) = \left(1 + 2\frac{x - x_1}{x_0 - x_1}\right)\left(\frac{x - x_0}{x_1 - x_0}\right)^2$$

由 $g_0(x_0) = 0, g(x_1) = g'(x_1) = 0$,可设 $g_0(x) = a(x - x_0)l_0^2(x)$。

由 $g'_0(x_0) = 1$,算出 $a = 1$。所以

$$g_0(x) = (x - x_0)l_0^2(x) \tag{1.21}$$

同理

$$g_1(x) = (x - x_1)l_1^2(x)$$

综上，得到以 x_0, x_1 为节点的 Hermite 插值为

$$H_3(x) = h_0(x)f(x_0) + h_1(x)f(x_1) + g_0(x)f'(x_0) + g_1(x)f'(x_1)$$

$$H_3(x) = \left(1 + 2\frac{x - x_0}{x_1 - x_0}\right)\left(\frac{x - x_1}{x_0 - x_1}\right)^2 f(x_0) + \left(1 + 2\frac{x - x_1}{x_0 - x_1}\right)\left(\frac{x - x_0}{x_1 - x_0}\right)^2 f(x_1)$$

$$+ (x - x_0)\left(\frac{x - x_1}{x_0 - x_1}\right)^2 f'(x_0) + (x - x_1)\left(\frac{x - x_0}{x_1 - x_0}\right)^2 f'(x_1) \tag{1.22}$$

容易证明，当 $f \in C^4[a,b]$ 时插值误差为

$$R(x) = f(x) - H_3(x) = \frac{f^{(4)}(\zeta)}{4!}(x - x_0)^2(x - x_1)^2, \zeta \in [a,b] \tag{1.23}$$

如果要构造 $f(x)$ 关于 $n+1$ 个节点 $x_i, i = 0,1,\cdots,n$ 的 $2n+1$ 次 Hermite 插值多项式，手法与构造两个节点的方法类似。

$$H_{2n+1}(x) = \sum_{i=0}^{n} h_i(x)f(x_i) + \sum_{i=0}^{n} g_i(x)f'(x_i)$$

这里 $h_i(x), g_i(x), i = 0,1,\cdots,n$ 分别为不高于 $2n+1$ 次插值多项式，分别满足

$$\begin{cases} h_i(x_j) = \delta_{ij} \\ h_i'(x_j) = 0 \end{cases} \text{及} \quad \begin{cases} g_i(x_j) = 0 \\ g_i'(x_j) = \delta_{ij} \end{cases}$$

由此可得到

$$h_i(x) = \left(1 - 2(x - x_i)\sum_{j \neq i}\frac{1}{x_i - x_j}\right)l_i^2(x) \tag{1.24}$$

$$g_i(x) = (x - x_i)l_i^2(x)$$

这里 $\{l_i(x)\}$ 为关于节点 $x_i, i = 0,1,\cdots,n$ 的拉格朗日基函数。

容易证明，当 $f \in C^{2n+2}[a,b]$ 时，误差为：

$$R(x) = f(x) - H_{2n+1}(x) = \frac{f^{(2n+2)}(\zeta)}{(2n+2)!}(x - x_0)^2(x - x_1)^2\cdots(x - x_n)^2, \zeta \in [a,b] \tag{1.25}$$

例 1.8：给定 $f(-1) = 0, f(1) = 4, f'(-1) = 2, f'(-1) = 0$，求 Hermite 插值多项式，并计算 $f(0.5)$。

解：$H_3(x) = g_0(x)\cdot 0 + g_1(x)\cdot 4 + h_0(x)\cdot 2 + h_1(x)\cdot 0$

显然本题不必计算 $g_0(x), h_1(x)$。

$$g_1(x) = \left(1 + 2\frac{(x-1)}{-1-1}\right)\left(\frac{x-(-1)}{1-(-1)}\right)^2 = (2 - x)(x+1)^2/4$$

$$h_0(x) = \left(x - (-1)\frac{x-1}{-1-1}\right)^2 = (x+1)(x-1)^2/4$$

$$H_3(x) = \frac{1}{4}(2 - x)(x+1)^2 + \frac{1}{4}(x+1)(x-1)^2$$

$$H_3(0.5) = 0.9375$$

利用构造基函数方法做插值多项式被广泛地应用在不同的插值条件中。

例 1.9:给定 $f(x_0)=y_0, f'(x_0)=m_0, f(x_1)=y_1$,构造二次插值多项式函数。

解:设 $P_2(x)=t_0(x)y_0+t_1(x)y_1+t_2(x)m_0$

这里 $t_0(x), t_1(x), t_2(x)$ 均为不高于二次的多项式,它们分别满足

$$\begin{cases} t_0(x_0)=1 \\ t_0'(x_0)=0, \\ t_0(x_1)=0 \end{cases} \quad \begin{cases} t_1(x_0)=0 \\ t_1'(x_0)=0, \\ t_1(x_1)=1 \end{cases} \quad \begin{cases} t_2(x_0)=0 \\ t_2(x_1)=0 \\ t_2'(x_0)=1 \end{cases}$$

于是 $P_2(x)$可表示为

$$P_2(x)=(a_0x+b_0)\frac{x-x_1}{x_0-x_1}y_0+a_1\left(\frac{x-x_0}{x_1-x_0}\right)^2y_1+a_2(x-x_0)(x-x_1)m_0$$

由 $t_0(x_0)=1$,得 $a_0x_0+b_0=1$

由 $t_0'(x_0)=0$,得 $a_0\dfrac{x_0-x_1}{x_0-x_1}+(a_0x_0+b_0)\dfrac{1}{x_0-x_1}=0$

所以 $a_0=-\dfrac{1}{x_0-x_1}, b_0=\dfrac{2x_0-x_1}{x_0-x_1}$

所以 $t_0(x)=\dfrac{(2x_0-x_1-x)}{x_0-x_1}\cdot\dfrac{(x-x_1)}{(x_0-x_1)}$

同理 $t_1(x)=\dfrac{(x-x_0)^2}{(x_1-x_0)^2}$

由 $t_2'(x_0)=1, a_2(x_0-x_1)=1$

所以 $a_2=\dfrac{1}{x_0-x_1}$

所以 $t_2(x)=\dfrac{(x-x_0)(x-x_1)}{x_0-x_1}$

$$P_2(x)=\frac{2x_0-x_1-x}{x_0-x_1}\cdot\frac{x-x_1}{x_0-x_1}y_0+\left(\frac{x-x_0}{x_1-x_0}\right)^2y_1+\frac{(x-x_0)(x-x_1)}{x_0-x_1}m_0$$

1.5 分段插值

1.5.1 龙格(Runge)现象

在构造插值多项式时,根据误差表达式(1.9),你是否认为多取插值点总比少取插值点的效果好呢?答案是不一定。如果被插函数是高次多项式,那么多取插值点比少取插值点效果好;但对有些函数来说,有时点取的越多,效果越不近人意。请看下面的例子。

例 1.10:给定函数 $f(x)=\dfrac{1}{1+25x^2}, x\in[-1,1]$。构造 10 次插值多项式$L_{10}(x)$。

对$[-1,1]$作等距分割,取 $h=\dfrac{2}{10}=0.2, x_i=-1+0.2i$,构造$\left(x_i,\dfrac{1}{1+25x_i^2}\right), i=0,1,\cdots,10$,10 次插值多项式 $L_{10}(x)$如图 1.3 所示。从图中可以看到,在零点附近,$L_{10}(x)$对$f(x)$的逼近效果较好,在 $x=-0.90,-0.70,0.70,0.90$ 时误差较大。

下面列出 $L_{10}(x)$和 $f(x)$的几个插值点的数值:

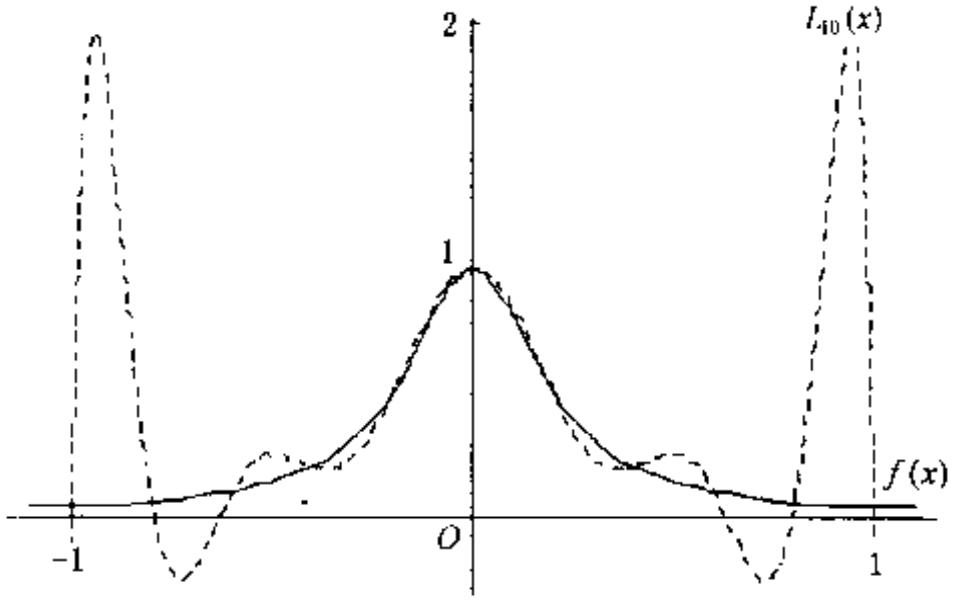


图 1.3　$L_{10}(x)$和 $f(x)$

x	-0.90	-0.70	-0.50	-0.30
$f(x)$	1.57872	0.07547	0.13793	0.30769
$L_{10}(x)$	0.04706	-0.22620	0.25376	0.23535

这个例子是由龙格提出的,也称插值多项式在插值区间内发生剧烈振荡的现象为龙格现象。龙格现象揭示了插值多项式的缺陷。它说明高次多项式的插值效果并不一定优于低次多项式的插值效果。

在插值过程中,误差由截断误差和舍入误差组成。式(1.9)给出的是截断误差,它是插值函数 $\varphi(x)$与原函数 $f(x)$的误差。另外由节点 y_i 和计算产生的舍入误差,在插值计算过程中可能被扩散或放大,这就是插值的稳定性问题。而高次多项式的稳定性一般比较差,这从另一角度说明了高次插值多项式的缺陷。

1.5.2　分段线性插值

既然增加插值点并不能提高插值函数的逼近效果,那么采用分段插值的效果又如何呢?

若对给定区间$[a,b]$作分割 $a = x_0 < x_1 < \cdots < x_n = b$,在每个小区间$[x_i, x_{i+1}]$上作 $f(x)$以 x_i, x_{i+1}为节点的线性插值,记这个插值函数为 $g(x) = g_i(x)$,则

$$g_i(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} f(x_i) + \frac{x - x_i}{x_{i+1} - x_i} f(x_{i+1}), x_i \leq x \leq x_{i+1} \tag{1.26}$$

把每个小区间的线性插值函数连接起来,就得到了 $f(x)$以剖分(节点)$a = x_0 < x_1 < \cdots < x_n = b$的分段线性函数$p(x)$。$p(x)$在$[x_i, x_{i+1}]$上为一个不高于一次的多项式,事实上 $p(x)$是平面上以点$(x_i, f(x_i))$为折点的折线。由线性插值误差公式,当 $x \in [x_i, x_{i+1}]$时,有

$$f(x) - p(x) = f(x) - s_i(x) = \frac{f^{(2)}(\zeta)}{2!}(x - x_i)(x - x_{i+1}) \tag{1.27}$$

因而

$$|f(x)-p(x)| \leq \frac{M_2}{2}|(x-x_i)(x-x_{i+1})|$$

$$\leq \frac{M_2}{2}\cdot\frac{1}{4}(x_{i+1}-x_i)^2 = \frac{M_2}{8}(x_{i+1}-x_i)^2$$

其中 $M_2=\max|f''(x)|$，$a\leq x\leq b$。

于是，当区间分割加密，$\max_i(x_{i+1}-x_i)\to 0$ 时，分段线性插值收敛于 $f(x)$。事实上，只要 $f(x)$连续，分段线性插值序列就能收敛于 $f(x)$。

分段线性插值算法简单，只要区间充分小，就能保证它的误差要求。它的一个显著优点是它的局部性质，如果修改了某节点$(x_i,f(x_i))$的值，仅在相邻的两个区间$[x_{i-1},x_i]$，$[x_i,x_{i+1}]$受到影响。分段线性插值的缺点是在插值节点处不光滑。

图 1.4 给出分段线性插值 $g_i(x)$(虚线表示)和 $f(x)$的图形，可以看到分段线性插值的效果明显好于整体的拉格朗日插值的效果。

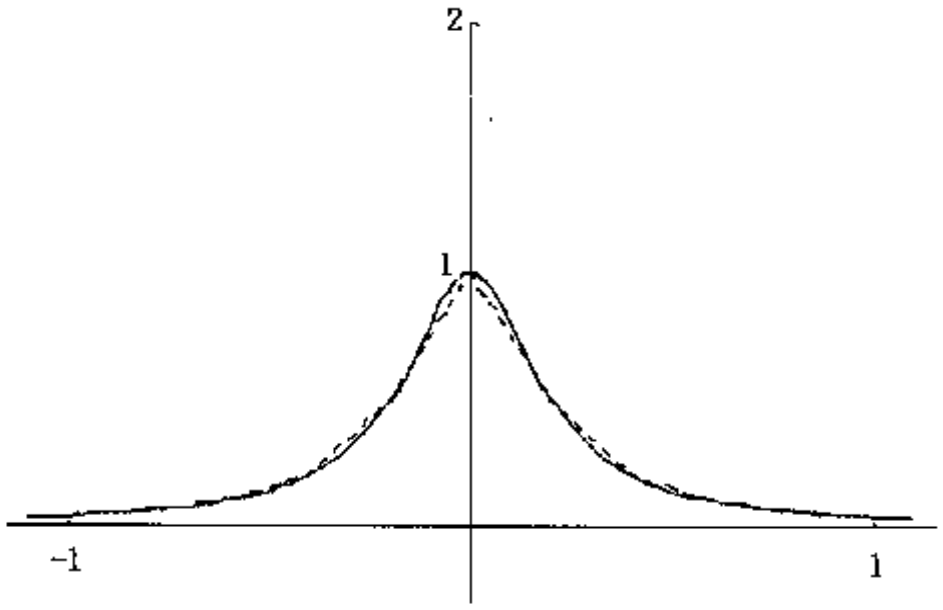


图 1.4　分段线性插值 $g(x)$和 $f(x)$

例 1.11：对下列数据作分段线性插值，并计算 $f(1.2)$，$f(3.3)$。

x_i	-3	-1	2	3	9
$f(x_i)$	12	12	1	6	12

解：$p(x)=g_i(x)=\dfrac{x-x_{i+1}}{x_i-x_{i+1}}f(x_i)+\dfrac{x-x_i}{x_{i+1}-x_i}f(x_{i+1}),x\in[x_i,x_{i+1}]$

$\because 1.2\in[-1,2]$

$\therefore p(1.2)=g_1(x)=\dfrac{1.2-2}{-1-2}\times 5+\dfrac{1.2+1}{2+1}\times 1=2.0667$

$\because 3.3\in[3,9]$

$\therefore p(3.3)=g_3(x)=\dfrac{3.3-9}{-6}\times 6+\dfrac{3.3-3}{6}\times 12=6.3$

1.6 三次样条函数

在制造船体和汽车外形等工艺中传统的设计方法是,首先由设计人员按外形要求,给出外形曲线的一组离散点值 $\{x_i, y_i\}$, $i = 0,1,\cdots,n$,施工人员准备好有弹性的样条(一般用竹条或有弹性的钢条)和压铁,将压铁放在点 $\{x_i, y_i\}$ 的位置上,调整竹条的形状,使其自然光滑,这时竹条表示一条插值曲线,我们称为样条函数。从数学上看,这一条近似于分段的 3 次多项式,在节点处具有一阶和二阶连续微商。样条函数的主要优点是它的光滑程度较高,保证了插值函数二阶导数的连续性,对于三阶导数的间断,人类的眼睛已难以辨认了。样条函数是一种隐式格式,最后需要解一个方程组,它的工作量大于多项式拉格朗日型式或牛顿型式等显式插值方法。

定义:给定区间 $[a,b]$ 上 $n+1$ 个节点 $a = x_0 < x_1 < \cdots < x_n = b$ 和这些点上的函数值 $f(x_i) = y_i$, $i = 0,1,\cdots,n$。若 $S(x)$ 满足 $S(x_i) = y_i, i = 0,1,\cdots,n$;$S(x)$ 在每个小区间 $[x_i, x_{i+1}]$ 上至多是一个三次多项式;$S(x)$ 在 $[a,b]$ 上有连续的二阶导数,则称 $S(x)$ 为 $f(x)$ 关于剖分 $a = x_0 < x_1 < \cdots < x_n = b$ 的三次样条插值函数,称 $x_0, x_1, \cdots, x_n$ 为样条节点。

要在每个子区间 $[x_i, x_{i+1}]$ 上构造三次多项式

$$S(x) = S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

$x \in [x_i, x_{i+1}]$, $i = 0,1,\cdots, n-1$,共需要 $4n$ 个条件,由插值条件 $S(x_i) = y_i, i = 0,1,\cdots, n$,提供了 $n+1$ 个条件;用每个内点的关系建立条件

$$S(x_i + 0) = S(x_i - 0)$$
$$S'(x_i + 0) = S'(x_i - 0)$$
$$S''(x_i + 0) = S''(x_i - 0), i = 1,\cdots, n-1$$

又得到 $3n-3$ 个条件;再附加两个边界条件,即可唯一确定样条函数了。用待定系数法确定了构造样条函数的存在性和唯一性。在具体构造样条函数时一般都不使用计算量大的待定系数法。下面给出构造三次样条插值的 M 关系式和 m 关系式的方法。

1.6.1 三次样条插值的 M 关系式

引入记号 $M_i = S''(x_i)$, $m_i = S'(x_i)$, $i = 0, 1,\cdots, n$。用节点处二阶导数表示样条插值函数时称为大 M 关系式,用一阶导数表示样条插值函数时称为小 m 关系式。

问题 1.8:给定插值点 (x_i, y_i), $i = 0, 1,\cdots, n$ 怎样构造用二阶导数表示的样条插值函数,即怎样构造 M 关系式?

假设 $S''(x_i) = M_i$, $i = 0, 1,\cdots, n$。由于 $S''(x)$ 在 $[x_i, x_{i+1}]$ 上为线性函数,故在 $[x_i, x_{i+1}]$ 上做 M 的分段线性插值函数:

$$S''(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} M_i + \frac{x - x_i}{x_{i+1} - x_i} M_{i+1}, i = 0,1,\cdots, n-1$$

令 $h_i = x_{i+1} - x_i$,得到

$$S''(x) = \frac{x_{i+1} - x}{h_i} M_i + \frac{x - x_i}{h_i} M_{i+1}, i = 0,1,2,\cdots,n-1 \tag{1.28}$$

对 $S''(x)$积分两次有

$$S(x) = \frac{(x_{i+1} - x)^3}{6h_i} M_i + \frac{(x - x_i)^3}{6h_i} M_{i+1} + cx + d$$

$$= \frac{(x_{i+1} - x)^3}{6h_i} M_i + \frac{(x - x_i)^3}{6h_i} M_{i+1} + C(x_{i+1} - x) + D(x - x_i) \tag{1.29}$$

将 $S(x_i) = y_i, S(x_{i+1}) = y_{i+1}$代入式(1.39)可解出

$$C = \frac{y_i}{h_i} - \frac{h_i M_i}{6}, D = \frac{y_{i+1}}{h_i} - \frac{h_i M_{i+1}}{6}$$

故在$[x_i, x_{i+1}]$上有

$$S(x) = \frac{(x_{i+1} - x)^3 M_i + (x - x_i)^3 M_{i+1}}{6h_i} + \frac{(x_{i+1} - x) y_i + (x - x_i) y_{i-1}}{h_i}$$

$$- \frac{h_i}{6}[(x_{i+1} - x) M_i + (x - x_i) M_{i+1}] \quad x \in [x_i, x_{i+1}], i = 1,2,\cdots,n \tag{1.30}$$

$S(x)$在每个小区间上具有不同的表达式,但由于 $S(x)$在整个区间$[a,b]$上是二阶光滑的,故有

$$S'(x_i + 0) = S'(x_i - 0), i = 1,2,\cdots,n-1$$

列出每一个关系式再经计算得:

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i, i = 1,2,\cdots,n-1 \tag{1.31}$$

其中:

$$\lambda_i = \frac{h_i}{h_i + h_{i-1}}$$

$$\mu_i = 1 - \lambda_i$$

$$d_i = \frac{6}{h_i + h_{i-1}} \left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right) = 6y[x_{i-1}, x_i, x_{i+1}]$$

由式(1.41) 得到 $n+1$ 个未知数的 $n-1$ 个方程组。现补充两个边界条件,使方程组只有唯一解。下面分三种情况讨论边界条件。

(1) 给定 M_0, M_n 的值($M_0 = 0, M_n = 0$时,称为自然边界条件),此时 $n-1$ 阶方程组有 $n-1$ 个未知量 $M_i, i = 1,2,\cdots,n-1$,即

$$\begin{bmatrix} 2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ & & & \mu_{n-1} & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 - \mu_1 M_0 \\ d_2 \\ \vdots \\ d_{n-2} \\ d_{n-1} - \lambda_{n-1} M_n \end{bmatrix} \tag{1.32}$$

(2) 给定 $S(x_0) = m_0, S'(x_n) = m_n$ 的值,它们分别代入 $S'(x)$在$[x_0, x_1], [x_{n-1}, x_n]$中的表达式,得到另外两个方程:

$$2M_0 + M_1 = \frac{6}{h_0}[y[x_0, x_1] - m_0] = d_0$$

$$M_{n-1} + 2M_n = \frac{6}{h_{n-1}}[m_n - y[x_{n-1}, x_n]] = d_n$$

于是需要解 $n+1$ 阶的方程组：

$$\begin{bmatrix} 2 & 1 & & & & \\ u_1 & 2 & \lambda_1 & & & \\ & u_2 & 2 & \lambda_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & u_{n-2} & 2 & \lambda_{n-1} \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{1.33}$$

(3) 被插函数以 $x_n - x_0$ 为基本周期时，即 $y_0 = y_n$，即 $S(x_0) = S(x_n)$，$S'(x_0) = S'(x_n)$，$S''(x_0) = S''(x_n)$；即 $m_0 = m_n$，$M_0 = M_n$。此时化为 n 个变量、n 个方程的方程组。

样条插值构造的 M 关系式是对角占优的三对角带状矩阵，可用第 5 章中的追赶法求解。

例 1.12：给出离散数值表：

x_i	1.1	1.2	1.4	1.5
y_i	0.4000	0.8000	1.6500	1.8000

取 $M_0 = M_n = 0$，构造三次样条插值的 M 关系式，并计算 $f(1.25)$。

解：由题中(x_i, y_i)的数值，计算得

$$h_0 = 0.1, h_1 = 0.2, h_2 = 0.1$$

$$\begin{cases} \lambda_1 = 0.6667, \\ \mu_1 = 0.3333, \end{cases} \quad \begin{cases} \lambda_2 = 0.3333 \\ \mu_2 = 0.6667 \end{cases}$$

$$d_1 = 5, \quad d_2 = -55$$

由 $M_0 = M_n = 0$ 的边界条件，得

$$\begin{bmatrix} 2 & 0.6667 \\ 0.6667 & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -55 \end{bmatrix}$$

解得 $M_1 = 13.125, M_2 = -31.875$。

因此，三次样条插值的分段表达式为

$$S(x) = \begin{cases} 21.875x^3 - 72.1875x^2 + 83.1875x - 32.875 & x \in [1.1, 1.2] \\ -37.5x^3 + 141.5625x^2 - 173.75x + 59.725 & x \in [1.2, 1.4] \\ 53.125x^3 - 239.0625x^2 + 358.0625x - 179.05 & x \in [1.4, 1.5] \end{cases}$$

特别地，$f(1.25) \approx S(1.25) = 1.0436$。

1.6.2 三次样条插值的 m 关系式

问题 1.9：给定插值点(x_i, y_i)，$i = 0, 1, \cdots, n$；怎样构造用节点处一阶导数表示的

样条插值函数,即怎样构造 m 关系式?

对给定的插值点$(x_i, y(x_i))$, $i=0,\cdots,n$,先假定已知 $S'(x_i)=m_i$,在每个小区间$[x_i, x_{i+1}]$上做 Hermite 插值,那么在整个$[x_0, x_n]$上是分段的 Hermite 插值,在$[x_i, x_{i+1}]$上$S(x)$的表达式为

$$S(x)=\left(1+2\frac{x-x_i}{x_{i+1}-x_i}\right)\left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right)^2 y_i+(x-x_i)\left(\frac{x-x_{i+1}}{x_i-x_{i+1}}\right)^2 m_i$$

$$+\left(1+2\frac{x-x_{i+1}}{x_i-x_{i+1}}\right)\left(\frac{x-x_i}{x_{i+1}-x_i}\right)^2 y_{i+1}+(x-x_{i+1})\left(\frac{x-x_i}{x_{i+1}-x_i}\right)^2 m_{i+1}$$

通过 $S''(x_i+0)=S''(x_i-0)$得到方程组

$$\lambda_i m_{i-1}+2m_i+\mu_i m_{i+1}=c_i, i=1,2,\cdots,n-1$$

其中:

$$\lambda_i=\frac{h_i}{h_i+h_{i-1}}$$

$$\mu_i=1-\lambda_i$$

$$c_i=3(\lambda_i y[x_{i-1}, x_i]+\mu_i y[x_i, x_{i+1}])$$

再附加两个边界条件,即可解出 m_i 的值。附加的边界条件情况同 M 关系式中的讨论类似,不再详述。

例 1.13:对 $f(x)=\frac{1}{1+25x^2}$ 作样条插值,插值效果见图 1.5。可以看到样条插值效果优于分段插值效果。

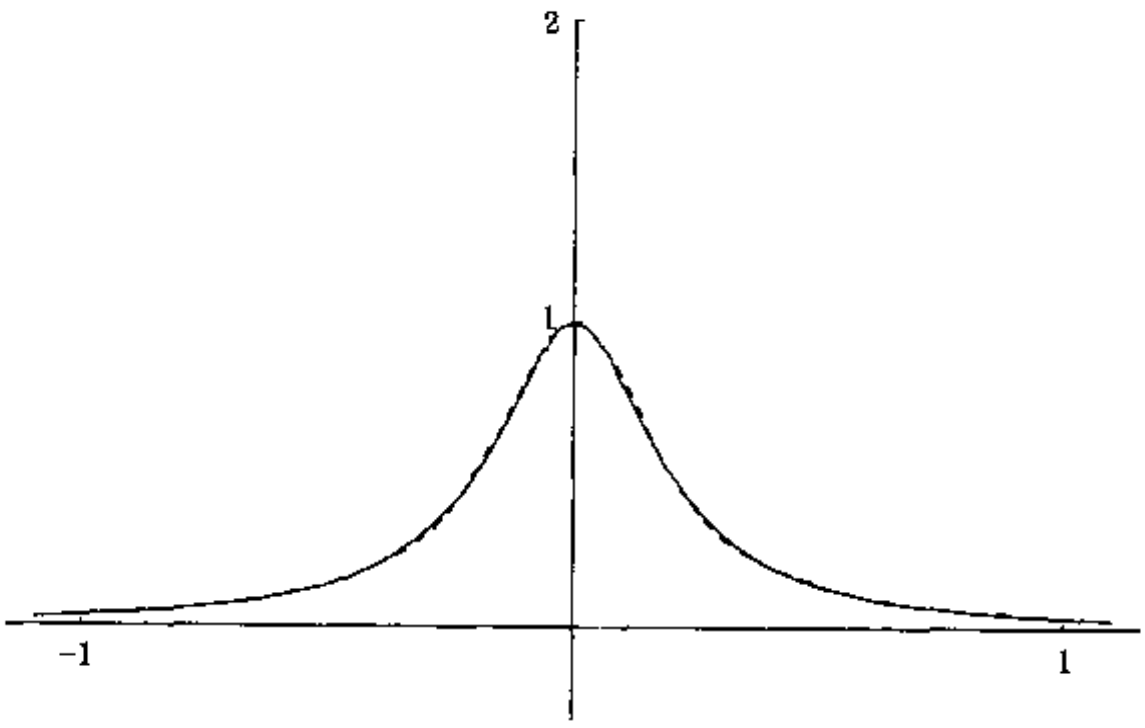


图 1.5 样条插值图示

1.7 程 序 示 例

程序 1 给定(x_i, y_i), $i=0,1,\cdots,n$,构造拉格朗日插值多项式 $L_n(x)$, 使得 $L_n(x_i)=y_i$, $i=0,1,\cdots,n$。

算法描述

输入 n 值,及$(x_i, y_i), i=0,1,\cdots,n$。

for i=0,1,…,n

对于给定 x,计算拉格朗日基函数 $l_i(x) = \prod_{\substack{0\le j\le n \\ j\neq i}} \frac{x - x_j}{x_i - x_j}$

计算 $L_n(x) = \sum_{i=0}^{n} l_i(x) y_i$

输出 $L_n(x)$。

程序源码

```
//////////////////////////////////////////////////////////////
// File : ch1_1.cpp (第 1 章 程序 1)                        //
// Purpose: (x_i,y_i)的拉格朗日插值多项式                   //
//////////////////////////////////////////////////////////////
# include <stdio.h>
# define MAX_N 20                          //定义(x_i,y_i)的最大维数
typedef struct tagPOINT                    //点的结构
{
    double x;
    double y;
} POINT;
int main()
{
    int n;
    int i, j;
    POINT points[MAX_N+1]; double l[MAX_N+1];
    double x, tmp, lagrange=0;
    printf("\nInput n value:");           //输入被插值点的数目
    scanf("%d", &n);
if (n>MAX_N)
{
    printf("The input n is larger then MAX_N, please redefine the MAX_N.\n");
    return 1;
}
if (n<=0)
{
    printf("Please input a number between 1 and %d.\n", MAX_N);
    return 1;
}
            //输入被插值点(x_i,y_i)
printf("Now input the (x_i, y_i), i=0,…,%d:\n", n);
for (i=0; i<=n; i++)
    scanf("%lf%lf", &points[i].x, &points[i].y);
```

```
printf("Now input the x value: "); //输入计算拉格朗日插值多项式的 x 值
scanf("%lf", &x);
for (i=0; i<=n; i++)
{
    for (j=0, tmp=1; j<=n; j++)
    {
  if (j==i) continue;
  tmp=tmp*(x-points[j].x)/(points[i].x-points[j].x);
}                                    //tmp 是拉格朗日基函数
    lagrange=lagrange+tmp*points[i].y;
}
printf("lagrange(%f)=%f\n", x, lagrange); //输出
return 0;
}
```

计算实例

给定 sin11°=0.190809,sin12°=0.207912,sin13°=0.224951，构造拉格朗日插值函数，计算 sin11°30′。

程序输入输出

```
Input n value: 2
Now input the (x_i,y_i), i=0,…,2;
11 0.190809 12 0.207912 13 0.224951
Now Input the x value: 11.5
lagrange(11.500000)=0.199369
```

程序 2　给定$(x_i, y_i), i=0,1,\cdots,n$,构造牛顿插值多项式 $N_n(x)$，使得 $N_n(x_i)=y_i, i=0,1,\cdots,n$。

算法描述

输入 n 值,及$(x_i, y_i), i=0,1,\cdots,n$;记 $f(x_i)=y_i$。

for i=0,1,…,n

计算差商 $f[x_0, x_1, \cdots, x_k]=\dfrac{f[x_1, x_2, \cdots, x_k]-f[x_0, x_1, \cdots, x_{k-1}]}{x_k-x_0}$，其中 $f[x_i]=f(x_i)$。

对给定的 x,由

$$N_n(x)=f(x_0)+(x-x_0)f[x_0, x_1]+(x-x_0)(x-x_1)f[x_0, x_1, x_2]+\cdots$$
$$+(x-x_0)(x-x_1)\cdots(x-x_0)f[x_0, x_1, \cdots, x_n]$$

计算出 $N_n(x)$的值。

输出 $N_n(x)$。

程序源码

```
//////////////////////////////////////////////////////
// Purpose: (x_i,y_i)的牛顿插值多项式 //
//////////////////////////////////////////////////////
```

```
# include <stdio.h>
# define MAX_N 20                          //定义(x_i,y_i)的最大维数
typedef struct tagPOINT                    //点的结构
{ double x;
  double y;
} POINT;
int main()
{ int n;
  int i, j;
    POINT points[MAX_N+1]; double diff[MAX_N+1];
    double x, tmp, newton=0;
    printf("\nInput n value:"); //输入被插值点的数目
    scanf("%d", &n);
    if (n>MAX_N)
    {
  printf("The input n is larger then MAX_N, please redefine the MAX_N.\n");
  return 1;
}
if (n<=0)
{
  printf("Please input a number between 1 and %d.\n", MAX_N);
  return 1;
}
        //输入被插值点(x_i,y_i)
printf("Now input the (x_i, y_i), i=0,…,%d:\n", n);
for (i=0; i<=n; i++)
    scanf("%lf%lf", &points[i].x, &points[i].y);
printf("Now input the x value:"); //输入计算牛顿插值多项式的x值
scanf("%lf", &x);
for (i=0; i<=n; i++) diff[i]=points[i].y;
for (i=0; i<n; i++)
    for (j=n; j>i; j--)
    {
  diff[j]=(diff[j]-diff[j-1])/(points[j].x-points[j-1-i].x);
}                                          //计算f(x_0,…,x_n)的差商
}
tmp=1; newton=diff[0];
for (i=0; i<n; i++)
{
  tmp=tmp*(x-points[i].x);
  newton=newton+tmp*diff[i+1];
}
```

```
        printf("newton(%f) = %f \n", x, newton); //输出
        return 0;
    }
```

计算实例

给定 $\sin11° = 0.190809$, $\sin12° = 0.207912$, $\sin13° = 0.224951$, 构造牛顿插值函数并计算 $\sin11°30'$。

程序输入输出

```
input n value: 2
Now input the (x_i,y_i), i=0,…,2:
11 0.190809 12 0.207912 13 0.224951
Now Input the x value: 11.5
newton(11.500000) = 0.199369
```

习题 1

1.1 作出插值点(-1.00, 3.00), (2.00, 5.00), (3.00, 7.00) 的二次拉格朗日插值多项式 $L_2(x)$, 并计算 $L_2(0)$。

1.2 作出插值点 (-2.00, 0.00), (2.00, 3.00), (5.00, 6.00)的二次拉格朗日插值多项式 $L_2(x)$, 并计算 $L_2(-1.2)$, $L_2(1.2)$。

1.3 作出下列插值点的三次拉格朗日插值多项式。

(1) (-1, 3), (0, -1/2), (1/2, 0), (1, 1)

(2) (-1, 2), (0, 0), (2, 1), (3, 3)

1.4 设 $f\in C^2[a,b]$, 且 $f(a)=f(b)=0$, 证明: $|f(x)| \leq \frac{1}{8}(b-a)^2 M_2$, $a \leq x \leq b$, 其中 $M_2 = \max\limits_{a\leq x\leq b} |f''(x)|$。

1.5 $f(x)=\sqrt{x}$ 在离散点有 $f(81)=9$, $f(100)=10$, $f(121)=11$, 用插值方法计算 $\sqrt{105}$ 的近似值, 并由误差公式给出误差界, 同时与实际误差作比较。

1.6 给出函数表:

x	-1.00	2.00	3.00	4.00
$f(x)$	3.00	5.00	7.00	5.00

作出差商表, 写出牛顿插值公式, 并计算 $f(1.2)$ 的近似值。

1.7 要作三角函数 $\log x$ 的函数值表, 已知表值有五位小数的近似值, 要求用线性插值引起的截断误差不超过表值的舍入误差, 试决定其最大允许步长。

1.8 $f(x)=x^7-125x^5+237x^3-999$, 计算差商 $f[2^0,2^1]$, $f[2^0,2^1,\cdots,2^7]$ 以及 $f[2^0,2^1,\cdots,2^8]$。

1.9 给出函数表:

x	1.05	1.10	1.15	1.20
$f(x)$	2.12	2.20	2.17	2.32

构造分段线性函数,并计算 $f(1.075)$ 和 $f(1.175)$的近似值。

1.10 给定数据 $f(0), f(1), f'(1)$,作出二次插值多项式,并写出插值余项。

1.11 给定数据 $f(3)=5.00, f(5)=15.00, f'(5)=7.00$,作出二次插值多项式,写出插值余项,并计算 $f(3.7)$的近似值。

1.12 给定数据 $f(0), f(1), f(3), f'(3)$,作出三次插值多项式,并写出插值余项。

1.13 给定数据 $f(0)=1.0, f(1)=0.75, f(3)=0.25, f'(3)=0.56$,作出三次插值多项式,并写出插值余项。

1.14 给定数据 $f(0)=0, f(1)=1, f'(0)=0, f'(1)=1, f''(1)=0$, 构造四次插值多项式,并写出插值余项。

1.15 给出下列数据:

x	-2.00	-1.00	1.00	2.00
$f(x)$	-4.00	3.00	5.00	12.00

试求满足上列数据和 $S''(-2)=0, S''(2)=0$ 的三次样条函数,并计算 $S(0)$的值。

1.16 给出下列数据:

x	-1.00	0.00	1.00	3.00
$f(x)$	2.00	3.00	4.00	29.00

试求满足上列数据和 $S'(-1)=5, S'(3)=29.00$ 的三次样条函数,并计算 $S(2)$的值。

第 2 章　数值微分和数值积分

2.1　数 值 微 分

2.1.1　差商与数值微分

当函数 $f(x)$是以离散点列给出时，当函数的表达式过于复杂时，常用数值微分近似计算 $f(x)$的导数 $f'(x)$。在微积分中，导数表示函数在某点上的瞬时变化率，它是平均变化率的极限；在几何上可解释为曲线的斜率；在物理上可解释为物体变化的速率。

以下是导数 $f'(x)$的三种定义形式：

$$f'(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h} = \lim_{h\to 0}\frac{f(x)-f(x-h)}{h} = \lim_{h\to 0}\frac{f(x+h)-f(x-h)}{2h} \tag{2.1}$$

在微积分中，用差商的极限定义导数；在数值计算中返朴归真，导数取用差商(平均变化率)作为其近似值。

最简单的计算数值微分的方法是用函数的差商近似函数的导数，即取极限的近似值。下面是与式(2.1)相应的三种差商形式的数值微分公式以及相应的截断误差。

向前差商

用向前平均变化率(差商)近似导数有：

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h} \tag{2.2}$$

其中 x_0+h 的位置在 x_0 的前面，因此称为向前差商。同理可得向后差商、中心差商的定义。

由泰勒展开

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}\times f''(\xi), x_0 \le \xi \le x_0+h$$

得向前差商的截断误差：

$$R(x) = f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} = -\frac{h}{2}f''(\xi) = O(h)$$

向后差商

用向后差商近似导数有：

$$f'(x_0) \approx \frac{f(x_0)-f(x_0-h)}{h} \tag{2.3}$$

与计算向前差商的方法类似，由泰勒展开得向后差商的截断误差：

$$R(x) = f'(x_0) - \frac{f(x_0)-f(x_0-h)}{h} = O(h)$$

中心差商

用中心平均变化率(差商)近似导数有：

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0-h)}{2h} \tag{2.4}$$

由泰勒展开

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f'''(\xi_1)$$

$$f(x_0-h) = f(x_0) - hf'(x_0) + \frac{h^2}{2!}f''(x_0) - \frac{h^3}{3!}f'''(\xi_2)$$

得中心差商的截断误差阶：

$$R(x) = f'(x_0) - \frac{f(x_0+h)-f(x_0-h)}{2h} = \frac{h^2}{12}[f'''(\xi_1)+f'''(\xi_2)]$$

$$= \frac{h^2}{6}f'''(\xi) = O(h^2) \qquad x_0 - h \leq \xi \leq x_0 + h$$

差商的几何意义

微积分中的极限定义 $f'(x_0) = \lim\limits_{h\to 0}\frac{f(x_0+h)-f(x_0)}{h}$，表示 $f(x)$ 在 $x = x_0$ 处切线的斜率，即图 2.1 中直线 P 的斜率；差商 $\frac{f(x_0+h)-f(x_0)}{h}$ 表示过 $(x_0, f(x_0))$ 和 $(x_0+h, f(x_0+h))$ 两点直线 Q 的斜率，是一条过 x_0 的割线。可见数值微分是用近似值内接弦的斜率代替准确值切线的斜率。

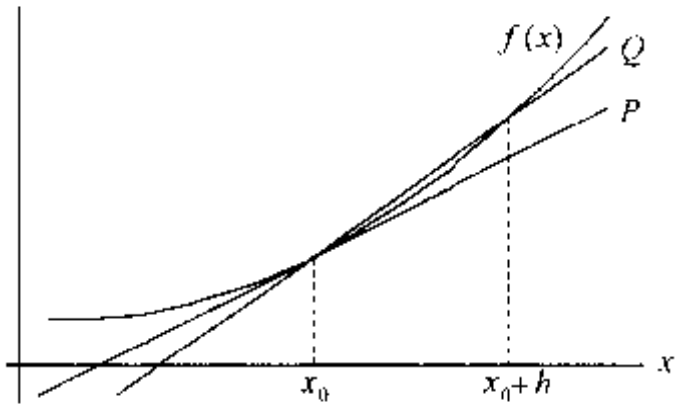


图 2.1　微商与差商

例 2.1：给出下列数据，计算 $f'(0.02), f'(0.06), f'(0.10), f''(0.08)$。

x	0.02	0.04	0.06	0.08	0.10
$f(x)$	5.06	5.07	5.065	5.05	5.055

解：$f'(0.02) \approx (5.07-5.06)/(0.04-0.02) = 0.5$

$f'(0.06) \approx (5.05-5.07)/(0.08-0.04) = -0.5$

$f'(0.10) \approx (5.05-5.055)/(0.08-0.10) = 0.25$

$f''(0.08) \approx (f'(0.10)-f'(0.06))/(0.10-0.06) = 18.75$

设定最佳步长

在计算数值导数时，它的误差由截断误差和舍入误差两部分组成。用差商或插值公式近似导数产生截断误差，由原始值 y_i 的数值近似产生舍入误差。在差商计算中，从截断误差的逼近阶的角度看，$|h|$ 越小，则误差也越小；但是太小的 $|h|$ 会带来较大的舍入误差。怎样选择最佳步长，使截断误差与舍入误差之和最小呢？

一般对计算导数的近似公式进行分析可得到误差的表示式，以中心差商为例，截断误差不超过

$$\frac{h^2}{6}M_3 = \frac{h^2}{6}\max|f'''(x)|$$

而舍入误差可用量$\frac{e}{h}$估计(证明略),其中 e 是函数 y_i 的原始值的绝对误差限,总误差为

$$\frac{h^2}{6}M_3 + \frac{e}{h}$$

当$\left(\frac{h^2}{6}M_3 + \frac{e}{h}\right)' = \frac{h}{3}M_3 - \frac{e}{h^2} = 0$时,总误差达到最小值,即

$$h = \sqrt[3]{\frac{3e}{M_3}} \qquad (*)$$

可以看到用误差的表达式确定步长,难度较大,难以实际操作。

通常用事后估计方法选取步长 h,例如,记 $D(h)$,$D\left(\frac{h}{2}\right)$为步长等于 h,$\frac{h}{2}$的差商计算公式,给定误差界 ϵ,当$\left|D(h) - D\left(\frac{h}{2}\right)\right| < \epsilon$时,$\frac{h}{2}$就是合适的步长。

例 2.2:对函数 $y = e^x$,取不同的步长计算 $f'(1.15)$,观察误差变化规律,从而确定最佳步长。

解:

h	$f'(1.15)$	误差	h	$f'(1.15)$	误差
0.10	3.1630	-0.0048	0.05	3.1590	-0.0008
0.09	3.1622	-0.0040	0.04	3.1588	-0.0006
0.08	3.1613	-0.0031	0.03	3.1583	-0.0001
0.07	3.1607	-0.0025	0.02	3.1575	0.0007
0.06	3.1600	-0.0018	0.01	3.1550	0.0032

表中数据显示,当步长 h 从 0.10 减少到 0.03 时,数值微分误差的绝对值从 0.0048 减少到 0.0001,而随着 h 的进一步减少,误差的绝对值又有所反弹,表明当步长 h 小于 0.03时,舍入误差起了主要作用。函数值取了四位数值,表明 $e = 0.5 \times 10^{-4}$,由(*)式可算出 $h = 0.035$。

在实际计算中是无法得到误差的准确数值的,这时以$|D(h_i) - D(h_j)|$最小为标准确定步长,本例中取 $h = 0.04$。

2.1.2 插值型数值微分

对于给定的 $f(x)$的函数表,建立插值函数 $L(x)$,用插值函数 $L(x)$的导数近似函数 $f(x)$的导数。

设 $x_i, i = 0,1,\cdots,n$ 为$[a,b]$上的节点,给定$(x_i, f(x_i)), i = 0,1,\cdots,n$,以$(x_i, f(x_i))$为插值点构造插值多项式 $L_n(x)$,以 $L_n(x)$的各阶导数近似 $f(x)$的相应阶的导数,即

$$f(x) = L_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i)$$

$$f'(x) \approx L_n'(x) = \sum_{i=0}^{n} l_i'(x) f(x_i)$$

当 $x = x_j$ 时，$f'(x_j) = \sum_{i=0}^{n} l_i'(x_j) f(x_i), j = 0, 1, \cdots, n$ (2.5)

误差项

$$R(x) = \frac{d}{dx}\left(\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)\right)$$

$$R(x_j) = \prod_{\substack{i=0 \\ i \neq j}}^{n}(x_j - x_i)\frac{f^{(n+1)}(\xi)}{(n+1)!}$$

例 2.3：给定$(x_i, f(x_i)), i = 0, 1, 2$，并有 $x_2 - x_1 = x_1 - x_0 = h$，计算 $f'(x_0), f'(x_1), f'(x_2)$。

解：作过$(x_i, f(x_i)), i = 0, 1, 2$ 的插值多项式：

$$L_2(x) = \frac{(x - x_1)(x - x_2)}{2h^2}f(x_0) + \frac{(x - x_0)(x - x_2)}{-h^2}f(x_1) + \frac{(x - x_0)(x - x_1)}{2h^2}f(x_2)$$

$$f'(x) = L_2'(x) = \frac{f(x_0)}{2h^2}(x - x_1 + x - x_2) - \frac{f(x_1)}{h^2}(x - x_0 + x - x_2) + \frac{f(x_2)}{2h^2}(x - x_0 + x - x_1)$$

将 $x = x_i$ 代入$f'(x)$得三点公式：

$$f'(x_0) = \frac{1}{2h}(-3f(x_0) + 4f(x_1) - f(x_2))$$

$$f'(x_1) = \frac{1}{2h}(-f(x_0) + f(x_2))$$

$$f'(x_2) = \frac{1}{2h}(f(x_0) - 4f(x_1) + 3f(x_2))$$

利用泰勒展开进行比较和分析，可得三点公式的截断误差是 $O(h^2)$。

2.1.3 样条插值数值微分

把离散点按大小排列成 $a = x_0 < x_1 < \cdots < x_n = b$，用 m 关系式构造插值点$(x_i, f(x_i)), i = 0, 12, \cdots, n$ 的样条函数$S(x)$：

$$S(x) = \left(1 + 2\frac{x - x_i}{x_{i+1} - x_i}\right)\left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 y_i + (x - x_i)\left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 m_i + \left(1 + 2\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)\left(\frac{x - x_i}{x_{i+1} - x_i}\right)^2 y_{i+1} + (x - x_{i+1})\left(\frac{x - x_i}{x_{i+1} - x_i}\right)^2 m_{i+1}$$

若 $x = x_i$ 则$f'(x_i) = m_i$；当 $x \in (x_i, x_{i+1})$时，可用 $S'(x) \approx f'(x)$计算导数。

2.2 数值积分

在微积分中用牛顿-莱布尼兹公式计算连续函数 $f(x)$的定积分：

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

但是，当被积函数是以点列$(x_i, f(x_i))$，$i = 0, 1, \cdots, n$ 的形式给出时；当被积函数$f(x)$的原函数 $F(x)$难以得到时，例如 $\int_1^2 \sin x^2\,dx$，则无法用牛顿-莱布尼兹积分公式计算。有时当被积函数的原函数过于复杂时，也不宜套用积分公式计算积分，而用数值积分公式计算定积分。

在微积分中，定积分是黎曼(Rimann)和的极限，它是分割小区间 n 趋于零时的极限，即

$$\int_a^b f(x)\,dx = \lim_{\Delta x_i \to 0}\left(\sum_{i=0}^{n-1} f(x_i)\Delta x_i\right)$$

在数值积分公式中，只能用有限项的和近似上面的极限，通常由函数在离散点函数值的线性组合形式给出。

记 $I(f) = \int_a^b f(x)\,dx$，$I_n(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$，在本章中，用 $I(f)$表示精确积分值，用$I_n(f)$表示近似积分值，$\{x_i\}$称为积分节点，α_i 称为积分系数。确定 $I_n(f)$中积分系数 α_i 的过程就是构造数值积分公式的过程。

怎样判断数值积分公式的效果？代数精度是衡量数值积分公式优劣的重要标准之一。

代数精度

记$[a, b]$上以$(x_i, f(x_i))$，$i = 0, 1, \cdots, n$ 为积分节点的数值积分公式为

$$I_n(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$$

若 $I_n(f)$满足 $E_n(x^k) = I(x^k) - I_n(x^k) = 0$，$k = 0, 1, \cdots, m$，而 $E_n(x^{m+1}) \neq 0$，则称$I_n(f)$具有 m 阶代数精度。

由此可知当 $I_n(f)$具有 m 阶代数精度时，对任意的 m 阶多项式都有 $I(f) = I_n(f)$。

2.2.1 插值型数值积分

对给定的被积函数在$[a, b]$上的点列$(x_i, f(x_i))$，$i = 0, 1, \cdots, n$，作拉格朗日插值多项式 $L_n(x)$，以 $\int_a^b L_n(x)\,dx$ 近似计算 $\int_a^b f(x)\,dx$，即

$$\int_a^b f(x)\,dx \approx \int_a^b L_n(x)\,dx = \int_a^b \sum_{i=0}^{n} l_i(x) f(x_i)\,dx = \sum_{i=0}^{n}\left[\int_a^b l_i(x)\,dx\right] f(x_i)$$

记 $\alpha_i = \int_a^b l_i(x)\,dx$，则有

$$I_n(f) = \int_a^b L_n(x)\,dx = \sum_{i=0}^{n} \alpha_i f(x_i)$$

数值积分误差，也就是对插值误差的积分值

$$E_n(f) = \int_a^b R_n(x)\,dx = \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\xi(x)) \prod_{i=0}^{n} (x - x_i)\,dx$$

或

$$E_n(f) = \int_a^b f[x_0, x_1, \cdots, x_n, x] \prod_{i=0}^{n} (x - x_i)\, dx$$

对一般的函数 $E_n(f) \neq 0$，但若 $f(x)$是一个不高于 n 次的多项式，由于 $f^{(n+1)}(x) = 0$，而有 $E_n(f) = 0$。因此，n 阶插值多项式型式的数值积分公式至少有 n 阶代数精度。

例 2.4: 建立[0,2]上节点为 $x_0 = 0, x_1 = 0.5, x_2 = 2$ 的数值积分公式。

解：由 $\alpha_i = \int_a^b l_i(x)\, dx$ 得

$$\alpha_0 = \int_0^2 l_0(x)\, dx = \int_0^2 \frac{(x - 0.5)(x - 2)}{(0 - 0.5)(0 - 2)}\, dx = -\frac{1}{3}$$

$$\alpha_1 = \int_0^2 l_1(x)\, dx = \int_0^2 \frac{(x - 0)(x - 2)}{(0.5 - 0)(0.5 - 2)}\, dx = \frac{16}{9}$$

$$\alpha_2 = \int_0^2 l_2(x)\, dx = \int_0^2 \frac{(x - 0)(x - 0.5)}{(2 - 0)(2 - 0.5)}\, dx = \frac{5}{9}$$

得到数值积分公式：

$$I_2(f) = \frac{1}{9}[-3f(0) + 16f(0.5) + 5f(2)]$$

2.2.2 牛顿-柯特斯(Newton-Cote's)积分

把积分区间$[a, b]$分成 n 等分，记步长为 $h = \frac{b - a}{n}$，取等分点 $x_i = a + ih\ (i = 0, 1, \cdots, n)$作为数值积分节点，构造拉格朗日插值多项式 $L_n(x)$，取

$$\int_a^b f(x)\, dx \approx \int_a^b L_n(x)\, dx$$

由此得到的数值积分称为牛顿－柯特斯积分。下面可以看到，牛顿-柯特斯积分系数和积分节点以及积分区间无直接关系，系数固定而易于计算。

梯形积分

以$(a, f(a))$和$(b, f(b))$为插值节点构造线性函数 $L_1(x)$，有

$$\int_a^b f(x)\, dx \approx \int_a^b L_1(x)\, dx$$

那么，

$$\int_a^b L_1(x)\, dx = \int_a^b (l_0(x)f(x_0) + l_1(x)f(x_1))\, dx$$

$$\alpha_0 = \int_a^b l_0(x)\, dx = \int_a^b \frac{x - b}{a - b} dx = \frac{1}{2}(b - a) = (b - a)c_0^{(1)}$$

$$\alpha_1 = \int_a^b l_1(x)\, dx = \int_a^b \frac{x - a}{b - a} dx = \frac{1}{2}(b - a) = (b - a)c_1^{(1)}$$

提取公因子$(b - a)$后，得到牛顿-柯特斯积分的组合系数：$c_0^{(1)} = \frac{1}{2}, c_1^{(1)} = \frac{1}{2}$，它们已与积分区间没有任何关系了。

$$\int_a^b f(x)\, dx \approx \frac{b - a}{2}[f(a) + f(b)]$$

记
$$T(f) = \frac{b-a}{2}[f(a)+f(b)] \tag{2.6}$$

称 $T(f)$为梯形积分公式。它的几何意义是用梯形面积近似代替积分值(图 2.2)。

怎样确定梯形积分公式的代数精度?

取 $f(x)=x$ 时,有

$$I(f) = \int_a^b x\,dx = \frac{b-a}{2}(f(a)+f(b)) = \frac{b^2-a^2}{2} = T(f)$$

取 $f(x)=x^2$ 时,有

$$I(f) = \int_a^b x^2\,dx \neq \frac{b-a}{2}(f(a)+f(b)) = T(f)$$

得梯形求积公式具有一阶代数精度。由

$$f(x) = L_1(x) + \frac{f''(\xi)}{2!}(x-a)(x-b), a \leq \xi \leq b$$

$$E_1(x) = \int_a^b \frac{f''(\xi)}{2!}(x-a)(x-b)dx$$

因为$(x-a)(x-b)$在$[a,b]$上不变号,由第二积分中值定理得到梯形求积公式的截断误差:

$$E_1(x) = \frac{f''(\eta)}{2!}\int_a^b (x-a)(x-b)dx = -\frac{f''(\eta)}{12}(b-a)^3, a \leq \eta \leq b \tag{2.7}$$

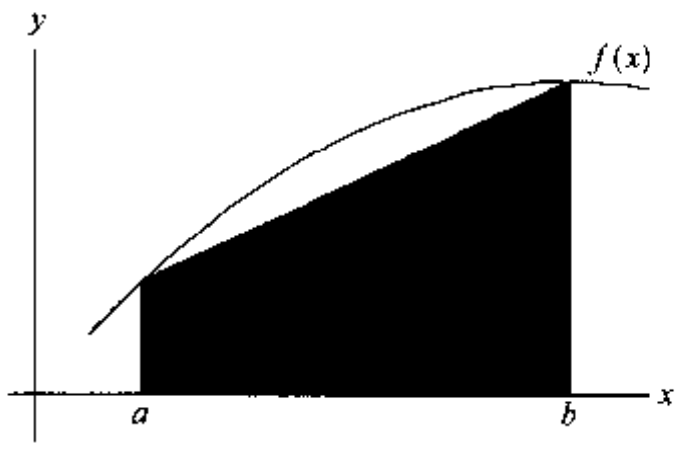


图 2.2 梯形面积

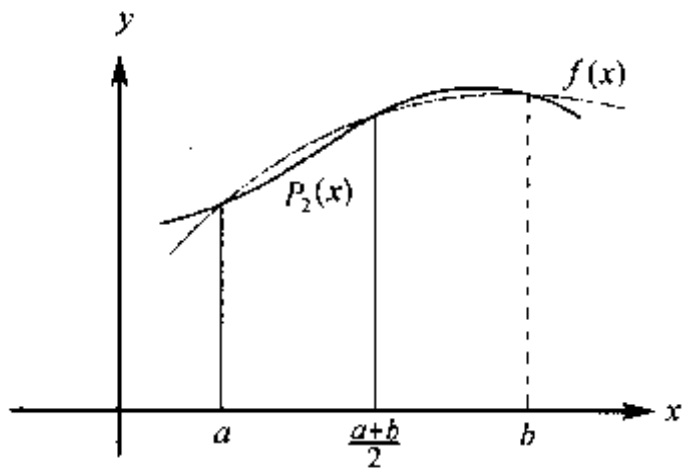


图 2.3 抛物线积分面积

辛普森(Simpson)积分

对区间$[a,b]$作二等分,记 $x_0=a, x_1=(a+b)/2, x_2=b$。以$(a,f(a))$,$((a+b)/2, f((a+b)/2))$和$(b,f(b))$为插值节点构造二次插值函数 $L_2(x)$,那么,有

$$\int_a^b L_2(x)dx = \int_a^b (l_0(x)f(x_0) + l_1(x)f(x_1) + l_2(x)f(x_2))dx$$

$$a_0 = \int_a^b l_0(x)dx = \int_a^b \frac{(x-(a+b)/2)(x-b)}{(a-(a+b)/2)(a-b)}dx = \frac{1}{6}(b-a) = (b-a)c_0^{(2)}$$

$$a_1 = \int_a^b l_1(x)dx = \frac{4}{6}(b-a) = (b-a)c_1^{(2)}$$

$$a_2 = \int_a^b l_2(x)dx = \frac{1}{6}(b-a) = (b-a)c_2^{(2)}$$

计算得到积分组合系数：$c_0^{(2)}=\frac{1}{6}, c_1^{(2)}=\frac{4}{6}, c_2^{(2)}=\frac{1}{6}$。

$$\int_a^b f(x)dx \approx I_2(f) = S(f) = \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] \tag{2.8}$$

$S(f)$称为辛普森或抛物线积分公式。它的几何意义是用过三点的抛物线面积近似代替积分的曲边面积(图 2.3)。

分别将 $f(x)=1, x, x^2, x^3, x^4$ 代入到 $I(f)$和 $S(f)$中，可以得到

$$S(f) = \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] = I(f)$$

表明辛普森公式对于次数不超过三次的多项式准确成立，$S(f)$具有三阶代数精度。因此可设一个三次多项式满足条件：

$$P_3(a)=f(a), P_3(b)=f(b)$$

$$P_3\left(\frac{a+b}{2}\right)=f\left(\frac{a+b}{2}\right), P_3'\left(\frac{a+b}{2}\right)=f'\left(\frac{a+b}{2}\right)$$

计算得到误差为：

$$f(x)-P_3(f)=\frac{f^{(4)}(\xi)}{4!}(x-a)\left(x-\frac{a+b}{2}\right)^2(x-b), \qquad a\leq\xi\leq b$$

于是有

$$E_2(f)=I(f)-S(f)=[I(f)-I(P_3)]+[I(P_3)-S(f)]$$

但 $I(P_3)=S(P_3)=\frac{b-a}{6}\left[P_3(a)+4P_3\left(\frac{a+b}{2}\right)+P_3(b)\right]=S(f)$

故辛普森求积公式的截断误差：

$$E_2(f)=I(f)-I(P_3)=\int_a^b \frac{f^{(4)}(\xi)(x-a)\left(x-\frac{a+b}{2}\right)^2(x-b)}{4!}dx$$

$$=\frac{f^{(4)}(\eta)}{4!}\int_a^b(x-a)\left(x-\frac{a+b}{2}\right)^2(x-b)$$

$$=-\frac{(b-a)^5}{2880}f^{(4)}(\eta), \qquad a\leq\eta\leq b \tag{2.9}$$

牛顿-柯特斯积分系数

n 等分区间$[a,b]$，取等分点为积分节点，$x_i=a+ih, i=0,1,\cdots,n$，其中 $h=\frac{b-a}{n}$。

以$(x_i, f(x_i)), i=1,2,\cdots,n$ 为插值节点构造插值函数 $L_n(x)$。

$$\int_a^b L_n(x)dx=\int_a^b\left(\sum_{i=0}^n l_i(x)f(x_i)\right)dx=\sum_{i=0}^n\left(\int_a^b l_i(x)dx\right)f(x_i)=\sum_{i=0}^n \alpha_i f(x_i)$$

其中

$$\alpha_i=\int_a^b l_i(x)dx=\int_a^b\frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}dx$$

令 $x=a+th, x_i=a+ih$，代入 α_i 得

$$\alpha_i=\int_0^n\frac{t(t-1)\cdots(t-i+1)(t-i-1)\cdots(t-n)}{i!(n-i)!(-1)^{n-i}}hdt$$

$$= \frac{(b-a)}{n}\frac{(-1)^{n-i}}{i!(n-i)!}\int_0^n t(t-1)\cdots(t-i+1)(t-i-1)\cdots(t-n)\,dt$$

$$= (b-a)c_i^{(n)} \tag{2.10}$$

这里称 $c_i^{(n)} = \frac{(-1)^{n-i}}{i!\ (n-i)!\ n}\int_0^n t(t-1)\cdots(t-i+1)(t-i-1)\cdots(t-n)\,dt$ 为牛顿-柯特斯系数。

可见在取等距节点时，积分系数 $c_i^{(n)}$ 与积分节点和积分区间无直接关系，只与插值的节点总数有关，而在例 2.3 中的积分系数是待定系数，这就简化了数值积分公式，而不必对每一组插值节点 x_i 都要计算一组相应的积分系数 α_i。在公式(2.10)中取 $n=1$，可算出梯形积分系数；取 $n=2$，可算出辛普森积分系数。在表 2.1 中列出 n 从 1 到 6 的牛顿-柯特斯系数。

表 2.1

n	$c_0^{(n)}$	$c_1^{(n)}$	$c_2^{(n)}$	$c_3^{(n)}$	$c_4^{(n)}$	$c_5^{(n)}$	$c_6^{(n)}$
1	$\frac{1}{2}$	$\frac{1}{2}$					
2	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$				
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$			
4	$\frac{7}{90}$	$\frac{16}{45}$	$\frac{2}{15}$	$\frac{16}{45}$	$\frac{7}{90}$		
5	$\frac{19}{288}$	$\frac{25}{96}$	$\frac{25}{144}$	$\frac{25}{144}$	$\frac{25}{96}$	$\frac{19}{288}$	
6	$\frac{41}{840}$	$\frac{9}{35}$	$\frac{9}{280}$	$\frac{34}{105}$	$\frac{9}{280}$	$\frac{9}{35}$	$\frac{41}{840}$

2.3 复化数值积分

由插值的龙格现象可知，高阶牛顿-柯特斯积分不能保证等距数值积分系列的收敛性，同时可证(略)高阶牛顿-柯特斯积分的计算是不稳定的。因此，实际计算中常用低阶复化梯形等积分公式。

2.3.1 复化梯形积分

把积分区间分割成若干小区间，在每个小区间 $[x_i, x_{i+1}]$ 上用梯形积分公式，再将这些小区间上的数值积分累加起来，称为复化梯形公式。复化梯形公式用若干个小梯形面积逼近积分 $\int_a^b f(x)\,dx$ 比用一个大梯形公式效果显然更好，如图 2.4 所示。这种作法使我们想起定积分定义，即它为被积函数无限分割的代数和。这也正是计算定积分最朴素的算法。

复化梯形积分计算公式

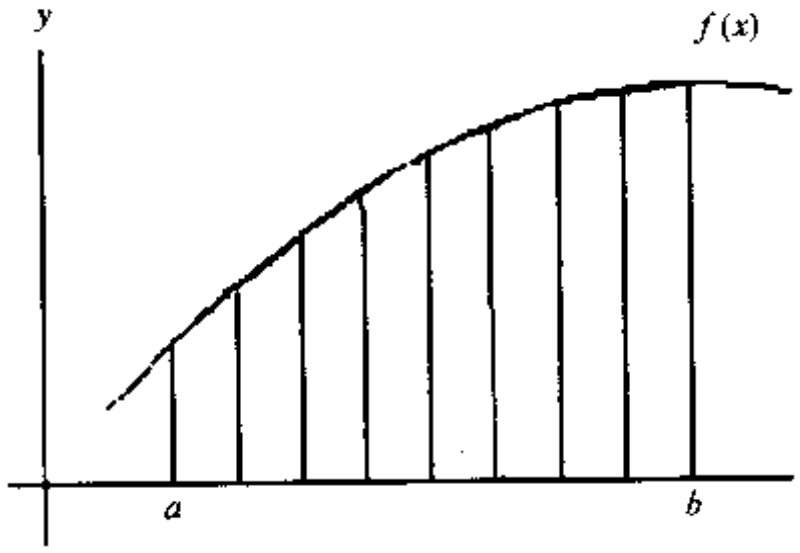


图 2.4　复化梯形公式积分视图

对$[a,b]$作等距分割,有 $h=\dfrac{b-a}{n}$, $x_i=a+ih$, $i=0,1,\cdots,n$, 于是

$$I(f)=\int_a^b f(x)\,dx=\sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} f(x)\,dx$$

在$[x_i,x_{i+1}]$上, $\int_{x_i}^{x_{i+1}} f(x)=\dfrac{h}{2}[f(x_i)+f(x_{i+1})]-f''(\xi_i)\dfrac{h^3}{12}$, 则有

$$I(f)=\sum_{i=0}^{n-1}\left\{\frac{h}{2}[f(x_i)+f(x_{i+1})]-f''(\xi_i)\frac{h^3}{12}\right\}$$

$$=h\left[\frac{1}{2}f(a)+\sum_{i=1}^{n-1}f(x_i)+\frac{1}{2}f(b)\right]-\sum_{i=0}^{n-1}f''(\xi_i)\frac{h^3}{12}$$

记 n 等分的复化梯形公式为 $T_n(f)$或 $T(h)$, 有

$$T(h)=T_n(f)=h\left[\frac{1}{2}f(a)+\sum_{i=1}^{n-1}f(a+ih)+\frac{1}{2}f(b)\right] \tag{2.11}$$

复化梯形公式截断误差

由 $E_n(f)=I(f)-T_n(f)=-\dfrac{h^3}{12}\sum_{i=0}^{n-1}f''(\xi_i)$, 根据均值定理, 当 $f\in C^2[a,b]$时, 存在 $\xi\in[a,b]$, 有 $\sum_{i=0}^{n-1}f''(\xi_i)=nf''(\xi)$, 于是

$$E_n(f)=-\frac{nh^3}{12}f''(\xi)=-\frac{h^2}{12}(b-a)f''(\xi)=-\frac{(b-a)^3}{12n^2}f''(\xi),\quad a\leq\xi\leq b \tag{2.12}$$

由此看到复化梯形公式的截断误差按照 h^2 或者$\dfrac{1}{n^2}$的速度下降, 事实上, 可以证明, 只要 $f(x)$在(a,b)上有界并黎曼可积, 当分点无限增多时, 复化梯形公式收敛到积分 $I(f)=\int_a^b f(x)\,dx$。

记 $M_2=\max\limits_{a\leq x\leq b}|f''(x)|$, 则有

$$|E_n(f)|\leq\frac{(b-a)^3}{12n^2}M_2=O\left(\frac{1}{n^2}\right)$$

对于任给的误差控制小量 $\varepsilon>0$,有

$$\frac{(b-a)^3}{12n^2}M_2<\varepsilon \quad 或 \quad n\geqslant\left[\sqrt{\frac{(b-a)^3M_2}{12\varepsilon}}\right]+1$$

就有 $|E_n(f)|<\varepsilon$,式中[·]表示取其最大整数。

2.3.2 复化辛普森积分

把积分区间分成偶数等分 $2m$,记 $n=2m$,其中 $n+1$ 是节点总数,m 是积分子区间的总数。

记 $h=\dfrac{b-a}{n}$,$x_i=a+ih$,$i=0,1,\cdots,n$,在每个子区间$[x_{2i},x_{2i+2}]$上用辛普森数值积分公式计算,则得到复化辛普森公式,记为 $S_n(f)$。

复化辛普森积分计算公式

$$I(f)=\int_a^b f(x)\,dx=\sum_{i=0}^{m-1}\int_{x_{2i}}^{x_{2i+2}}f(x)\,dx$$

而 $\displaystyle\int_{x_{2i}}^{x_{2i+2}}f(x)\,dx=\frac{2h}{6}[f(x_{2i})+4f(x_{2i+1})+f(x_{2i+2})]-\frac{(2h)^5}{2880}f^{(4)}(\zeta_i)$,称

$$\begin{aligned}S_n(f)&=\sum_{i=0}^{m-1}\frac{2h}{6}[f(x_{2i})+4f(x_{2i+1})+f(x_{2i+2})]\\&=\frac{h}{3}\Big[f(a)+4\sum_{i=0}^{m-1}f(x_{2i+1})+2\sum_{i=1}^{m-1}f(x_{2i})+f(b)\Big]\end{aligned}\tag{2.13}$$

为复化辛普森积分公式,它是 $f(x)$在$[x_{2i},x_{2i+2}]$上采用辛普森积分公式叠加而得。下面用图 2.5 显示复化辛普森积分计算公式中节点与系数的关系,取 $n=8$,在每个积分区间上提出因子$\dfrac{2h}{6}$后,三个节点的系数分别是 1,4,1;将 4 个积分区间的系数按节点的位置累加,可以清楚地看到,首尾节点的系数是 1,奇数点的系数是 4,偶数点的系数是 2。

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	4	1						
		1	4	1				
				1	4	1		
						1	4	1
1	4	2	4	2	4	2	4	1

图 2.5 复化辛普森积分系数

复化辛普森公式的截断误差

设 $f\in C^4[a,b]$,在$[x_{2i},x_{2i+1}]$上的误差为

$$-\frac{(2h)^5}{2880}f^{(4)}(\zeta_i),x_{2i}\leqslant\zeta_i\leqslant x_{2i+2}$$

因此,有

$$\begin{aligned}I(f)-S_n(f)&=-\frac{(2h)^5}{2880}\sum_{i=0}^{m-1}f^{(4)}(\zeta_i)=-\frac{(2h)^5m}{2880}f^{(4)}(\zeta)\\&=\frac{-(b-a)^5}{2880m^4}f^{(4)}(\zeta)=\frac{-(b-a)^5}{180n^4}f^{(4)}(\zeta),\quad a\leqslant\zeta\leqslant b\end{aligned}\tag{2.14}$$

与复化梯形公式类似，误差的截断误差按照 h^4 或者 $\frac{1}{n^4}$ 的速度下降。可以证明，只要 $f(x)$ 在 (a,b) 上有界并黎曼可积，当分点无限增多时，复化辛普森公式收敛到积分 $I(f)=\int_a^b f(x)dx$。

记 $M_4=\max\limits_{a\le x\le b}|f^{(4)}(x)|$，则有

$$|E_n(f)|\le\frac{(b-a)^5}{2880m^4}M_4=O\left(\frac{1}{m^4}\right)$$

对任给的误差控制小量 $\varepsilon>0$，只要

$$\frac{(b-a)^5}{2880m^4}M_4<\varepsilon\quad 或\ m\ge\left[\sqrt[4]{\frac{(b-a)^5M_4}{2880\varepsilon}}\right]+1$$

就有 $|E_n(f)|<\varepsilon$。

例 2.5： $I(f)=\int_0^1 e^x dx$，计算中要求有 5 位有效数字。用复化梯形和复化辛普森求积公式的分点应取多少？

解：

$$f(x)=e^x,f''(x)=f^{(4)}(x)=e^x$$
$$|f''(x)|=|f^{(4)}(x)|\le e,\quad 0\le x\le 1$$

由复化梯形误差公式得到：

$$|I(f)-T_n(f)|\le\frac{(b-a)^3}{12n^2}M_2=\frac{1}{12n^2}e\le\frac{1}{2}10^{-4}$$

计算出 $n=67.3$，复化梯形公式至少要在 $[0,1]$ 等分 $n=68$。

由复化辛普森误差公式，有

$$|I(f)-S_n(f)|\le\frac{1}{2880m^4}e\le\frac{1}{2}10^{-4}$$

在复化辛普森公式中取 $m=\left[\frac{3}{2}\right]+1=2$ 或 $n=4$。

2.3.3 复化积分的自适应算法

复化积分的误差公式表明，截断误差随分点 n 的增大而减小，对于给定的误差量 ε，用估计函数导数的界的方法可计算出 n。用误差公式计算满足精度的分点数，像是在做一道计算导数 $|f^{(n)}(\zeta)|$ 上界的微积分习题（如例 2.5 所示）。但是在实际运算中，一般难以估计出函数的各阶导数界，也就无法确定分点数 n。在计算中常用误差的事后估计方法，即用 $|T_{2n}(f)-T_n(f)|$ 估计误差。

$T_{2n}(f)$ 的计算公式

对定积分 $\int_a^b f(x)dx$，取分点 $n=1$，计算得

$$T_1(f)=\frac{b-a}{2}(f(a)+f(b))$$

取分点 $n=2$,计算得

$$T_2(f)=\frac{b-a}{2}\left(\frac{f(a)}{2}+\frac{f(b)}{2}+f(x_2)\right)=\frac{1}{2}T_1(f)+\frac{b-a}{2}f(x_2)$$

这里,$x_2=\frac{a+b}{2}$。可以看到,$T_2(f)$的值是 $T_1(f)$与新增分点 $f(x_2)$的组合。

取分点 $n=4$,计算得

$$\begin{aligned}T_4(f)&=\frac{b-a}{4}\left(\frac{f(a)}{2}+\frac{f(b)}{2}+f(x_1)+f(x_2)+f(x_3)\right)\\&=\frac{T_2}{2}+\frac{b-a}{4}(f(x_1)+f(x_3))\end{aligned}$$

这里,$x_1=\frac{1}{2}(a+x_2)$,$x_3=\frac{1}{2}(x_2+b)$。

同理,计算 $T_4(f)$时只要在 $T_2(f)$的基础上计算新增分点 $f(x_1)$,$f(x_3)$的值再做组合,如图 2.6 所示。

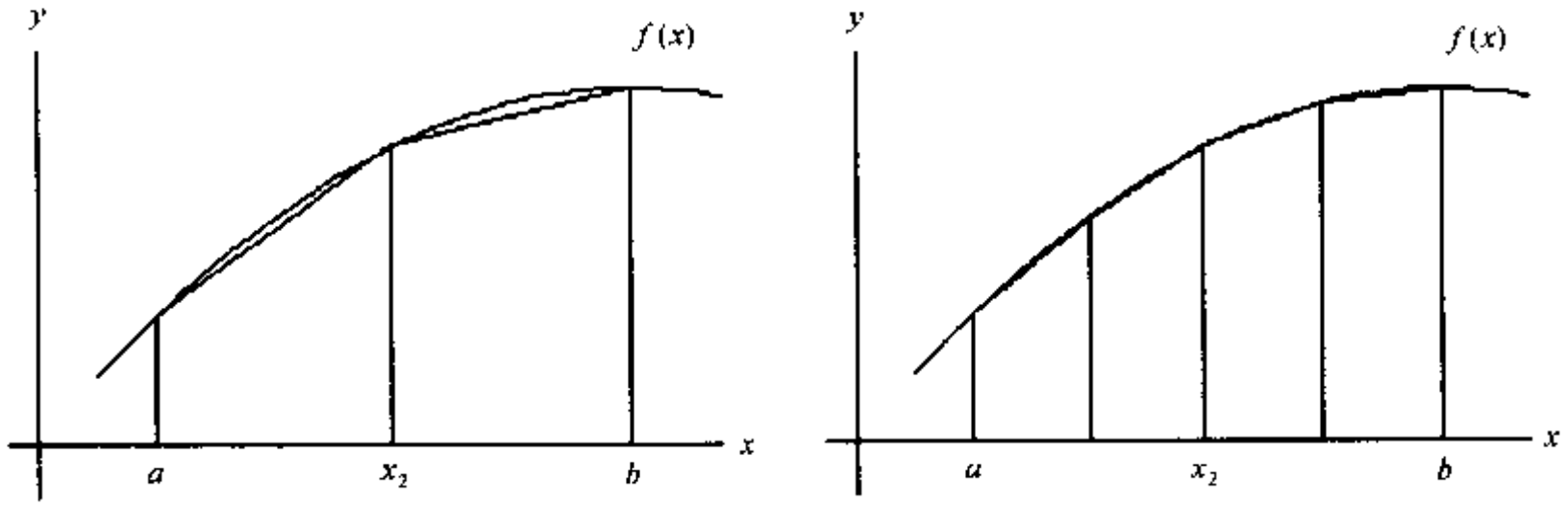


图 2.6 $T_2(f)$与 $T_4(f)$

一般地,每次总对前一次的小区间分半,分点加密一倍,并可充分利用老分点上的函数值,每次只需计算新增分点的和。

对$[a,b]$上 n 等分,$h_n=\frac{b-a}{n}$,则有

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

记$[x_i,x_{i+1}]$上的中点为 $x_{i+\frac{1}{2}}$,则

$$\begin{aligned}T_{2n}(f)&=\frac{h_n}{2}\left(\frac{1}{2}f(a)+\sum_{i=1}^{n-1}f(x_i)+\sum_{i=0}^{n-1}f(x_{i+1/2})+\frac{1}{2}f(b)\right)\\&=\frac{h_n}{2}\left(\frac{1}{2}f(a)+\sum_{i=1}^{n-1}f(x_i)+\frac{1}{2}f(b)\right)+\frac{h_n}{2}\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}})\end{aligned}\tag{2.15}$$

$$T_{2n}(f)=\frac{1}{2}(T_n(f)+H_n(f))$$

其中 $H_n(f)=h_n\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}})$。

或 $$T_{2n} = \frac{T_n}{2} + h_{2n}\sum_{i=1}^{n} f(a + (2i-1)h_{2n})$$

其中 $h_{2n} = \frac{b-a}{2n}$。

类似地，可得积分节点为 $n, 2n$ 的辛普森求积公式的关系式：

$$S_{2n}(f) = \frac{1}{2}S_n(f) + \frac{1}{6}(4H_{2n}(f) - H_n(f)) \tag{2.16}$$

$|T_{2n}(f) - T_n(f)|$ 与 $|I(f) - T_{2n}(f)|$

由误差公式：

$$I(f) - T_n(f) = -\frac{(b-a)}{12}h^2 f''(\zeta)$$

$$I(f) - T_{2n}(f) = -\frac{(b-a)}{12}\left(\frac{h}{2}\right)^2 f''(\eta)$$

由于 $f''(\xi) = \frac{1}{n}\sum_{i=1}^{n-1} f''(\xi_i)$，$f''(\eta) = \frac{1}{2n}\sum_{i=0}^{2n-1} f''(\eta_i)$，分别为 n 及 $2n$ 个点上的均值，可视 $f''(\xi) \approx f''(\eta)$，于是有

$$I(f) - T_n(f) \approx 4(I(f) - T_{2n}(f))$$

上式表明 $T_n(f)$ 的误差大约是 $T_{2n}(f)$ 误差的 4 倍。

或 $$I(f) - T_{2n}(f) \approx \frac{1}{3}(T_{2n}(f) - T_n(f)) \tag{2.17}$$

由此得到启发，对任给的误差控制量 $\varepsilon > 0$，要 $|I(f) - T_{2n}(f)| < \varepsilon$，只需 $|T_{2n}(f) - T_n(f)| < 3\varepsilon$ 即可，而用 $|T_{2n}(f) - T_n(f)|$ 作为控制手段简单直接，序列 $T_n(f)$，$T_{2n}(f)$，…在计算机上也不难实现。

复化积分的自适应算法描述

从数值积分的误差公式可以看到，截断误差随分点 n 的增长而减少，控制计算的精度也就是确定分点数 n。用数值积分的误差公式确定分点数 n 是一种理论模式，在实际问题中难以实现。在计算中用 $|T_{2n}(f) - T_n(f)| < 3\varepsilon$ 作为控制，通过增加分点自动满足精度的方法称为数值积分公式的自适应算法或自动积分法。即在计算中构造序列 T_n，$T_{2n}, T_{4n}, \ldots$，每次计算一个积分值 T_{2m} 都要计算与前一次的积分值 T_m 的误差，直到 $|T_{2m} - T_m| < \varepsilon$ 或 $\frac{|T_{2m} - T_m|}{|T_{2m}|} < \varepsilon$ 时停止计算，并取 $I(f) \approx T_m(f)$。

下面描述复化数值积分公式的自适应算法，详细程序和算例请看本章 2.6 节。

1. 输入：误差控制精度 e = eps；初始分点值 $n = m$。

2. 计算 n 分点的复化梯形积分 T_n，$T2 = T_n$

```
T1 = T2 + 100          !迭代计算中 T1 和 T2 分别表示 Tn 和 T2n
```

3. while |T1 − T2| > 3ε

```
T1 = T2
H = Hn          ! 计算新增节点的值 Hn(f) = hn Σ_{i=0}^{n-1} f(x_{i+1/2})
T2 = (T1 + H)/2
```

```
        h = h/2, n = 2n          ! 将区间一分为二
        end while
```

4. 输出积分值 T2。

在自适应算法中初始分点值不宜过小,以防假收敛。

2.3.4 龙贝格(Romberg)积分

前面得到的关系式(2.17)将$(T_n(f) - T_{2n}(f))$作为 $T_{2n}(f)$的修正值补充到 $I(f)$,即

$$I(f) \approx T_{2n}(f) + \frac{1}{3}(T_{2n}(f) - T_n(f)) = \frac{4}{3}T_{2n} - \frac{1}{3}T_n = S_n \tag{2.18}$$

其结果是将梯形求积公式组合成辛普森求积公式,截断误差由$O(h^2)$提高到$O(h^4)$。这种手段称为外推算法。外推算法在不增加计算量的前题下提高了误差的精度,是计算方法中的一种常用手法。

不妨对 $S_{2n}(f)$,$S_n(f)$再做一次组合。由

$$I(f) - S_{2n}(f) = -\frac{f^{(4)}(\eta)}{2880}\left(\frac{h}{2}\right)^4(b-a) \approx d\left(\frac{h}{2}\right)^4$$

$$I(f) - S_n(f) = -\frac{f^{(4)}(\zeta)}{2880}h^4(b-a) \approx dh^4$$

得到

$$\begin{aligned} I(f) - S_{2n}(f) &\approx \frac{1}{15}(S_{2n}(f) - S_n(f)) \\ I(f) &\approx \frac{16}{15}S_{2n}(f) - \frac{1}{15}S_n(f) = C_n(f) \end{aligned} \tag{2.19}$$

复化辛普森公式组成复化柯特斯公式,其截断误差是 $O(h^6)$。同理对柯特斯公式进行组合:

$$I(f) - C_n(f) = eh^6$$

$$I(f) - C_{2n}(f) = e\left(\frac{h}{2}\right)^6$$

得到具有 7 次代数精度和截断误差是 $O(h^8)$的龙贝格公式:

$$R_n(f) = \frac{64}{63}C_{2n}(f) - \frac{1}{63}C_n(f) \tag{2.20}$$

还可以继续对 $R_n(f)$做下去。

为了在计算机上实现龙贝格算法,将 T_n,S_n,C_n,R_n 都用一种符号 T 表示,按照表2.2

表 2.2

$T^{(0)}$	$T^{(1)}$	$T^{(2)}$	$T^{(3)}$	$T^{(4)}$	$T^{(5)}$
1. $T^{(0)}$(h)	3. $T^{(1)}$(h)	6. $T^{(2)}$(h)	10. $T^{(3)}$(h)	15. $T^{(4)}$(h)	…
2. $T^{(0)}$(h/2)	5. $T^{(1)}$(h/2)	9. $T^{(2)}$(h/2)	14. $T^{(3)}$(h/2)		
4. $T^{(0)}$(h/4)	8. $T^{(1)}$(h/4)	13. $T^{(2)}$(h/4)			
7. $T^{(0)}$(h/8)	12. $T^{(1)}$(h/8)				
11. $T^{(0)}$(h/16)					
⋮					

左上角标号的次序进行运算，每做一步后都要判别相邻的两步误差是否满足给定的控制精度，直到 $\left| T^{(m)}\left(\frac{h}{2}\right) - T^{(m+1)}(h) \right| < \varepsilon$ 时停止。

这里，$T^{(k+1)}(h) = T^{(k)}\left(\frac{h}{2}\right) + \frac{1}{4^{k+1}-1}\left[T^{(k)}\left(\frac{h}{2}\right) - T^{(k)}(h)\right]$。

2.4 重积分计算

在微积分中计算二重积分是用化为累次积分的方法进行的。计算二重数值积分也是计算累次数值积分的过程。为了简化问题，我们仅讨论矩形域上的二重积分。有很多非矩形域上的二重积分可作变换将其转换到矩形域上。

$$\int_a^b\int_c^d f(x,y)\,dxdy \tag{2.21}$$

其中：a,b,c,d 是常数，$f(x,y)$ 在 D 上连续。像在微积分中一样，将二重积分化为累次积分：

$$\int_a^b\int_c^d f(x,y)\,dxdy = \int_a^b\left(\int_c^d f(x,y)\,dy\right)dx \tag{2.22}$$

或

$$\int_a^b\int_c^d f(x,y)\,dxdy = \int_c^d\left(\int_a^b f(x,y)\,dx\right)dy$$

二重积分的复化梯形公式

对区间$[a,b]$和$[c,d]$分别选取正整数 m 和 n，在 x 轴和 y 轴上分别有步长

$$h = \frac{b-a}{m}, \quad k = \frac{d-c}{n}$$

用复化梯形公式计算 $\int_c^d f(x,y)\,dy$，计算中将 x 当作常数，有

$$\int_c^d f(x,y)\,dy \approx k\left[\frac{1}{2}f(x,y_0) + \frac{1}{2}f(x,y_n) + \sum_{j=1}^{n-1} f(x,y_j)\right] \tag{2.23}$$

再将 y 当作常数，在 x 方向上计算式(2.23)中每一项的积分，有

$$\frac{1}{2}\int_a^b f(x,y_0)\,dx \approx \frac{h}{2}\left[\frac{1}{2}f(x_0,y_0) + \frac{1}{2}f(x_m,y_0) + \sum_{i=1}^{m-1} f(x_i,y_0)\right]$$

$$\frac{1}{2}\int_a^b f(x,y_n)\,dx \approx \frac{h}{2}\left[\frac{1}{2}f(x_0,y_n) + \frac{1}{2}f(x_m,y_n) + \sum_{i=1}^{m-1} f(x_i,y_n)\right]$$

$$\int_a^b \sum_{j=1}^{n-1} f(x,y_j)\,dx = \sum_{j=1}^{n-1}\int_a^b f(x,y_j)\,dx$$

$$\approx h\sum_{j=1}^{n-1}\left[\frac{1}{2}f(x_0,y_j) + \frac{1}{2}f(x_m,y_j) + \sum_{i=1}^{m-1} f(x_i,y_j)\right]$$

$$= h\sum_{j=1}^{n-1}\left[\frac{1}{2}f(x_0,y_j) + \frac{1}{2}f(x_m,y_j)\right] + h\sum_{j=1}^{n-1}\sum_{i=1}^{m-1} f(x_i,y_j)$$

则

$$\int_a^b\int_c^d f(x,y)\,dxdy \approx hk\left\{\frac{1}{4}[f(x_0,y_0) + f(x_m,y_0) + f(x_0,y_n) + f(x_m,y_n)\right]$$

$$+\frac{1}{2}\Big[\sum_{i=1}^{m-1}f(x_i,y_0)+\sum_{i=1}^{m-1}f(x_i,y_n)+\sum_{j=1}^{n-1}f(x_0,y_j)+\sum_{j=1}^{n-1}f(x_m,y_j)\Big]$$

$$+\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}f(x_i,y_j)\Big\}=hk\sum_{i=0}^{m}\sum_{j=0}^{n}c_{i,j}f(x_i,y_j)$$

积分区域的4个角点的系数是1/4,4个边界的系数是1/2,内部节点的系数是1。

例2.6:用复化梯形公式计算二重积分$\int_0^1\int_1^2\sin(x^2+y)\,dxdy$,取$h=k=0.25$。

解:$f(x,y)$如下表所示:

x \ y	1.00	1.25	1.50	1.75	2.00
0.00	0.841471	0.948985	0.997495	0.983986	0.909297
0.25	0.873575	0.966827	0.999966	0.970932	0.88153
0.50	0.948985	0.997495	0.983986	0.909297	0.778073
0.75	0.999966	0.970932	0.88153	0.737319	0.547265
1.00	0.909297	0.778073	0.598472	0.381661	0.14112

$C_{i,j}$的数值列表如下:

J \ I	,0	1	2	3	4
0	1/4	1/2	1/2	1/2	1/4
1	1/2	1	1	1	1/2
2	1/2	1	1	1	1/2
3	1/2	1	1	1	1/2
4	1/4	1/2	1/2	1/2	1/4

$$\therefore\quad \int_0^1\int_1^2\sin(x^2+y)\,dxdy=0.25\times0.25\Big[\frac{1}{4}(0.841471+0.909297+0.909297+0.14112)$$

$$+\frac{1}{2}(0.948985+0.997495+0.983986)$$

$$+\frac{1}{2}(0.778073+0.598472+0.381661)+\frac{1}{2}(0.873575+0.948985+0.999966)$$

$$+\frac{1}{2}(0.88153+0.778073+0.547265)+(0.966827+0.999966+0.970932$$

$$+0.997495+0.983986+0.909297+0.970932+0.88153+0.737319)\Big]$$

$$=1.84375$$

$\int_0^1\int_1^2\sin(x^2+y)\,dxdy$的准确值是1.833333。

2.5* 高斯型积分公式介绍

在牛顿-柯特斯积分公式中要求节点是等距的,其优点是计算积分系数的公式规则相

同,缺点是制约了求积公式的代数精度。可以证明:当节点个数 n 为偶数时,求积公式具有 $n-1$ 阶的代数精度;当节点个数 n 为奇数时,求积公式具有 n 阶的代数精度。

当取积分节点为正交多项式的零点时,则节点个数是 n 的求积公式具有 $2n-1$ 阶的代数精度。并称积分节点为正交多项式的零点的近似积分公式为高斯型积分公式。

为了一般性,考虑积分

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

其中 $W(x)$($W(x) \geq 0$)称为权函数。当 $W(x)=1$ 时,即是普通的积分。

对于不同的权函数 $W(x)$选定的节点也不相同。

对给定的$[a,b]$及权函数 $W(x)$,由 Schemite 正交化过程作出正交多项式 $P_0(x)$, $P_1(x),\cdots,P_n(x)$;解出正交多项式 $P_n(x)$的 n 个零点 $x_1,x_2,\cdots,x_n$,这 n 个零点就是积分节点;以这些节点构造插值多项式,计算积分系数

$$\alpha_i = \int_a^b l_i(x) W(x)\,dx, \quad i = 1,2,\cdots,n$$

其中 $l_i(x)$是插值基函数。

记高斯型求积公式为

$$G_n(f) = \sum_{i=1}^{n} \alpha_i f(x_i)$$

高斯型积分公式的优点是它的代数精度高,特别是对无穷区间或瑕积分更有效,但计算正交多项式的零点即积分节点有一定的难度,好在数值学家们已算出一些特定的函数的积分节点和积分系数,在计算中我们可以查表直接得到这些数值。

本章并不构造各种高斯型积分公式,有关的详细内容请参考有关的教材。

下面给出$[a,b]=[-1,1]$上,取权函数 $W(x)=1$ 的高斯型积分。

$[-1,1]$上权函数 $W(x)=1$ 的正交多项式为 Legendre 多项式:

$$L_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[(x^2-1)^n], x \in [-1,1]$$

Guass-Legendre $n=2,4,6$ 相应的积分节点和积分系数表如下:

n	x_k	a_k
2	$x_1 = -0.5773503, x_2 = 0.5773503$	1.0000000, 1.0000000
4	$x_1 = -0.8611363, x_2 = -0.3399810$ $x_3 = 0.3399810, x_4 = 0.8611363$	0.3478548, 0.6521452 0.6521452, 0.3478548
5	$-x_1 = x_5 = 0.9061798$ $-x_2 = x_4 = 0.538469$ $x_3 = 0.0$	0.2369269, 0.2369269 0.4786287, 0.4786287 0.5688889

要计算一般区间$[a,b]$上的积分 $\int_a^b f(x)\,dx$,只需作变量代换 $x = \frac{a+b}{2} + \frac{b-a}{2}t$ 则有 $\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} g(t)\,dt$,其中 $g(t) = f\left(\frac{a+b}{2} + \frac{b-a}{2}t\right)$,这样, $I(f) = \int_a^b f(x)\,dx$ 仍可用高斯积分求积,即

$$G_n(f) = \frac{b-a}{2}G_n(g) = \frac{b-a}{2}\sum_{i=1}^{n}\alpha_i g(x_i)$$

例 2.7:应用两点 Guass-Legendre 积分公式计算 $I = \int_{-1}^{1} u^2 \cos u du$。

解:$I = (-0.5773503)^2\cos(-0.5773503) + (0.5773503)^2\cos(0.5773503)$
$= 0.558608$

例 2.8:应用两点 Guass-Legendre 积分公式计算 $I = \int_0^1 \frac{dx}{1+x^2}$。

解:令 $x = \frac{1}{2} + \frac{1}{2}t$

得到积分 $I = 2\int_{-1}^{1} \frac{dt}{4+(t+1)^2} = 2(f(x_1) + f(x_2)) = 0.786885$

2.6 程 序 示 例

程序 3 复化梯形公式 $T_n = h\left[\frac{1}{2}f(a) + \sum_{i=1}^{n-1} f(a+ih) + \frac{1}{2}f(b)\right]$ 的自适应算法。

算法描述

初始值 $n = m, h = (b-a)/n$;

$T2 = T_n = h\left[\frac{1}{2}f(a) + \sum_{i=1}^{n-1} f(a+ih) + \frac{1}{2}f(b)\right]$;

T1 = T2 + 100

while|T1 − T2| > 3ε

T1 = T2;

$H = H_n = h_n \sum_{i=0}^{n-1} f(x_{i+1/2})$

T2 = (T1 + H)/2, h = h/2, n = 2n

end while

输出 T2

程序源码

```
////////////////////////////////////////////////////////
//      Purpose: 梯形公式的自适应算法          //
////////////////////////////////////////////////////////
 # include <stdio.h>
 # include <math.h>
 # define f(x) (sin(x))                    //f(x)由 define 给出
 # define m   100
 # define a   1.0
 # define b   2.0
 # define epsilon 0.00001

void main()
```

```
{   int n = m;
    int i;
    double T_n, H_n, T1, T2;
    double h = (b - a)/n;
    T_n = (f(a) + f(b))/2;
    for (i = 1; i < n; i + + )
        T_n + = f(a + h * i);
    T_n * = h;
    T2 = T_n;
    T1 = T2 + 100;
    do
    {   T1 = T2;
        for (i = 0, H_n = 0; i < n; i + + )
            H_n + = f(a + h * i + h/2);
        H_n * = h;
        T2 = (T1 + H_n)/2;
        h = h/2; n = n * 2;
    } while (fabs(T1 - T2) > 3 * epsilon);
    printf("T = %lf \n", T2);
}
// - - - - - - - - - End of File - - - - - - - - - - - - -
```

计算实例

对于 $f(x) = \sin(x)$,在区间$[a, b]$上验证梯形公式的自适应公式。

程序输入输出

(对于不同 $f(x)$, 修改程序 # define f(x)项,本例 $f(x) = \sin(x)$,初始 $m = 100$,区间$[a, b] = [0, 1]$)

$T = 0.956447$

程序 4 龙贝格积分算法

计算公式

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

算法描述

对于给定 $f(x)$和初始 h,用复化梯形公式

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

求得 $T^{(0)}(h)$, $T^{(0)}(h/2)$,外推法得出 $T^{(1)}(h)$。对于已经计算出来的 $T^{(0)}(h/2^{m-1})$、$T^{(1)}(h/2^{m-2})$、…、$T^{(m-1)}(h)$,计算 $T^{(0)}(h/2^m)$,并用外推法依次计算 $T^{(1)}(h/2^{m-1})$、$T^{(2)}(h/2^{m-2})$、…、$T^{(m-1)}(h/2)$、$T^{(m)}(h)$,直到$|T^{(m-1)}(h/2) - T^{(m)}(h)|$小于给定 ε。

输出 $T^{(m)}(h)$

程序源码

```
////////////////////////////////////////////////////
//      Purpose: RomBerg 算法                     //
////////////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
# define f(x)                    (sin(x))
# define N_H                     20
# define MAXREPT                 10
# define a                       1.0
# define b                       2.0
# define epsilon                 0.00001

double computeT(double aa, double bb, long int n)        //复化梯形公式
{
    int i; double sum, h = (bb - aa)/n;
    for (i = 1; i < n; i + + )
        sum + = f(aa + i * h);
    sum + = (f(aa) + f(bb))/2;
    return (h * sum);
}

void main()
{
    int i;
    long int n = N_H, m = 0;
    double T[MAXREPT + 1][2];
    T[0][1] = computeT(a, b, n);
    n* = 2;
    for (m = 1; m < MAXREPT; m + + )
    {
      for (i = 0; i < m; i + + )
      { T[i][0] = T[i][1]; }
      T[0][1] = computeT(a, b, n);              //计算 T{m-1}(h/2)
      n* = 2;
                                                //T^m(h)
      for (i = 1; i < = m; i + + )
        T[i][1] = T[i-1][1] + (T[i-1][1] - T[i-1][0])/(pow(2, 2 * m) - 1);
      if ((T[m-1][1] < T[m][1] + epsilon) && (T[m-1][1] > T[m][1] - epsilon))
      {       printf("The Integrate is %lf \n", T[m][1]);        //输出
              return;       }
    }
    printf("Return no solved... \n");
}
// - - - - - - - - - End of File - - - - - - - - - - - - -
```

计算实例

利用龙贝格积分法计算 $\int_0^1 \sin(x)\,dx$ 。

程序输入输出

(对于不同 $f(x)$, 修改程序 # define f(x)项,本例 $f(x)=\sin(x)$,区间$[a,b]=[0,1]$,初始 $h=(b-a)/n=(b-a)/20$。)

对于本程序的给定,输出结果。

$T=0.956447$

习题 2

2.1 设函数 $f(x)$出下表给出,分别向前、向后差商,并计算 $f'(0.02)$,$f'(0.06)$。

x	0.00	0.02	0.04	0.06
$f(x)$	11.00	9.00	7.00	10.00

2.2 设函数 $f(x)$的数值由下表给出,用差商公式计算 $f''(0)$,$f''(0.20)$,$f''(0.40)$。

x	0.00	0.10	0.20	0.30	0.40
$f(x)$	1.70	1.50	1.60	2.00	1.90

2.3 设函数 $f(x)$的数值由下表给出:

x	$f(x)$	x	$f(x)$
0.06	0.871475	0.11	0.892995
0.07	0.876022	0.12	0.896943
0.08	0.880446	0.13	0.90078
0.09	0.884748	0.14	0.904506
0.10	0.88893	0.15	0.90812

取不同的步长计算 $f'(0.10)$,观察误差变化规律,从而确定最佳步长。

2.4 写出下列两种矩形公式的余项:

(1) $\int_a^b f(x)\,dx \approx f(a)(b-a)$

(2) $\int_a^b f(x)\,dx \approx f\left(\frac{a+b}{2}\right)(b-a)$

2.5 构造积分 $I(f)=\int_{-h}^{2h} f(x)\,dx$ 的数值积分公式。

$$I(f) = a_{-1}f(-h) + a_0 f(0) + a_1 f(2h)$$

2.6 用梯形公式计算：

(1) $\int_0^1 \sqrt{x^4+1}\,dx$

(2) $\int_1^2 \sin(\cos x^2)\,dx$

2.7 用辛普森公式计算下列积分：

(1) $\int_0^{\frac{\pi}{6}} \sqrt{2-\sin^2 x}dx$

(2) $\int_1^2 e^{-x^2/2}\,dx$

2.8 设函数由下表给出，分别用复化梯形和复化辛普森公式计算 $\int_{0.6}^{1.8} f(x)\,dx$。

x	0.6	0.8	1.0	1.2	1.4	1.6	1.8
$f(x)$	5.7	4.6	3.5	3.7	4.9	5.2	5.5

2.9 $I(\ln x)=\int_1^2 \ln x dx$，取 $\varepsilon=10^{-4}$，$h=1$，试用龙贝格公式计算积分直到 $\left|T^{(k-1)}\left(\frac{h}{2}\right)-T^{(k)}(h)\right|<\varepsilon$ 时停止，并作出龙贝格积分数值表。

2.10 用复化梯形公式计算二重积分：

(1) $\int_{-1}^{1}\int_{-1}^{1} xydxdy$，取 $h=k=4$

(2) $\int_1^2\int_1^2 \frac{1}{x+y}dxdy$，取 $h=k=5$

2.11 用具有三阶代数精度的 Gauss-Legendre 积分公式计算下列数值积分值：

(1) $\int_{-\pi}^{\pi} \sqrt{1-\cos 2x+\sin x^2}\,dx$

(2) $\int_0^{\infty} e^{-x}x^2\,dx$

第 3 章　曲线拟合的最小二乘法

3.1　拟 合 曲 线

通过观察或测量得到一组离散数据序列(x_i, y_i), $i = 1,2,\cdots,m$,当所得数据比较准确时,可构造插值函数 $\varphi(x)$逼近客观存在的函数 $y = y(x)$,构造的原则是要求插值函数通过这些数据点,即 $\varphi(x_i) = y_i$, $i = 1,2,\cdots,m$。此时,序列 $Q = (\varphi(x_1), \varphi(x_2), \cdots, \varphi(x_m))^T$ 与 $Y = (y_1, y_2, \cdots, y_m)^T$ 是相等的。

如果数据序列(x_i, y_i), $i = 1,2,\cdots,m$,含有不可避免的误差(或称“噪音”),如图 3.1 所示;如果数据序列无法同时满足某特定函数,如图 3.2 所示,那么,只能要求所做逼近函数 $\varphi(x)$最优地靠近样点,即向量 $Q = (\varphi(x_1), \varphi(x_2), \cdots, \varphi(x_m))^T$ 与 $Y = (y_1, y_2, \cdots, y_m)^T$ 的误差或距离最小。按 Q 与 Y 之间误差最小原则作为“最优”标准构造的逼近函数,称为拟合函数。

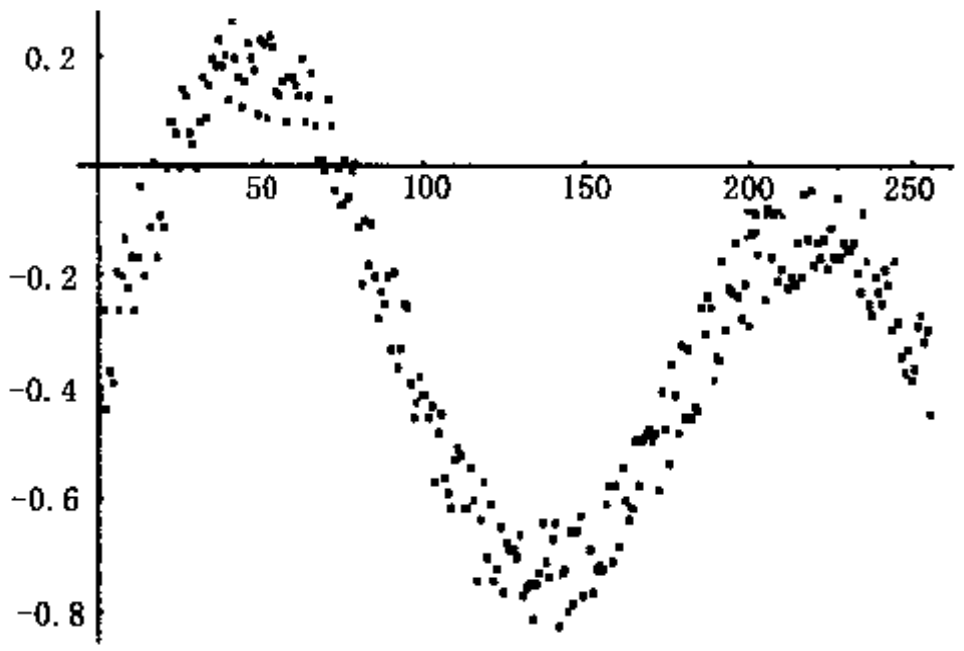


图 3.1　含有“噪音”的数据



图 3.2　一条直线公路与多个景点

插值和拟合是构造逼近函数的两种方法。

向量 Q 与 Y 之间的误差或距离有各种不同的定义方法。例如：

用各点误差绝对值的和表示：

$$R_1 = \sum_{i=1}^{m} | \varphi(x_i) - y_i |$$

用各点误差按模的最大值表示：

$$R_\infty = \max_{1 \le i \le n} | \varphi(x_i) - y_i |$$

用各点误差的平方和表示：

$$R = R_2 = \sum_{i=1}^{m} (\varphi(x_i) - y_i)^2 \text{ 或 } R = \| Q(x) - Y \|_2^2 \tag{3.1}$$

其中 R 称为均方误差。由于计算均方误差的最小值的原则容易实现而被广泛采用。按均方误差达到极小构造拟合曲线的方法称为最小二乘法。本章主要讲述用最小二乘法构造拟合曲线的方法。

在实际问题中，怎样由测量的数据设计和确定“最贴近”的拟合曲线？关键在于选择适当的拟合曲线类型，有时根据专业知识和工作经验即可确定拟合曲线类型；在对拟合曲线一无所知的情况下，不妨先绘制数据的粗略图形，或许从中观测出拟合曲线的类型；更一般地，对数据进行多种曲线类型的拟合，并计算均方误差，用数学实验的方法找出在最小二乘法意义下的误差最小的拟合函数。

例如，某风景区要在已有的景点之间修一条规格较高的主干路，景点与主干路之间由各具特色的支路联接。设景点的坐标为点列 (x_i, y_i)，$i = 1,2,\cdots,m$；设主干路为一条直线 $\varphi(x) = a + bx$，即拟合函数是一条直线。通过计算均方误差 $Q(a,b)$ 最小值而确定直线方程（见图 3.2）。

$$Q(a,b) = \sum_{i=1}^{m} (\varphi(x_i) - y_i)^2 = \sum_{i=1}^{m} (a + bx_i - y_i)^2$$

3.2 线性拟合和二次拟合函数

线性拟合

给定一组数据 (x_i, y_i)，$i = 1,2,\cdots,m$，做拟合直线 $p(x) = a + bx$，均方误差为

$$Q(a,b) = \sum_{i=1}^{m} (p(x_i) - y_i)^2 = \sum_{i=1}^{m} (a + bx_i - y_i)^2 \tag{3.2}$$

在微积分理论中，$Q(a,b)$ 的极小值要满足

$$\frac{\partial Q(a,b)}{\partial a} = 2\sum_{i=1}^{m} (a + bx_i - y_i) = 0$$

$$\frac{\partial Q(a,b)}{\partial b} = 2\sum_{i=1}^{m} (a + bx_i - y_i) x_i = 0$$

整理得到拟合曲线满足的方程：

$$\begin{cases} ma + \left(\sum_{i=1}^{m} x_i\right) b = \sum_{i=1}^{m} y_i \\ \left(\sum_{i=1}^{m} x_i\right) a + \left(\sum_{i=1}^{m} x_i^2\right) b = \sum_{i=1}^{m} x_i y_i \end{cases} \tag{3.3}$$

或

$$\begin{pmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{pmatrix}$$

称式(3.3)为拟合曲线的法方程。用消元法或克莱姆方法解出方程:

$$a = \begin{vmatrix} \sum_{i=1}^{m} y_i & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i y_i & \sum_{i=1}^{m} x_i^2 \end{vmatrix} \Bigg/ \begin{vmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{vmatrix}$$

$$= \left(\sum_{i=1}^{m} y_i \sum_{i=1}^{m} x_i^2 - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} x_i y_i\right) \Big/ \left(m \sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2\right)$$

$$b = \left(m \sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i\right) \Big/ \left(m \sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2\right)$$

例 3.1:下表为 P.Sale 及 R.Dybdall 在某处作的鱼类抽样调查,表中 x 为鱼的数量,y 为鱼的种类。请用线性函数拟合鱼的数量和种类。

x	13	15	16	21	22	23	25	29	30	31	36
y	11	10	11	12	12	13	13	12	14	16	17
x	40	42	55	60	62	64	70	72	100	130	
y	13	14	22	14	21	21	24	17	23	34	

解:设拟合直线 $p(x) = a + bx$,并计算得下表:

编号	x	y	xy	x^2
1	13	11	143	121
2	15	10	150	100
3	16	11	176	121
4	21	12	252	144
5	22	12	264	144
⋮	⋮	⋮	⋮	⋮
21	130	34	4420	1156
$\sum$	956	344	18913	61640

将数据代入法方程组(3.3)中,得到:

$$\begin{pmatrix} 21 & 956 \\ 956 & 61640 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 344 \\ 18913 \end{pmatrix}$$

解方程得:$a = 8.2084, b = 0.1795$。

拟合直线为:$p(x) = 8.2084 + 0.1795x$。

二次拟合函数

给定数据序列(x_i, y_i),$i = 1,2,\cdots,m$,用二次多项式函数拟合这组数据。

设 $p(x) = a_0 + a_1 x + a_2 x^2$,作出拟合函数与数据序列的均方误差:

$$Q(a_0, a_1, a_2) = \sum_{i=1}^{m} (p(x_i) - y_i)^2 = \sum_{i=1}^{m} (a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2 \qquad (3.4)$$

由多元函数的极值原理,$Q(a_0, a_1, a_2)$的极小值满足

$$\begin{cases} \dfrac{\partial Q}{\partial a_0} = 2\sum\limits_{i=1}^{m} (a_0 + a_1 x_i + a_2 x_i^2 - y_i) = 0 \\ \dfrac{\partial Q}{\partial a_1} = 2\sum\limits_{i=1}^{m} (a_0 + a_1 x_i + a_2 x_i^2 - y_i) x_i = 0 \\ \dfrac{\partial Q}{\partial a_2} = 2\sum\limits_{i=1}^{m} (a_0 + a_1 x_i + a_2 x_i^2 - y_i) x_i^2 = 0 \end{cases}$$

整理得二次多项式函数拟合的法方程:

$$\begin{pmatrix} m & \sum\limits_{i=1}^{m} x_i & \sum\limits_{i=1}^{m} x_i^2 \\ \sum\limits_{i=1}^{m} x_i & \sum\limits_{i=1}^{m} x_i^2 & \sum\limits_{i=1}^{m} x_i^3 \\ \sum\limits_{i=1}^{m} x_i^2 & \sum\limits_{i=1}^{m} x_i^3 & \sum\limits_{i=1}^{m} x_i^4 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum\limits_{i=1}^{m} y_i \\ \sum\limits_{i=1}^{m} x_i y_i \\ \sum\limits_{i=1}^{m} x_i^2 y_i \end{pmatrix} \qquad (3.5)$$

解此方程得到在均方误差意义下的拟合函数 $p(x)$。方程组(3.5)称为多项式拟合的法方程,法方程的系数矩阵是对称的。当拟合多项式阶 $n > 5$ 时,法方程的系数矩阵是病态的,在计算中要用双精度或一些特殊算法以保护解的准确性。

例 3.2:给定一组数据,如下表。用二次多项式函数拟合这组数据。

x	-3	-2	-1	0	1	2	3
y	4	2	3	0	-1	-2	-5

解:设 $p(x) = a_0 + a_1 x + a_2 x^2$,由计算得下表:

x	y	xy	x^2	$x^2 y$	x^3	x^4
-3	4	-12	9	36	-27	81
-2	2	-4	4	8	-8	16

续表

$\sum$						

将数据代入式(3.5)，相应的法方程为：

$$\begin{cases} 7a_0 + 0a_1 + 28a_2 = -1 \\ 0a_0 + 28a_1 + 0a_2 = -39 \\ 28a_0 + 0a_1 + 196a_2 = -7 \end{cases}$$

解方程得：$a_0 = 0.66667$，$a_1 = -1.39286$，$a_2 = -0.13095$

$\therefore p(x) = 0.66667 - 1.39286x - 0.13095x^2$

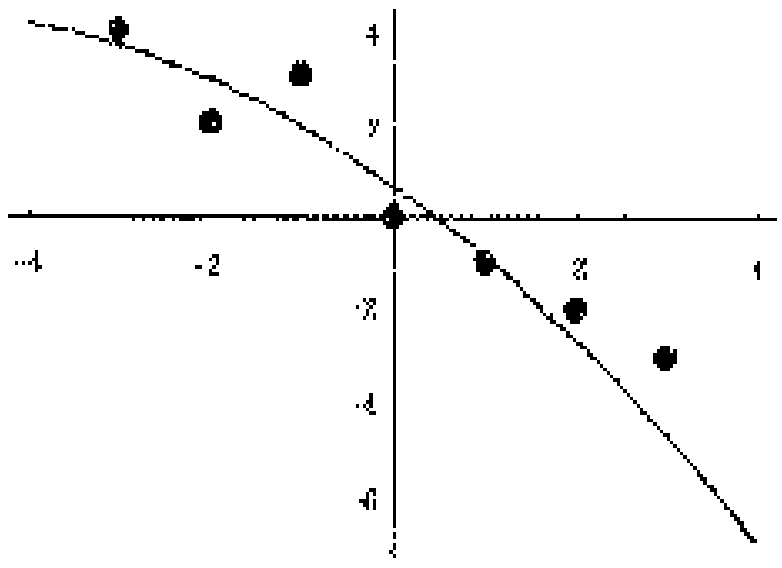


图 3.3 拟合曲线与数据示例

拟合曲线的均方误差：$\sum_{i=1}^{7} \delta_i^2 = \sum_{i=1}^{7} (p(x_i) - y_i)^2 = 3.09524$

结果见图 3.3。

3.3 解矛盾方程组

在 3.2 节中用最小二乘法构造拟合函数，本节中用最小二乘法来解线性矛盾方程的方法构造拟合函数。

给定数据序列 (x_i, y_i)，$i = 1, 2, \cdots, m$，作拟合直线 $p(x) = a_0 + a_1 x$，如果要直线 $p(x)$ 过这些点，那么有 $p(x_i) = a_0 + a_1 x_i = y_i$，$i = 1, 2, \cdots, m$，即

$$\begin{cases} a + a_1 x_1 = y_1 \\ a + a_1 x_2 = y_2 \\ \cdots \\ a + a_1 x_m = y_m \end{cases}$$

写成矩阵形式：

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

上述方程组中有 2 个未知量 m 个方程($m \gg 2$)，一般地，将含有 n 个未知量 m 个方程的线性方程组

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n = y_1 \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n = y_2 \\ \cdots \\ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = y_m \end{cases}$$

写成矩阵形式：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \cdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

在线性代数中，我们知道，若秩$(A, b) \neq$ 秩(A)，则方程组无解，这时方程组称为矛盾方程组。在数值代数中对矛盾方程计算的是在均方误差 $\min \| AX - b \|_2^2$ 极小意义下的解，也就是在最小二乘法意义下的矛盾方程的解。定理 3.1 将证明，方程组 $A^T AX = A^T b$ 的解就是矛盾方程组 $AX = b$ 在最小二乘法意义下的解。

例如，拟合直线 $p(x) = a_0 + a_1 x$ 的矛盾方程组 $A^T AX = A^T b$ 的形式如下：

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_m \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_m \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

化简得到与式(3.3)相同的法方程：

$$\begin{pmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{pmatrix}$$

定理 3.1：

1) $A^T AX = A^T b$ 称为矛盾方程 $AX = b$ 的法方程，法方程恒有解。

2) X 是 $\min \| AX - b \|_2^2$ 的解，当且仅当 X 满足 $A^T AX = A^T b$，即 X 是法方程的解。

证明：

1）法方程 $A^TAX = A^Tb$ 的增广矩阵的秩

$$\mathrm{rank}(A^TA, A^Tb) = \mathrm{rank}A^T(A, b) = \mathrm{rank}A^T = \mathrm{rank}A$$

以及系数矩阵的秩

$$\mathrm{rank}(A^TA) = \mathrm{rank}A^T = \mathrm{rank}A$$

故 $A^TAX = A^Tb$ 恒有解，又 $A^TAX = A^Tb$ 是含有 n 个未知量的线性方程组，若 $\mathrm{rank}A^T = \mathrm{rank}A = n$ 则方程组有唯一解，若 $\mathrm{rank}A < n$ 则方程组有无穷多组解。

2)设 X 满足 $A^TAX = A^Tb$，任取 Y，记 $Y = X + (Y - X) = X + e$，则

$$\begin{aligned}\| AY - b \|_2^2 &= (AX - b + Ae)^T(AX - b + Ae)\\ &= (AX - b)^T(AX - b) + 2(Ae)^T(AX - b) + (Ae)^T(Ae)\\ &= \| AX - b \|_2^2 + \| Ae \|_2^2 + 2e^T(A^TAX - A^Tb)\\ &= \| AX - b \|_2^2 + \| Ae \|_2^2 \geq \| AX - b \|_2^2\end{aligned}$$

由于 Y 是任取的，故法方程组 $A^TAX = A^Tb$ 的解为极小问题 $\min \| AX - b \|_2^2$ 的解。事实上，对离散数据 (x_i, y_i)，$i = 1, 2, \cdots, m$；作 n 次多项式曲线拟合，要计算 $Q(a_0, a_1, \cdots, a_n) = \sum_{i=1}^{m} (a_0 + a_1 x_i + \cdots + a_n x_i^n - y_i)^2$ 的极小问题。这与解矛盾方程组

$$\begin{cases} a_0 + a_1 x_1 + \cdots + a_n x_1^n = y_1 \\ a_0 + a_1 x_2 + \cdots + a_n x_2^n = y_2 \\ \qquad\qquad \cdots \\ a_0 + a_1 x_m + \cdots + a_n x_m^n = y_m \end{cases}$$

或

$$A\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

求 $\| A\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} - b \|^2 = \sum_{i=1}^{m} (a_0 + a_1 x_i + \cdots + a_n x_i^n - y_i)^2$ 的极小问题是一回事。

在这里 $A = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ & & \cdots & \\ 1 & x_m & \cdots & x_m^n \end{pmatrix}$

故对离散数据 (x_i, y_i)，$i = 1, 2, \cdots, m$；所作的 n 次拟合曲线 $a_0 + a_1 x + \cdots + a_n x^n$，可通过解下列方程组求得：

$$A^TA\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = A^T\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

例 3.3：给定一组数据，见下表，用二次多项式函数拟合这组数据。

x	-3	-2	-1	0	1	2	3
y	4	2	3	0	-1	-2	-5

解：记二次拟合曲线为 $f(x) = a_0 + a_1x + a_2x^2$，形成法方程：

$$A^TA\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = A^T\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_7 \end{pmatrix}$$

而

$$A^TA = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_7 \\ x_1^2 & x_2^2 & \cdots & x_7^2 \end{pmatrix}\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ & \cdots & \\ 1 & x_7 & x_7^2 \end{pmatrix} = \begin{pmatrix} 7 & \sum_{i=1}^{7} x_i & \sum_{i=1}^{7} x_i^2 \\ \sum_{i=1}^{7} x_i & \sum_{i=1}^{7} x_i^2 & \sum_{i=1}^{7} x_i^3 \\ \sum_{i=1}^{7} x_i^2 & \sum_{i=1}^{7} x_i^3 & \sum_{i=1}^{7} x_i^4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 9 & 4 & 1 & 0 & 1 & 4 & 9 \end{pmatrix}\begin{pmatrix} 1 & -3 & 9 \\ 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix} = \begin{pmatrix} 7 & 0 & 28 \\ 0 & 28 & 0 \\ 28 & 0 & 196 \end{pmatrix}$$

$$A^TY = \begin{pmatrix} \sum_{i=1}^{7} y_i \\ \sum_{i=1}^{7} x_iy_i \\ \sum_{i=1}^{7} x_i^2y_i \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 9 & 4 & 1 & 0 & 1 & 4 & 9 \end{pmatrix}\begin{pmatrix} 4 \\ 2 \\ 3 \\ 0 \\ -1 \\ -2 \\ -5 \end{pmatrix} = \begin{pmatrix} 1 \\ -39 \\ -7 \end{pmatrix}$$

得到：

$$\begin{pmatrix} 7 & 0 & 28 \\ 0 & 28 & 0 \\ 28 & 0 & 196 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -39 \\ -7 \end{pmatrix}$$

解方程得：$a_0 = 0.66667, a_1 = -1.39286, a_2 = -0.13095$

$\therefore \quad f(x) = 0.66667 - 1.39286x - 0.13095x^2$

例 3.4：给出一组数据，见下表。用最小二乘法求形如 $f(x) = a + bx^3$ 的经验公式。

x	-3	-2	-1	2	4
y	14.3	8.3	4.7	8.3	22.7

解：列出法方程：$A^TA\begin{pmatrix} a \\ b \end{pmatrix} = A^TY$

$$A^TA = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 & x_5^3 \end{pmatrix}\begin{pmatrix} 1 & x_1^3 \\ 1 & x_2^3 \\ 1 & x_3^3 \\ 1 & x_4^3 \\ 1 & x_5^3 \end{pmatrix} = \begin{pmatrix} 5 & \sum_{i=1}^{5} x_i^3 \\ \sum_{i=1}^{5} x_i^3 & \sum_{i=1}^{5} x_i^6 \end{pmatrix} = \begin{pmatrix} 5 & 36 \\ 36 & 4954 \end{pmatrix}$$

$$A^TY = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -27 & -8 & -1 & 8 & 64 \end{pmatrix}\begin{pmatrix} 14.3 \\ 8.3 \\ 4.7 \\ 8.3 \\ 22.7 \end{pmatrix} = \begin{pmatrix} 58.3 \\ 1062 \end{pmatrix}$$

法方程为：

$$\begin{pmatrix} 5 & 36 \\ 36 & 4954 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 58.3 \\ 1062 \end{pmatrix}$$

解方程得到：$a = 10.675, b = 0.137$

拟合曲线为：$f(x) = 10.678 + 0.137x^3$

有些非线性函数经过转换以后可化为线性函数计算。例如，$\frac{1}{y} = a + b\frac{1}{x}$，令 $u = \frac{1}{y}$，$v = \frac{1}{x}$，则化拟合曲线为：$u = a + bv$。

例 3.5：求一个形如 $y = ae^{bx}$ 的经验函数公式，使它能够和下列数据相拟合。

x	1	2	3	4	5	6	7	8
y	15.3	20.5	27.4	36.6	49.1	65.6	87.8	117.6

解：化经验公式为线性形式，对经验公式的两边取自然对数有

$$\ln y = \ln a + bx$$

由矛盾方程组

$$\begin{cases} \ln a + bx_1 = \ln y_1 \\ \ln a + bx_2 = \ln y_2 \\ \cdots \\ \ln a + bx_8 = \ln y_8 \end{cases}$$

得到法方程组

$$\begin{pmatrix} m & \sum_{i=1}^{8} x_i \\ \sum_{i=1}^{8} x_i & \sum_{i=1}^{8} x_i^2 \end{pmatrix} \begin{pmatrix} \ln a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{8} \ln y_i \\ \sum_{i=1}^{8} x_i \ln y_i \end{pmatrix}$$

即

$$\begin{pmatrix} 8 & 36 \\ 36 & 204 \end{pmatrix} \begin{pmatrix} \ln a \\ b \end{pmatrix} = \begin{pmatrix} 13.0197 \\ 63.9003 \end{pmatrix}$$

解方程组得：$\ln a = 2.4369, b = 0.2912, a = e^{2.4369} = 11.4375$

$\therefore\ y = 11.4375e^{0.2912x}$

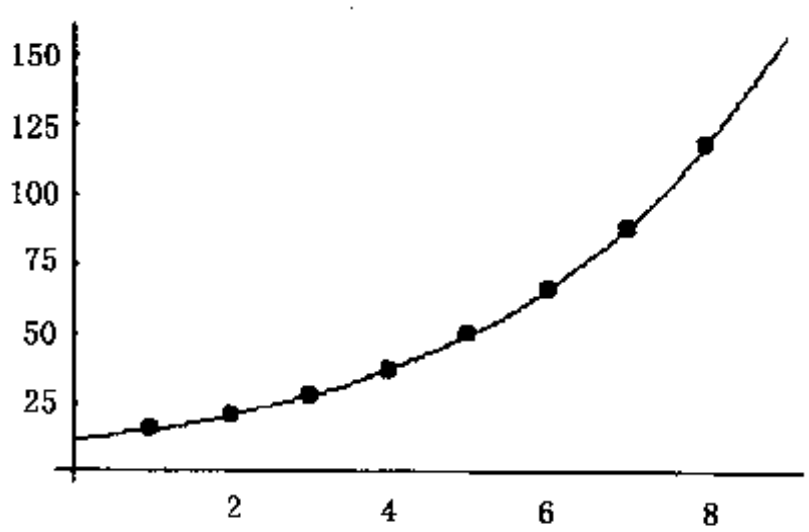


图 3.4　拟合曲线图示

拟合曲线的均方误差为：

$$\sum_{i=1}^{8} \delta_i^2 = \sum_{i=1}^{8} (y(x_i) - y_i)^2 = 0.0262401$$

计算结果见图 3.4。

例 3.6：解矛盾方程组

$$\begin{cases} x_1 + x_2 + x_3 = 2 \\ x_1 + 3x_2 - x_3 = -1 \\ 2x_1 + 5x_2 + 2x_3 = 1 \\ 3x_1 - x_2 + 5x_3 = -2 \end{cases}$$

解：写出法方程组 $A^TAX = A^TY$，即

$$\begin{pmatrix}1 & 1 & 2 & 3\\ 1 & 3 & 5 & -1\\ 1 & -1 & 2 & 5\end{pmatrix}\begin{pmatrix}1 & 1 & 1\\ 1 & 3 & -1\\ 2 & 5 & 2\\ 3 & -1 & 5\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}=\begin{pmatrix}1 & 1 & 2 & 3\\ 1 & 3 & 5 & -1\\ 1 & -1 & 2 & 5\end{pmatrix}\begin{pmatrix}2\\ -1\\ 1\\ -2\end{pmatrix}$$

得

$$\begin{pmatrix}15 & 11 & 19\\ 11 & 36 & 3\\ 19 & 3 & 31\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}=\begin{pmatrix}-1\\ 6\\ -5\end{pmatrix}$$

解法方程得：$x_1=1.5917, x_2=0.5899, x_3=0.7572$。

3.4 程序示例

程序 5 用线性函数 $p(x)=a+bx$ 拟合给定数据$(x_i, y_i), i=0,1,\cdots,m-1$。

算法描述

输入 m 值，及$(x_i, y_i), i=0,1,\cdots,m-1$。

解方程组：

$$\begin{pmatrix}m & \sum_{i=0}^{m-1}x_i\\ \sum_{i=0}^{m-1}x_i & \sum_{i=0}^{m-1}x_i^2\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix}=\begin{pmatrix}\sum_{i=0}^{m-1}y_i\\ \sum_{i=0}^{m-1}x_iy_i\end{pmatrix}$$

输出 $p(x)=a+bx$。

程序源码

```
////////////////////////////////////////////////////////
//      Purpose: (x_i,y_i) 线性拟合函数     //
////////////////////////////////////////////////////////
# include <stdio.h>
# define MAX_N 25                    //(x_i,y_i)的最大维数
typedef struct tagPOINT              //点的结构
{
    double x;
    double y;
} POINT;
int main()
{
    int m;
    int i;
    POINT points[MAX_N];
    static double u11, u12, u21, u22, c1, c2;
    double a, b, tmp;
    printf("\nInput m value: ");          //输入点数 m
```

```
scanf("%d", &m);
if (m > MAX_N)
{
    printf("The input m is larger then MAX_N, please redefine the MAX_N. \n");
    return 1;
}
if (m <= 0)
{       printf("Please input a number between 1 and %d. \n", MAX_N);
        return 1;   }
printf("Now input the (x_i, y_i), i = 0,...,%d: \n", m-1);      //输入点
for (i=0; i<m; i++)
{
    scanf("%lf", &tmp);
    points[i].x = tmp;
        scanf("%lf", &tmp);
        points[i].y = tmp;
//          scanf("%lf%lf", &points[i].x, &points[i].y);
  }
for (i=0; i<m; i++)                    //列出方程 U(a, b)' = c
{
    u21 += points[i].x;
    u22 += points[i].x * points[i].x;
    c1 += points[i].y;
    c2 += points[i].x * points[i].y;
}
u12 = u21;
u11 = m;
                                //求解
a = (c1 * u22 - c2 * u12)/(u11 * u22 - u12 * u21);
b = (c1 * u21 - c2 * u11)/(u21 * u12 - u22 * u11);
printf("Solve: p(x) = %f + %f x \n", a, b);          //输出
return 0;
}  // - - - - - - - - - End of File - - - - - - - - - - - - -
```

计算实例

用线性函数拟合下列数据(例 3.1)。

x	13	15	16	21	22	23	25	29	30	31	36
y	11	10	11	12	12	13	13	12	14	16	17
x	40	42	55	60	62	64	70	72	100	130	
y	13	14	22	14	21	21	24	17	23	34	

程序输入输出

```
input m value: 21
Now input the (x_i,y_i), i=0,...,20:
13 11 15 10 16 11 21 12 22 12 23 13 25 13 29 12
30 14 31 16 36 17 40 13 42 14 55 22 60 14 62 21
64 21 70 24 72 17 100 23 130 34
Solve: p(x) = 8.208408 + 0.179522 x
```

程序 6 用形如 $p(x)=ae^{bx}$ 的函数拟合给定数据 $(x_i,y_i),i=0,1,\cdots,m-1$。

算法描述

输入 m 值,及 $(x_i,y_i),i=0,1,\cdots,m-1$。

解方程组:

$$\begin{pmatrix} m & \sum_{i=0}^{m-1} x_i \\ \sum_{i=0}^{m-1} x_i & \sum_{i=0}^{m-1} x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{m-1} \log y_i \\ \sum_{i=0}^{m-1} x_i \log y_i \end{pmatrix}$$

输出所求拟合函数 $p(x)=ae^{bx}$。

程序源码

```
//////////////////////////////////////////////////////////////////////////
//      Purpose: (x_i,y_i) 形如 y = aExp(bx)的拟合函数      //
//////////////////////////////////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
# define MAX_N 25                    //(x_i,y_i)的最大维数
typedef struct tagPOINT               //点的结构
{
   double x;
    double y;
} POINT;
int main()
{   int m;
    int i;
POINT points[MAX_N];
static double u11, u12, u21, u22, c1, c2;
double A, B, tmp;
printf("\nInput m value: ");          //输入点数 m
scanf("%d", &m);
if (m > MAX_N)
{
    printf("The input m is larger then MAX_N, please redefine the MAX_N.\n");
    return 1;
}
if (m <= 0)
```

```
    }
        printf("Please input a number between 1 and %d. \n", MAX_N);
        return 1;
    }
    printf("Now input the (x_i, y_i), i=0,...,%d: \n", m-1);      //输入点
    for (i=0; i<m; i++)
    {   scanf("%lf", &tmp);
        points[i].x=tmp;
        scanf("%lf", &tmp);
        points[i].y=tmp;
//          scanf("%lf%lf", &points[i].x, &points[i].y);
    }
    for (i=0; i<m; i++)                          //列出方程 X(a, b)' = Y
    {                                  //其中 Y 为对原方程取对数后的数据
        u21+=points[i].x;
        u22+=points[i].x*points[i].x;
        c1+=log(points[i].y);
        c2+=points[i].x*log(points[i].y);
    }
    u12=u21;
    u11=m;
                                //求解
    A=(c1*u22-c2*u12)/(u11*u22-u12*u21);
    B=(c1*u21-c2*u11)/(u21*u12-u22*u11);
    printf("Solve: p(x)=%lfexp(%lfx)\n", exp(A), B);          //输出
    return 0;
} // ---------- End of File ----------
```

计算实例

求一个形如 $y=ae^{bx}$ 的经验函数公式,使它能够和下列数据相拟合(例 3.5)。

x	1	2	3	4	5	6	7	8
y	15.3	20.5	27.4	36.6	49.1	65.6	87.8	117.6

程序输入输出

```
input m value: 8
Now input the (x_i,y_i), i=0,...,7:
1 15.3 2 20.5 3 27.4 4 36.6 5 49.1 6 65.6 7 87.8 8 117.6
Solve: p(x)=11.437069exp(0.291215x)
```

习题 3

3.1 下表中的资料说明各种等级汽车在碰撞时严重损伤的比率,求此数据的最小平方近

似直线。

车　型	车　重（磅）	严重损伤(%)
美国制豪华型	4800	3.1
美国制中级型	3700	4.0
美国制经济型	3400	5.2
美国制轻便型	2800	6.4
其它国制轻便型	1900	9.6

3.2　下表给出一组数据：

x_i	-1.00	-0.50	0.00	0.25	0.75
y_i	0.2200	0.800	2.000	2.5000	3.8000

分别用一次、二次多项式拟合这些数据，并给出最佳平方误差。

3.3　给出下列数据：

x	-3	-2	-1	2	4
y	14.3	8.3	4.7	8.3	22.7

用最小二乘法求形如 $y = a + bx^2$ 的经验公式。

3.4　给出下列数据：

x_i	-0.70	-0.50	0.25	0.75
y_i	0.99	1.21	2.57	4.23

用最小二乘法求形如 $y = ae^{bx}$ 的经验公式。

3.5　用最小二乘法求解矛盾方程组。

$$(1)\begin{cases} x_1 + 2x_2 = 5 \\ 2x_1 + x_2 = 6 \\ x_1 + x_2 = 4 \end{cases} \qquad (2)\begin{cases} x_1 - 2x_2 = 1 \\ x_1 + 5x_2 = 13.1 \\ 2x_1 + x_2 = 7.9 \\ x_1 + x_2 = 5.1 \end{cases}$$

第 4 章　非线性方程求根

与线性方程相比，非线性方程问题无论是从理论上还是从计算公式上，都要复杂得多。对于一般的非线性方程 $f(x)=0$，计算方程的根既无一定章程可寻也无直接法可言。例如，求解高次方程组 $7x^6-x^3+x-1.5=0$ 的根，求解含有指数和正弦函数的超越方程 $e^x-\cos(\pi x)=0$ 的零点。解非线性方程或非线性方程组也是计算方法中的一个主题。一般地，我们用符号 $f(x)$ 来表示方程左端的函数，方程的一般形式表示为 $f(x)=0$，方程的解称为方程的根或函数的零点。

通常，非线性方程的根不止一个，而任何一种方法一次只能算出一个根。因此，在求解非线性方程时，要给定初始值或求解范围。

4.1　实根的对分法

使用对分法的条件

对分法或称二分法是求方程近似解的一种简单直观的方法。设函数 $f(x)$ 在 $[a,b]$ 上连续，且 $f(a)f(b)<0$，则 $f(x)$ 在 $[a,b]$ 上至少有一零点，这是微积分中的介值定理，也是使用对分法的前题条件。计算中通过对分区间，缩小区间范围的步骤搜索零点的位置。

例 4.1：用对分法求 $f(x)=x^3-7.7x^2+19.2x-15.3$ 在区间[1,2]之间的根。

解：

1) $f(1)=-2.8$，$f(2)=0.3$，由介值定理可得有根区间 $[a,b]=[1,2]$。

2) 计算 $x_1=(1+2)/2=1.5$，$f(1.5)=-0.45$，有根区间 $[a,b]=[1.5,2]$。

3) 计算 $x_2=(1.5+2)/2=1.75$，$f(1.75)=0.078125$，有根区间 $[a,b]=[1.5,1.75]$。

一直做到 $|f(x_n)|<\varepsilon$（计算前给定的精度）或 $|a-b|<\varepsilon$ 时停止。详细计算结果见表 4.1。

表 4.1

k	x	$f(x)$	求解区间	$\lvert x_k-x_{k-1}\rvert$
0	1	-2.8		
1	2	0.3	[1, 2]	
2	1.5	-0.45	[1.5,2]	0.5
3	1.75	0.078125	[1.5, 1.75]	0.25
4	1.625	-0.141797	[1.625,1.75]	0.125
5	1.6875	-0.0215332	[1.6875,1.75]	0.0625
6	1.71875	0.03078	[1.6875,1.71875]	0.03125
7	1.70312	0.00525589	[1.6875,1.70312]	0.015625

对分法求根算法

计算 $f(x)=0$ 的一般计算步骤如下：

1：输入求根区间$[a,b]$和误差控制量 ε，定义函数 $f(x)$。

IF $(f(a)f(b)<0)$THEN　做步骤 2

ELSE　退出选用其它求根方法

2：WHILE $|a-b|>\varepsilon$ 时

(1) 计算中点 $x=(a+b)/2$ 以及 $f(x)$的值

(2) 分情况处理

$|f(x)|<\varepsilon$：　　停止计算 $x^*=x$，转向步骤 4

$f(a)f(x)<0$：　　修正区间$[a,x]\rightarrow[a,b]$

$f(x)f(b)<0$：　　修正区间$[x,b]\rightarrow[a,b]$

ENDWHILE

3：$x^*=\dfrac{a+b}{2}$

4：输出近似根 x^*。

图 4.1 给出对分法的示意图。

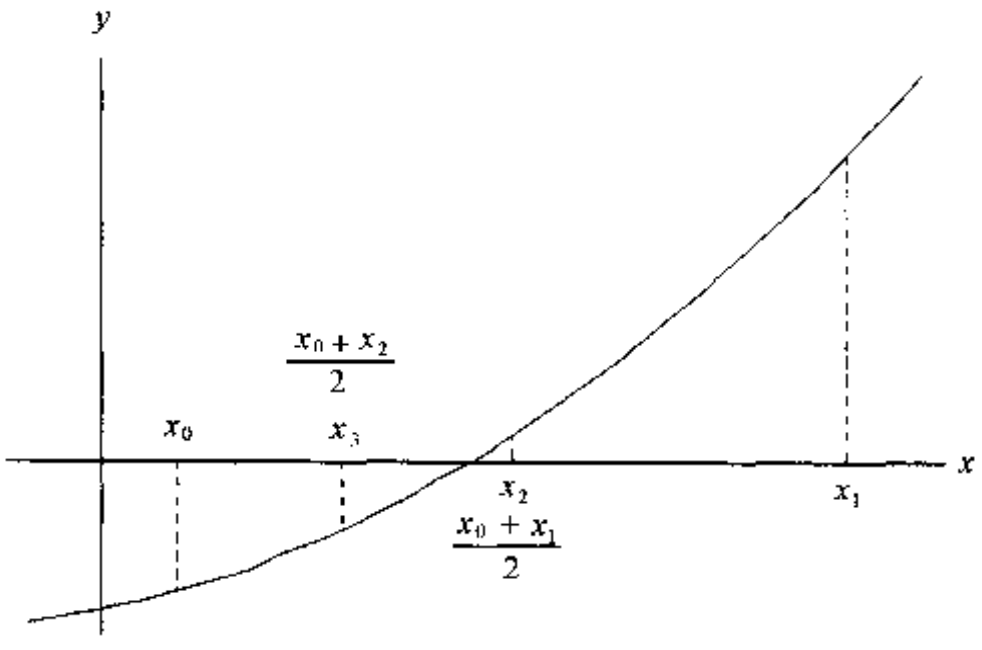


图 4.1　逐步对分区间

对分法的算法简单，然而，若 $f(x)$在$[a,b]$上有几个零点时，只能算出其中一个零点；另一方面，即使 $f(x)$在$[a,b]$上有零点，也未必有 $f(a)f(b)<0$。这就限制了对分法的使用范围。对分法只能计算方程 $f(x)=0$ 的实根。

4.2　迭　代　法

对给定的方程 $f(x)=0$，将它转换成等价形式：$x=\varphi(x)$。给定初值 x_0，由此来构造迭代序列 $x_{k+1}=\varphi(x_k)$，$k=1,2,\cdots$，如果迭代收敛，即 $\lim\limits_{k\to\infty}x_{k+1}=\lim\limits_{k\to\infty}\varphi(x_k)=\alpha$，有 $\alpha=\varphi(\alpha)$，则 α 就是方程$f(x)=0$ 的根。在计算中当$|x_{k+1}-x_k|$小于给定的精度控制量时，取 $\alpha=x_{k+1}$为方程的根。

例如，代数方程 $x^3-2x-5=0$ 的三种等价形式及其迭代格式如下：

(1) $x^3 = 2x + 5, x = \sqrt[3]{2x+5}$,迭代格式 $x_{k+1} = \sqrt[3]{2x_k + 5}$

(2) $2x = x^3 - 5$,迭代格式 $x_{k+1} = \dfrac{x_k^3 - 5}{2}$

(3) $x^3 = 2x + 5, x = \dfrac{2x+5}{x^2}$,迭代格式 $x_{k+1} = \dfrac{2x_k + 5}{x_k^2}$

对于方程 $f(x)=0$ 构造的多种迭代格式 $x_{k+1} = \varphi(x_k)$,怎样判断构造的迭代格式是否收敛? 收敛是否与迭代的初值有关?

定理 1:若 $\varphi(x)$定义在$[a,b]$上,如果 $\varphi(x)$满足

1) 当 $x\in[a,b]$有 $a\leq\varphi(x)\leq b$;

2) $\varphi(x)$在$[a,b]$上可导,并且存在正数 $L<1$,使对任意的 $x\in[a,b]$,有 $|\varphi'(x)|\leq L$;

则在$[a,b]$上有唯一的点 x^* 满足 $x^* = \varphi(x^*)$,称 x^* 为 $\varphi(x)$的不动点。而且迭代格式 $x_{k+1} = \varphi(x_k)$对任意的初值 $x_0\in[a,b]$均收敛于 $\varphi(x)$的不动点 x^*,并有误差估计式

$$|x^* - x_k| \leq \frac{L^k}{1-L}|x_1 - x_0| \tag{4.1}$$

证明:(1) 令 $\Psi(x) = x - \varphi(x)$,则有

$$\Psi(a) = a - \varphi(a) \leq 0$$
$$\Psi(b) = b - \varphi(b) \geq 0$$

故有 $a\leq x^*\leq b$,使得

$$\Psi(x^*) = x^* - \varphi(x^*) = 0 \text{ 或 } x^* = \varphi(x^*)$$

另一方面,若另有 x^{**} 满足 $x^{**} = \varphi(x^{**})$,则由

$$|x^* - x^{**}| = |\varphi(x^*) - \varphi(x^{**})|$$
$$= |\varphi'(\xi)(x^* - x^{**})| \leq L|x^* - x^{**}|, \xi\in[a,b]$$

以及 $L<1$,得到 $x^* = x^{**}$。

(2) 当 $x_0\in[a,b]$时可用归纳法证明,迭代序列$\{x_k\}\subset[a,b]$,于是由微分中值定理

$$x_{k+1} - x^* = \varphi(x_k) - \varphi(x^*) = \varphi'(\xi)(x_k - x^*), \xi\in[a,b]$$

和 $|\varphi'(x)|\leq L$,得

$$\begin{aligned}|x_{k+1} - x^*| &\leq L|x_k - x^*| = L|\varphi(x_{k-1}) - \varphi(x^*)| \\ &\leq L^2|x_{k-1} - x^*| \cdots \leq L^{k+1}|x_0 - x^*|\end{aligned} \tag{4.2}$$

因为 $L<1$,所以当 $k\to\infty$ 时,$L^{k+1}\to 0$,$x_{k+1}\to x^*$,即迭代格式 $x_{k+1} = \varphi(x_k)$收敛。

(3)误差估计式:

$$|x_{k+1} - x_k| = |\varphi(x_k) - \varphi(x_{k-1})| \leq L|x_k - x_{k-1}| \cdots \leq L^k|x_1 - x_0|$$

设 k 固定,对于任意的正整数 p 有:

$$|x_{k+p} - x_k| \leq |x_{k+p} - x_{k+p-1}| + |x_{k+p-1} - x_{k+p-2}| + \cdots |x_{k+1} - x_k|$$
$$\leq (L^{k+p-1} + L^{k+p-2} + \cdots + L^k)|x_1 - x_0| = \frac{L^k(1-L^p)}{1-L}|x_1 - x_0|$$

由于 p 的任意性及 $\lim\limits_{p\to\infty} x_{k+p} = x^*$,故有

$$|x^* - x_k| \leq \frac{L^k}{1-L}|x_1 - x_0|$$

要构造满足定理条件的等价形式一般是难于做到的。事实上,如果 x^* 为 $f(x)$ 的零点,若能构造等价形式 $x=\varphi(x)$,而 $|\varphi'(x^*)|<1$,由 $\varphi'(x)$ 的连续性,一定存在 x^* 的邻域 $[x^*-\rho, x^*+\rho]$,其上有 $|\varphi'(x)|<L<1$,这时若初值 $x_0 \in [x^*-\rho, x^*+\rho]$ 迭代也就收敛了。由此构造收敛迭代格式有两个要素,其一,等价形式 $x=\varphi(x)$ 应满足 $|\varphi'(x^*)|<1$;其二,初值必须取自 x^* 的充分小邻域,这个邻域大小决定于函数 $f(x)$,及做出的等价形式 $x=\varphi(x)$。

例 4.2:求代数方程 $x^3-2x-5=0$,在 $x_0=2$ 附近的实根。

解:1) $x^3=2x+5$

$$x_{k+1} = \sqrt[3]{2x_k+5}$$

$\because \varphi'(x)=\frac{1}{3}\cdot\frac{1}{(2x+5)^{\frac{2}{3}}}$, $|\varphi'(x)|<1$, 当 $x\in[1.5,2.5]$

$\therefore$ 构造的迭代序列收敛。取 $x_0=2$,则

$$x_1 = 2.08008, \quad x_2 = 2.09235, \quad x_3 = 2.094217,$$
$$x_4 = 2.094494, \quad x_5 = 2.094543, \quad x_6 = 2.094550$$

准确的解是 $x=2.09455148150$。

2) 将迭代格式写为

$$x_{n+1} = \frac{x_n^3-5}{2}, \qquad \varphi_2(x) = \frac{x^3-5}{2}$$

$\because |\varphi_2'(x)| = \left|\frac{3x^2}{2}\right| > 1$, 当 $x\in[1.5,2.5]$

$\therefore$ 迭代格式 $x_{n+1}=\varphi_2(x_n)$ 不能保证收敛。

4.3 牛顿迭代法

对方程 $f(x)=0$ 可构造多种迭代格式 $x_{k+1}=\varphi(x_k)$,牛顿迭代法是借助于对函数 $f(x)=0$ 作泰勒展开而构造的一种迭代格式。

将 $f(x)=0$ 在初始值 x_0 作泰勒展开:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots$$

取展开式的线性部分作为 $f(x)\approx 0$ 的近似值,则有:

$$f(x_0) + f'(x_0)(x-x_0) \approx 0$$

设 $f'(x_0)\neq 0$,则

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

令

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

类似地,再将 $f(x)=0$ 在 x_1 作泰勒展开并取其线性部分得到:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

一直做下去得到牛顿迭代格式：

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 1,2,\cdots \tag{4.3}$$

牛顿迭代格式对应于 $f(x)=0$ 的等价方程是

$$x = \varphi(x) = x - \frac{f(x)}{f'(x)}$$

$$\varphi'(x) = \frac{f(x)f''(x)}{(f'(x))^2} \tag{4.4}$$

若 α 是 $f(x)$ 的单根时，$f(\alpha)=0, f'(\alpha)\neq 0$，则有 $|\varphi'(\alpha)|=0$，只要初值 x_0 充分接近 α，牛顿迭代都收敛。牛顿迭代是二阶迭代方法。可以证明，α 为 $f(x)$ 的 p 重根时，迭代也收敛，但这已是一阶迭代，收敛因子为 $1-\frac{1}{p}$，若这时取下面迭代格式，它仍是二阶方法：

$$x_{k+1} = x_k - p\frac{f(x_k)}{f'(x_k)}, \quad k = 1,2,\cdots$$

牛顿法的几何意义

以 $f'(x_0)$ 为斜率作过 $(x_0, f(x_0))$ 点的直线，即作 $f(x)$ 在 x_0 点的切线方程

$$y - f(x_0) = f'(x_0)(x - x_0)$$

令 $y=0$，则得此切线与 x 轴的交点 x_1，即

$$x_1 = x_0 - f(x_0)/f'(x_0)$$

再作 $f(x)$ 在 x_1 处的切线，得交点 x_2，逐步逼近方程的根 α。如图 4.2 所示。

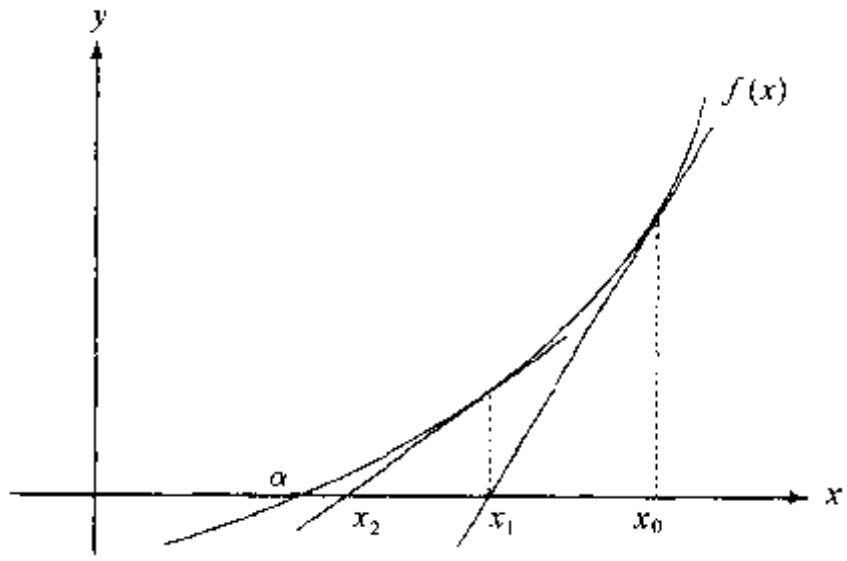


图 4.2　牛顿切线法示意图

在区域 $[x_0, x_0+h]$ 的局部“以直代曲”是处理非线性问题的常用手法。在泰勒展开中，截取函数展开的线性部分替代 $f(x)$。

例 4.3：用牛顿迭代法求方程 $f(x)=x^3-7.7x^2+19.2x-15.3$，在 $x_0=1$ 附近的根。

解：$x_{k+1} = x_k - \dfrac{x^3 - 7.7x_k^2 + 19.2x_k - 15.3}{3x_k^2 - 15.4x_k + 19.2}$

计算结果列于表 4.2 中。

表 4.2

k	x_k	$f(x)$
0	1.00	-2.8
1	1.41176	-0.727071
2	1.62424	-0.145493
3	1.6923	-0.0131682
4	1.69991	-0.0001515
5	1.7	0

比较表 4.1 和表 4.2 的数值,可以看到牛顿迭代法的收敛速度明显快于对分法。

牛顿迭代法也有局限性。在牛顿迭代法中,选取适当迭代初始值 x_0 是求解的前题,当迭代的初始值 x_0 在某根的附近时迭代才能收敛到这个根,有时会发生从一个根附近跳向另一个根附近的情况,尤其在导数 $f'(x_0)$数值很小时,如图 4.3 所示。

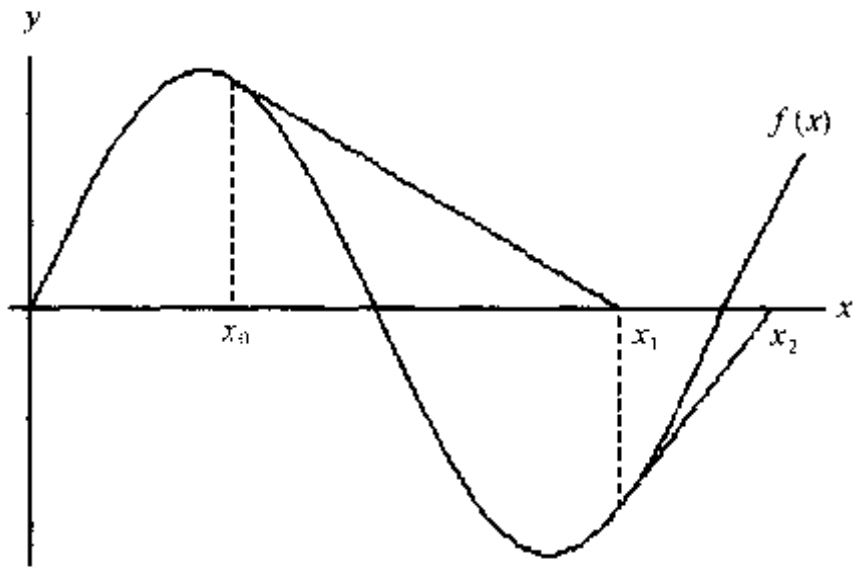


图 4.3 失效的牛顿法

如果 $f(x)=0$ 没有实根,初始值 x_0 是实数,则迭代序列不收敛。图 4.4 给出迭代函数 $f(x)=2+x^2$,初始值 $x_0=2$ 的发散的迭代序列。

牛顿迭代算法

1:定义函数 $f(x)$,输入或定义迭代初始值 x_k0 和控制精度 epsilon。

2:FOR i = 0,1,…,MAXREPT

$x_k1:=f(x_k0)$ //迭代

IF ($|x_k1-x_k0|$ < epsilon)

THEN {输出满足给定精度的近似解 x_k1,结束}

$x_k0:=x_k1$

ENDFOR

3:输出:在 x_k0 附近 $f(x)$无根。

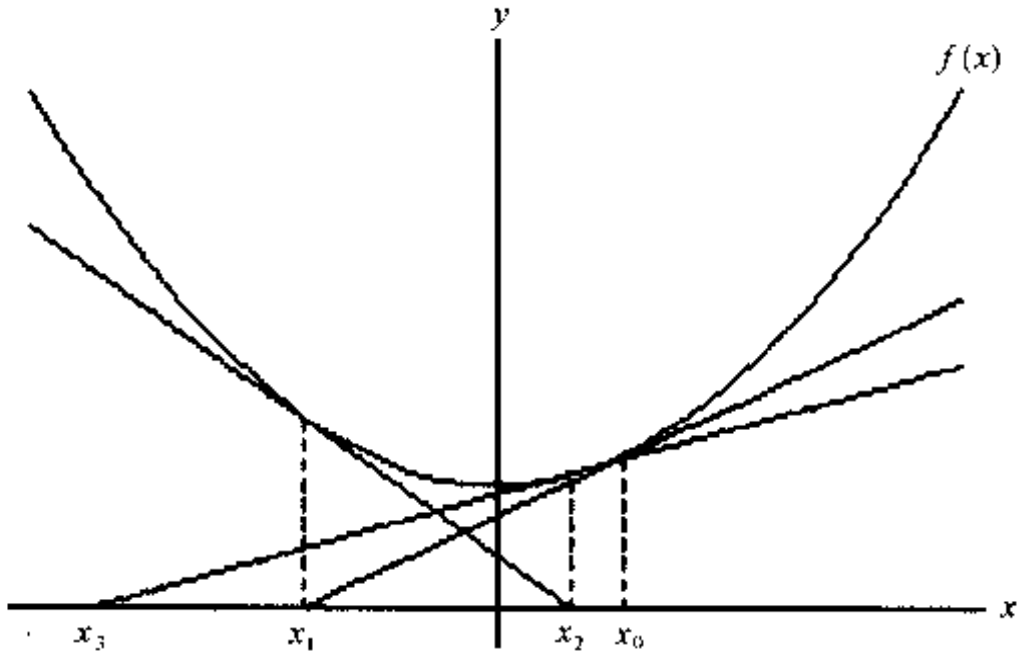


图 4.4　牛顿迭代序列不收敛

4.4　弦　截　法

弦截法迭代格式

在牛顿迭代格式中：$x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}, k = 1,2,\cdots$，用差商 $f[x_{k-1},x_k] = \dfrac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}$ 代替导数 $f'(x_k)$，并给定两个初始值 x_0 和 x_1，那么迭代格式可写成如下形式：

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, \quad k = 1,2,\cdots \tag{4.5}$$

上式称为弦截法。

用弦截法迭代求根，每次只需计算一次函数值，而用牛顿迭代法每次要计算一次函数值和一次导数值。但弦截法收敛速度稍慢于牛顿迭代法，弦截法为 1.618 阶迭代方法。

弦截法的几何意义

做过两点$(x_0,f(x_0))$和$(x_1,f(x_1))$的一条直线(弦)，该直线与 x 轴的交点就是生成

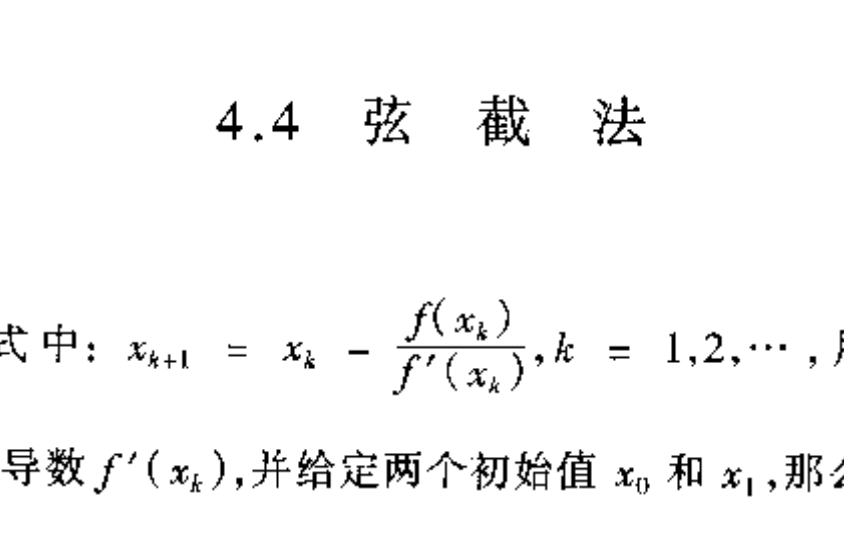


图 4.5　弦截法示意图

的迭代点 x_2，再做过$(x_1, f(x_1))$和$(x_2, f(x_2))$的一条直线，x_3 是该直线与 x 轴的交点，继续做下去得到方程的根 $f(\alpha)=0$，如图 4.5 所示。

例 4.4：用弦截法求方程 $f(x)=x^3-7.7x^2+19.2x-15.3$ 的根，取 $x_0=1.5, x_1=4.0$。

解：$x_{k+1}=x_k-\dfrac{f(x_k)(x_k-x_{k-1})}{f(x_k)-f(x_{k-1})}$

计算结果列于表 4.3 中。

表 4.3

k	x_k	$f(x)$
0	1.5	-0.45
1	4	2.3
2	1.90909	0.248835
3	1.65543	-0.0805692
4	1.71748	0.0287456
5	1.70116	0.00195902
6	1.69997	-0.0000539246
7	1.7	$9.459\ 10^{-8}$

弦截法算法

1：定义函数 $f(x)$，输入或定义控制精度 epsilon 和迭代初始值 x_k1, x_k2。

计算 $f1:=f(x_k1)$ ！x_k1, x_k2 表示 x_{k-1}, x_k

2：FOR i = 0,1,…, MAXREPT

2.1 $f2:=f(x_k2)$

2.2 $x_k:=x_k2-f2(x_k2-x_k1)/(f2-f1)$ ！x_k 表示 x_{k+1}

2.3 IF ($|x_k-x_k2|<$ epsilon)

THEN {输出满足给定精度的近似解 x_k，结束}

2.4 $f1:=f2$ ！ 为下一次迭代准备数值

$x_k1:=x_k2$

$x_k2:=x_k$

ENDFOR

3：输出：在初始值 x_k1, x_k2 附近 $f(x)$无根。

4.5 非线性方程组的牛顿方法

为了叙述简单，我们以解二阶非线性方程组为例演示解题的方法和步骤，类似地可以得到解更高阶非线性方程组的方法和步骤。

设二阶方程组

$$\begin{cases} f_1(x,y)=0 \\ f_2(x,y)=0 \end{cases} \tag{4.6}$$

其中 x,y 为自变量。为了方便起见,将方程组写成向量形式:

$$F(w) = \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}, \text{其中 } w = \begin{pmatrix} x \\ y \end{pmatrix}$$

将 $f_1(x,y), f_2(x,y)$ 在 (x_0,y_0) 附近作二元泰勒展开,并取其线性部分,得到方程组

$$\begin{cases} f_1(x_0,y_0) + (x-x_0)\dfrac{\partial f_1(x_0,y_0)}{\partial x} + (y-y_0)\dfrac{\partial f_1(x_0,y_0)}{\partial y} = 0 \\ f_2(x_0,y_0) + (x-x_0)\dfrac{\partial f_2(x_0,y_0)}{\partial x} + (y-y_0)\dfrac{\partial f_2(x_0,y_0)}{\partial y} = 0 \end{cases}$$

令 $x - x_0 = \Delta x, y - y_0 = \Delta y$,则有

$$\begin{cases} \Delta x \dfrac{\partial f_1(x_0,y_0)}{\partial x} + \Delta y \dfrac{\partial f_1(x_0,y_0)}{\partial y} = -f_1(x_0,y_0) \\ \Delta x \dfrac{\partial f_2(x_0,y_0)}{\partial x} + \Delta y \dfrac{\partial f_2(x_0,y_0)}{\partial y} = -f_2(x_0,y_0) \end{cases} \tag{4.7}$$

如果 $|J(x_0,y_0)| = \left| \begin{matrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} \end{matrix} \right|_{(x_0,y_0)} \neq 0$,解出 $\Delta x, \Delta y$,得

$$w_1 = w_0 + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} x_0 + \Delta x \\ y_0 + \Delta y \end{pmatrix}$$

再列出方程组:

$$\begin{cases} \dfrac{\partial f_1(x_1,y_1)}{\partial x}(x-x_1) + \dfrac{\partial f_1(x_1,y_1)}{\partial y}(y-y_1) = -f_1(x_1,y_1) \\ \dfrac{\partial f_2(x_1,y_1)}{\partial x}(x-x_1) + \dfrac{\partial f_2(x_1,y_1)}{\partial y}(y-y_1) = -f_2(x_1,y_1) \end{cases}$$

解出 $\Delta x = x - x_1, \Delta y = y - y_1$,得出

$$w_2 = \begin{pmatrix} x_1 + \Delta x \\ y_1 + \Delta x \end{pmatrix}$$

继续做下去,每一次迭代都是解一个方程组

$$J(x_k,y_k)\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} f_1(x_k,y_k) \\ f_2(x_k,y_k) \end{pmatrix} \tag{4.8}$$

$$\Delta x = x_{k+1} - x_k, \Delta y = y_{k+1} - y_k$$

即 $x_{k+1} = x_k + \Delta x, y_{k+1} = y_k + \Delta y$,直到 $\max(|\Delta x|, |\Delta y|) < \epsilon$ 为止。

例 4.5:求解非线性方程组

$$\begin{cases} f_1(x,y) = 4 - x^2 - y^2 = 0 \\ f_2(x,y) = 1 - e^x - y = 0 \end{cases}$$

取初始值 $w_0 = \begin{pmatrix} 1 \\ -1.7 \end{pmatrix}$。

$$\text{解：} J(x,y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{pmatrix} -2x & -2y \\ -e^x & -1 \end{pmatrix}$$

$$J(x_0,y_0) = \begin{pmatrix} -2 & 3.4 \\ -2.71828 & -1 \end{pmatrix}$$

$$F(w_0) = \begin{pmatrix} f_1(x_0,y_0) \\ f_2(x_0,y_0) \end{pmatrix} = \begin{pmatrix} 0.11 \\ -0.01828 \end{pmatrix}$$

$$\begin{cases} -2\Delta x + 3.4\Delta y = -0.11 \\ -2.71828\Delta x - \Delta y = 0.01828 \end{cases}$$

$$\text{解方程得 } \Delta w = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 0.004256 \\ -0.029849 \end{pmatrix}$$

$$\therefore\ w_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \Delta w = \begin{pmatrix} 1 \\ -1.7 \end{pmatrix} + \begin{pmatrix} 0.004256 \\ -0.029849 \end{pmatrix} = \begin{pmatrix} 1.004256 \\ -1.729849 \end{pmatrix}$$

继续做下去，直到 $\max(|\Delta x|,|\Delta y|) < 10^{-5}$时停止。

4.6 程 序 示 例

程序 7 用牛顿迭代法求 $f(x)$在 x_0 附近的根。

算法描述

给定 $f(x)$，从 x_0 开始，根据牛顿迭代公式

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0,1,\cdots$$

计算 $f(x)$在 x_0 附近的根。

程序源码

```
////////////////////////////////////////////////////////
//        Purpose: 牛顿迭代法求根                      //
////////////////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
                    // iterate(x)为 x - f(x)/f'(x)
# define iterate(x)     (x - ((x * x - 3) * x - 1)/(3 * x * x - 3))
# define x0   1.5        //迭代初值 x0
# define MAXREPT 1000       //最大迭代次数
# define epsilon 0.0001       //精度
void main()
{       int i;
        double x_k = x0, x_k1 = x0;
        for (i = 0; i < MAXREPT; i++)
```

```
        {
            printf("Got... %f\n", x_k1);
            x_k1 = iterate(x_k);          //迭代
            if (x_k1 - x_k < epsilon && x_k1 - x_k > - epsilon)
            {
                printf("! Root: %f\n", x_k1);  //满足精度,输出
                return;
            }
            x_k = x_k1;
        }
    printf("After %d repeate, no solved.\n", MAXREPT);
    }
//------------------------------ End of File ------------------------------
```

计算实例

已知 $f(x)=x^3-3x-1$,取 $\varepsilon=10^{-4}$,$x_0=1.5$,用牛顿迭代法计算 $f(x)$的根。

程序输入输出

由本程序预设的 $f(x)$及 $x_0=1.5$,得到输出

! Root: 1.879385

程序 8 用弦截法求 $f(x)$在 x_0,x_1 附近的根。

算法描述

给定 $f(x)$,从 x_0,x_1 开始,根据弦截法迭代公式

$$x_{k+1}=x_k-\frac{f(x_k)(x_k-x_{k-1})}{f(x_k)-f(x_{k-1})},k=1,2,\cdots$$

求得 $f(x)$在其附近的根。

程序源码

```
////////////////////////////////////////////////////////////////
//        Purpose: 弦截法求根                                   //
////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <math.h>
#define f(x) (x*x*x-7.7*x*x+19.2*x-15.3)  //f函数
#define x0 0.0                            //初值 x0, x1
#define x1 1.0
#define MAXREPT 1000                      //最大迭代次数
#define epsilon 0.00001                   //求解精度
void main()
{   int i;
    double x_k = x0, x_k1 = x1, x_k2 = x1;
    for (i=0; i<MAXREPT; i++)
    {
```

```
        printf("Got... %f \n", x_k2);
        x_k2 = x_k1 - (f(x_k1) * (x_k1 - x_k))/(f(x_k1) - f(x_k));   //弦截求新 x_n
        if (x_k2 - x_k1 < epsilon && x_k2 - x_k1 > - epsilon)
    {
            printf("! Root: %f \n", x_k2);      //满足精度，输出
            return;
        }
        x_k = x_k1; x_k1 = x_k2;          //反复
    }
    printf("After %d repeate, no solved. \n", MAXREPT);
}    //------------------------------ End of File ------------------------------
```

计算实例

用弦截法求方程 $f(x) = x^3 - 7.7x^2 + 19.2x - 15.3$ 的根，取 $x_0 = 1.5, x_1 = 4.0$。

程序输入输出

由本程序的 $f(x)$及 x_0, x_1，得到输出

! Root: 1.700000

习题 4

4.1 $f(x) = x - \frac{1}{2} - \sin x$，作等价形式 $x = \varphi(x) = \frac{1}{2} + \sin x$，用迭代 $x_{k+1} = \varphi(x_k)$，求解 $f(x)$在[1,2]的根，计算到$|x_{k+1} - x_k| < 10^{-3}$时停止。

4.2 方程 $2x^3 - 5x^2 - 19x + 42 = 0$，在 $x = 3.0$ 附近有根，写出该方程的三种不同的等价形式，并判断迭代格式在 $x = 3.0$ 的收敛性。

4.3 已知 $f(x) = x^3 + x^2 - 1$，取 $\varepsilon = 10^{-3}$，用二分法求 $f(x)$在[0,1]上的根。

4.4 正实数 α 的平方根为$f(x) = x^2 - \alpha$ 的零点，用牛顿迭代法

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{\alpha}{x_k}\right)$$

计算 $\sqrt{11}$ 的平方根，取 $x_0 = 4$，误差控制量 $\varepsilon = 10^{-4}$。

4.5 用牛顿迭代法计算$\sqrt[5]{9}$，取 $x_0 = 2$，计算到$|x_{k+1} - x_k| < 10^{-2}$时停止。

4.6 已知 $f(x) = x^3 - 3x - 2$，取 $\varepsilon = 10^{-3}$，$x_0 = 1.5$，用牛顿迭代法计算 $f(x)$的根。

4.7 已知 $f(x) = x^3 - 3x - 2$，取 $\varepsilon = 10^{-3}$，$x_0 = 1$，$x_1 = 3$，用弦截法计算 $f(x)$的根。

4.8 用牛顿迭代法求解非线性方程组

$$\begin{cases} x^2 + y^2 - 1 = 0 \\ x^3 - y = 0 \end{cases}$$

取 $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0.8 \\ 0.6 \end{pmatrix}$，误差控制量 $\| \Delta W \| < 10^{-3}$。

第 5 章　解线性方程组的直接法

在数值计算中，许多问题的关键一步是求解一个 n 元线性代数方程。在数值计算和工程应用中，求解线性代数方程都具有中心的作用。例如，样条插值中形成的 M 关系式，曲线拟合形成的法方程等，都落实到解一个 n 元线性方程组，即求线性方程组(5.1)的未知量 $x_1, x_2, \cdots, x_n$ 的数值，

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{5.1}$$

其中 a_{ij}, b_i 为常数。上式可写成矩阵形式 $Ax = b$，即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \cdots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{5.2}$$

其中，$A = (a_{ij})$ 为系数矩阵，$x = (x_1, x_2, \cdots, x_n)^T$ 为解向量，$b = (b_1, b_2, \cdots, b_n)^T$ 为常数向量。当 $\det A = D \neq 0$ 时，由线性代数中的克莱姆法则，方程组 $Ax = b$ 的解存在且唯一，其中

$$x_i = D_i / D, \qquad i = 1, 2, \cdots, n$$

D_i 为系数矩阵 A 的第 i 列元素以 b 代替的矩阵的行列式的值。克莱姆法则在建立线性方程组解的理论基础中功不可没，但是在实际计算中，我们难以承受它的计算量。例如，解一个 100 阶的行列式，乘除法次数约为 $(101 \cdot 100^2 \cdot 99!)$，即使以每秒 10^{33} 的运算速度，也需要近 10^{120} 年的时间。在石油勘探、天气预报等问题中常常出现成百上千阶的方程组，也就产生了各种形式方程组数值解法的需求。研究大型方程组的解是目前计算数学中的一个重要方向和课题。

解方程组的方法可归纳为直接解法和迭代解法。从理论上来说，直接法经过有限次四则运算，假定每一步运算过程中没有舍入误差，那么，最后得到方程组的解就是精确解。但是，在计算过程中，完全杜绝舍入误差是不可能的，只能控制和约束舍入误差的增长和危害，这样直接法得到的解也不一定是绝对精确的。

迭代法是将方程组的解看作某种极限过程的向量极限的值，像第 4 章中非线性方程求解一样，极限过程是用迭代过程完成的，只不过将迭代式中单变量 $x_{k+1} = \varphi(x_k)$ 换成向量而已。在用迭代算法时，我们不可能将极限过程算到底，只能将迭代进行有限多次，得到满足一定精度要求的方程组的近似解。

在数值计算历史上，直接解法和迭代解法交替生辉。一种解法的兴旺与计算机的硬件环境和问题规模是密切相关的。一般说来，对同等规模的线性方程组，直接法对计算机

的要求高于迭代法。对于中等规模的线性方程组($n<200$),由于直接法的准确性和可靠性高,一般都用直接法求解。对于高阶方程组和稀疏方程组(非零元素较少),一般用迭代法求解。

5.1 消 元 法

5.1.1 三角形方程组的解

形如下面三种形式的线性方程组较容易求解。

对角形方程组

$$\begin{cases} a_{11}x_1 & & & = b_1 \\ & a_{22}x_2 & & = b_2 \\ & & \cdots & \\ & & a_{nn}x_n & = b_n \end{cases} \tag{5.3}$$

设 $a_{ii}\neq 0$,对每一个方程,$x_i = b_i/a_{ii}$,$i=1,2,\cdots,n$。

显然,求解 n 阶对角方程的运算量为 $O(n)$。

下三角方程组

$$\begin{cases} l_{11}x_1 & & & = b_1 \\ l_{21}x_1 + l_{22}x_2 & & & = b_2 \\ & & \cdots & \\ l_{n1}x_1 + l_{n2}x_2 + \cdots + l_{nn}x_n & & & = b_n \end{cases} \tag{5.4}$$

按照方程组的顺序,从第一个方程至第 n 个方程,逐个解出 x_i,$i=1,2,\cdots,n$。

由方程 $l_{11}x_1 = b_1$,得 $x_1 = b_1/l_{11}$。将 x_1 的值代入到第二个方程

$$l_{21}x_1 + l_{22}x_2 = b_2$$

得

$$x_2 = (b_2 - l_{21}x_1)/l_{22}$$

将 $x_1, x_2, \cdots, x_{i-1}$ 的值代入到第 i 个方程

$$l_{i1}x_1 + \cdots + l_{ii}x_i = b_i$$

得

$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right)\Big/ l_{ii}, \quad i = 1,2,\cdots,n$$

计算 x_i 需要 i 次乘法或除法运算,$i=1,2,\cdots,n$。因此,求解过程中的运算量为

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \approx O(n^2)$$

上三角方程组

$$\begin{cases} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n} & = b_1 \\ u_{22}x_2 + \cdots + u_{2n}x_n & = b_2 \\ \cdots & \\ u_{nn}x_n & = b_n \end{cases} \tag{5.5}$$

与计算下三角方程组的次序相反,从第 n 个方程至第一个方程,逐个解出 x_i,$i=n, n-1,\cdots,2,1$。

由第 n 个方程 $u_{nn}x_n = b_n$，得 $x_n = b_n/u_{nn}$。将 x_n 的值代入到第 $n-1$ 个方程

$$u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = b_{n-1}$$

得
$$x_{n-1} = (b_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1}$$

将 $x_n, x_{n-1}, \cdots, x_{i+1}$ 的值代入到第 i 个方程

$$u_{ii}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{in}x_n = b_i$$

得解的通式

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii}, \qquad i = n, n-1, \cdots, 2, 1$$

计算 x_i 需要 $n-i+1$ 次乘法或除法运算，$i = n, n-1, \cdots, 2, 1$。因此，求解过程中的运算量为

$$\sum_{i=n}^{1}(n-i+1) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = O(n^2)$$

消元法的基本思想就是通过对方程组做初等变换，把一般形式的方程组化为等价的具有上述形式的易解方程组。

5.1.2 高斯(Gauss)消元法与列主元消元法

高斯消元法

高斯消元法是我们熟悉的古老、简单而有效的解方程组的方法。下面是中学阶段解二元方程组(高斯消元法)的步骤：

$$\begin{cases} x_1 - x_2 = 1 & ① \\ 3x_1 + 2x_2 = 8 & ② \end{cases}$$

方程①乘以 -3 加到第②个方程中得

$$\begin{cases} x_1 - x_2 = 1 \\ 0 + 5x_2 = 5 \end{cases}$$

$\therefore x_2 = 1$，代入①得 $x_1 = 2$。

其作用相当于对方程组的增广矩阵做行的初等变换：

$$(A, b) = \begin{pmatrix} 1 & -1 & 1 \\ 3 & 2 & 8 \end{pmatrix} \xrightarrow{-3\cdot①+②} \begin{pmatrix} 1 & -1 & 1 \\ 0 & 5 & 5 \end{pmatrix} = (\tilde{A}, \tilde{b})$$

$\tilde{A}$ 已是上三角矩阵，而

$$\begin{cases} x_1 - x_2 = 1 \\ 5x_2 = 5 \end{cases}$$

为原方程组的等价方程组，已化成易解的方程组形式。再用回代方法求解，得到：$x_2 = 1$，$x_1 = 2$。

这就是高斯消元法解方程组的消元和回代过程。

一般地，可对线性方程组(5.1)施行以下一系列变换：

(1) 对换某两个方程的次序；

(2) 对其中某个方程的两边同乘一个不为零的数；

(3) 把某一个方程两边同乘一个常数后加到另一个方程的两边。

记变换后的方程组为：

$$\begin{cases} \tilde{a}_{11}x_1 + \tilde{a}_{12}x_2 + \cdots + \tilde{a}_{1n}x_n = \tilde{b}_1 \\ \tilde{a}_{21}x_1 + \tilde{a}_{22}x_2 + \cdots + \tilde{a}_{2n}x_n = \tilde{b}_2 \\ \cdots \\ \tilde{a}_{n1}x_1 + \tilde{a}_{n2}x_2 + \cdots + \tilde{a}_{nn}x_n = \tilde{b}_n \end{cases} \tag{5.6}$$

显然方程组(5.1)与(5.6)是等价方程组，或者说它们有相同的解。分别记方程组(5.1)与(5.6)的增广矩阵为：

$$(A,b) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \cdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{pmatrix}$$

$$(\tilde{A},\tilde{b}) = \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \cdots & \tilde{a}_{1n} & \tilde{b}_1 \\ \tilde{a}_{21} & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} & \tilde{b}_2 \\ & & \cdots & & \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \cdots & \tilde{a}_{nn} & \tilde{b}_n \end{pmatrix}$$

可以看出，$(\tilde{A},\tilde{b})$实际上是由(A,b)按一系列初等变换后得到的：

(1)对换(A,b)某两行元素；

(2)(A,b)中的某行乘以一个不为零的数；

(3)把(A,b)的某一行乘一个常数后加到另一行。

高斯消元法就是通过以上(3)的变换，把(A,b)化为等价的上三角形式。

下面我们以 $n=4$ 为例演示消元过程。

设方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases} \tag{5.7}$$

其增广矩阵为：

$$(A,b) = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & b_4 \end{pmatrix}$$

①若 $a_{11} \neq 0$，则将第一行乘以 $-a_{21}/a_{11}$ 加到第二行上；将第一行乘以 $-a_{31}/a_{11}$ 加到第三行上；将第一行乘以 $-a_{41}/a_{11}$ 加到第四行上，得到

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{33}^{(1)} & b_2^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} & b_3^{(1)} \\ 0 & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} & b_4^{(1)} \end{pmatrix} \tag{5.8}$$

即 $$A^{(1)}x = b^{(1)}$$

其中：$a_{ij}^{(1)} = a_{ij} - a_{1j}a_{i1}/a_{11}, i,j = 2,3,4$

$b_i^{(1)} = b_i - b_1 a_{i1}/a_{11}, i = 2,3,4$

②若 $a_{22}^{(1)} \neq 0$,则将第二行乘以 $-a_{32}^{(1)}/a_{22}^{(1)}$ 加到第三行上;将第二行乘以 $-a_{42}^{(1)}/a_{22}^{(1)}$ 加到第四行上,得到

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} & b_2^{(1)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} & b_3^{(2)} \\ 0 & 0 & a_{43}^{(2)} & a_{44}^{(2)} & b_4^{(2)} \end{pmatrix} \tag{5.9}$$

其中：$a_{ij}^{(2)} = a_{ij}^{(1)} - a_{2j}^{(1)} \cdot a_{i2}^{(1)}/a_{22}^{(1)}, i,j = 3,4$

$b_i^{(2)} = b_i^{(1)} - b_2^{(1)} a_{i2}^{(1)}/a_{22}^{(1)}, i = 3,4$

③若 $a_{33}^{(2)} \neq 0$,则将第三行乘以 $-a_{43}^{(2)}/a_{33}^{(2)}$ 加到第四行上,得到

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} & b_2^{(1)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} & b_3^{(2)} \\ 0 & 0 & 0 & a_{44}^{(3)} & b_4^{(3)} \end{pmatrix} = (\tilde{A}, \tilde{b}) \tag{5.10}$$

其中：$a_{44}^{(3)} = a_{44}^{(2)} - a_{34}^{(2)} \cdot a_{43}^{(2)}/a_{33}^{(2)}$

$b_4^{(3)} = b_4^{(2)} - b_3^{(2)} \cdot a_{43}^{(2)}/a_{33}^{(2)}$

$\tilde{A}$已是上三角矩阵,于是得到了与原方程等价的易解形式的方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 = b_2^{(1)} \\ a_{33}^{(2)}x_3 + a_{34}^{(2)}x_4 = b_3^{(2)} \\ a_{44}^{(3)}x_4 = b_4^{(3)} \end{cases} \tag{5.11}$$

再对方程组(5.11)依次回代解出 x_4, x_3, x_2, x_1。

从式(5.10)可以得到系数矩阵 A 的行列式的值为 $\tilde{A}$ 的对角元素的乘积。即

$$\det(A) = a_{11}a_{22}^{(1)}a_{33}^{(2)}a_{44}^{(3)}$$

这也正是在计算机上计算方阵 A 的行列式的常规方法。

要将上述解方程步骤推广到 n 阶方程组,只需将对控制量“4”的操作改成对控制量 n 的操作即可。

n 元方程组高斯消元法的步骤如下：

对于 $k = 1,2,\cdots,n-1$,约定 $a_{ij}^{(0)} = a_{ij}$,有

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - \left(a_{ik}^{(k-1)}/a_{kk}^{(k-1)}\right) * a_{kj}^{(k-1)} \quad k+1 \leqslant i \leqslant n, k+1 \leqslant j \leqslant n \\ b_i^{(k)} &= b_i^{(k-1)} - \left(a_{ik}^{(k-1)}/a_{kk}^{(k-1)}\right) * b_k^{(k-1)} \quad k+1 \leqslant i \leqslant n \end{aligned} \tag{5.12}$$

经过以上消元，我们已得到与 $Ax = b$ 等价的方程组 $\tilde{A}x = \tilde{b}$，其中 $\tilde{A}$ 已是一个上三角矩阵。为简单起见，仍记 $\tilde{A}$ 的元素为 a_{ij}，$\tilde{b}$ 的元素为 b_i，回代

$$x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right)/a_{ii}, \quad i = n, n-1, \cdots, 1 \tag{5.13}$$

即已得到原方程组的解。

高斯消元法算法

在算法中略去了变量、矩阵和向量的定义部分。在消元过程中，将 $a_{ij}^{(k)}$ 仍放在 $a_{ij}^{(k-1)}$ 元素的位置上。

1：输入：方程组阶数 n、方程组系数矩阵 A 和常数向量 b。

```
2:FOR k := 1 TO n-1            ! 消元过程
    {   FOR i: = k+1 TO n
        {   t := a_ik / a_kk
            FOR j: = k+1 TO n
              {  a_ij := a_ij - t * a_kj  }      ! ENDFORJ
            b_i := b_i - t * b_k
            }                                    ! ENDFORI
    }                                            ! ENDFORK
3:FOR i: = n TO 1          ! 回代求解 x_i = (b_i - Σ_{j=i+1}^{n} a_ij x_j) / a_ii
    {  s := b_i
       FOR j: = i+1 TO n DO s = s - a_ij * x_j
       x_i := s / a_ii }
```

4：输出方程组的解 $x_i, i = 1, 2, \cdots, n$。

高斯消元法的运算量

整个消元过程即式(5.12)的乘法和除法的运算量为

$$\sum_{k=1}^{n-1}(n-k)(n-k+1) + \sum_{k=1}^{n-1}(n-k) + \frac{n(n-1)}{2}$$

回代过程即式(5.13)的乘除运算量为

$$\frac{n(n+1)}{2}$$

故高斯消元法的运算量为

$$\frac{n^3}{3} + n^2 - \frac{1}{3}n = O(n^3) \tag{5.14}$$

高斯消元法的可行性

在上面的消元法中，未知量是按照在方程组中的自然顺序消去的，也叫顺序消元法。

在消元过程中假定对角元素 $a_{kk}^{(k-1)} \neq 0$，消元步骤才能顺利进行，由于顺序消元不改

变 A 的主子式值,故高斯消元法可行的充分必要条件为 A 的各阶主子式不为零。但是,实际上只要 $\det A \neq 0$,方程组 $Ax = b$ 就有解。故高斯消元法本身具有局限性。

另一方面,即使高斯消元法可行,如果 $|a_{kk}^{(k-1)}|$ 很小,在运算中用它作为除法的分母($a_{ij} \leftarrow a_{ij} - a_{ik} \cdot a_{kj}/a_{kk}$),会导致其它元素数量级的严重增长和舍入误差的扩散。这是高斯消元法的另一缺陷。

例 5.1:方程组
$$\begin{cases} 0.0003x_1 + 3.0000x_2 = 2.0001 & ① \\ 1.0000x_1 + 1.0000x_2 = 1.0000 & ② \end{cases}$$
的精确解为:$x_1 = \frac{1}{3}$,$x_2 = \frac{2}{3}$。在高斯消元法计算中取 5 位有效数字。

解:方程①$\times(-1)/0.0003$ + 方程②得:
$$\begin{cases} 0.0003x_1 + 3.0000x_2 = 2.0001 \\ 9999.0x_2 = 6666.0 \end{cases}$$
$\therefore x_2 = 0.6667$,代入方程①得 $x_1 = 0$。由此得到的解完全失真,如果交换两个方程的顺序,得到等价方程组
$$\begin{cases} 1.0000x_1 + 1.0000x_2 = 1.0000 \\ 0.0003x_1 + 3.0000x_2 = 2.0001 \end{cases}$$
经高斯消元后有
$$\begin{cases} 1.0000x_1 + 1.0000x_2 = 1.0000 \\ 2.9997x_2 = 1.9998 \end{cases}$$
$\therefore x_2 = 0.6667$, $x_1 = 0.3333$

由此可看到,在有些情况下,调换方程组的次序对方程组的解是有影响的,在消元法中抑制舍入误差的增长是十分重要的。

如果不调换方程组的次序,取 6 位有效数字计算方程组的解,得到
$$x_2 = 0.666667, x_1 = 0.33$$
取 9 位有效数字计算方程组的解,得到
$$x_2 = 0.666667, x_1 = 0.333333$$
由此可见有效数字的作用。

列主元消元法

如果在一列中选取按模最大的元素,将其调到主干方程位置再做消元,则称为列主元消元法。调换方程组的次序是依照在运算中做分母量的绝对值尽量的大,以减少舍入误差的影响。

用列主元方法可以克服高斯消元法的额外限制,只要方程组有解,列主元消元法就能畅通无阻地顺利求解,同时又提高了解的精确度。

更具体地,第一步在第一列元素中选出绝对值最大的元素 $\max\limits_{1 \le i \le n} \{|a_{i1}|\} = |a_{m1}|$,交换第一行和第 m 行的所有元素,再做化简 $a_{21}, a_{31}, \cdots, a_{n1}$ 为零的操作。

对于每个 $k, k = 1,2,\cdots,n-1$ 在实现消元前,选出 $\{|a_{kk}^{(k-1)}|, |a_{k+1,k}^{(k-1)}|, \cdots, |a_{nk}^{(k-1)}|\}$ 中绝对值最大的元素 $|a_{m,k}^{(k-1)}|$,对 k 行和 m 行交换后,再做消元操作,这就是列主元消元法的操作步骤。由于 $\det A \neq 0$,可证 $a_{kk}^{(k-1)}, a_{k+1,k}^{(k-1)}, \cdots, a_{nk}^{(k-1)}$ 中至少有一个元素不为零,因此,列

主元消元总是畅通无阻的。列主元消元法与高斯消元法相比，只增加了选列主元和交换两个方程组(即两行元素)的过程。

列主元消元法算法

1:输入:方程组阶数 n、方程组系数矩阵 A 和常数向量项 b。

2:FOR k: = 1 TO n - 1　　! 选主元的消元过程

{ ! 选择 $\max\{|a_{k,k}|,|a_{k+1,k}|,\cdots,|a_{n,k}|\}=|a_{m,k}|$

{ $s=|a_{kk}|, m=k$

FOR u: = k + 1 TO n

IF $|a_{uk}|>s$ THEN $\{m=u; s=|a_{uk}|\}$

! 交换第 k 行和第 m 行

FOR v: = k TO n

{{ $t:=a_{kv}; a_{kv}:=a_{mv}; a_{mv}:=t$ |

$t:=b_k; b_k:=b_m; b_m:=t$

|

FOR i: = k + 1 TO n

| $t:=a_{ik}/a_{kk}$

FOR j: = k + 1, n

{ $a_{ij}:=a_{ij}-t\cdot a_{kj}$ }　　! ENDFORJ

$b_i:=b_i-t\cdot b_k$

|　　! ENDFORI

}　　! ENDFORK

3:FOR i = n TO 1　　! 回代求解

$$x_i:=\left(b_i-\sum_{j=i+1}^{n}u_{ij}x_j\right)\Big/u_{ii}$$

4:输出方程组的解 $x_i, i=1,2,\cdots,n$。

如果对于第 k 步，从 k 行至 n 行和从 k 列至 n 列中选取按模最大的 $|a_{ij}|, k\le i\le n, k\le j\le n$，记 $|a_{iw}^{(k-1)}|=\max\limits_{k\le i\le j\le n}\{|a_{ij}^{(k-1)}|\}$，对第 k 行和第 u 行交换，对第 k 列和第 v 列交换，这就是全主元消元法。在 k 列和第 v 列交换时，还要记录下 v 的序号，以便恢复未知量 x_k 和 x_v 的位置。

5.1.3 Guass-Jordan 消元法

高斯消元法将系数矩阵化为上三角矩阵，再进行回代求解；Guass-Jordan 消元法是将系数矩阵化为对角矩阵，再进行求解，即

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ & & a_{33}^{(2)} & a_{34}^{(2)} \\ & & & a_{44}^{(3)} \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(3)} \end{pmatrix}$$

对上式[即式(5.11)],用第4行乘 $-a_{34}^{(2)}/a_{44}^{(3)}$ 加到第3行上,用第4行乘 $-a_{24}^{(1)}/a_{44}^{(3)}$ 加到第2行上,用第4行乘 $-a_{14}/a_{44}^{(3)}$ 加到第1行上,得到

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ & a_{22}^{(1)} & a_{23}^{(1)} & 0 \\ & & a_{33}^{(2)} & 0 \\ & & & a_{44}^{(3)} \end{pmatrix} x = \begin{pmatrix} b_1^{(4)} \\ b_2^{(4)} \\ b_3^{(4)} \\ b_4^{(3)} \end{pmatrix}$$

用第3行乘 $-a_{23}^{(1)}/a_{33}^{(2)}$ 加到第2行上,用第3行乘 $-a_{13}/a_{33}^{(2)}$ 加到第1行上,再用第2行乘 $-a_{12}/a_{22}^{(1)}$ 加到第1行上,得到

$$\begin{pmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22}^{(1)} & 0 & 0 \\ 0 & 0 & a_{33}^{(2)} & 0 \\ 0 & 0 & 0 & a_{44}^{(3)} \end{pmatrix} x = \begin{pmatrix} b_1^{(6)} \\ b_2^{(5)} \\ b_3^{(4)} \\ b_4^{(3)} \end{pmatrix} \tag{5.15}$$

为方便起见,仍记式(5.15)系数矩阵元素为 a_{ij},常数项元素为 b_i。那么

$$x_i = b_i/a_{ii}, i = 1,2,\cdots,n$$

用初等变换化系数矩阵为对角矩阵的方法称为 Gauss-Jordan 消元法。从形式上看对角矩阵(5.15)比上三角矩阵(5.11)更为简单,易于计算 x_i。但是将上三角矩阵(5.11)化为对角矩阵(5.15)的工作量$\left(\frac{n^3}{3}\right)$大于上三角矩阵回代的工作量$\left(\frac{n^2}{2}\right)$。所以,用回代解方程组比 Gauss-Jordan 继续消元更适用。

5.2 直接分解法

仍以 $n=4$ 为例,在高斯消元法中,从对方程组进行初等变换的角度观察方程组系数矩阵的演变过程。第一次消元步骤将方程组(5.7)化为方程组 (5.8),相当于用了三个初等矩阵左乘 A 和 b。记 $l_{i1} = a_{i1}/a_{11}, i=2,3,4$,容易验证,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -l_{41} & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -l_{31} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -l_{21} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ -l_{21} & 1 & 0 & 0 \\ -l_{31} & 0 & 1 & 0 \\ -l_{41} & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$= T_1 A = A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{33}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ 0 & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{pmatrix}$$

由 $T_1 Ax = T_1 b$,得到 $A^{(1)} x = b^{(1)}$,其中

$$T_1 b = b^{(1)} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{pmatrix}$$

将方程组 (5.8)化为方程组 (5.9),相当于用了两个初等矩阵左乘 A 和 b。

记 $l_{i2} = a_{i2}^{(1)} / a_{22}^{(1)}$, $i = 3,4$,有

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -l_{42} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -l_{32} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot A^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -l_{32} & 1 & 0 \\ 0 & -l_{42} & 0 & 1 \end{pmatrix} \cdot A^{(1)}$$

$$= T_2 A^{(1)} = A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ & & a_{33}^{(2)} & a_{34}^{(2)} \\ & & a_{43}^{(2)} & a_{44}^{(2)} \end{pmatrix}$$

$$T_2 b^{(1)} = b^{(2)} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(2)} \end{pmatrix}$$

同理,将方程组 (5.9)化为方程组 (5.10),相当于用一个初等矩阵左乘 A 和 b。

记 $l_{43} = -a_{43}^{(2)} / a_{33}^{(2)}$,有

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -l_{43} & 1 \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ & & a_{33}^{(2)} & a_{34}^{(2)} \\ & & a_{43}^{(2)} & a_{44}^{(2)} \end{pmatrix} = T_3 A^{(2)}$$

$$= A^{(3)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ & & a_{33}^{(2)} & a_{34}^{(2)} \\ & & & a_{44}^{(3)} \end{pmatrix} = \tilde{A}$$

$$T_3 b^{(2)} = b^{(3)} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(3)} \end{pmatrix}$$

完成了消元过程,有

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ & & a_{33}^{(2)} & a_{34}^{(2)} \\ & & & a_{44}^{(3)} \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(3)} \end{pmatrix}$$

亦即所有的消元步骤表示为:(A,b)左乘一系列下三角初等矩阵;容易验证,这些下三角矩阵的乘积仍为下三角矩阵,即

$$T = T_3 T_2 T_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -l_{21} & 1 & 0 & 0 \\ -l_{31} & -l_{32} & 1 & 0 \\ -l_{41} & -l_{42} & -l_{43} & 1 \end{pmatrix}$$

$$T_3 T_2 T_1 A = \widetilde{A}$$

于是有

$$\widetilde{A} = TA$$

或

$$A = T^{-1} \widetilde{A}$$

这里 T^{-1}仍为下三角矩阵,其对角元素为 1,称为单位下三角矩阵,而$\widetilde{A}$已是上三角矩阵。

记 $T^{-1} = L, \widetilde{A} = U$,则有

$$A = LU$$

结果表明若消元过程可行,可以将 A 分解为单位下三角矩阵 L 与上三角矩阵 U 的乘积。由此派生出解方程组的直接分解法。

由高斯消元法得到启发,对 A 消元的过程相当于将 A 分解为一个上三角矩阵和一个下三角矩阵的过程。如果直接分解 A 得到 L 和 U,$A = LU$。这时方程 $Ax = b$ 化为 $LUx = b$,令 $Ux = y$,由 $Ly = b$ 解出 y;再由 $Lx = y$,解出 x。这就是直接分解法。

将方阵 A 分解为 $A = LU$,当 L 是单位下三角矩阵,U 是上三角矩阵时,称为 Doolittle 分解;当 L 是下三角矩阵,U 是单位上三角矩阵时,称为 Courant 分解。分解的条件是若方阵 A 的各阶主子式不为零,则 Doolittle 分解或 Courant 分解是可行的。

5.2.1 Doolittle 分解

Doolittle 分解步骤

若 A 的各阶主子式不为零,A 可分解为 $A = LU$,其中 L 为单位下三角矩阵,U 为上三角矩阵,即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \cdots & \cdots & \cdots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \cdots & \cdots \\ & & & u_{nn} \end{pmatrix} \tag{5.16}$$

矩阵 L 和 U 共有 n^2 个未知元素，按照一定的顺序，对每个 a_{ij} 列出式(5.16)两边对应的元素关系式，一个关系式解出一个 L 或 U 的元素。按下列步骤计算 L 和 U 的元素：

(1) 计算 U 的第一行元素 $u_{11}, u_{12}, \cdots, u_{1n}$

要计算 u_{11}，则列出式(5.16)等号两边的第 1 行第 1 列元素的关系式：

$$a_{11} = \sum_{r=1}^{n} l_{1r}u_{r1} = (1 \quad 0 \quad \cdots \quad \cdots \quad 0)\begin{pmatrix} u_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = u_{11}$$

故 $u_{11} = a_{11}$。一般地，由 U 的第一行元素的关系式

$$a_{1j} = \sum_{r=1}^{n} l_{1r}u_{rj} = (1 \quad 0 \quad \cdots \quad \cdots \quad 0)\begin{pmatrix} u_{1j} \\ u_{2j} \\ \vdots \\ 0 \end{pmatrix} = u_{1j}$$

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

(2) 计算 L 的第一列元素 $l_{21}, l_{31}, \cdots, l_{n1}$

要计算 l_{21}，则列出式(5.16)等号两边的第 2 行第 1 列元素的关系式：

$$a_{21} = \sum_{r=1}^{n} l_{2r}u_{r1} = (l_{21} \quad 1 \quad 0 \quad \cdots \quad 0)\begin{pmatrix} u_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = l_{21}u_{11}$$

故 $l_{21} = a_{21}/u_{11}$。一般地，由 L 的第 1 列元素的关系式

$$a_{i1} = \sum_{r=1}^{n} l_{ir}u_{r1} = (l_{i1}\cdots l_{i,i-1} \quad 1 \quad 0 \quad \cdots \quad 0)\begin{pmatrix} u_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = l_{i1}u_{11}$$

得到 $l_{i1} = a_{i1}/u_{11}, i = 2,3,\cdots,n$。

(3) 计算 U 的第 2 行元素

(4) 计算 L 的第 2 列元素

……

若已算出 U 的前 $k-1$ 行，L 的 $k-1$ 列元素，则

(5) 计算 U 的第 k 行元素 $u_{kk}, u_{k,k+1}, \cdots, u_{k,n}$

由 U 的第 k 行元素的关系式：

$$a_{kj} = \sum_{r=1}^{n} l_{kr}u_{rj} = (l_{k1} \quad l_{k2} \quad \cdots \quad l_{k,k-1} \quad 1 \quad 0 \cdots 0)\begin{pmatrix} u_{1j} \\ \vdots \\ u_{jj} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

∵ U 是上三角矩阵，∴ 列标 $j \geq$ 行标 k。

$$a_{kj} = \sum_{r=1}^{n} l_{kr}u_{rj} = \sum_{r=1}^{k} l_{kr}u_{rj} = \sum_{r=1}^{k-1} l_{kr}u_{rj} + u_{kj}$$

得到

$$u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}, j = k, k+1, \cdots, n \tag{5.17}$$

(6) 计算 L 的第 k 列元素 $l_{k+1,k}, l_{k+2,k}, \cdots, l_{n,k}$

由 L 的第 k 列元素的关系式：

$$a_{ik} = \sum_{r=1}^{n} l_{ir}u_{ik} = (l_{i1}, \cdots, l_{i,i-1}, 1, 0, \cdots, 0)\begin{pmatrix} u_{1k} \\ \vdots \\ u_{kk} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

∵ L 是下三角矩阵，∴ 行标 $i >$ 列标 k。

$$a_{ik} = \sum_{r=1}^{n} l_{ir}u_{rk} = \sum_{r=1}^{k} l_{ir}u_{rk} = \sum_{r=1}^{k-1} l_{ir}u_{rk} + l_{ik}u_{kk}$$

得到

$$l_{ik} = \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk}\right) \Big/ u_{kk}, i = k+1, k+2, \cdots, n \tag{5.18}$$

一直做到 L 的第 $n-1$ 列，U 的第 n 行为止。

用 LU 直接分解方法求解方程组所需要的计算量仍为 $\frac{1}{3}n^3 + O(n^2)$，和用高斯消元法的计算量基本相同。

可以看到在分解中 A 的每个元素只在式(5.17)或(5.18)中做而且仅做一次贡献，如果需要节省空间，可将 U 以及 L 的元素直接放在矩阵 A 相应元素的位置上。

用直接分解法解方程 $Ax = b$，首先作出分解 $A = LU$，令 $Ux = y$，解方程组 $Ly = b$。由于 L 是单位下三角矩阵，容易得到

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij}x_j, i = 1, 2, \cdots, n \tag{5.19}$$

再解方程组 $Ux = y$，其中

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \Big/ u_{ii}, i = n, n-1, \cdots, 2, 1 \tag{5.20}$$

对 A 进行 LU 分解时，并不涉及常数项 b。因此，若需要解具有相同系数的一系列线性方程组时，使用直接分解法可以达到事半功倍的效果。

Doolittle 直接分解算法

1：输入：方程组阶数 n、系数矩阵 A 和常数项 b。

```
2: FOR k: = 1 TO n
       {    FOR j: = k TO n          ! 计算 U 的第 k 行元素
```

$$u_{kj} := a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}$$

FOR i: = k + 1 TO n　　! 计算 L 的第 k 列元素

$$l_{ik} := \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk}\right)/u_{kk}$$

}　　! 完成 *LU* 分解

3:FOR i: = 1 TO n　　! 解方程组 $Ly = b$

$$y_i := b_i - \sum_{j=1}^{i-1} l_{ij}y_j$$

4:FOR i: = n TO 1　　! 解方程组 $Ux = y$

$$x_i := \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii}$$

5:输出方程组的解 $x_i, i = 1,2,\cdots,n$。

例 5.2:用 Doolittle 分解求解方程组:

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ x_1 + 3x_2 + 2x_3 = 6 \\ x_1 + 2x_2 + 2x_3 = 5 \end{cases}$$

解:
$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix}\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

对 $k = 1$,有

$$u_{1j} = a_{1j}, j = 1,2,3$$

$$u_{11} = 2, u_{12} = 1, u_{13} = 1$$

$$l_{i1} = a_{i1}/u_{11}, i = 2,3$$

$$l_{21} = 1/2 = 0.5, l_{31} = 1/2 = 0.5$$

对 $k = 2$,有

$$u_{22} = a_{22} - l_{21}u_{12} = 3 - 0.5 = 2.5$$

$$u_{23} = a_{23} - l_{21}u_{13} = 2 - 0.5 = 1.5$$

$$l_{32} = (a_{32} - l_{31}u_{12})/u_{22} = (2 - 0.5)/2.5 = 0.6$$

对 $k = 3$,有

$$u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} = 0.6$$

$$\therefore \quad LU = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.6 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & 2.5 & 1.5 \\ 0 & 0 & 0.6 \end{pmatrix}$$

解 $Ly = b$,即

$$\begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.6 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 5 \end{pmatrix}$$

$\therefore\ y_1=4, y_2=4, y_3=0.6$

解 $Ux=y$，即

$$\begin{pmatrix}2 & 1 & 1\\ 0 & 2.5 & 1.5\\ 0 & 0 & 0.6\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}=\begin{pmatrix}4\\ 4\\ 0.6\end{pmatrix}$$

$\therefore\ x_3=1, x_2=1, x_1=1$

5.2.2 Courant 分解

Courant 分解步骤

在矩阵 $A=LU$ 的 Courant 分解形式中，L 是下三角矩阵，U 是单位上三角矩阵，记

$$L=\begin{pmatrix}l_{11} & & & \\ l_{21} & l_{22} & & \\ \cdots & \cdots & \cdots & \\ l_{n1} & l_{n2} & \cdots & l_{nn}\end{pmatrix},\quad U=\begin{pmatrix}1 & u_{12} & \cdots & u_{1n}\\ & 1 & \cdots & u_{2n}\\ & & \cdots & \cdots\\ & & & 1\end{pmatrix} \tag{5.21}$$

矩阵 Courant 分解的次序与 Doolittle 分解的次序不同，设 A 的各阶主子式不为零，按计算 L 的第 1 列，U 的第 1 行，……，L 的第 k 列，U 的第 k 行的顺序进行计算。类似于 Doolittle 分解的手法，作出 $A=LU$ 两边矩阵元素的关系式 $a_{ij}=\sum_{r=1}^{n}l_{ir}u_{rj}$，可导出计算公式。对 $k=1,2,\cdots,n$，有

$$l_{ik}=a_{ik}-\sum_{r=1}^{k-1}l_{ir}u_{rk},\ i=k,k+1,\cdots,n \tag{5.22}$$

$$u_{kj}=\left(a_{kj}-\sum_{r=1}^{k-1}l_{kr}u_{rj}\right)\Big/l_{kk},\ j=k+1,k+2,\cdots,n \tag{5.23}$$

如果 A 的各阶主子式不为零，则式(5.23)中的 $l_{kk}\neq 0$，Courant 分解才能顺利进行。在线性代数中我们已知，系数矩阵 A 的行列式不为零，方程组的解存在而且唯一，并没有对主子式有任何要求。在直接分解中也能采用与列主元消元法类似的方法进行处理，避免 A 的某阶主子式为零或 $|l_{kk}|$ 很小。即每求一列 L 的元素后，求出该列中模最大的元素 l_{uk}，然后将矩阵 A 和 L 的第 k 行和第 u 行元素逐个交换，并交换常数项 b_k 与 b_u。

对于方程组 $Ax=b$，作 Courant 分解后，方程组可写成 $LUx=b$，仍记 $Ux=y$，由 $Ly=b$，得到

$$y_i=\left(b_i-\sum_{j=1}^{i-1}l_{ij}y_j\right)\Big/l_{ii},\ i=1,2,\cdots,n \tag{5.24}$$

由 $Ux=y$，得到

$$x_i=y_i-\sum_{j=i+1}^{n}u_{ij}x_j,\quad j=n,n-1,\cdots,2,1 \tag{5.25}$$

Courant 列主元直接分解算法

1：输入：方程组阶数 n、系数矩阵 A 和常数项 b。

2：FOR k: = 1 TO n

{ FOR i: = k TO n ！计算 L 的第 k 列元素

$$l_{ik} := a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk}$$

FOR j: = k + 1 TO n　　! 计算 U 的第 k 行元素

$$u_{kj} := \left(a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}\right)/l_{kk}$$

}　　! 完成 LU 分解

3:FOR i: = 1 TO n　　! 解方程组 $Ly = b$

$$y_i := \left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right)/l_{ii}$$

4:FOR i: = n TO 1　　! 解方程组 $Ux = y$

$$x_i := y_i - \sum_{j=i+1}^{n} u_{ij}x_j$$

5:输出方程组的解 $x_i, i = 1,2,\cdots,n$。

例 5.3:用 Courant 分解求解方程组:

$$\begin{pmatrix} 1 & 2 & 1 \\ -2 & -1 & -5 \\ 0 & -1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ -63 \\ 50 \end{pmatrix} \text{ 和 } \begin{pmatrix} 1 & 2 & 1 \\ -2 & -1 & -5 \\ 0 & -1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 11 \\ -1 \\ -17 \end{pmatrix}$$

解:
$$\begin{pmatrix} 1 & 2 & 1 \\ -2 & -1 & -5 \\ 0 & -1 & 6 \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix}\begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

$k = 1$,

计算 L 的第 1 列　$l_{i1} = a_{i1}, i = 1,2,3$

$l_{11} = 1, l_{21} = -2, l_{31} = 0$

计算 U 的第 1 行　$u_{12} = a_{12}/l_{11} = 2, u_{13} = a_{13}/l_{11} = 1$

$k = 2$,

计算 L 的第 2 列　$l_{22} = a_{22} - l_{21}u_{12} = 3$

$l_{32} = a_{32} - l_{31}u_{12} = -1$

计算 U 的第 2 行　$u_{23} = (a_{23} - l_{21}u_{13})/l_{22} = -1$

$k = 3$,

$$l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} = 5$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 3 & 0 \\ 0 & -1 & 5 \end{pmatrix}, U = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

分别解方程:$LY_1 = b_1, UX_1 = Y_1$ 和 $LY_2 = b_2, UX_2 = Y_2$,即

$$LY_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 3 & 0 \\ 0 & -1 & 5 \end{pmatrix}Y_1 = \begin{pmatrix} 24 \\ -63 \\ 50 \end{pmatrix}, \qquad Y_1 = \begin{pmatrix} 24 \\ -5 \\ 9 \end{pmatrix}$$

$$UX_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} X_1 = Y_1, \qquad X_1 = \begin{pmatrix} 7 \\ 4 \\ 9 \end{pmatrix}$$

$$LY_2 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 3 & 0 \\ 0 & -1 & 5 \end{pmatrix} Y_2 = \begin{pmatrix} 11 \\ -1 \\ -17 \end{pmatrix}, \qquad Y_2 = \begin{pmatrix} 11 \\ 7 \\ -2 \end{pmatrix}$$

$$UX_2 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} X_2 = Y_2, \qquad X_2 = \begin{pmatrix} 3 \\ 5 \\ -2 \end{pmatrix}$$

5.2.3 追赶法

形如

$$A = \begin{pmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{pmatrix} \tag{5.26}$$

的矩阵称为三对角线矩阵。其特征是非零元素仅分布在对角线及对角线两侧的位置。在样条插值函数的 M 关系式中就出现过这类矩阵。事实上，许多连续问题经离散化后得到的线性方程组，其系数矩阵就是三对角或五对角形式的对角矩阵。

利用矩阵直接分解法，求解具有三对角线系数矩阵的线性方程组十分简单而有效。采用 Courant 分解法，分解形如(5.26)的三对角线矩阵 $A = LU$，容易验证 L, U 具有形式

$$L = \begin{pmatrix} u_1 & & & \\ w_2 & u_2 & & \\ & \ddots & \ddots & \\ & & w_n & u_n \end{pmatrix}, \quad U = \begin{pmatrix} 1 & v_1 & & \\ & 1 & \ddots & \\ & & \ddots & v_{n-1} \\ & & & 1 \end{pmatrix}$$

比较 $A = L \cdot U$ 两边元素，可得到

$$w_i = c_i, i = 2,3,\cdots,n$$

$$u_1 = a_1, v_1 = b_1/u_1$$

$$u_2 = a_2 - c_2 v_1, v_2 = b_2/u_2$$

若规定 $c_1 = 0$，可得到一般计算公式

$$\begin{cases} u_i = a_i - c_i v_{i-1} \\ v_i = b_i/u_i \end{cases}, \quad i = 1,2,\cdots,n \tag{5.27}$$

考虑三对角线系数矩阵的线性方程组

$$Ax = f$$

这里 $x = (x_1, x_2, \cdots, x_n)^T, f = (f_1, f_2, \cdots, f_n)^T$。令 $Ux = y$，则有

$$Ly = f$$

于是有

$$y_i = (f_i - c_i y_{i-1})/u_i, \quad i = 1,2,\cdots,n \tag{5.28}$$

规定 $v_n = 0$,由 $Ux = y$ 可得到

$$x_i = y_i - v_i x_{i+1}, i = n, n-1, \cdots, 2, 1$$

事实上,由式(5.27),每计算出一个 u_i,就可按式(5.28)计算出 y_i。若规定 $c_1 = 0$,由式(5.27)和式(5.28)可合并成

$$\begin{cases} u_i = a_i - c_i v_{i-1} \\ v_i = b_i / u_i \qquad\qquad i = 1,2,\cdots,n \\ y_i = (f_i - c_i y_{i-1})/u_i \end{cases} \tag{5.29}$$

$$x_k = y_k - v_k x_{k+1}, \quad k = n, n-1, \cdots, 2, 1 \tag{5.30}$$

计算公式(5.29)和(5.30)称为追赶法。

5.2.4 对称矩阵的 LDL^T 分解

对称正定矩阵也是很多物理问题产生的一类矩阵,正定矩阵的各阶主子式大于零。由线性代数中的定理,若 A 正定,则存在下三角矩阵 U,使 $A = UU^T$,直接分解 $A = UU^T$ 的分解方法,称为平方根法。由于在平方根法中含有计算平方根的步骤,因此很少采用平方根法的分解形式。对于对称矩阵,常用的是 LDL^T 分解。

对 A 作 Doolittle 分解 $A = LU$,即

$$A = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \cdots & \cdots & \cdots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \cdots & \cdots \\ & & & u_{nn} \end{pmatrix} \quad \text{/ 提出 } U \text{ 矩阵的对角元素 /}$$

$$= \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ & \cdots & & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & & & \\ & u_{22} & & \\ & & \ddots & \\ & & & u_{nn} \end{pmatrix} \begin{pmatrix} 1 & \overline{u}_{12} & \cdots & \overline{u}_{1n} \\ & 1 & \cdots & \overline{u}_{2n} \\ & & \ddots & \\ & & & 1 \end{pmatrix}$$

由 A 对称正定,可得 $u_{ii} > 0$,令

$$D = \begin{pmatrix} u_{11} & & & \\ & u_{22} & & \\ & & \ddots & \\ & & & u_{nn} \end{pmatrix} = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

可证

$$\begin{pmatrix} 1 & \overline{u}_{12} & \cdots & \overline{u}_{1n} \\ & 1 & \cdots & \overline{u}_{2n} \\ & & \ddots & \\ & & & 1 \end{pmatrix} = L^T$$

即 $A = LDL^T$。

$$A = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ & \cdots & & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} \begin{pmatrix} 1 & l_{12} & \cdots & l_{1n} \\ & 1 & \cdots & 1_{2n} \\ & & \ddots & \\ & & & 1 \end{pmatrix} \tag{5.31}$$

L 是对角元素为 1 的单位下三角矩阵。

对矩阵 A 作 Doolittle 或 Courant 分解，共计算 n^2 个矩阵元素；对称矩阵的 LDL^T 分解，只需计算 $\frac{n(n-1)}{2}$ 个元素，减少了近一半的工作量。借助于 Doolittle 或 Courant 分解计算公式，容易得到 LDL^T 分解计算公式。

设 A 有 Doolittle 分解形式：

$$A = L(DL^T) = L\widetilde{U} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} d_1 & d_1 l_{12} & \cdots & d_1 l_{1n} \\ & d_2 & \cdots & d_2 l_{2n} \\ & & \ddots & \vdots \\ & & & d_n \end{pmatrix}$$

其中 $\widetilde{u}_{ij} = d_i l_{ij} = d_i l_{ji}$。

在分解中可套用 Doolittle 分解公式，只要计算下三角矩阵 L 和 D 的对角元素 d_k。计算中只需保存 $L=(l_{ij})$ 的元素，L^T 的 i 行 j 列的元素用 L 的 l_{ji} 表示。由于对称正定矩阵的各阶主子式大于零，直接调用 Doolittle 或 Courant 分解公式可完成 LDL^T 分解计算，而不必借助于列主元的分解算法。

对于 $k=1,2,\cdots,n$，有

$$d_k = \widetilde{u}_{kk} = a_{kk} - \sum_{r=1}^{k-1} l_{kr}\widetilde{u}_{rk} = a_{kk} - \sum_{r=1}^{k-1} l_{kr} d_r l_{rk}$$

$$d_k = a_{kk} - \sum_{r=1}^{k-1} d_r l_{kr}^2 \tag{5.32}$$

$$l_{ik} = \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir}\widetilde{u}_{rk}\right)\Big/\widetilde{u}_{kk} = \left(a_{ik} - \sum_{r=1}^{k-1} d_r l_{ir} l_{kr}\right)\Big/d_k$$

$$l_{ik} = \left(a_{ik} - \sum_{r=1}^{k-1} d_r l_{ir} l_{kr}\right)\Big/d_k, \quad i = k+1,\cdots,n \tag{5.33}$$

对称矩阵的 LDL^T 分解算法

1：输入：方程组阶数 n、系数矩阵 A 和常数项 b。

2：FOR k：= 1 TO n

$$d_k := a_{kk} - \sum_{r=1}^{k-1} d_r l_{kr}^2$$

FOR i：= k + 1 TO n

$$l_{ik} := \left(a_{ik} - \sum_{r=1}^{k-1} d_r l_{ir} l_{kr}\right)\Big/d_k$$

3：略去解方程组步骤。

由 $L(DL^T)x = b$，解方程组 $Ax = b$ 可分三步完成：

(1) 解方程组 $Lz = b$，其中 $z = DL^T x$。

$$z_i = b_i - \sum_{j=1}^{i-1} l_{ij} z_j, \qquad i = 1,2,\cdots,n \tag{5.34}$$

(2) 解方程组 $Dy = z$,其中 $y = L^T x$。

$$y_i = z_i / d_i, \qquad i = 1,2,\cdots,n \tag{5.35}$$

(3)解方程组 $L^T x = y$。

$$x_i = y_i - \sum_{j=i+1}^{n} l_{ji} x_j, \quad i = n, n-1,\cdots,1 \tag{5.36}$$

从矩阵分解角度看,直接分解法与消元法本质上没有多大区别,但实际计算时它们各有所长。一般来说,如果仅用单字长进行计算,列主元消元法具有运算量较少、精度高的优点,故是常用的方法。但是,为了提高精度往往采取单字长数双倍内积的办法(即作向量内积计算时,采用双倍位加法,最终结果再舍入成单字长数),这时直接分解法能获得较高的精度。

例 5.4:用 LDL^T 分解求解方程组:

$$\begin{pmatrix} 1 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 4.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -8 \\ 12 \end{pmatrix}$$

解: $k = 1$ 时, $d_1 = a_{11} = 1$

$$l_{21} = a_{21}/d_1 = -1$$

$$l_{31} = a_{31}/d_1 = 1$$

$k = 2$ 时, $d_2 = a_{22} - l_{21}^2 d_1 = 2$

$$l_{32} = (a_{32} - l_{31} l_{21} d_1)/d_2 = -0.5$$

$k = 3$ 时, $d_3 = a_{33} - l_{31}^2 d_1 - l_{32}^2 d_2 = 3$

$$\therefore L = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -0.5 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & & \\ & 2 & \\ & & 3 \end{pmatrix}$$

由 $LDL^T x = b$,有

$$Lz = b, z = (4, -4, 6)^T$$

$$Dy = z, y = (4, -2, 2)^T$$

$$L^T x = y, x = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

5.3* 矩阵的条件数

在解方程组时,我们总是假定系数矩阵 A 和常数项 b 是准确的,而在实际问题中,系数矩阵 A 和常数项 b 往往是从前面的近似计算所得,元素的误差是不可避免的。这些误差会对方程组 $Ax = b$ 的解 x 产生多大的影响? 矩阵的条件数将给出一种粗略的衡量尺度。

定义：若 A 非奇异，称 $\mathrm{Cond}_p(A) = \| A \|_p \| A^{-1} \|_p$ 为 A 的条件数。其中 $\| \cdot \|_p$ 表示矩阵的某种范数。

用矩阵 A 及其逆矩阵 A^{-1} 的范数的乘积表示矩阵的条件数，由于矩阵范数的定义不同，因而其条件数也不同，但是由于矩阵范数的等价性，故在不同范数下的条件数也是等价的。矩阵条件数的大小是衡量矩阵“好”或“坏”的标志。

对于线性方程组 $Ax = b$，若系数矩阵有小扰动 δA，这时方程组的解也有扰动，于是 $(A + \delta A)(x + \delta x) = b$。$\delta A$ 对 δx 的影响可表示为：

$$\frac{\| \delta x \|}{\| x \|} \leq \frac{\mathrm{Cond}(A) \dfrac{\| \delta A \|}{\| A \|}}{1 - \mathrm{Cond}(A) \dfrac{\| \delta A \|}{\| A \|}} \tag{5.37}$$

若常数 b 有小扰动 δb，其对 δx 的影响表示为：

$$\frac{\| \delta x \|}{\| x \|} \leq \mathrm{Cond}(A) \frac{\| \delta b \|}{\| b \|} \tag{5.38}$$

因此，$\mathrm{Cond}(A)$ 大的矩阵称为“坏矩阵”或“病态矩阵”。对于 $\mathrm{Cond}(A)$ 大的矩阵，小的误差可能会引起解的失真。一般说来，若 A 的按模最大特征值与按模最小特征值之比值较大时，矩阵就会呈病态。特别地，当 $\det A$ 很小时，A 总是病态的。

例 5.5：方程组

$$\begin{cases} 0.2161x_1 + 0.1441x_2 = 0.1440 \\ 1.2969x_1 + 0.8648x_2 = 0.8642 \end{cases}$$

方程组的解为

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$$

对常数项 b 引入小扰动 $\delta b = (0.00000001, -0.00000001)^T$，则解为

$$\begin{pmatrix} x_1 + \delta x_1 \\ x_2 + \delta x_2 \end{pmatrix} = \begin{pmatrix} 0.991 \\ -0.4870 \end{pmatrix}$$

即 $\delta x = (-1.009, 1.5130)^T$。虽然 δb 很小，但解已完全失真。

$$\det A = -10^{-8}$$

$$A^{-1} = -10^{-8} \begin{pmatrix} 0.8648 & -0.1441 \\ -1.2969 & 0.2161 \end{pmatrix}$$

$$\| A \|_\infty = 2.1617, \| A^{-1} \|_\infty = 1.513 \times 10^8$$

故 $\mathrm{Cond}_\infty(A) = 3.2706521 \times 10^8$，$A$ 的病态是显而易见的。

5.4 程序示例

程序 9 用列主元高斯消元法求解方程组：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

算法描述

输入:矩阵 a 及列向量 b。

分解过程:对 k = 1,2,…,n - 1

{

选择 $\max\{|a_{k,k}|,|a_{k+1,k}|,\cdots,|a_{n,k}|\} = a_{m,k}$,交换第 k 行和第 m 行

对 i = k + 1,…,n;j = k + 1,…,n

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)}/a_{kk}^{(k)}) \times a_{kj}^{(k)}$$

对 i = k + 1,…,n

$$b_i^{(k+1)} = b_i^{(k)} - (a_{ik}^{(k)}/a_{kk}^{(k)}) \times b_k^{(k)}$$

}

回代过程:对 i = n,n - 1,…,1,解得

$$x_i = b_i^{(n)} - \sum_{j=i+1}^{n} a_{ij}^{(k)} x_j^{(k)}$$

程序源码

```
////////////////////////////////////////////////////////
//        Purpose: 列主元高斯消元法解  Ax = b          //
////////////////////////////////////////////////////////
# include < stdio.h >
# include < math.h >
# define MAX _ N 20          //(x _ i,y _ i)的最大维数
int main()
{
    int n;
    int i, j, k;
    int mi; double mx, tmp;
    static double a[MAX _ N][MAX _ N], b[MAX _ N], x[MAX _ N];
    printf(" \ nInput n value(dim of Ax = b): "); //输入 Ax = b 的维数
    scanf("%d", &n);
    if (n > MAX _ N)
    {
        printf("The input n is larger then MAX _ N, please redefine the MAX _ N. \ n");
        return 1;
    }
    if (n< = 0)
    {
        printf("Please input a number between 1 and %d. \ n", MAX _ N);
        return 1;
    }
                        //输入 Ax = b 的 A 矩阵
    printf("Now input the matrix a(i,j), i,j = 0,…,%d: \ n", n - 1);
    for (i = 0; i < n; i + + )
        for (j = 0; j < n; j + + )
```

```
        scanf("%lf", &a[i][j]);
                        //输入 b 矩阵
printf("Now input the matrix b(i), i=0,…,%d: \n", n-1);
for (i=0; i<n; i++) scanf("%lf", &b[i]);
for (i=0; i<n-1; i++)
{
    for (j=i+1, mi=i, mx=fabs(a[i][i]); j<n; j++)  //找主元素
    if (fabs(a[j][i])>mx)
    {
      mi=j;
      mx=fabs(a[j][i]);
    }
    if (i<mi)                                      //交换两行
    {
      tmp=b[i]; b[i]=b[mi]; b[mi]=tmp;
      for (j=i; j<n; j++)
      {
        tmp=a[i][j];
        a[i][j]=a[mi][j];
        a[mi][j]=tmp;
      }
        }
    for (j=i+1; j<n; j++)                          //高斯消元
    {  tmp=-a[j][i]/a[i][i];
       b[j]+=b[i]*tmp;
      for (k=i; k<n; k++)
      a[j][k]+=a[i][k]*tmp;
    }  }
x[n-1]=b[n-1]/a[n-1][n-1];                         //求解方程
for (i=n-2; i>=0; i--)
{     x[i]=b[i];
      for (j=i+1; j<n; j++)
      x[i]-=a[i][j]*x[j];
      x[i]/=a[i][i];
    }
    printf("Solve... x_i= \n");                    //输出
    for (i=0; i<n; i++) printf("%f \n", x[i]);
    return 0;
}
// ------------------------------ End of File ------------------------------
```

计算实例

用列主元高斯消元法求解方程组：

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ x_1 + 3x_2 + 2x_3 = 6 \\ x_1 + 2x_2 + 2x_3 = 5 \end{cases}$$

程序输入输出

```
Input n value(dim of Ax = b): 3
Now input the matrix a(i,j), i,j = 0,…,2:
2  1  1  1  3  2  1  2  2
Now input the matrix b(i), i = 0,…,2:
4 6 5
Solve... x_i =
1.000000
1.000000
1.000000
```

程序 10　用 Courant 分解公式求解方程组：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

算法描述

输入矩阵 a 及列向量 b；

将矩阵 a 分解为 $a = LU$，其中

$$L = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} 1 & u_{12} & \cdots & u_{1n} \\ & 1 & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & 1 \end{pmatrix}$$

对 $k = 1,2,\cdots,n$，

$$l_{ik} = a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk}, \quad i = k,k+1,\cdots,n$$

$$u_{kj} = \left(a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}\right) \Big/ l_{kk}, \quad j = k+1,k+2,\cdots,n$$

记 $Ux = y$，解 $Ly = b$，得

$$y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right) \Big/ l_{ii}, \quad i = 1,2,\cdots,n$$

再由 $Ux = y$ 得

$$x_i = y_i - \sum_{j=i+1}^{n} u_{ij}x_j, \quad i = n,n-1,\cdots,2,1$$

程序源码

```
////////////////////////////////////////////////////
//          Purpose: Courant 分解解 Ax = b          //
////////////////////////////////////////////////////
```

```
# include < stdio.h >
# include < math.h >
# define MAX _ N 20                    //(x _ i,y _ i)的最大维数
int main( )
{   int n;
  int i, j, k;
  int mi; double mx, tmp;
  static double a[MAX _ N][MAX _ N], b[MAX _ N], x[MAX _ N], y[MAX _ N];
  static double l[MAX _ N][MAX _ N], u[MAX _ N][MAX _ N];
  printf(" \ nInput n value(dim of Ax = b): "); //输入 Ax = b 的维数
  scanf("%d", &n);
  if (n > MAX _ N)
    {   printf("The input n is larger then MAX _ N, please redefine the MAX _ N. \ n");
    return 1;
  }
if (n < = 0)
{   printf("Please input a number between 1 and %d. \ n", MAX _ N);
  return 1;
}
                          //输入 Ax = b 的 a 矩阵
printf("Now input the matrix a(i,j), i,j = 0,…,%d: \ n", n - 1);
for (i = 0; i < n; i + + )
  for (j = 0; j < n; j + + )
  scanf("%lf", &a[i][j]);
                          //输入 b 矩阵
printf("Now input the matrix b(i), i = 0,…,%d: \ n", n - 1);
for (i = 0; i < n; i + + ) scanf("%lf", &b[i]);
for (i = 0; i < n; i + + ) u[i][i] = 1;        //u 矩阵对角元素为 1
for (k = 0; k < n; k + + )
{   for (i = k; i < n; i + + )                 //计算 L 矩阵
  {
  l[i][k] = a[i][k];
  for (j = 0; j < = k - 1; j + + )
    l[i][k] - = (l[i][j] * u[j][k]);
  }
  for (j = k + 1; j < n; j + + )                        //计算 u 矩阵
  {
  u[k][j] = a[k][j];
  for (i = 0; i < = k - 1; i + + )
    u[k][j] - = (l[k][i] * u[i][j]);
    u[k][j] / = l[k][k];
    }
}
```

```
for (i=0; i<n; i++)                                    //Ly=b 计算 y
{   y[i]=b[i];
  for (j=0; j<=i-1; j++)
  y[i]-=(l[i][j]*y[j]);
  y[i]/=l[i][i];
}
for (i=n-1; i>=0; i--)                                 //uX=y 计算 X
{ x[i]=y[i];
  for (j=i+1; j<n; j++)
  x[i]-=(u[i][j]*x[j]);
}
printf("Solve... x_i=\n");                             //输出
for (i=0; i<n; i++) printf("%f\n", x[i]);
return 0;
}
```

计算实例

用 Courant 分解公式求解方程组：

$$\begin{pmatrix} 1 & 2 & 1 \\ -2 & -1 & -5 \\ 0 & -1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 24 \\ -63 \\ 50 \end{pmatrix}$$

程序输入输出

```
Input n value(dim of Ax=b): 3
Now input the matrix a(i,j), i,j=0,…,2:
1 2 1 -2 -1 -5  0 -1 6
Now input the matrix b(i), i=0,…,2:
24 -63 50
Solve... x_i=
7.000000
4.000000
9.000000
```

程序 11 用 LDL^T 分解求解方程组：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}=\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

算法描述

输入矩阵 A 及列向量 b。

将矩阵 A 分解为 $A=LDL^T$，其中

$$L=\begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix},\quad D=\begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}$$

对 $k = 1,2,\cdots,n$,

$$\begin{cases} d_k = a_{kk} - \sum_{r=1}^{k-1} l_{kr}^2 d_r, \\ l_{ik} = \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir} l_{kr} d_r \right) / d_k, \end{cases} \quad i = k+1,\cdots,n$$

解 $Ax = b$,先解 $Lz = b$:

$$z_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} z_j \right), i = 1,2,\cdots,n$$

再由 $Dy = z$ 得

$$y_i = z_i / d_i, \quad i = 1,2,\cdots,n$$

最后由 $L^T x = y$ 得

$$x_i = y_i - \sum_{j=i-1}^{n} l_{ji} x_j, \quad i = n, n-1,\cdots,2,1$$

程序源码

```
////////////////////////////////////////////////////
//          Purpose: LDL'分解解方程组              //
////////////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
# define MAX_N 20                    //(x_i,y_i)的最大维数

int main()
 { int n;
   int i, j, k;
   int mi; double mx, tmp;
   static double a[MAX_N][MAX_N],b[MAX_N],x[MAX_N],y[MAX_N], z[MAX_N];
   static double L[MAX_N][MAX_N], d[MAX_N];
   printf(" \ nInput n value(dim of Ax = b): ");          //输入 Ax = b 的维数
   scanf("%d", &n);
   if (n > MAX_N)
   {
     printf("The input n is larger then MAX_N, please redefine the MAX_N. \ n");
     return 1;
   }
   if (n < = 0)
    {   printf("Please input a number between 1 and %d. \ n", MAX_N);
     return 1;
   }
                                      //输入 Ax = b 的 A 矩阵
   printf("Now input the matrix a(i,j), i,j = 0,…,%d: \ n", n-1);
   for (i = 0; i < n; i + + )
     for (j = 0; j < n; j + + )
```

```
    scanf("%lf", &a[i][j]);
                                    //输入 b 矩阵
printf("Now input the matrix b(i), i = 0,…,%d: \n", n-1);
for (i = 0; i < n; i++) scanf("%lf", &b[i]);

for (i = 0; i < n; i++) L[i][i] = 1;          //u 矩阵对角元素为 1

for (k = 0; k < n; k++)
{
   d[k] = a[k][k];                            //计算 d_i
   for (j = 0; j <= k-1; j++)
   d[k] -= (L[k][j] * L[k][j] * d[j]);
   for (i = k+1; i < n; i++)                          //计算 L 矩阵
   {
   L[i][k] = a[i][k];
   for (j = 0; j <= k-1; j++)
      L[i][k] -= (L[i][j] * L[k][j] * d[j]);
   L[i][k] /= d[k];
   }
}

for (i = 0; i < n; i++)                               //求解 L'z = b 的 z
{
   z[i] = b[i];
   for (j = 0; j <= i-1; j++)
   z[i] -= L[i][j] * z[j];
}
for (i = 0; i < n; i++)                              //求解 Dy = z 的 y
   y[i] = z[i]/d[i];
for (i = n-1; i >= 0; i--)                            //求解 Ly = x
{  x[i] = y[i];
   for (j = i+1; j < n; j++)
      x[i] -= L[j][i] * x[j];
   }
   printf("Solve... x_i= \n");                        //输出
   for (i = 0; i < n; i++) printf("%f \n", x[i]);
   return 0;
}
```

计算实例

用 LDL^T 分解求解方程组：

$$\begin{pmatrix} 1 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 4.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -8 \\ 12 \end{pmatrix}$$

程序输入输出

```
Input n value(dim of Ax = b): 3
Now input the matrix a(i,j), i,j = 0,…,2:
1 -1 1 -1 3 -2  1 -2 4.5
Now input the matrix b(i), i = 0,…,2:
4 -8 12
Solve... x_i =
1.000000
-1.000000
2.000000
```

习题 5

5.1 分别用高斯消元法和列主元方法解方程(计算中取四位有效数字):

$$(1)\begin{cases}0.002x_1 + 87.13x_2 = 87.15\\4.453x_1 - 7.26x_2 = 37.27\end{cases}\qquad(2)\begin{cases}0.01x_1 - 69.47x_2 = -138.93\\2.01x_1 + 8.51x_2 = 15.01\end{cases}$$

5.2 用高斯消元法编制程序解下列方程组:

$$\begin{cases}7.2x_1 + 2.3x_2 - 4.4x_3 + 0.5x_4 = 15.1\\1.3x_1 + 6.3x_2 - 3.5x_3 + 2.8x_4 = 1.8\\5.6x_1 + 0.9x_2 + 8.1x_3 - 1.3x_4 = 16.6\\1.5x_1 + 0.4x_2 + 3.7x_3 + 5.9x_4 = 36.9\end{cases}$$

5.3 用高斯消元法计算下列行列式的值:

$$(1)\ A = \begin{vmatrix}-1 & 3 & 2\\2 & 1 & -2\\3 & 6 & 2\end{vmatrix}\qquad(2)\ B = \begin{vmatrix}10 & -2 & -1\\-2 & 10 & -1\\-1 & -2 & 5\end{vmatrix}$$

5.4 用 Doolittle 分解求解方程组:

$$(1)\begin{pmatrix}2 & 1 & 2\\-2 & 2 & -1\\2 & 4 & 6\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = \begin{pmatrix}18\\-39\\24\end{pmatrix}\qquad(2)\begin{pmatrix}3 & 1 & 2\\-3 & 1 & -1\\6 & -4 & 2\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = \begin{pmatrix}23\\-10\\12\end{pmatrix}$$

5.5 用 Courant 方法求解下列方程组:

$$(1)\begin{pmatrix}5 & 1 & 2\\1 & 3 & -1\\2 & 3 & 5\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = \begin{pmatrix}10\\2\\15\end{pmatrix}$$

$$(2)\begin{pmatrix}2 & 4 & 6\\1 & 4 & 7\\3 & 8 & 12\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = \begin{pmatrix}26\\25\\46\end{pmatrix},\qquad\begin{pmatrix}2 & 4 & 6\\1 & 4 & 7\\3 & 8 & 12\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = \begin{pmatrix}40\\34\\71\end{pmatrix}$$

5.6 用 LDL^T 分解求解方程组:

(1) $\begin{cases} -6x_1 + 3x_2 + 2x_3 = -4 \\ 3x_1 + 5x_2 + x_3 = 11 \\ 2x_1 + x_2 + 6x_3 = -8 \end{cases}$ (2) $\begin{cases} x_1 + 2x_2 + 3x_3 = -3 \\ 2x_1 + x_2 - 2x_3 = 10 \\ 3x_1 - 2x_2 + x_3 = 7 \end{cases}$

5.7 用追赶法求解下列三对角方程组：

(1) $\begin{pmatrix} 1 & 3 & 0 & 0 \\ 2 & 7 & 3 & 0 \\ 0 & 2 & 7 & 3 \\ 0 & 0 & 2 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -2 \\ -8 \\ -6 \\ 5 \end{pmatrix}$ (2) $\begin{pmatrix} 10 & 5 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ 0 & 1 & 10 & 5 \\ 0 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 27 \\ 6 \end{pmatrix}$

5.8 用 Gauss-Jordan 消元法求解方程组：

(1) $\begin{pmatrix} 5 & 2 & 2 \\ -1 & 3 & 0 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \\ 3 \end{pmatrix}$ (2) $\begin{pmatrix} 5 & 2 & 2 \\ 0 & 2 & 1 \\ 1 & -1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -4 \\ 5 \\ -1 \end{pmatrix}$

第 6 章　解线性方程组的迭代法

用迭代法求解线性方程组 $AX = y$ 与第 4 章非线性方程求根的方法相似，对方程组 $AX = y$ 进行等价变换，构造同解方程组 $X = MX + g$（对 $AX = y$ 可构造各种等价方程组，分解 $A = N - P$，N 可逆，则由 $AX = y$ 得到 $X = N^{-1}PX + N^{-1}y = MX + g$），以此构造迭代关系式

$$X^{(k+1)} = MX^{(k)} + g$$

任取初始向量 $X^{(0)} = (x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)})^T$，代入迭代式中，经计算得到迭代序列 $X^{(1)}, X^{(2)}, \cdots$。

若迭代序列 $\{X^{(k+1)}\}$ 收敛，设 $\{X^{(k)}\}$ 的极限为 X^*，对迭代式两边取极限

$$\lim_{k\to\infty} X^{(k+1)} = \lim_{k\to\infty}(MX^{(k)} + g)$$

即 $X^* = MX^* + g$，X^* 是方程组 $AX = y$ 的解，此时称迭代法收敛，否则称迭代法发散。我们将看到，不同于非线性方程的迭代方法，解线性方程组的迭代收敛与否完全决定于迭代矩阵的性质，与迭代初始值的选取无关。迭代法的优点是占用存储空间少，程序实现简单，尤其适用于大型稀疏矩阵；不尽人意之处是要面对判断迭代是否收敛和收敛速度的问题。

可以证明迭代矩阵的谱半径 $\rho(M) = \max\limits_{1\le i\le n} |\lambda_i| < 1$ 是迭代收敛的充分必要条件，其中 λ_i 是矩阵 M 的特征根。事实上，若 X^* 为方程组 $AX = y$ 的解，则有

$$X^* = MX^* + g$$

再由迭代式 $X^{(k+1)} = MX^{(k)} + g$，可得到

$$X^* - X^{(k+1)} = M(X^* - X^{(k)}) = M^2(X^* - X^{(k-1)}) = \cdots = M^{k+1}(X^* - X^{(0)})$$

由线性代数定理，$\lim\limits_{k\to\infty} M^k = O$ 的充分必要条件为 $\rho(M) < 1$。

因此，称谱半径小于 1 的矩阵为收敛矩阵。一个矩阵的谱半径，需要求解矩阵的特征值才能得到。通常这是较为繁重的工作。但是可以通过计算矩阵的范数等方法简化判断收敛的工作。前面已经提到过，若 $\| A \|_p$ 为矩阵 A 的范数，则总有 $\| A \|_p \ge \rho(A)$。因此，若 $\| A \|_p < 1$，则 A 必为收敛矩阵。计算矩阵的 1 范数和 ∞ 范数的方法比较简单，其中：

$$\| A \|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$$

$$\| A \|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$$

于是，只要迭代矩阵 M 满足 $\| M \|_1 < 1$ 或 $\| M \|_\infty < 1$，就可以判断迭代序列 $X^{(k+1)} = MX^{(k)} + g$ 是收敛的。

要注意的是，当 $\| M \|_1 > 1$ 或 $\| M \|_\infty > 1$ 时，不能判断迭代序列发散。

在计算中当相邻两次的误差向量的某种范数 $\| X^{(k+1)} - X^{(k)} \|_p$ 小于给定精度时，则

停止迭代计算，视 $X^{(k+1)}$ 为方程组 $AX=y$ 的近似解(有关范数的详细定义请看 0.4 节。)

6.1 简单(Jacobi)迭代

6.1.1 Jacobi 迭代格式

Jacobi 迭代计算

n 元线性方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = y_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = y_2 \\ \cdots \quad \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = y_n \end{cases} \tag{6.1}$$

写成矩阵形式为 $AX=y$。若 $a_{ii}\neq 0, i=1,2,\cdots,n$，将式(6.1)中每个方程的 $a_{ii}x_i$ 留在方程的左边，其余各项都移到方程的右边；方程两边除以 a_{ii}，则得到下列同解方程组：

$$\begin{cases} x_1 = -\dfrac{a_{12}}{a_{11}}x_2 - \cdots - \dfrac{a_{1n}}{a_{11}}x_n + \dfrac{y_1}{a_{11}} \\ x_2 = -\dfrac{a_{21}}{a_{22}}x_1 - \cdots - \dfrac{a_{2n}}{a_{22}}x_n + \dfrac{y_2}{a_{22}} \\ \cdots \\ x_n = -\dfrac{a_{n1}}{a_{nn}}x_1 - \dfrac{a_{n2}}{a_{nn}}x_2 - \cdots - \dfrac{a_{n,n-1}}{a_{nn}}x_{n-1} + \dfrac{y_n}{a_{nn}} \end{cases}$$

记 $b_{ij}=-a_{ij}/a_{ii}$，$g_i=y_i/a_{ii}$，构造迭代形式

$$\begin{cases} x_1^{(k+1)} = b_{12}x_2^{(k)} + b_{13}x_3^{(k)} + \cdots + b_{1n}x_n^{(k)} + g_1 \\ x_2^{(k+1)} = b_{21}x_1^{(k)} + b_{23}x_3^{(k)} + \cdots + b_{2n}x_n^{(k)} + g_2 \\ \cdots \\ x_n^{(k+1)} = b_{n1}x_1^{(k)} + b_{n2}x_2^{(k)} + \cdots + b_{n,n-1}x_{n-1}^{(k)} + g_n \end{cases} \tag{6.2}$$

或

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} 0 & b_{12} & \cdots & b_{1n} \\ b_{21} & 0 & \cdots & b_{2n} \\ & & \ddots & \\ b_{n1} & b_{n2} & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} + \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}$$

迭代计算式(6.2)称为简单迭代或 Jacobi 迭代。任取初始向量 $X^{(0)}$，由式(6.2)可得到迭代向量序列 $\{X^{(k)}\}$。

Jacobi 迭代矩阵

若令

$$D = \begin{pmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix}$$

由 $AX=(D+A-D)X=y$,得到等价方程组:

$$DX=(D-A)X+y$$

$$X^{(k+1)}=D^{-1}(D-A)X^{(k)}+D^{-1}y$$

记 $B=D^{-1}(D-A)=I-D^{-1}A$, $g=D^{-1}y$

不难看出,B 正是迭代式(6.2) 的迭代矩阵,g 是常数项向量。于是式(6.2) 可写成矩阵形式:

$$X^{(k+1)}=BX^{(k)}+g \tag{6.3}$$

其中:$B=\begin{pmatrix} 0 & -a_{1,2}/a_{11} & \cdots & -a_{1,n}/a_{11} \\ -a_{2,1}/a_{22} & 0 & \cdots & -a_{2,n}/a_{22} \\ & & \vdots & \\ -a_{n,1}/a_{nn} & -a_{n,2}/a_{nn} & \cdots & 0 \end{pmatrix}$, $g=\begin{pmatrix} y_1/a_{11} \\ y_2/a_{22} \\ \vdots \\ y_n/a_{nn} \end{pmatrix}$

Jacobi 迭代算法

下面描述解线性方程组 $AX=y$ 的 Jacobi 迭代算法,为了简单起见,在算法中假定矩阵 A 满足 Jacobi 迭代要求,即 $a_{ii}\neq 0, i=1,2,\cdots,n$,并设由系数矩阵 A 构造的迭代矩阵 B 是收敛的。

1:定义和输入系数矩阵 A 与常数项向量 Y 的元素。

2:FOR i := 1,2,…, n

{$g_i := y_i/a_{ii}$! 假定 $a_{ii}\neq 0, i=1,2,\cdots,n$,形成常数项向量

FOR j := 1,2,…, n

$b_{ij} := -a_{ij}/a_{ii}$

$b_{ii} := 0$} ! 形成迭代矩阵元素

3:x1 := {0,0,…, 0}^{T}, x2 := {1,1,…, 1}^{T}

! 赋初始值,x1 和 x2 分别表示 $X^{(k)}$ 和 $X^{(k+1)}$

4:WHILE $\| x1-x2 \|_\infty > 10^{-6}$

x1 := x2

x2 := B * x1 + g ! FOR u := 1,2,…,n

! s := 0;

! FOR v := 1,2,…,n s := s + b[u,v] * x1[v];

! x2[u] := s + g[u]

ENDWHILE

5:输出方程组的解 $x2_i, i=1,2,\cdots,n$。

例 6.1 :用 Jacobi 方法解下列方程组:

$$\begin{cases} 2x_1-x_2-x_3=-5 \\ x_1+5x_2-x_3=8 \\ x_1+x_2+10x_3=11 \end{cases}$$

解:方程的迭代格式:

$$\begin{cases} x_1^{(k+1)} = 0.5x_2^{(k)}+0.5x_3^{(k)}-2.5 \\ x_2^{(k+1)} = -0.2x_1^{(k)}+0.2x_3^{(k)}+1.6 \\ x_3^{(k+1)} = -0.1x_1^{(k)}-0.1x_2^{(k)}+1.1 \end{cases}$$

或

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{pmatrix} = \begin{pmatrix} 0 & 0.5 & 0.5 \\ -0.2 & 0 & 0.2 \\ -0.1 & -0.1 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{pmatrix} + \begin{pmatrix} -2.5 \\ 1.6 \\ 1.1 \end{pmatrix}$$

$\because \| B \|_1 = 0.7, \therefore$ Jacobi 迭代收敛。

取初始值 $X^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$，计算结果由表 6.1 所示。

表 6.1

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\| X^{(k)} - X^{(k-1)} \|_\infty$
0	1	1	1	
1	-1.5	1.6	0.9	0.6
2	-1.25	2.08	1.09	0.48
3	-0.915	2.068	1.017	0.355
4	-0.9575	1.9864	0.9847	0.0425
5	-1.01445	1.98844	0.99711	0.05695
6	-1.00722	2.00231	1.0026	0.00723
7	-0.997543	2.00197	1.00049	0.001

方程组的准确解是 $\{-1,2,1\}$。

6.1.2 Jacobi 迭代收敛条件

对于方程组 $Ax = y$，构造 Jacobi 迭代格式 $X^{(k+1)} = BX^{(k)} + g$，其中 $B = I - D^{-1}A$，$g = D^{-1}y$。当迭代矩阵的谱半径 $\rho = \max\limits_{1 \le i \le n} |\lambda_i| < 1$ 时，迭代收敛，这是收敛的充分必要条件。迭代矩阵的某范数 $\| B \| < 1$ 时，迭代收敛。要注意的是范数小于 1 只是判断迭代矩阵收敛的充分条件，当迭代矩阵的一种范数使 $\| B \| > 1$，并不能确定迭代矩阵是收敛还是发散。例如，$B = \begin{pmatrix} 0.9 & 0 \\ 0.2 & 0.8 \end{pmatrix}$，则 $\| B \|_\infty = 1.0$，$\| B \|_1 = 1.1$，但它的特征值是 0.9 和 0.8。$\rho(B) < 1$，B 是收敛的。

当方程组的系数矩阵 A 具有某些性质时，可直接判定由它生成的 Jacobi 迭代矩阵是收敛的。

定理 6.1：若方程组 $AX = y$ 的系数矩阵 A，满足下列条件之一，则其 Jacobi 迭代收敛。

(1) A 为行对角优阵，即 $|a_{ii}| > \sum\limits_{j \ne i} |a_{ij}|, i = 1,2,\cdots,n$。

(2) A 为列对角优阵，即 $|a_{jj}| > \sum\limits_{i \ne j} |a_{ij}|, j = 1,2,\cdots,n$。

证明：(1)Jacobi 迭代矩阵 $B = (b_{ij})$，其中 $b_{ij} = -a_{ij}/a_{ii}, j \ne i, b_{ii} = 0$。

$$\| B \|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |b_{ij}| = \max_{1 \le i \le n} \sum_{j \ne i} \frac{|a_{ij}|}{|a_{ii}|} < 1$$

$\therefore$ Jacobi 迭代收敛。

(2) A 为列对角优阵，$\therefore A^T$ 为行对角优阵，由系数矩阵 A^T 构造的迭代矩阵 $C = I - D^{-1}A^T$，A^T 为行对角优阵，则有

$$\| C \|_\infty = \| I - D^{-1}A^T \|_\infty < 1$$

又 $\rho(I - D^{-1}A^T) < \| I - D^{-1}A^T \|_\infty$

得到 $\rho(I - D^{-1}A^T) < 1$

而 $\rho(I - D^{-1}A) = \rho(I - D^{-1}A^T)$，

得 $\rho(B) = \rho(I - D^{-1}A) < 1$

$\therefore$ 由系数矩阵 A 构造的 Jacobi 迭代矩阵收敛。

6.2 Gauss-Seidel 迭代

Gauss-Seidel 迭代计算

在 Jacobi 迭代中，用 $(x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})^T$ 的值代入方程(6.2)中计算出 $x_i^{(k+1)}$，$i = 1, 2, \cdots, n$ 的值，$x_i^{(k+1)}$ 的计算公式是

$$x_i^{(k+1)} = \sum_{j=1}^{n} b_{ij}x_j^{(k)} + g_i$$

事实上，在计算 $x_i^{(k+1)}$ 前，已经得到 $x_1^{(k+1)}, \cdots, x_{i-1}^{(k+1)}$ 的值，不妨将已算出的分量直接代入迭代式中，及时使用最新计算出的分量值。因此 $x_i^{(k+1)}$ 的计算公式可改为：

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} b_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} b_{ij}x_j^{(k)} + g_i$$

即用向量 $(x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})^T$ 计算出 $x_1^{(k+1)}$ 的值，用向量 $(x_1^{(k+1)}, x_2^{(k)}, \cdots, x_n^{(k)})^T$ 计算出 $x_2^{(k+1)}$ 的值，……，用向量 $(x_1^{(k+1)}, \cdots, x_{i-1}^{(k+1)}, x_i^{(k)}, \cdots x_n^{(k)})^T$ 计算出 $x_i^{(k+1)}$ 的值，这种迭代格式称为 Gauss-Seidel 迭代。

对于方程组 $AX = y$，如果由它构造 Gauss-Seidel 迭代和 Jacobi 迭代都收敛，那么，一般 Gauss-Seidel 迭代比 Jacobi 迭代的收敛效果要好，但是情况并非总是如此。

构造方程组 $AX = y$ 的 Gauss-Seidel 迭代格式的步骤与 Jacobi 类似，设 $a_{ii} \neq 0$，$i = 1, 2, \cdots, n$；将式(6.1)中每个方程的 $a_{ii}x_i$ 留在方程的左边，其余各项都移到方程的右边；方程两边除以 a_{ii}，得到下列同解方程组：

$$\begin{cases} x_1 = -\dfrac{a_{12}}{a_{11}}x_1 - \cdots - \dfrac{a_{1n}}{a_{11}}x_n + \dfrac{y_1}{a_{11}} \\ x_2 = -\dfrac{a_{21}}{a_{22}}x_1 - \cdots - \dfrac{a_{2n}}{a_{22}}x_n + \dfrac{y_2}{a_{22}} \\ \cdots \\ x_n = -\dfrac{a_{n1}}{a_{nn}}x_1 - \dfrac{a_{n2}}{a_{nn}}x_2 - \cdots - \dfrac{a_{n,n-1}}{a_{nn}}x_n + \dfrac{y_n}{a_{nn}} \end{cases}$$

记 $b_{ij} = -a_{ij}/a_{ii}$，$g_i = y_i/a_{ii}$，对方程组对角线以上的 x_i 取第 k 步迭代的数值，对角线以下的 x_i 取第 $k+1$ 步迭代的数值，构造 Gauss-Seidel 迭代形式：

$$\begin{cases} x_1^{(k+1)} = & b_{12}x_2^{(k)} + b_{13}x_3^{(k)} + \cdots + b_{1n}x_n^{(k)} + g_1 \\ x_2^{(k+1)} = b_{21}x_1^{(k+1)} & + b_{23}x_3^{(k)} + \cdots + b_{2n}x_n^{(k)} + g_2 \\ \cdots & \\ x_n^{(k+1)} = b_{n1}x_1^{(k+1)} + b_{n2}x_2^{(k+1)} + \cdots & + b_{n,n-1}x_{n-1}^{(k+1)} + g_n \end{cases} \tag{6.4}$$

例6.2：用 Gauss-Seidel 方法解方程组：

$$\begin{cases} 2x_1 - x_2 - x_3 = -5 \\ x_1 + 5x_2 - x_3 = 8 \\ x_1 + x_2 + 10x_3 = 11 \end{cases}$$

解：方程的迭代格式：

$$\begin{cases} x_1^{(k+1)} = & 0.5x_2^{(k)} + 0.5x_3^{(k)} - 2.5 \\ x_2^{(k+1)} = -0.2x_1^{(k+1)} & + 0.2x_3^{(k)} + 1.6 \\ x_3^{(k+1)} = -0.1x_1^{(k+1)} - 0.1x_2^{(k+1)} & + 1.1 \end{cases}$$

取初始值 $x^{(0)} = (0,0,0)$，有

$k = 1$ 时，

$$x_1^{(1)} = 0.5 \cdot 0 + 0.5 \cdot 0 - 2.5 = -2.5$$

$$x_2^{(1)} = -0.2 \cdot (-2.5) - 0.2 \cdot 0 + 1.6 = 2.1$$

$$x_3^{(1)} = -0.1 \cdot (-2.5) - 0.1 \cdot 2.1 + 1.1 = 1.04$$

$k = 2$ 时，

$$x_1^{(2)} = 0.5 \cdot 2.1 + 0.5 \cdot 1.04 - 2.5 = -0.93$$

$$x_2^{(2)} = -0.2 \cdot (-0.93) + 0.2 \cdot 2.1 + 1.6 = 1.994$$

$$x_3^{(2)} = -0.1 \cdot (-0.93) - 0.1 \cdot 1.994 + 1.1 = 0.9936$$

计算结果由表 6.2 所示。

表 6.2

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|x^{(k)} - x^{(k-1)}\|_\infty$
0	0	0	0	
1	-2.5	2.1	1.04	1.5
2	-0.93	1.994	0.9936	0.57
3	-1.0062	1.9996	0.999964	0.07
4	-0.999708	2.0007	0.999964	0.006

Gauss-Seidel 迭代矩阵

设 $A = D + L + U$

$$= \begin{pmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & & & \\ a_{21} & 0 & & \\ \vdots & & \ddots & \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ & 0 & & a_{2n} \\ & & \ddots & \vdots \\ & & & 0 \end{pmatrix}$$

写成等价矩阵表达式：

$$AX = (D + L + U)X = (D + L)X + UX = y$$

$$(D + L)X = -UX + y$$

构造迭代形式：

$$(D + L)X^{(k+1)} = -UX^{(k)} + y$$

有

$$X^{(k+1)} = -(D + L)^{-1}UX^{(k)} + (D + L)^{-1}y \tag{6.5}$$

则 Gauss-Seidel 迭代式(6.4)为

$$X^{(k+1)} = SX^{(k)} + f \tag{6.6}$$

记 $S = -(D + L)^{-1}U, f = (D + L)^{-1}y$，$S$ 称为 Gauss-Seidel 迭代矩阵。

Gauss-Seidel 迭代算法

Gauss-Seidel 迭代的程序实现与 Jacobi 迭代步骤大致相同，对于方程组 $AX = y$，在前面的 Jacobi 算法中，假定 Jacobi 迭代矩阵为 B，$x1$ 表示 $X^{(k)}$，$x2$ 表示 $X^{(k+1)}$，其迭代的核心部分是 $x2 = B * x1 + g$。下面的算法给出由 B 和 $x1$ 计算 $x2$ 的过程，省略了形成迭代矩阵 B 和对 $x1, x2$ 初始化的部分。

Jacobi 迭代的核心部分：

```
WHILE  ‖x1 - x2‖∞ > 10^-6
  FOR (u := 1; u <= n, u++)   x1[u]:=x2[u]
  FOR (i := 1; i <= n; i++)
    { s:= g[i];
    FOR ( j := 1; j <= n; j++ )
    {s := s + b[i][j] * x1[j] }     ! 注意 x1[j]
    x2[i] := s }
  ENDWHILE
```

在 Gauss-Seidel 迭代计算中并不需要形成迭代矩阵 S，由式(6.4)可知在计算中只要形成矩阵 $B = (b_{ij}) = (-a_{ij}/a_{ii})$，在程序中令向量 $x2 = (x_1^{(k+1)}, \cdots, x_{i-1}^{(k+1)}, x_i^{(k)}, \cdots x_n^{(k)})^T$，$x1 = (x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})^T$。它的核心部分是计算迭代式 $x2 = B * x2 + g$，计算中只需及时将 $x_i^{(k+1)}$ 放到 $x_i^{(k)}$ 的位置上。

Gauss-Seidel 迭代的核心部分：

```
WHILE  ‖x1 - x2‖∞ > 10^-6
  FOR ( u:= 1; u <= n; u++)
    x1[u]:= x2[u]
  FOR (i:= 1; i <= n; i++ )
    { s:= g[i];
    FOR ( j:= 1; j <= n; j++ )
        {s:= s+ b[i][j] * x2[j] }   ! 注意 x2[j]
    x2[i]:= s }
  ENDWHILE
```

上列算法是在假定迭代收敛的前提下，使用当型(WHILE)结构控制循环。更一般地，

可将上列算法中 WHILE 循环改为 FOR 循环，通过控制循环次数和观测计算误差终止循环。也可将上列算法中的 WHILE 语句改为

WHILE　$\| x1 - x2 \|_\infty > 10^{-6}$　OR　循环次数 < m

这时在程序中要增加循环变量的设定和运算。

判断 Gauss-Seidel 迭代收敛的方法与判断 Jacobi 迭代收敛类似，一方面从 Gauss-Seidel 迭代矩阵 S 获取信息，当 $\rho(S)<1$ 或 S 的某种范数 $\| S \| < 1$ 时，迭代收敛；另一方面，直接根据方程组系数矩阵的特点作出判断。

定理 6.2：若方程组系数矩阵为列或行对角优时，则 Gauss-Seidel 迭代收敛。

定理 6.3：若方程组系数矩阵 A 为正定阵，则 Gauss-Seidel 迭代收敛。

6.3　松弛迭代

松弛迭代计算

方程组 $AX = y$ 的 Jacobi 迭代形式 $X^{(k+1)} = BX^{(k)} + g$，记 $B = \widetilde{L} + \widetilde{U}$，其中 $\widetilde{L}$ 是下三角矩阵，$\widetilde{U}$ 是上三角矩阵。

得 Gauss-Seidel 的迭代形式：

$$X^{(k+1)} = \widetilde{L}X^{(k+1)} + \widetilde{U}X^{(k)} + g$$

记 $\Delta X = X^{(k+1)} - X^{(k)}$，有

$$X^{(k+1)} = X^{(k)} + \Delta X^{(k)}$$

这样 $\Delta X^{(k)}$ 可视为 $X^{(k)}$ 的修正量，而 $X^{(k+1)}$ 恰是由 $X^{(k)}$ 加修正量 $\Delta X^{(k)}$ 而得，如果将 $X^{(k+1)}$ 改为 $X^{(k)}$ 加上修正量 $\Delta X^{(k)}$ 乘一个因子 ω，迭代格式为：

$$X^{(k+1)} = X^{(k)} + \omega\Delta X^{(k)}$$

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

$$X^{(k+1)} = X^{(k)} + \omega(\widetilde{L}X^{(k+1)} + \widetilde{U}X^{(k)} + g - X^{(k)})$$

整理得

$$X^{(k+1)} = (1-\omega)X^{(k)} + \omega(\widetilde{L}X^{(k+1)} + \widetilde{U}X^{(k)} + g) \tag{6.7}$$

这里 ω 为修正因子，称为松弛因子，而式(6.7)称为松弛迭代。迭代的分量形式为

$$\begin{cases} x_1^{(k+1)} = (1-\omega)x_1^{(k)} + \omega(b_{12}x_2^{(k)} + b_{13}x_3^{(k)} + \cdots + b_{1n}x_n^{(k)} + g_1) \\ x_2^{(k+1)} = (1-\omega)x_2^{(k)} + \omega(b_{21}x_1^{(k-1)} + b_{23}x_3^{(k)} + \cdots + b_{2n}x_n^{(k)} + g_2) \\ \cdots \\ x_n^{(k+1)} = (1-\omega)x_n^{(k)} + \omega(b_{n1}x_1^{(k-1)} + b_{n2}x_2^{(k+1)} + \cdots + b_{n,n-1}x_{n-1}^{(k+1)} + g_n) \end{cases} \tag{6.8}$$

式(6.8)称为松弛迭代的计算格式。

松弛迭代算法

下列算法假定迭代矩阵收敛，否则要在 WHILE 循环中增加判断条件。

1：定义和输入系数矩阵 A 与常数项向量 y 的元素，输入因子 ω 的值。

2：FOR i := 1,2,…, n

　　$g_i := y_i / a_{ij}$　！假定 $a_{ii} \neq 0, i = 1,2,\cdots,n$，形成常数项向量

```
        FOR j := 1,2,…, n
            b_{i,j} := - a_{i,j} / a_{i,i}
        ⋮
3: x1 := {0,0,…, 0}^T, x2 := {1,1,…, 1}^T
4: WHILE ‖ x1 - x2 ‖_∞ > 10^-6
        { FOR ( u = 1; u < = n; u + + )
            x1[u] = x2[u]
          FOR (i = 1; i < = n, i + + )
            { s = g[i] ;
              FOR ( j = 1; j < = n; j + + )
                {s = s + b[i][j] * x2[j] }
              x2[i] := (1 - ω) x1[i] + ω · s }
        } ENDWHILE
```

5:输出 $x2_i, i = 1,2,\cdots,n$。

松弛迭代矩阵

将式(6.7)中含有 $X^{(k+1)}$ 与 $X^{(k)}$ 的项分别放在方程两边:

$$(I - \omega \tilde{L}) X^{(k+1)} = ((1 - \omega) I + \omega \tilde{U}) X^{(k)} + \omega g$$

用 $\tilde{L} = -D^{-1}L, \tilde{U} = -D^{-1}U$ 代入上式,有

$$(I + \omega D^{-1} L) X^{(k+1)} = ((1 - \omega) I - \omega D^{-1} U) X^{(k)} + \omega g$$

$$X^{(k+1)} = (I + \omega D^{-1} L)^{-1} ((1 - \omega) I - \omega D^{-1} U) X^{(k)} + \omega (I + \omega D^{-1} L)^{-1} g$$

则松弛因子为 ω 的迭代矩阵为

$$X^{(k+1)} = S_\omega X^{(k)} + f$$

其中

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

$$f = \omega (I + \omega D^{-1} L)^{-1} Y$$

定理 6.4:松弛迭代收敛的必要条件 $0 < \omega < 2$。

定理 6.5:若 A 为正定矩阵,则当 $0 < \omega < 2$ 时,松弛迭代恒收敛。

以上定理给出了松弛迭代因子的范围。对于每个给定的方程组,确定 ω 究竟取多少为最佳,这是比较困难的问题,对某些特定的方程组,我们可以得到一些理论结果。

通常,把 $0 < \omega < 1$ 的迭代称为亚松弛迭代,把 $\omega = 1$ 的迭代称为 Gauss-Seided 迭代,而把 $1 < \omega < 2$ 的迭代称为超松弛迭代。

6.4 逆矩阵计算

在线性代数中逆矩阵是按其伴随矩阵定义的,若 $|A| \neq 0$,则方阵 A 可逆,且 $A^{-1} = \frac{1}{|A|} A^*$,其中 A^* 为 A 的伴随矩阵。要计算 n^2 个 $n-1$ 阶的行列式才能得到一个伴随矩阵,在数值计算中因其计算工作量大而不被采用。通常对 (A, I) 做行的初等变换,在将 A 化成 I 的过程中得到 (I, A^{-1})。在数值计算中,这仍然是一种行之有效的方法。

由逆矩阵的定义 $A \cdot A^{-1} = I$，令 $A = (a_{ij})$，$A^{-1} = (x_{ij})$，有

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

化为 n 个方程组

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = e_j, j = 1,2,\cdots,n$$

e_j 是第 j 个分量为 1，其余分量为 0 的 n 维向量。或记为：$Ax_j = e_j, j = 1,2,\cdots,n$。

用直接法或迭代法算出 $x_j, j = 1,2,\cdots,n$，也就完成了逆矩阵 A^{-1} 的计算。

如果依次对 $(A, e_1), (A, e_2), \cdots, (A, e_n)$ 用 Gauss-Jordan 消元法，组合起来看有（当然也能组合起来做）：

$$(A, e_1, e_2, \cdots, e_n) = (A, I)$$

这正是在线性代数中用初等变换计算逆矩阵的方法。

由此可见，计算一个 n 阶逆矩阵的工作量相当于解 n 个线性方程组。在数值计算中，常常将计算矩阵逆的问题转化为解线性方程组的问题。

例如，已知方阵 A 和向量 $X^{(k)}$ 有迭代关系式 $X^{(k+1)} = A^{-1} X^{(k)}$，在计算中不是先算出 A^{-1}，再作 A^{-1} 与 $X^{(k)}$ 的乘积得到 $X^{(k+1)}$；而是将 A 作为线性方程组系数矩阵，求解方程组 $AX^{(k+1)} = X^{(k)}$，$X^{(k)}$ 作为常数项解出 $X^{(k+1)}$。

6.5 程序示例

程序 12 用 Jacobi 迭代求解方程组：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$$

算法描述

输入系数矩阵 A 及常数项向量 c。

按 Jacobi 迭代公式：

$$\begin{cases} x_1^{(k+1)} = (-a_{12} x_2^{(k)} - \cdots - a_{1n} x_n^{(k)} + c_1)/a_{11} \\ x_2^{(k+1)} = (-a_{21} x_2^{(k)} - \cdots - a_{2n} x_n^{(k)} + c_2)/a_{22} \\ \cdots \\ x_n^{(k+1)} = (-a_{n1} x_n^{(k)} - \cdots - a_{n,n-1} x_{n-1}^{(k)} + c_n)/a_{nn} \end{cases}$$

求解 $Ax = c$。

程序源码

```
////////////////////////////////////////////////////////////////
//      Purpose: Jacobi 迭代求解线性方程组                    //
////////////////////////////////////////////////////////////////
 # include < stdio.h >
 # include < math.h >
 # define MAX _ N 20              //方程的最大维数
 # define MAXREPT 100
 # define epsilon 0.00001                //求解精度
int main()
{   int n;
    int i, j, k;
    double err;
    static double a[MAX _ N][MAX _ N], b[MAX _ N][MAX _ N], c[MAX _ N], g[MAX _ N];
    static double x[MAX _ N], nx[MAX _ N];
    printf(" \ nInput n value(dim of Ax = c): ");    //输入方程的维数
    scanf("%d", &n);
    if (n> MAX _ N)
    {
        printf("The input n is larger then MAX _ N, please redefine the MAX _ N. \ n");
        return 1;
    }
if (n< =0)
    { printf("Please input a number between 1 and %d. \ n", MAX _ N); return 1; }
//输入 Ax = c 的系数矩阵 A.
printf("Now input the matrix a(i,j), i,j=0,…,%d: \ n", n-1);
for (i=0; i< n; i++)
  for (j=0; j< n; j++)
  scanf("%lf", &a[i][j]);
          //输入 c 矩阵
printf("Now input the matrix b(i), i=0,…,%d: \ n", n-1);
for (i=0; i< n; i++) scanf("%lf", &c[i]);
  for (i=0; i< n; i++)          //形成 x _{k+1} = bx _{k} + g 迭代矩阵 b
    for (j=0; j< n; j++)
    {
b[i][j] = -a[i][j]/a[i][i];
g[i] = c[i]/a[i][i];      //为了简化程序，假设 a[i][i]! =0,
        //否则要附加对 a[i][i]的判断及其相应的处理
    }
for (i=0; i< MAXREPT; i++)
{ for (j=0; j< n; j++)
  nx[j] = g[j];
  for (j=0; j< n; j++)
```

```
}
for (k=0; k<n; k++)
{
  if (j==k) continue;
  nx[j] += b[j][k] * x[k];              //迭代
}
 }
err=0;
for (j=0; j<n; j++)
if (err<fabs(nx[j]-x[j])) err=fabs(nx[j]-x[j]);    //误差
for (j=0; j<n; j++)
x[j]=nx[j];
if (err<epsilon)
{
  printf("Solve... x_i= \n");              //输出
  for (i=0; i<n; i++) printf("%f \n", x[i]);
  return 0;
  }
}
printf("After %d repeat, no result... \n", MAXREPT);//输出
return 1; }
```

计算实例

用 Jacobi 方法解下列方程组：

$$\begin{cases} 64x_1 - 3x_2 - x_3 = 14 \\ 2x_1 - 90x_2 + x_3 = -5 \\ x_1 + x_2 + 40x_3 = 20 \end{cases}$$

程序输入输出

```
Input n value(dim of Ax=b): 3
Now input the matrix a(i,j), i,j=0,…,2:
64 -3 -1  2 -90 1 1 1 40
Now input the matrix c(i), i=0,…,2:
14 -5 20
Solve... x_i=
0.229547
0.066130
0.492608
```

程序 13 用超松弛迭代(ω 作参数)求解方程组：

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$$

算法描述

输入矩阵 A 及列向量 c。

按因子为 ω 的超松弛迭代公式：

$$\begin{cases} x_1^{(k+1)} = (1-\omega)x_1^{(k)} - \omega(a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} - c_1)/a_{11} \\ x_2^{(k+1)} = (1-\omega)x_2^{(k)} - \omega(a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)} \cdots + a_{2n}x_n^{(k)} - c_2)/a_{22} \\ \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ x_n^{(k+1)} = (1-\omega)x_n^{(k)} - \omega(a_{n1}x_1^{(k+1)} + \cdots + a_{n,n-1}x_{n-1}^{(k+1)} - c_n)/a_{nn} \end{cases}$$

求解 $Ax = c$。

程序源码

```
////////////////////////////////////////////////////////
// Purpose: 超松弛迭代求解线性方程组      //
////////////////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
# define MAX_N 20                    //方程的最大维数
# define MAXREPT 100
# define epsilon 0.00001             //求解精度
int main()
{ int n;
  int i, j, k;
  double err, w;
  static double a[MAX_N][MAX_N], b[MAX_N][MAX_N], c[MAX_N], g[MAX_N];
  static double x[MAX_N], nx[MAX_N];
  printf("\nInput n value(dim of Ax = c): ");       //输入方程的维数
  scanf("%d", &n);
  if (n > MAX_N)
  {    printf("The input n is larger then MAX_N, please redefine the MAX_N.\n");
       return 1;
  }
  if (n <= 0)
  {    printf("Please input a number between 1 and %d.\n", MAX_N);
       return 1; }
                //输入 Ax = c 的 A 矩阵
printf("Now input the matrix a(i,j), i,j=0,…,%d:\n", n-1);
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  scanf("%lf", &a[i][j]);
                //输入 c 矩阵
printf("Now input the matrix c(i), i=0,…,%d:\n", n-1);
for (i=0; i<n; i++) scanf("%lf", &c[i]);
printf("Now input the w value: ");
```

```
scanf("%lf", &w);
if (w<=1 || w>=2)
{
  printf("w must between 1 and 2.\n");
  return 1;
}
for (i=0; i<n; i++)                    //改造 x_{k+1} = bx_{k} + g 迭代矩阵
  for (j=0; j<n; j++)
  {
  b[i][j] = -a[i][j]/a[i][i];
  g[i] = e[i]/a[i][i];
  }
for (i=0; i<MAXREPT; i++)
{
  for (j=0; j<n; j++)
nx[j] = g[j];
for (j=0; j<n; j++)
{
for (k=0; k<j; k++)
  nx[j] += b[j][k] * nx[k];            //迭代
for (k=j+1; k<n; k++)
  nx[j] += b[j][k] * x[k];
nx[j] = (1-w) * x[j] + w * nx[j];
}
err = 0;
for (j=0; j<n; j++)
if (err<fabs(nx[j]-x[j])) err = fabs(nx[j]-x[j]);      //误差
    for (j=0; j<n; j++)
x[j] = nx[j];
if (err<epsilon)
  {
  printf("Solve... x_i=\n");           //输出
      for (i=0; i<n; i++) printf("%f\n", x[i]);
  return 0;
  }
}
printf("After %d repeat, no result...\n", MAXREPT);          //输出
return 1;
}
```

计算实例

解下列方程组：

$$\begin{cases} 64x_1 - 3x_2 - x_3 = 14 \\ 2x_1 - 90x_2 + x_3 = -5 \\ x_1 + x_2 + 40x_3 = 20 \end{cases}$$

程序输入输出

```
Input n value(dim of Ax = b): 3
Now input the matrix a(i,j), i,j = 0,…,2:
64 -3 -1 2 -90 1 1 1 40
Now input the ? matrix c(i), i = 0,…,2:
14 -5 20
Now input the w value: 1                    ?
Solve... x_i =
0.229547
0.066130
0.492608
```

习题 6

6.1 计算下列矩阵的 $\|A\|_1$, $\|A\|_2$, $\|A\|_\infty$ 三种范数：

$$(1)A = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \qquad (2)A = \begin{pmatrix} 5 & 1 & 1 \\ 0 & 3 & 0 \\ -1 & 1 & 6 \end{pmatrix} \qquad (3)A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

6.2 计算下列矩阵的谱半径：

$$(1)B = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \qquad (2)B = \begin{pmatrix} 5 & 2 & 2 \\ 2 & 6 & 0 \\ 2 & 0 & 4 \end{pmatrix}$$

6.3 已知方程组：

$$\begin{cases} 10x_1 - x_2 = 1 \\ -x_1 + 10x_2 - x_3 = 0 \\ -x_2 + 10x_3 - x_4 = 1 \\ -x_3 + 10x_4 = 2 \end{cases}$$

(1) 写出解方程组的 Jacobi 迭代计算式，并以 $X^{(0)} = (0,0,0,0)^T$ 为初值，迭代计算 $X^{(1)}, X^{(2)}, X^{(3)}$。

(2) 写出解方程组的 Gauss-Seidel 迭代计算式，并以 $X^{(0)} = (0,0,0,0)^T$ 为初值，迭代计算 $X^{(1)}, X^{(2)}, X^{(3)}$。

(3) 分别写出 Jacobi 迭代、Gauss-Seidel 迭代的矩阵，并讨论其迭代收敛性。

6.4 以 $X^{(0)} = (0,0,0,0)^T$ 为初值，用 Jacobi 迭代求解下列方程组的 $X^{(1)}, X^{(2)}$。

$$(1)\begin{cases} 2x_1 - x_2 + x_3 = -1 \\ 3x_1 + 3x_2 + 9x_3 = 0 \\ 3x_1 + 3x_2 + 5x_3 = 4 \end{cases} \qquad (2)\begin{cases} 5x_1 - x_2 - x_3 = -1 \\ 3x_1 + 6x_2 + 2x_3 = 0 \\ x_1 - x_2 + 2x_3 = 4 \end{cases}$$

6.5 用 Gauss-Seidel 迭代求解：

$$(1)\begin{cases}10x_1-2x_2-x_3=0\\-2x_1+10x_2-x_3=-21\\-x_1-2x_2+5x_3=-20\end{cases}$$

自取初始值，当 $\|X^{(k+1)}-X^{(k)}\|_\infty<10^{-4}$ 时迭代停止。

$$(2)\begin{cases}5x_1-x_2-x_3=16\\3x_1+6x_2+2x_3=11\\x_1-x_2+2x_3=-2\end{cases}$$

以 $X^{(0)}=\begin{pmatrix}1\\1\\-1\end{pmatrix}$ 为初值，当 $\|X^{(k+1)}-X^{(k)}\|_\infty<10^{-3}$ 时迭代停止。

6.6 已知方程组 $\begin{cases}x_1+tx_2=b_1\\tx_1+2x_2=b_2\end{cases}$。

(1) 写出解方程组的 Jacobi 迭代的迭代矩阵，并讨论迭代收敛条件。

(2) 写出解方程组的 Gauss-Seidel 迭代的迭代矩阵，并讨论迭代收敛条件。

6.7 设有系数矩阵：

$$A=\begin{pmatrix}1&2&-2\\1&1&1\\2&2&1\end{pmatrix}\text{和 }B=\begin{pmatrix}2&-1&1\\1&1&1\\1&1&-2\end{pmatrix}$$

证明(1)系数矩阵 A 对 Jacobi 迭代收敛，而 Gauss-Seidel 迭代不收敛。

(2)系数矩阵 B 对 Jacobi 迭代不收敛，而 Gauss-Seidel 迭代收敛。

6.8* 选用一种直接方法或一种迭代方法，在计算机上编制程序计算下列矩阵的逆：

$$(1)A=\begin{pmatrix}1&2&0&1\\-1&0&2&3\\2&3&1&5\\0&1&7&3\end{pmatrix}\qquad(2)B=\begin{pmatrix}3&3&-2&1\\2&5&3&-4\\-3&2&6&1\\1&2&-1&3\end{pmatrix}$$

第 7 章　计算矩阵的特征值和特征向量

在线性代数中，一个 n 阶矩阵 A，若有数 λ 及非零 n 维向量 v 满足 $Av=\lambda v$，则称 λ 为 A 的特征值，v 为属于特征值 λ 的特征向量。在线性代数中，先计算矩阵 A 的特征多项式，即计算 $\det(\lambda I-A)=\lambda^n+\cdots+(-1)^n\det A$ 的根，算出 A 的 n 个特征值 λ_i，$i=1,2,\cdots,n$。然后解线性方程组 $(A-\lambda_i I)v=0$，计算出对应于 λ_i 的特征向量。由于求解高次多项式的根是件困难的事，上述方法一般无法解出阶数略大（$n>4$）的矩阵特征值的精确解，在实际计算中难以按定义计算矩阵特征值。

本章介绍一些简单有效的计算矩阵特征值和特征向量的近似值的数值方法。

7.1　幂　　法

7.1.1　幂法运算

在实际问题中，矩阵的按模最大特征值往往起重要的作用。例如矩阵的谱半径即矩阵的按模最大特征值的值，决定了迭代矩阵是否收敛。因此，矩阵的按模最大的特征值比其余的特征值的地位更加重要。幂法是计算按模最大特征值及相应的特征向量的数值方法。

简单地说，任取初始向量 $X^{(0)}$，进行迭代计算

$$X^{(k+1)}=AX^{(k)}$$

得到迭代序列 $\{X^{(k)}\}$，再分析 $X^{(k+1)}$ 与 $X^{(k)}$ 之间的关系，就可得到 A 的按模最大特征值及特征向量的近似解。

例 7.1：已知矩阵 $A=\begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix}$，用迭代序列 $X^{(k+1)}=AX^{(k)}$ 计算 A 的最大特征值。

解：计算 A 的特征多项式可得到 A 的特征值为：1.61803 和 0.61803。取

$$X^{(0)}=\begin{pmatrix}1\\ 1\end{pmatrix},\quad X^{(1)}=A\cdot X^{(0)}=\begin{pmatrix}1\\ 2\end{pmatrix},\quad X^{(2)}=A\cdot X^{(1)}=\begin{pmatrix}2\\ 3\end{pmatrix},$$

$$X^{(3)}=A\cdot X^{(2)}=\begin{pmatrix}3\\ 5\end{pmatrix},\quad X^{(4)}=\begin{pmatrix}5\\ 8\end{pmatrix},\quad X^{(5)}=\begin{pmatrix}8\\ 13\end{pmatrix},\cdots$$

在表 7.1 中列出迭代序列 $X^{(0)},X^{(1)},\cdots,X^{(12)}$ 以及 $x_2^{(k)}/x_2^{(k-1)}$ 的值。

观察前后两个向量对应分量的比值：

$$\frac{x_1^{(12)}}{x_1^{(11)}}=\frac{233}{144}=1.61805$$

$$\frac{x_2^{(12)}}{x_2^{(11)}}=\frac{377}{233}=1.61803$$

表 7.1

k	$X^{(k)}$		$x_2^{(k)}/x_2^{(k-1)}$	k	$X^{(k)}$		$x_2^{(k)}/x_2^{(k-1)}$
0	1	1		7	21	34	1.61905
1	1	2	2	8	34	55	1.61761
2	2	3	1.5	9	55	89	1.61818
3	3	5	1.66667	10	89	144	1.61798
4	5	8	1.6	11	144	233	1.61806
5	8	13	1.625	12	233	377	1.61803
6	13	21	1.61538				

在本题中，可以看到当 k 充分大的时候，$X^{(k+1)}$ 与 $X^{(k)}$ 对应分量的比值趋于最大特征值 1.618。但是，并非对每个矩阵 A，$X^{(k+1)}$ 与 $X^{(k)}$ 对应元素的比值都趋于矩阵 A 的最大特征值。这与矩阵特征值的分布有关。

在幂法中，假设矩阵 A 有特征值 λ_i，$i=1,2,\cdots,n$，其中 $|\lambda_1|\geq|\lambda_2|\geq|\lambda_3|\geq\cdots\geq|\lambda_n|$，并有 n 个线性无关的特征向量 v_i，即 $Av_i=\lambda_i v_i$，$i=1,2,\cdots,n$。

任取初始向量 $X^{(0)}$，$X^{(0)}$ 可表示成 A 的 n 个线性无关的特征向量 v_i 的线性组合，即

$$X^{(0)}=\alpha_1 v_1+\alpha_2 v_2+\cdots+\alpha_n v_n \tag{7.1}$$

那么，$X^{(1)}=AX^{(0)}=\alpha_1\lambda_1 v_1+\alpha_2\lambda_2 v_2+\cdots+\alpha_n\lambda_n v_n$，一般地有：

$$X^{(k)}=AX^{(k-1)}=\alpha_1\lambda_1^k v_1+\alpha_2\lambda_2^k v_2+\cdots+\alpha_n\lambda_n^k v_n \tag{7.2}$$

$X^{(k)}$ 的变化趋势与特征值的分布关系有关，幂法根据 $X^{(k)}$ 的变化趋势计算矩阵按模最大的特征值。下面讨论幂法中两种比较简单的情况。

(1)按模最大的特征值只有一个，且是单实根

设 $|\lambda_1|>|\lambda_2|\geq|\lambda_3|\geq\cdots\geq|\lambda_n|$，由式(7.2)得到

$$\begin{aligned}X^{(k)}&=\alpha_1\lambda_1^k v_1+\alpha_2\lambda_2^k v_2+\cdots+\alpha_n\lambda_n^k v_n\\&=\lambda_1^k\left[\alpha_1 v_1+\alpha_2\left(\frac{\lambda_2}{\lambda_1}\right)^k v_2+\cdots+\alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^k v_n\right]\end{aligned} \tag{7.3}$$

若 $\alpha_1\neq 0$，由于 $\left|\frac{\lambda_i}{\lambda_1}\right|<1$，$i=2,3,\cdots,n$，对充分大的 k 有 $\left|\left(\frac{\lambda_i}{\lambda_1}\right)^k\right|\ll 1$，$i=2,3,\cdots,n$，故

$$\begin{aligned}X^{(k)}&\approx\lambda_1^k\alpha_1 v_1\\X^{(k+1)}&\approx\lambda_1^{k+1}\alpha_1 v_1=\lambda_1\lambda_1^k\alpha_1 v_1=\lambda_1 X^{(k)}\end{aligned} \tag{7.4}$$

得到按模最大的特征值

$$\lambda_1\approx x_i^{(k+1)}/x_i^{(k)},\ i=1,2,\cdots,n \tag{7.5}$$

相应的特征向量近似地为 $X^{(k)}$。

由式(7.3)可知，$\{x_i^{(k+1)}/x_i^{(k)}\}$ 收敛于 λ_1 的速度取决于比值 $\left|\frac{\lambda_2}{\lambda_1}\right|$ 的大小。

例 7.2：计算矩阵 A 的按模最大的特征值和它的特征向量。

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

解:用表 7.2 给出计算结果。

表 7.2

k	$X^{(k)}$		k	$X^{(k)}$	
0	1	1	8	-84	86
1	2	0	9	172	-170
2	0	2	10	-340	342
3	4	-2	11	684	-682
4	-4	6	12	-1364	1366
5	12	-10	13	2732	-2730
6	-20	22	14	-5460	5462
7	44	-42	15	10924	-10922

$x_1^{(14)}/x_1^{(13)} = -5460/2732 = -1.9985$

$x_2^{(14)}/x_2^{(13)} = 5462/(-2730) = -2.00073$

$x_1^{(15)}/x_1^{(14)} = 10924/(-5460) = -2.0073$

$x_2^{(15)}/x_2^{(14)} = -10922/5462 = -1.99963$

得到 A 按模最大的特征值 $\lambda_1 = -2.00073$，和相应的特征向量 $v_1 \approx (-5460, 5462)$。继续算下去,可越来越靠近按模最大特征值的准确值 $\lambda_1 = -2$ 和特征向量 $v_1 = (1, -1)$。

(2)按模最大的特征值是互为反号的实根

设 $\lambda_1 > 0$,且 $\lambda_1 = -\lambda_2$,即 $|\lambda_1| = |\lambda_2| > |\lambda_3| \geq \cdots |\lambda_n|$,这时有

$$X^{(k)} = \lambda_1^k\left(\alpha_1 v_1 + (-1)^k\alpha_2 v_2 + \alpha_3\left(\frac{\lambda_3}{\lambda_1}\right)^k v_3 + \cdots + \alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^k v_n\right) \tag{7.6}$$

当 k 充分大时有:

$$X^{(k)} \approx \lambda_1^k(\alpha_1 v_1 + (-1)^k\alpha_2 v_2)$$

$$X^{(k+2)} \approx \lambda_1^{k+2}(\alpha_1 v_1 + (-1)^{k+2}\alpha_2 v_2) \approx \lambda_1^2 X^{(k)} \tag{7.7}$$

$X^{(k)}$ 呈现有规律的摆动,对充分大的 k,$X^{(k)}$ 与 $X^{(k+2)}$ 几乎相差一个常数因子 λ_1^2。

$$\therefore \quad \lambda_1 = \sqrt{x_i^{(k+2)}/x_i^{(k)}}$$

由 $\begin{cases} X^{(k+1)} = \lambda_1^{k+1}(\alpha_1 v_1 + (-1)^{k+1}\alpha_2 v_2) \\ X^{(k)} = \lambda_1^k(\alpha_1 v_1 + (-1)^k\alpha_2 v_2) \end{cases}$ 得

$$\begin{cases} X^{(k+1)} + \lambda_1 X^{(k)} \approx 2\lambda_1^{k+1}\alpha_1 v_1 \\ X^{(k+1)} - \lambda_1 X^{(k)} \approx (-1)^{(k+1)}2\lambda_1\alpha_2 v_2 \end{cases} \tag{7.8}$$

得到相应的特征向量:

$$\begin{cases} v_1 = X^{(k+1)} + \lambda_1 X^{(k)} \\ v_2 = X^{(k+1)} - \lambda_1 X^{(k)} \end{cases} \tag{7.9}$$

还有很多更复杂的情况，可参考有关书籍或选用其它方法。本文中只讨论幂法计算的两种情况：在计算中若 $x_i^{(k+1)}/x_i^{(k)}$ 的比值趋于一个稳定的值，则属于第(1)种情况；如果 $x_i^{(k+1)}/x_i^{(k)}$ 不能趋于一个稳定的值，但 $x_i^{(2k+2)}/x_i^{(2k)}$ 的比值趋于一个稳定的值，则属于第(2)种情况。

7.1.2 幂法的规范运算

在幂法迭代计算 $X^{(k+1)} = AX^{(k)}$ 中，若 A 的按模最大特征值的绝对值较大时，在迭代过程中 $X^{(k)}$ 的某些分量迅速增大(如例 7.2 所示)，以致当 k 充分大时，或许会超过计算机实数的值域；若 A 的按模最大特征值的绝对值较小时，$X^{(k)}$ 的分量迅速缩小，当 k 充分大时，或许会被计算机当零处理。因此，分量过大或过小都不利于计算的精确度。在实际计算中，通常采用规范运算，即对 $X^{(k)}$ 的每个元素除以 $X^{(k)}$ 按模最大的分量 $\| X^{(k)} \|_\infty = \max\limits_{1\le i\le n} |x_i^{(k)}|$。

规范运算可按下面公式进行：

$$\begin{cases} X^{(k+1)} = AY^{(k)} \\ Y^{(k+1)} = X^{(k+1)}/\| X^{(k+1)} \|_\infty \quad k = 0,1,2,\cdots \\ \text{初值 } Y^{(0)} = X^{(0)}/\| X^{(0)} \|_\infty \end{cases} \tag{7.10}$$

规范化运算保证了 $\| Y^{(k)} \|_\infty = 1$，即 $Y^{(k)}$ 按模最大分量的值保持为 1 或 -1。下面给出在规范运算中迭代序列的几种情况：

(1) 如果 $\{X^{(k)}\}$ 收敛，则 A 的特征值按模最大分量的值仅有一个，且 $\lambda_1 > 0$，对充分大的 k，按模最大分量 $x_j^{(k)}$ 不变号，这时

$$\lambda_1 \approx \frac{|x_i^{(k+1)}|}{|y_i^{(k)}|}, i = 1,2,\cdots,n$$

即 $\lambda_1 \approx \max\limits_{1\le i\le n} |x_i^{(k)}| = |x_j^{(k)}|$

相应的特征向量是 $v_1 \approx Y^{(k)}$。

(2) 如果 $\{X^{(2k)}\}$，$\{X^{(2k+1)}\}$ 分别收敛于互为反号的向量，则按模最大的特征值也仅有一个单实根，且 $\lambda_1 < 0$，即对充分大的 k，若 $x_j^{(k)}$ 的符号交替变号，则 λ_1 为负值。

$$\lambda_1 \approx -\max\limits_{1\le i\le n} |x_i^{(k)}| = -|x_j^{(k)}|$$

相应的特征向量是 $v_1 \approx Y^{(k)}$。

(3)如果 $\{X^{(2k)}\}$，$\{X^{(2k+1)}\}$ 分别收敛于两个不同的向量[与(2)不同]，则按模最大的特征值有两个，是互为反号的一对实根。这时，对充分大的 k，再作一次非规范运算则

$$X^{(k+1)} = AX^{(k)}$$

$$\lambda_1 \approx \sqrt{x_i^{(k+1)}/y_i^{(k-1)}}, \lambda_2 = -\lambda_1 \tag{7.11}$$

而仍有

$$\begin{cases} v_1 = X^{(k+1)} + \lambda_1 X^{(k)} \\ v_2 = X^{(k+1)} - \lambda_1 X^{(k)} \end{cases}$$

(4) 如果 $\{X^{(k)}\}$ 的趋势无一定的规律，这时 A 的按模最大的特征值的情况更为复杂，需要另行处理。

例 7.3：用规范运算计算矩阵 A 的按模最大的特征值和它的特征向量。

$$A=\begin{pmatrix}9 & -3\\ 4 & 1\end{pmatrix}$$

解：用表 7.3 给出计算结果。

表 7.3

k	$Y^{(k)}$		$X^{(k+1)}$	
0	1	1	6	5
1	1.	0.833333	6.5	4.83333
2	1.	0.74359	6.76923	4.74359
3	1.	0.700758	6.89773	4.70076
4	1.	0.681493	6.95552	4.18149
5	1.	0.673061	6.98081	4.67306
6	1.	0.669415	6.99176	4.66942
7	1.	0.66782	6.99654	4.66782
8	1.	0.667161	6.99852	4.66716
9	1.	0.666879	6.99936	4.66688

得到按模最大的特征值 $\lambda_1\approx 6.99936$，及其特征向量 $v_1=\begin{pmatrix}1.0\\ 0.666879\end{pmatrix}$。

例 7.4：用规范运算计算矩阵 A 的按模最大的特征值和它的特征向量。

$$A=\begin{pmatrix}4 & -1 & 1\\ 16 & -2 & -2\\ 16 & -3 & -1\end{pmatrix}$$

解：计算结果列在表 7.4 中。

表 7.4

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$y_1^{(k)}$	$y_2^{(k)}$	$y_3^{(k)}$
0	0.5	0.5	1	0.5	0.5	1
1	2.5	5	5.5	0.454545	0.909091	1
2	1.909089	3.454538	3.553628	0.537222	0.972116	1
3	2.176772	4.65132	4.679104	0.465201	0.994091	1
4	2.176772	3.455134	3.461124	0.539392	0.998269	1
5	2.159299	4.633734	4.635485	0.465721	0.999627	1
6	1.862661	3.452282	3.452655	0.539487	0.999892	1
7	2.158056	4.632000	4.632116	0.465890	0.999975	1
8	1.863585	3.454290	3.454315	0.534950	0.999993	1
9	2.157985	4.631926	4.631926	0.465893	0.999999	1
10	1.863573	3.454290	3.454291	0.539495	1	1
11	2.157980	4.631920	4.631920	0.465893	1	1
12	1.863572	3.454288	3.454288			
13	7.454288	16.	16.			

$$\lambda_1=\sqrt{x_2^{(13)}/y_2^{(11)}}=4,\lambda_2=-4$$

$$v_1=X^{(13)}+\lambda_1 X^{(12)}=(14.908576,29.817152,29.817152)$$

$$v_2=X^{(13)}-\lambda_1 X^{(12)}=(0,2.182848,2.182848)$$

7.1.3* 关于幂法的初始值

用迭代法求解非线性方程的根，初始值必需选择在根的附近，才能保证迭代收敛；用迭代法求解线性方程组，可任选初始值。在幂法计算中，一般认为初始值 $X^{(0)}$ 是任取的，其实从理论上讲也隐含了条件。由特征值和特征向量的定义要求 $X^{(0)}$ 不能是零向量，在幂法中，设 $X^{(0)} = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n$，要求 $\alpha_1 \neq 0$，然而在迭代计算中，由于舍入误差的影响，即使初始值 $X^{(0)}$ 按 $v_1, v_2, \cdots, v_n$ 分解时，v_1 的系数 $\alpha_1 = 0$，对充分大的 k，$X^{(k)}$ 按 $v_1, v_2, \cdots, v_n$ 分解时，v_1 的系数也不是零，不过这时需要的迭代次数可能很大。

7.2 反 幂 法

反幂法是计算矩阵按模最小的特征值以及相应的特征向量的数值方法。

设矩阵 A 可逆，λ 和 v 分别为 A 的特征值以及相应的特征向量，并设 $\lambda_i \neq 0, i = 1, 2, \cdots, n$。对 $Av = \lambda v$，两边同乘 A^{-1}，得 $A^{-1}v = \frac{1}{\lambda}v$，可见 A 和 A^{-1} 的特征值互为倒数，而且 v 也是 A^{-1} 的特征值 $\frac{1}{\lambda}$ 的特征向量。A^{-1} 的按模最大的特征值正是 A 的按模最小的特征值的倒数，用幂法计算 A^{-1} 按模最大的特征值而得到 A 的按模最小的特征值的方法，称为反幂法。

用幂法计算 A^{-1} 按模最大的特征值仍可用规范方法：任取 $X^{(0)}$，规范迭代

$$\begin{cases} Y^{(k)} = X^{(k)} / \| X^{(k)} \|_\infty \\ X^{(k+1)} = A^{-1} Y^{(k)} \end{cases}, k = 0, 1, \cdots \tag{7.12}$$

在实际计算中不是先算出 A^{-1}，再作乘积 $A^{-1}Y^{(k)}$，而是解方程 $AX^{(k+1)} = Y^{(k)}$，求得 $X^{(k+1)}$。由于需要反复求解，一般不用高斯消元法，而用直接分解法解方程。将 A 分解为 $A = LU$，记 $UX^{(k+1)} = Z^{(k+1)}$，由 $LZ^{(k+1)} = Y^{(k)}$ 解出 $Z^{(k+1)}$，再由 $UX^{(k+1)} = Z^{(k+1)}$ 解出 $X^{(k+1)}$。

例 7.5：用规范运算计算矩阵 A 的按模最小的特征值和它的特征向量。

$$A = \begin{pmatrix} 9 & -3 \\ 4 & 1 \end{pmatrix}$$

解：计算结果列在表 7.5 中。

表 7.5

k	$Y^{(k)}$		$X^{(k+1)}$	
0	1	1	0.1904766	0.238095
1	0.8	1.	0.180952	0.27619
2	0.655172	1.	0.174056	0.303777
3	0.572973	1	0.170142	0.319434
4	0.532636	1.	0.168221	0.327117
5	0.514253	1.	0.167345	0.330618
6	0.514253	1.	0.16696	0.33216
7	0.502649	1.	0.166793	0.332829
8	0.501137	1.	0.166721	0.333117

由表可知 $\mu = 0.333117$　　为 A^{-1} 的按模最大的特征值。

故　　$\lambda_2 = 1/0.333117 = 3.0019$　　为 A 的按模最小的特征值。

　　$v_2 = \{0.501137, 1\}$　　为其近似的特征向量。

7.3　实对称矩阵的 Jacobi 方法

在客观世界中,矩阵的特征值都有它的物理意义。不同类型矩阵特征值的分布特点也不相同。对称矩阵是一类常见的矩阵,对称矩阵的一些良好特性给计算它的全部特征值提供了便利的条件。Jacobi 方法是计算对称矩阵全部特征值的方法。

我们注意到以下事实:

(1)若 A 为 n 阶对角阵,$A = \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_n \end{pmatrix}$,则 $a_1, a_2, \cdots, a_n$ 就是 A 的 n 个特征值;

(2)若 A 为 n 阶矩阵,P 为任意 n 阶可逆矩阵,则称 A 与 $P^{-1}AP$ 相似,相似矩阵具有相同的特征值;

(3)若 A 为 n 阶对称矩阵,则存在正交矩阵 Q,使 $Q^TAQ = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$,即对称矩阵正交相似于一个对角阵。

但是,对于给定的对称矩阵 A,寻求正交矩阵 Q,使 $Q^TAQ = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$ 是件十分困难的事情。可以构造一系列特殊形式的正交矩阵 $Q_1, Q_2, \cdots, Q_k$;对 A 作若干次正交相似变换,使矩阵的非对角线元素比重逐次减少,同时使矩阵的对角线元素比重逐次增大。一直到每个非对角线元素已小得无足轻重时,其对角元素即可看作 A 的特征值。

Jacobi 方法就是实现以上思想的一种数值方法,它通过一系列平面旋转变换(也是正交变换),逐渐减弱非对角线元素的比重。计算对称矩阵 A 全部特征值的 Jacobi 方法是构造一系列的正交矩阵 Q_k 的过程。

例 7.6　计算矩阵 $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ 的特征值和相应的特征向量。

解:记 $B = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$,有

$$B^TAB = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$= \begin{pmatrix} -2\sin\theta\cos\theta & \cos^2\theta - \sin^2\theta \\ \cos^2\theta - \sin^2\theta & 2\sin\theta\cos\theta \end{pmatrix}$$

做旋转变换的目的是要 $\cos^2\theta - \sin^2\theta = 0$。若取 $\theta = \frac{\pi}{4}$，这时

$$\cos\theta = \sin\theta = \frac{\sqrt{2}}{2}, \qquad B = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

而 $B^TAB = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

得到 A 的特征值 $\lambda_1 = 1, \lambda_2 = -1$，相应的特征向量为

$$v_1 = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix}, \qquad v_2 = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$$

上例中 B 是一个二阶正交矩阵，事实上是一个平面旋转变换矩阵，称为 Givens 矩阵。对任意一个二阶对称矩阵，只需适当选取 θ，就可通过正交相似变换把矩阵化为对角矩阵。对高于二阶的对称矩阵，情况远为复杂。

记 $Q(p,q,\theta) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \cos\theta & & \sin\theta & & \\ & & & 1 & & & \\ & & & & \ddots & & \\ & & & & & 1 & \\ & & -\sin\theta & & \cos\theta & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$ → p 行，→ q 行

$Q(p,q,\theta)$是一个正交矩阵，称为 Givens 旋转变换。下面分析 Givens 变换作用到对称矩阵后正交相似的变换效果。

记

$A = (a_{ij})$

$B = Q^T(p,q,\theta)AQ(p,q,\theta) = (b_{ij})$

$A = (A_1, A_2, \cdots, A_p, \cdots, A_q, \cdots, A_n)$，其中 $A_k = \begin{pmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{nk} \end{pmatrix}$

$$Q^T(p,q,\theta)=\begin{pmatrix} e_1 \\ \vdots \\ e_p \\ \vdots \\ e_q \\ \vdots \\ e_n \end{pmatrix}$$

$$e_i=(0,\cdots,0,\overset{i}{1},0,\cdots,0),\ i\neq p,q$$

$$e_p=(0,\cdots,0,\underset{p}{\cos\theta},0,\cdots,0,\underset{q}{-\sin\theta},0,\cdots,0)$$

$$e_q=(0,\cdots,0,\underset{p}{\sin\theta},0,\cdots,0,\underset{q}{\cos\theta},0,\cdots,0)$$

$$\widetilde{B}=Q^T(p,q,\theta)A=\begin{pmatrix} e_1 \\ \vdots \\ e_p \\ \vdots \\ e_q \\ \vdots \\ e_n \end{pmatrix}(A_1\cdots A_p\cdots A_q\cdots A_n)$$

$$=\begin{pmatrix} e_1A_1 & \cdots & e_1A_p & \cdots & e_1A_q & \cdots & e_1A_n \\ & \cdots & & \cdots & & & \\ e_pA_1 & \cdots & e_pA_p & \cdots & e_pA_q & \cdots & e_pA_n \\ & \cdots & & \cdots & & & \\ e_qA_1 & \cdots & e_qA_p & \cdots & e_qA_q & \cdots & e_qA_n \\ & \cdots & & \cdots & & & \\ e_nA_1 & \cdots & e_nA_p & \cdots & e_nA_q & \cdots & e_nA_n \end{pmatrix}$$

其中：$e_iA_j=(0\ \cdots\ 1\ \cdots\ 0)\begin{pmatrix} a_{1j} \\ \vdots \\ a_{ij} \\ \vdots \\ a_{nj} \end{pmatrix}=a_{ij},\ i\neq p,q$

$$e_pA_j=(0\ \cdots\ \cos\theta\ \cdots\ -\sin\theta\ \cdots\ 0)\begin{pmatrix} a_{1j} \\ \vdots \\ a_{ij} \\ \vdots \\ a_{nj} \end{pmatrix}=a_{pj}\cos\theta-a_{qj}\sin\theta$$

$$e_q A_j = (0 \cdots \sin\theta \cdots \cos\theta \cdots 0)\begin{pmatrix} a_{1j} \\ \vdots \\ a_{ij} \\ \vdots \\ a_{nj} \end{pmatrix} = a_{pj}\sin\theta + a_{qj}\cos\theta$$

可以看到 Q^T 左乘 A 的作用结果是 A 的 p 行和 q 行元素有所变化，其余行的元素依然如故；Q 右乘 Q^TA 的作用结果是 Q^TA 的 p 列和 q 列元素有所变化，其中 p、q 的行和列的交叉元素 a_{pq}，a_{qp}，a_{pp}，a_{qq} 被作用两次。由 Q^TAQ 的对称性容易得到(证明略)下列公式：

$$\begin{cases} b_{ip} = b_{pi} = a_{pi}\cos\theta - a_{qi}\sin\theta, i \neq p, q \\ b_{iq} = b_{qi} = a_{pi}\sin\theta + a_{qi}\cos\theta, i \neq p, q \\ b_{pp} = a_{pp}\cos^2\theta + a_{qq}\sin^2\theta - a_{pq}\sin 2\theta \\ b_{qq} = a_{pp}\sin^2\theta + a_{qq}\cos^2\theta + a_{pq}\sin 2\theta \\ b_{pq} = b_{qp} = a_{pq}\cos 2\theta + \dfrac{a_{pp} - a_{qq}}{2}\sin 2\theta \end{cases} \tag{7.13}$$

利用 Civens 变换的目的是要 $b_{pq}=0, b_{qp}=0$，即

$$a_{pq}\cos 2\theta + \frac{a_{pp} - a_{qq}}{2}\sin 2\theta = 0$$

记 $s = \dfrac{a_{qq} - a_{pp}}{2a_{pq}}$，$t = \mathrm{tg}\theta$，由 b_{pq} 的表达式(7.13)可知，当 t 取

$$t = \begin{cases} t^2 + 2st - 1 = 0 \text{ 的按模较小根}, s \neq 0 \\ 1, \qquad\qquad\qquad s = 0 \end{cases} \tag{7.14}$$

可使 $b_{pq}=0, b_{qp}=0$，即当 $a_{pp} \neq a_{qq}$ 时，$\mathrm{tg}\theta$ 取方程 $t^2 + 2st - 1 = 0$ 的按模最小根；当 $a_{pp} = a_{qq}$ 时，取 $\mathrm{tg}\theta = 1, \theta = \dfrac{\pi}{4}$。

由 $t = \mathrm{tg}\theta$，可得

$$\begin{cases} \cos\theta = \dfrac{1}{\sqrt{1 + t^2}} \\ \sin\theta = \dfrac{t}{\sqrt{1 + t^2}} \end{cases}$$

记 $\begin{cases} c = \cos\theta \\ d = \sin\theta \end{cases}$，则当 t 按式(7.14)取值时，式(7.13)化简成

$$\begin{cases} b_{ip} = b_{pi} = ca_{pi} - da_{qi}, i \neq p, q \\ b_{iq} = b_{qi} = da_{pi} + ca_{qi}, i \neq p, q \\ b_{pp} = a_{pp} - ta_{pq} \\ b_{qq} = a_{pp} + ta_{pq} \\ b_{pq} = b_{qp} = 0 \\ b_{ij} = a_{ij} \quad i \neq p, q; j \neq p, q \end{cases} \tag{7.15}$$

由式(7.15)，容易证明

$$\sum_{i\neq j} b_{ij}^2 = \sum_{i\neq j} a_{ij}^2 - 2a_{pq}^2$$

$$\sum_{i=1}^{n} b_{ii}^2 = \sum_{i=1}^{n} a_{ii}^2 + 2a_{pq}^2$$

即 B 的非对角元素比重小于 A 的非对角元素比重。

Jacobi 方法就是对 A 连续施行以上变换的方法。如果选取 p,q 使 $|a_{pq}| = \max_{i\neq j}|a_{ij}|$，那么实施以上变换的效率将更高。

不妨记 $A=(a_{ij})=A^{(0)}=(a_{ij}^{(0)})$，将按式(7.14)和(7.15)计算得到的矩阵记为 B，$B=(b_{ij})=A^{(1)}=(a_{ij}^{(1)})$。再对 $A^{(1)}$ 实施类似的变换，得到 $A^{(2)}$，继续做下去，得到正交相似序列 $A^{(0)},A^{(1)},A^{(2)},\cdots$，而且后面矩阵的非对角元素比重均小于前面矩阵的非对角元素比重。若任给误差控制量 $\varepsilon>0$，可以证明，必存在充分大的 k，使得 $\sum_{i\neq j}(a_{ij}^{(k)})^2 < \varepsilon$，这时 $A^{(k)}$ 的对角元素 $a_{ii}^{(k)}, i=1,2,\cdots,n$ 可视为 A 的特征值。

Jacobi 算法描述

计算目标：给定控制精度 ε，计算 n 阶对称方阵 $A=(a_{ij})$ 的全部特征值。

1：输入：矩阵阶数 n，对称方阵 $A=(a_{ij})$ 的元素，控制精度 ε。

2：　WHILE $\sum_{i\neq j} a_{ij}^2 > \varepsilon$

2.1　选取非对角线按模最大元素 $|a_{pq}| = \max_{i\neq j}|a_{ij}|$

2.2　确定旋转角度 θ，其中 $t=\mathrm{tg}\theta$

$$s=\frac{a_{qq}-a_{pp}}{2a_{pq}},$$

IF $s=0$ THEN $t=1$

ELSE $\{t1=-s-\sqrt{s^2-1}, t2=-s+\sqrt{s^2-1}\}$

IF $|t1|>|t2|$ THEN $t=t2$

ELSE $t=t1$ ENDIF

ENDIF

$$\begin{cases} c=\dfrac{1}{\sqrt{1+t^2}} & ! \ c=\cos\theta \\ d=\dfrac{t}{\sqrt{1+t^2}} & ! \ d=\sin\theta \end{cases}$$

2.3　计算 Q^TAQ 的 p,q 行和 p,q 列的元素；计算 Q^TAQ 的对角元素 a_{pp}, a_{qq}，并将 Q^TAQ 存放在 A 中。

FOR　$i=1,2,\cdots,n(i\neq p,q)$

$\{a_{ip}=a_{pi} \leftarrow ca_{pi}-da_{qi}$

$a_{iq}=a_{qi} \leftarrow ca_{qi}+da_{pi}\}$

$a_{pp} \leftarrow a_{pp}-ta_{pq}$

$a_{qq} \leftarrow a_{qq}+ta_{pq}$

ENDWHILE

3:输出 A 的特征值 $a_{ii}=\lambda_i, i=1,2,\cdots,n$。

上列算法还可以进一步细化,例如,给出选取非对角线按模最大元素 $|a_{pq}|=\max\limits_{i\neq j}|a_{ij}|$ 具体的策略。在做要 a_{pq} 为零的旋转变换中,常会使原为零的非对角线元素 a_{uv} 变为非零元素。因此,当阶数大于 2 时,一般不可能通过 Jacobi 方法得到纯对角矩阵。

Jacobi 方法求得的计算结果精度一般都比较高,特征向量的正交性也较好,它的缺点是当计算稀疏矩阵的特征值时,旋转以后常不能保持原稀疏的性质,因此,Jacobi 方法一般用于阶不很高的"满矩阵"情形。

例 7.7:用 Jacobi 方法计算矩阵 $A=\begin{pmatrix}3&1&2\\1&3&4\\2&4&6\end{pmatrix}$ 的全部特征值。

解:取 $p=1,q=2;s=\dfrac{a_{22}-a_{11}}{2a_{12}}=\dfrac{3-3}{2}=0;t=1$

$$c=\sqrt{2}/2,d=\sqrt{2}/2$$

$$\begin{cases}a_{13}^{(1)}=a_{31}^{(1)}=a_{13}c-a_{23}d=-1.4142\\a_{23}^{(1)}=a_{32}^{(1)}=a_{23}c+a_{13}d=4.2426\\a_{11}^{(1)}=a_{11}-ta_{12}=2\\a_{22}^{(1)}=a_{22}+ta_{12}=4\\a_{33}^{(1)}=a_{33}=6\\a_{12}^{(1)}=a_{21}^{(1)}=0\end{cases}$$

即 $A^{(1)}=Q^TAQ=\begin{pmatrix}c&-s&0\\s&c&0\\0&0&1\end{pmatrix}A\begin{pmatrix}c&s&0\\-s&c&0\\0&0&1\end{pmatrix}=\begin{pmatrix}2&0&-1.4142\\0&4&4.2426\\-1.4142&4.2426&6\end{pmatrix}$

取 $p=2,q=3;s=\dfrac{a_{33}-a_{22}}{2a_{23}}=\dfrac{6-4}{2\times4.2426}=0.2357$

$$t=-s+\sqrt{s^2+1}=0.7917$$

$$c=1/\sqrt{1+t^2}=0.7840,d=t/\sqrt{1+t^2}=0.6207$$

即 $A^{(2)}=Q^TA^{(1)}Q=\begin{pmatrix}c&-s&0\\s&c&0\\0&0&1\end{pmatrix}A^{(1)}\begin{pmatrix}c&s&0\\-s&c&0\\0&0&1\end{pmatrix}$

$$=\begin{pmatrix}2&0.8778&-1.1088\\0.8778&0.6411&0\\-1.1088&0&9.3589\end{pmatrix}$$

第一步将 $p=1,q=2$ 位置的元素化为零,第二步将 $p=2,q=3$ 位置的元素化为零后 $p=1,q=2$ 位置的元素 $a_{12}^{(2)}$ 又变为非零元素了,但是 $|a_{12}^{(2)}|$ 比 $|a_{12}^{(0)}|$ 的数值小。继续做下去可求出 A 的特征值为:9.52,2.29,0.183。

如果按选取非对角线按模最大元素 $|a_{pq}|=\max\limits_{i\neq j}|a_{ij}|$ 的策略,那么,第一步选取 $p=2$,$q=3$ 位置的元素,将 $a_{23}^{(1)}$ 和 $a_{32}^{(1)}$ 化为零。通常,以按模最大元素的顺序进行旋转变换,其

计算速度会更快一些。

7.4 程序示例

程序 14 用幂法和反幂法计算矩阵按模最大和按模最小的特征值及其特征向量。

算法描述

输入矩阵 A,利用幂法

$$\begin{cases} x^{(k+1)} = Ay^{(k)} \\ y^{(k+1)} = x^{(k+1)} \Big/ \| x^{(k+1)} \|_\infty \end{cases} \quad k = 0,1,2,\cdots$$

求矩阵 A 的按模最大的特征值 $\max\limits_{1 \le i \le n} | x_i^{(k)} |$ 及其特征向量 $v = y^{(k)}$(k 足够大);利用反幂法求得其按模最小的特征值及其特征向量(即求矩阵 A^{-1} 的按模最大的特征值)。其中 $x^{(k+1)} = A^{-1}y^{(k)}$ 是用直接分解法:用公式 $LUx^{(k+1)} = y^{(k)}$ 计算 $Ax^{(k+1)} = y^{(k)}$。

程序源码

```
////////////////////////////////////////////////////////////////////////////////
// Purpose: 用幂法和反幂法计算矩阵按模最大和按模最小的特征值及其特征向量。//
////////////////////////////////////////////////////////////////////////////////
# include < stdio.h >
# include < math.h >
# define MAX _ N 20            //方程的最大维数
# define MAXREPT 100
# define epsilon 0.00001       //求解精度
int main( )
{ int n;
  int i, j, k;
  double err;
  double xmax, oxmax;
  static double a[MAX _ N][MAX _ N];
  static double l[MAX _ N][MAX _ N], u[MAX _ N][MAX _ N];
  static double x[MAX _ N], ox[MAX _ N];
  printf(" \ nInput n value(dim of Ax = c): ");      //输入方程的维数
  scanf("%d", &n);
  if (n > MAX _ N)
  {
    printf("The input n is larger then MAX _ N, please redefine the MAX _ N. \ n");
    return 1;
  }
  if (n < = 0)
  {
    printf("Please input a number between 1 and %d. \ n", MAX _ N);
    return 1;
  }
```

```
                                    //输入矩阵 A
printf("Now input the matrix a(i,j), i,j = 0,...,%d: \n", n-1);
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
   scanf("%lf", &a[i][j]);
for (i=0; i<n; i++) x[i]=1;
oxmax=0;
for (i=0; i<MAXREPT; i++)
{
   for (j=0; j<n; j++)                              //幂乘
   {
   nx[j]=0;
   for (k=0; k<n; k++)
      nx[j]+=a[j][k]*x[k];
   }
   xmax=0.0;
   for (j=0; j<n; j+-)                              //规范化
   if (fabs(nx[j])>xmax) xmax=fabs(nx[j]);
   for (j=0; j<n; j++)
   nx[j]/=xmax;
   for (j=0; j<n; j++)
   x[j]=nx[j];
   if (fabs(xmax-oxmax) < epsilon)
   {
   printf("Solve... max lamda=%lf \n", xmax);          //输出
   printf("The vector is: \n");
   for (i=0; i<n; i++) printf("%f \n", nx[i]);
   break;
// return 0;
    }
      oxmax=xmax;
  }
// printf("After %d repeat, max no result... \n", MAXREPT); //输出
  for (i=0; i<n; i++) u[i][i]=1;        //U 矩阵对角元为 1
  for (k=0; k<n; k++)
  {
    for (i=k; i<n; i++)                 //计算 L 矩阵
    {
    l[i][k]=a[i][k];
    for (j=0; j<=k-1; j++)
      l[i][k]-=(l[i][j]*u[j][k]);
    }
    for (j=k+1; j<n; j++)               //计算 U 矩阵
```

```
    }
    u[k][j] = a[k][j];
    for (i = 0; i < = k - 1; i + + )
      u[k][j] - = (l[k][i] * u[i][j]);
    u[k][j]/ = l[k][k];
    }
  }
for (i = 0; i < n; i + + ) x[i] = 1;
oxmax = 0;
for (i = 0; i < MAXREPT; i + + )
  {
    for (j = 0; j < n; j + + )                    //反幂乘
    {
    nx[j] = x[j];
for (k = 0; k < = j - 1; k + + )
    nx[j] - = l[j][k] * nx[k];
nx[j]/ = l[j][j];
  }
for (j = n - 1; j > = 0; j - - )
  { x[j] = nx[j];
for (k = j + 1; k < n; k + + )
    x[j] - = u[j][k] * x[k];
  }
xmax = 0.0;
for (j = 0; j < n; j + + )                        //规范化
if (fabs(x[j]) > xmax) xmax = fabs(x[j]);
for (j = 0; j < n; j + + )
x[j]/ = xmax;
if (fabs(xmax - oxmax) < epsilon)
{
printf("Solve... min lamda = %lf \ n", 1/xmax);      //输出
printf("The vector is: \ n");
      for (i = 0; i < n; i + + ) printf("%f \ n", x[i]);
      break;
//    return 0;
      }
      oxmax = xmax;
    }
    return 1;
  }
```

计算实例

用规范运算计算矩阵 $A = \begin{pmatrix} 9 & -3 \\ 4 & 1 \end{pmatrix}$ 的按模最大的特征值和它的特征向量以及按模

最小的特征值和它的特征向量。

程序输入输出

```
Input n value(dim of aX = c): 2
Now input the matrix a(i,j), i,j = 0,...,2:
9 -3 4 1
Solve... max lamda = 6.999996          (按模最大的特征值 6.999996 )
The vector is:
1.000000
0.666667                               (特征向量(1.000000,0.666667)^T)
Solve... min lamda = 3.000066          (按模最小的特征值 3.000066 )
The vector is:
0.500016
1.000000                               (特征向量(0.500016,1.000000)^T)
```

习题 7

7.1 用幂法计算下列矩阵按模最大的特征值和相应的特征向量：

$$(1)A=\begin{pmatrix}5 & -3\\ -6 & -2\end{pmatrix} \qquad (2)B=\begin{pmatrix}1 & 2\\ 4 & 1\end{pmatrix}$$

$$(3)C=\begin{pmatrix}-4 & 4 & 1\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \qquad (4)D=\begin{pmatrix}3 & 0 & 2\\ 1 & 0 & -1\\ 0 & 1 & 5\end{pmatrix}$$

7.2 用幂法计算下列矩阵按模最小的特征值和相应的特征向量：

$$(1)A=\begin{pmatrix}7 & 3\\ 3 & 1\end{pmatrix} \qquad (2)B=\begin{pmatrix}5 & 3\\ -2 & 0\end{pmatrix}$$

$$(3)C=\begin{pmatrix}2 & 0 & 2\\ 1 & 0 & -1\\ 0 & 1 & 5\end{pmatrix} \qquad (4)D=\begin{pmatrix}10 & 0 & 2\\ 2 & 3 & 0\\ 0 & 2 & 1\end{pmatrix}$$

7.3 用 Jacobi 方法计算下列矩阵全部特征值：

$$(1)A=\begin{pmatrix}3 & 1\\ 1 & 5\end{pmatrix} \qquad (2)B=\begin{pmatrix}4 & -3\\ -3 & 4\end{pmatrix}$$

$$(3)C=\begin{pmatrix}1 & -1 & 0\\ -1 & 2 & 2\\ 0 & 2 & 3\end{pmatrix} \qquad (4)D=\begin{pmatrix}2 & -1 & 3\\ -1 & 5 & 0\\ 3 & 0 & 1\end{pmatrix}$$

第 8 章　常微分方程数值解

物体冷却过程的数学模型可用下式表示：

$$\frac{du}{dt} = -k(u - u_0)$$

它含有自变量 t、未知函数 u 以及它的一阶导数 $\frac{du}{dt}$，是一个常微分方程。在微分方程中我们称自变量函数只有一个的微分方程为常微分方程，自变量函数个数为两个或两个以上的微分方程为偏微分方程。给定微分方程及其初始条件，称为初值问题；给定微分方程及其边界条件，称为边值问题。

本章主要讨论常微分方程的初值问题：

$$\begin{cases} y'(x) = f(x,y) \\ y(a) = y_0 \end{cases}, (a \leq x \leq b) \tag{8.1}$$

或记为

$$\begin{cases} \frac{dy}{dx} = f(x,y) \\ y(a) = y_0 \end{cases}, (a \leq x \leq b)$$

只有一些特殊形式的 $f(x,y)$，才能找到它的解析解；对于大多数常微分方程的初值问题，只能计算它的数值解。常微分方程初值问题的数值解是求 $y(x)$ 在求解区间 $[a,b]$ 上剖分点列 x_n，$n = 1,2,\ldots,m$ 的数值解 y_n。在计算中约定 $y(x_n)$ 表示常微分方程准确解的值，y_n 表示 $y(x_n)$ 的近似值。

8.1　欧拉(Euler)公式

8.1.1　基于数值微商的欧拉公式

对于常微分方程初值问题[式(8.1)]，在求解区间 $[a,b]$ 上作等距分割的剖分，步长 $h = \frac{b-a}{m}$，记 $x_n = x_{n-1} + h, n = 1,2,\cdots,m$。用数值微商的方法，即用差商近似导数计算常微分方程。

用向前差商近似 $y'(x)$

做出 $y(x)$ 的在 $x = x_0$ 处的一阶向前差商式：

$$y'(x_0) \approx \frac{y(x_1) - y(x_0)}{h}$$

又 $y'(x_0) = f(x_0, y(x_0))$，于是得到

$$\frac{y(x_1) - y(x_0)}{h} \approx f(x_0, y(x_0))$$

而 $y(x_1)$ 的近似值 y_1 可按

$$\frac{y_1 - y_0}{h} = f(x_0, y_0) \text{ 或 } y_1 = y_0 + hf(x_0, y_0)$$

求得。类似地，由

$$y'(x_n) \approx \frac{y(x_{n+1}) - y(x_n)}{h} \text{ 以及 } y'(x_n) = f(x_n, y(x_n))$$

得到计算 $y(x_{n+1})$近似值 y_{n+1}的向前欧拉公式：

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{8.2}$$

由 y_n 直接算出 y_{n+1}值的计算格式称为显式格式，向前欧拉公式是显式格式。

欧拉方法的几何意义

以 $f(x_0, y_0)$为斜率，通过点(x_0, y_0)做一条直线，它与直线 $x = x_1$ 的交点就是 y_1。依此类推，y_{n+1}是以 $f(x_n, y_n)$为斜率过点(x_n, y_n)的直线与直线 $x = x_n$ 的交点。欧拉法也称为欧拉折线法，如图 8.1 所示。

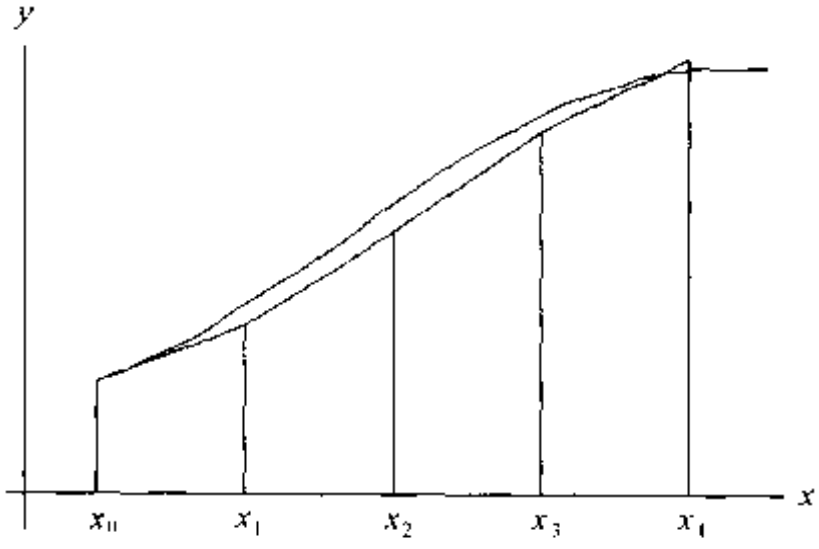


图 8.1　欧拉折线法

例 8.1：假定某公司的净资产因资产本身产生了利息而以 4% 的年利率增长，同时，该公司以每年 100 万的数额支付职工工资。净资产的微分方程：

$$\frac{dw}{dt} = 0.04w - 100 \quad (t \text{ 以年为单位})$$

分别以初始值 $x(0) = 1500$ 万，$y(0) = 2500$ 万，$z(0) = 3500$ 万，用欧拉公式预测公司 24 年后的净资产趋势。

解：$w_{n+1} = w_n + h(0.04w_n - 100) = 1.04w_n - 100, h = 1$。

w_0 分别以 $x_0 = 1500, y_0 = 2500, z_0 = 3500$ 代入，计算结果见表 8.1。

表 8.1

n	x_n	y_n	z_n	n	x_n	y_n	z_n
1	1460	2500.	3540	5	1283.35	2500.	3716.65
2	1418.4	2500.	3581.6	6	1234.68	2500.	3765.32
3	1375.14	2500.	3624.86	7	1184.07	2500.	3815.93
4	1330.14	2500.	3669.86	8	1131.43	2500.	3868.57

续表

n	x_n	y_n	z_n	n	x_n	y_n	z_n
9	1076.69	2500.	3923.31	17	552.1	2500.	4447.9
10	1019.76	2500.	3980.24	18	474.183	2500.	4525.82
11	960.546	2500.	4039.45	19	393.151	2500.	4606.85
12	898.968	2500.	4101.03	20	308.877	2500.	4691.12
13	834.926	2500.	4165.07	21	221.232	2500.	4778.77
14	768.324	2500.	4231.68	22	130.081	2500.	4869.92
15	699.056	2500.	4300.94	23	35.2845	2500.	4964.72
16	627.019	2500.	4372.98	24	-63.3042	2500.	5063.3

从表 8.1 可以看到当利息赢利低于工资的支出,公司的净资产逐年减少,以至净资产为负值;当利息赢利与工资的支出平衡时,公司的净资产每年保持不变;当利息赢利超过工资的支出,公司的净资产稳步增长。在图 8.2 中 $L1, L2, L3$ 分别表示初始值 3500,2500 和 1500 的三条净资产趋势曲线。

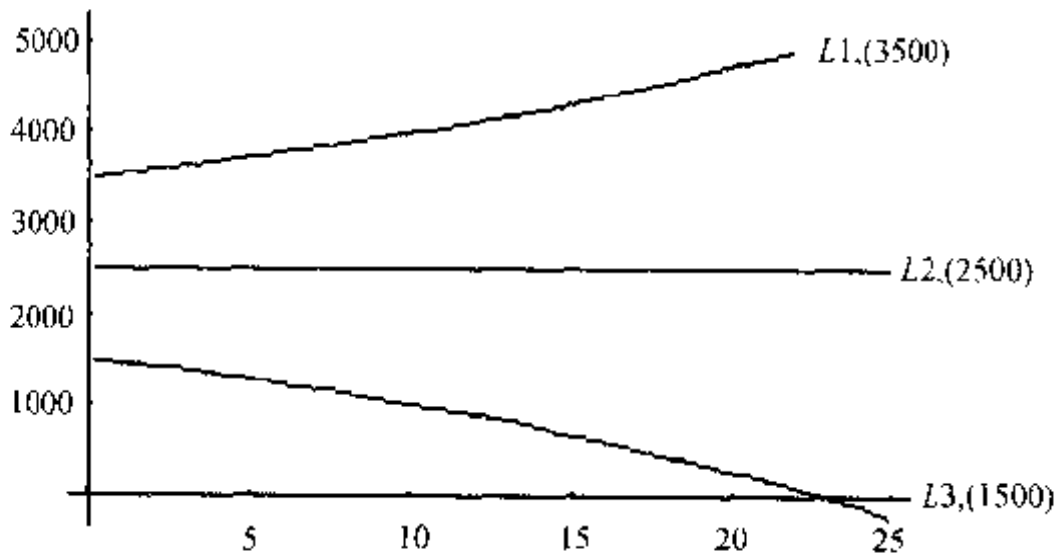


图 8.2　三种初始值的净资产趋势

用向后差商近似 $y'(x)$

做出 $y(x)$的在 $x = x_0$ 处的一阶向后差商式:

$$y'(x_1) \approx \frac{y(x_1) - y(x_0)}{h}$$

又 $y'(x_1) = f(x_1, y(x_1))$,得到 $y(x_1)$ 的近似值 y_1 的计算公式:

$$y_1 = y_0 + hf(x_1, y_1)$$

类似地,可得到计算 $y(x_{n+1})$近似值 y_{n+1}的计算公式:

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \tag{8.3}$$

公式(8.3)称为向后欧拉公式。通常 $f(x, y)$为 y 的非线性函数,因此式(8.3)是关于 y_{n+1}的非线性方程,称为隐式欧拉公式,需要通过迭代法求得 y_{n+1}。其中,初始值 $y_{n+1}^{(0)}$可由向前欧拉公式提供。

最简单的迭代公式为:

$$\begin{cases} y_{n+1}^{(0)} = y_n + hf(x_n, y_n) \\ y_{n+1}^{(k+1)} = y_n + hf(x_{n+1}, y_{n+1}^{(k)}) \end{cases}, k = 0,1,2,\ldots, \text{直到} \left| y_{n+1}^{(k+1)} - y_{n+1}^{(k)} \right| < \text{给定精度}$$

可以证明，h 充分小时，以上迭代收敛。事实上，记 $\phi(y) = y_n + hf(x_{n+1}, y)$，则

$$\phi'(y) = hf_y(x_n, y)$$

h 充分小时，可以保证 $|hf_y(x_n, y)| \leq hL < 1$，其中 L 为李普希兹条件。

用中心差商近似 $y'(x)$

做出 $y(x)$的在 $x = x_1$ 处的中心差商式：

$$y'(x_1) \approx \frac{y(x_2) - y(x_0)}{2h}$$

又 $y'(x_1) = f(x_1, y(x_1))$，可得到 $y(x_2)$ 的近似值 y_2 的计算公式：

$$y_2 = y_0 + 2hf(x_1, y_1)$$

类似地，可得到计算 $y(x_{n+1})$ 近似值 y_{n+1} 的计算公式：

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n) \tag{8.4}$$

公式(8.4)称为中心格式。按公式(8.4)，需要知道 y_{n-1}, y_n 的值才能求得 y_{n+1}的值。因此，要先用其它公式算出 y_1，再用中心格式算出 $y_2, y_3, \cdots$。

8.1.2* 欧拉公式的收敛性

局部截断误差

对 $y(x_{n+1})$在 x_n 做泰勒展开：

$$y(x_{n+1}) = y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(\xi_n), x_n \leq \xi_n \leq x_{n+1} \tag{8.5}$$

欧拉公式 (8.2)是由以上展开式中截断 $\frac{h^2}{2!}y''(\xi_n)$ 而得，故

$$T_{n+1} = \frac{h^2}{2!}y''(\xi_n) \tag{8.6}$$

称为欧拉公式的截断误差或称局部截断误差。

若 $y(x)$为线性函数，则 $y''(x) \equiv 0$，这时局部截断误差为 0，由欧拉公式得到的解为精确解，故欧拉公式是一阶方法。

由于欧拉公式(8.2)是由式(8.5)截断 $\frac{h^2}{2!}y''(\xi_n)$ 而得，故 y_{n+1}只是 $y(x_{n+1})$的近似值。记 $e_{n+1} = y(x_{n+1}) - y_{n+1}$，它是由截断引起的解误差，我们将估计这一误差。

由微分方程理论，为保证微分方程解的存在唯一性及稳定性，通常 $f(x, y)$对 y 应满足李普希兹条件，即对任意 $y, \bar{y}$，存在 $L > 0$，$f(x, y)$满足

$$|f(x, y) - f(x, \bar{y})| < L | y - \bar{y} |$$

于是由式(8.5)与(8.2)两式相减得到

$$e_{n+1} = y(x_{n+1}) - y_{n+1} = y(x_n) - y_n + h[f(x_n, y(x_n)) - f(x_n, y_n)] + \frac{h^2}{2}y''(\xi_n)$$

曾记 $T_{n+1} = \frac{h^2}{2!}y''(\xi_n)$，故有

$$|e_{n+1}| \leq |e_n| + h|f(x_n, y(x_n)) - f(x_n, y_n)| + |T_{n+1}| \leq |e_n| + hL|e_n| + |T_{n+1}|$$

记 $T = \max_k |T_k| = O(h^2)$，则有

$$|e_{n+1}| \leq (1 + Lh)|e_n| + T$$

因此有

$$\begin{aligned}
|e_{n+1}| &\leq (1 + Lh)[(1 + Lh)|e_{n-1}| + T] + T \leq \cdots \\
&\leq (1 + Lh)^{n+1}|e_0| + T + (1 + Lh)T + (1 + Lh)^2 T + \cdots + (1 + Lh)^n T \\
&= (1 + Lh)^{n+1} \cdot e_0| + \frac{1 - (1 + Lh)^{n+1}}{1 - (1 + Lh)} T \\
&< (1 + Lh)^{n+1}|e_0| + \frac{(1 + Lh)^{n+1}}{Lh} T \\
&= (1 + Lh)^{n+1}\left(|e_0| + \frac{T}{Lh}\right)
\end{aligned}$$

对 $z > 0$，由公式 $(1 + z)^n \leq e^{nz}$，最终可得：

$$|e_{n+1}| \leq e^{(n+1)Lh}\left(|e_0| + \frac{T}{Lh}\right) \leq e^{L(b-a)}\left(|e_0| + \frac{T}{Lh}\right)$$

其中 $e^{L(b-a)}|e_0|$ 由原始误差引起，当初始值为精确值时，这一项的值是 0；$e^{L(b-a)}\frac{T}{Lh}$ 仍是由截断引起，由于 $T = O(h^2)$，当 $h \to 0$ 时，$e^{L(b-a)}\frac{T}{Lh} \to 0$，即欧拉公式是收敛的。

8.1.3 基于数值积分的近似公式

对常微分方程(8.1)，即 $\frac{dy}{dx} = f(x, y)$ 两边在区间 $[x_n, x_{n+1}]$ 上积分得：

$$y(x_{n+1}) - y(x_n) = \int_{x_n}^{x_{n+1}} f(x, y)\,dx$$

即

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x, y)\,dx = y(x_n) + \int_{x_n}^{x_{n+1}} y'(t)\,dt$$

用矩形近似积分公式计算 $\int_{x_n}^{x_{n+1}} y'(x)\,dx$

取 $y(x) \approx y(x_n) = f(x_n, y(x_n))$，有

$$\int_{x_n}^{x_{n+1}} y'(t)\,dt \approx (x_{n+1} - x_n)y'(x_n) = hf(x_n, y(x_n))$$

则得向前欧拉公式：

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{8.7}$$

取 $y(x) \approx y(x_{n+1}) = f(x_{n+1}, y(x_{n+1}))$，有

$$\int_{x_n}^{x_{n+1}} y'(t)\,dt \approx (x_{n+1} - x_n)y'(x_{n+1}) = hf(x_{n+1}, y(x_{n+1}))$$

则得向后欧拉公式：

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \tag{8.8}$$

用梯形近似积分公式计算 $\int_{x_n}^{x_{n+1}} y'dx$

$$\int_{x_n}^{x_{n+1}} y'(t)\,dt \approx \frac{1}{2}(x_{n+1}-x_n)(y'(x_{n+1})+y'(x_n))$$

$$= \frac{h}{2}(f(x_n,y(x_n))+f(x_{n+1},y(x_{n+1})))$$

则得到梯形公式：

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n,y_n)+f(x_{n+1},y_{n+1})) \tag{8.9}$$

梯形公式也是隐式格式，计算中为了保证一定的精确度，又避免用迭代过程不菲的计算量，一般先用显式公式算出初始值，再用隐式公式进行一次修正。称为预估-校正过程。例如，下面是用显式的欧拉公式和隐式的梯形公式给出的预估-校正公式：

$$\begin{cases} y_{n+1} = y_n + hf(x_n,y_n) \\ y_{n+1} = y_n + \dfrac{h}{2}[f(x_n,y_n)+f(x_{n+1},\overline{y}_{n+1})] \end{cases} \tag{8.10}$$

式(8.10)也称为改进的欧拉公式，它可合并写成

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n,y_n)+f(x_{n+1},y_n+hf(x_n,y_n)))$$

例 8.2：分别用梯形公式和预估-校正公式(改进的欧拉公式)解初值问题：

$$\begin{cases} \dfrac{dy}{dx} = y^2 \\ y(0) = 1 \end{cases}, 0.0 \le x \le 0.4$$

解：

$$y_0 = 1, h = 0.1。$$

用下面的迭代公式，对每个点迭代4次，$k=1,2,3,4$。

$$\begin{cases} y_{n+1}^{(0)} = y_n + hy_n^2 \\ y_{n+1}^{(k+1)} = y_n + \dfrac{h}{2}[y_n^2 + (y_{n+1}^{(k)})^2] \end{cases}$$

该方程的精确解是 $y=\dfrac{1}{1-x}$，计算结果如表8.2所示。

表 8.2

n	x_n	y_n	$y(x_n)$	$\lvert y_n - y(x_n)\rvert$
1	0.1	1.1118	1.1111	0.0007
2	0.2	1.2520	1.2500	0.0020
3	0.3	1.4331	1.4236	0.0095
4	0.4	1.6763	1.6667	0.0004

8.2 龙格-库塔方法

8.2.1 二阶龙格－库塔方法

常微分方程初值问题：

$$\begin{cases} y'(x) = f(x,y) \\ y(x_0) = y_0 \end{cases} \qquad a \leq x \leq b$$

做 $y(x+h)$在 x 点的泰勒展开：

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \cdots + \frac{h^p}{p!}y^{(p)}(x) + \frac{h^{(p+1)}}{(p+1)!}y^{(p+1)}(x+\theta h)$$

$$= y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \cdots + \frac{h^p}{p!}y^{(p)}(x) + \widetilde{T}$$

这里 $0 \leq \theta \leq 1, \widetilde{T} = O(h^{p+1})$。取 $x = x_n$，就有

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \cdots + \frac{h^p}{p!}y^{(p)}(x) + \widetilde{T}_{n+1} \qquad (8.11)$$

若取 $p=1$，有

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \widetilde{T}_{n+1} = y(x_n) + hf(x_n, y(x_n)) + \widetilde{T}_{n+1}$$

截断 $\widetilde{T}_{n+1}$可得到 $y(x_{n+1})$近似值 y_{n+1}的计算公式，即欧拉公式：

$$y_{n+1} = y_n + hf(x_n, y_n)$$

若取 $p=2$，式(8.11)可写成：

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \widetilde{T} = y(x_n) + hf(x_n, y(x_n))$$

$$+ \frac{h^2}{2!}[f_x(x_n, y(x_n)) + f_y(x_n, y(x_n))f(x_n, y(x_n))] + \widetilde{T}_{n+1}$$

或

$$y(x_{n+1}) = y(x_n) + h\{f(x_n, y(x_n))$$

$$+ \frac{h}{2!}[f_x(x_n, y(x_n)) + f_y(x_n, y(x_n))f(x_n, y(x_n))]\} + \widetilde{T}_{n+1} \qquad (8.12)$$

截断 $\widetilde{T}_{n+1}$可得到 $y(x_{n+1})$近似值 y_{n+1}的计算公式：

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2!}[f_x(x_n, y_n) + f_y(x_n, y_n)f(x_n, y_n)] \qquad (8.13)$$

或

$$y_{n+1} = y_n + h\{f(x_n, y_n) + \frac{h}{2}[f_x(x_n, y_n) + f_y(x_n, y_n)f(x_n, y_n)]\} \qquad (8.14)$$

上式为二阶方法，一般优于一阶的欧拉公式(8.2)，但是在计算 y_{n+1}时，需要计算 f，f_x, f_y 在(x_n, y_n)点的值，因此，此法不可取。

龙格-库塔设想用 $f(x,y)$在点$(x_n, y(x_n))$和$(x_n + ah, y(x_n) + bhf(x_n, y(x_n)))$值的线性组合逼近式(8.12)的主体，即用

$$c_1 f(x_n, y(x_n)) + c_2 f(x_n + ah, y(x_n) + bhf(x_n, y(x_n))) \qquad (8.13)$$

逼近

$$f(x_n, y(x_n)) + \frac{h}{2}[f_x(x_n, y(x_n)) + f_y(x_n, y(x_n))f(x_n, y(x_n))] \qquad (8.14)$$

然后对照式(8.12)可得到数值公式：

$$y_{n+1} = y_n + h[c_1 f(x_n, y_n) + c_2 f(x_n + ah, y_n + bhf(x_n, y_n))]$$

或更一般地写成

$$\begin{cases} y_{n+1} = y_n + h(c_1 k_1 + c_2 k_2) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + ah, y_n + bhk_1) \end{cases}$$

对式(8.13)在$(x_n, y(x_n))$点展开得到：

$$c_1 f(x_n, y(x_n)) + c_2[f(x_n, y(x_n)) + ahf_x(x_n, y(x_n)) + bhf_y(x_n, y(x_n))$$

$$f(x_n, y(x_n)) + O(h^2)] = (c_1 + c_2)f(x_n, y(x_n)) + \frac{h}{2}[2c_2 af_x(x_n, y(x_n))$$

$$+ 2c_2 bhf_y(x_n, y(x_n))f(x_n, y(x_n)) + O(h^2)]$$

将上式与式(8.13)比较，知当 c_1, c_2, a, b 满足

$$\begin{cases} c_1 + c_2 = 1 \\ 2c_2 a = 1 \\ 2c_2 b = 1 \end{cases}$$

时有最好的逼近效果，此时式(8.13) - 式(8.14) = $O(h^2)$。这是四个未知数的三个方程，显然方程组有无数组解。

若取 $c_1 = \frac{1}{2}, c_2 = \frac{1}{2}, a = 1, b = 1$，则有二阶龙格-库塔公式：

$$\begin{cases} y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + h, y_n + hk_1) \end{cases} \tag{8.15}$$

若取 $c_1 = 0, c_2 = 1, a = \frac{1}{2}, b = \frac{1}{2}$，则得另一种形式的二阶龙格-库塔公式：

$$\begin{cases} y_{n+1} = y_n + hk_2 \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \end{cases} \tag{8.16}$$

从公式建立过程中可看到，二阶龙格-库塔公式的局部截断误差仍为 $O(h^3)$，是二阶精度的计算公式。类似地，可建立高阶的龙格-库塔公式，同理可知四阶龙格-库塔公式的局部截断误差界为 $O(h^5)$，是四阶精度的计算公式。

8.2.2 四阶龙格-库塔公式

下面列出常用的三阶、四阶龙格-库塔计算公式。

三阶龙格-库塔公式

$$(1)\begin{cases} y_{n+1} = y_n + \frac{h}{6}(k_1 + 4k_2 + k_3) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\ k_3 = f(x_n + h, y_n - hk_1 + 2hk_2) \end{cases} \tag{8.17}$$

(2)
$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{4}(k_1 + 3k_3) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{1}{3}h, y_n + \dfrac{1}{3}hk_1) \\ k_3 = f(x_n + \dfrac{2}{3}h, y_n + \dfrac{2}{3}hk_2) \end{cases} \tag{8.18}$$

(3)
$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{9}(2k_1 + 3k_2 + 4k_3) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}hk_1) \\ k_3 = f(x_n + \dfrac{3}{4}h, y_n + \dfrac{3}{4}hk_2) \end{cases} \tag{8.19}$$

四阶龙格-库塔公式

(1)
$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}hk_1) \\ k_3 = f(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}hk_2) \\ k_4 = f(x_n + h, y_n + hk_3) \end{cases} \tag{8.20}$$

(2)
$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{8}(k_1 + 3k_2 + 3k_3 + k_4) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{1}{3}h, y_n + \dfrac{1}{3}hk_1) \\ k_3 = f(x_n + \dfrac{2}{3}h, y_n + \dfrac{1}{3}hk_1 + hk_2) \\ k_4 = f(x_n + h, y_n + hk_1 - hk_2 + hk_3) \end{cases} \tag{8.21}$$

例 8.3:用四阶龙格-库塔公式(8.20)解初值问题:

$$\begin{cases} \dfrac{dy}{dx} = y^2\cos x \\ y(0) = 1 \end{cases}, 0.1 \leq x \leq 0.8$$

解:取步长 $h = 0.2$,计算公式为:

$$\begin{cases} y_{n+1} = y_n + \dfrac{0.2}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 = y_n^2\cos x_n \\ k_2 = (y_n + 0.1k_1)^2\cos(x_n + 0.1) \\ k_3 = (y_n + 0.1k_2)^2\cos(x_n + 0.1) \\ k_4 = (y_n + 0.2k_3)^2\cos(x_n + 0.2) \end{cases}$$

计算结果列表 8.3 中。

表 8.3

n	x_n	y_n	$y(x_n)$	$\lvert y_n - y(x_n)\rvert$
1	0.2	1.24789	1.24792	0.00003
2	0.4	1.63762	1.63778	0.00016
3	0.6	2.29618	2.29696	0.00078
4	0.8	3.53389	3.53802	0.00413

8.2.3 步长的自适应

欧拉方法和龙格-库塔方法在计算 y_{n+1} 时仅用到前一步 y_n 的值，我们称这样的方法为单步法。在单步法计算中根据需要可以取等步长或变步长。对于 $y(x)$ 变化平缓的区段，步长可以取大一些；而对于 $y(x)$ 变化剧烈的区段，则步长可以取小一些。

以计算 y_1 为例，已知 y_0 的数值，取 $h = h_0$ 做初始步长，ε 是给定的控制精度，用欧拉或龙格-库塔方法的计算公式。

记 $y_1^{(h)}$ 为由 y_0 和取步长 h 算出的 y_1，记 $y_1^{(h/2)}$ 为由 y_0 和取步长 $h/2$ 算出的 y_1，即 $y_1^{(h)}, y_1^{(h/2)}$ 均是 $y(x_0 + h)$ 的近似值，当 $|y_1^{(h)} - y_1^{(h/2)}| < \varepsilon$，逐步放大步长，$h \leftarrow 2h$，直到 $|y_1^{(h)} - y_1^{(h/2)}| > \varepsilon$ 时为止，最后取 $h/2$ 为步长的值。而当 $|y_1^{(h)} - y_1^{(h/2)}| > \varepsilon$，逐步缩小步长，$h \leftarrow h/2$，直到 $|y_1^{(h)} - y_1^{(h/2)}| < \varepsilon$ 时为止，取 h 为步长的值。

下面给出自适应算法的描述：

给定控制精度 ε 和初始步长 h_0。

对于 $n = 1,2,\cdots,m$

$h = h_0$

IF $|y_n^{(h)} - y_n^{(h/2)}| \leq \varepsilon$ THEN

WHILE $|y_n^{(h)} - y_n^{(h/2)}| \leq \varepsilon$

$h = 2 \cdot h$ ENDWHILE

$h = h/2$

ELSE

WHILE $|y_n^{(h)} - y_n^{(h/2)}| > \varepsilon$

$h = h/2$ ENDWHILE

$h = h$

ENDIF

8.3 线性多步法

常微分方程初值问题[式(8.1)]与积分 $y(x) = y(x^*) + \int_{x^*}^{x} y'(t)dt$ 等价，在 8.1.3 节中用一些简单的数值积分公式建立了欧拉和梯形等数值方法，下面给出一些更一般的

用数值积分工具求解常微分方程初值问题的方法。当积分节点中包含 x_{n+1} 时称为隐式公式，否则称为显式公式。分别取 x, x^* 为剖分点 x_{n+1}, x_{n-p}，有

$$y(x_{n+1}) = y(x_{n-p}) + \int_{x_{n-p}}^{x_{n+1}} y'(x)\,dx$$

若用积分节点 $x_n, x_{n-1}, \cdots, x_{n-q}$ 构造插值多项式近似 $y'(x)$，在区间 $[x_{n-p}, x_{n+1}]$ 上计算数值积分 $\int_{x_{n-p}}^{x_{n+1}} y'(x)\,dx$，则称构造的计算 y_{n+1} 的方法为线性多步法。这是显式公式。

若以 $x_{n+1}, x_n, x_{n-1}, \cdots, x_{n+1-q}$ 积分节点构造插值多项式近似 $y'(x)$，在区间 $[x_{n-p}, x_{n+1}]$ 上计算数值积分 $\int_{x_{n-p}}^{x_{n+1}} y'(x)\,dx$，则称为隐式公式。

与通常数值积分不同的是，在线性多步法公式中，有两个控制量 p 和 q，p 控制积分区间，q 控制插值节点。

特别取 $p=0$，有

$$y(x_{n+1}) = y(x_n) = \int_{x_n}^{x_{n+1}} y'(x)\,dx$$

若积分 $\int_{x_n}^{x_{n+1}} y'(x)\,dx$ 用关于积分节点 $x_n, x_{n-1}, \cdots, x_{n-q}$ 的数值积分近似，就可得到显式的阿达姆斯公式。

例 8.4：取 $q=1$，有

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} [l_0(x)y'(x_n) + l_1(x)y'(x_{n-1}) + R(x)]\,dx$$

这里

$$\int_{x_n}^{x_{n+1}} l_0(x)\,dx = \int_{x_n}^{x_{n+1}} \frac{x - x_{n-1}}{x_n - x_{n-1}}\,dx = \frac{3}{2}h$$

$$\int_{x_n}^{x_{n+1}} l_1(x)\,dx = \int_{x_n}^{x_{n+1}} \frac{x - x_n}{x_{n-1} - x_n}\,dx = -\frac{1}{2}h$$

$$T_{n+1} = \int_{x_n}^{x_{n+1}} R(x)\,dx = \int_{x_n}^{x_{n+1}} \frac{y^{(3)}(\eta)}{2}(x - x_n)(x - x_{n-1})\,dx = \frac{5}{12}y^{(3)}(\xi)h^3$$

即

$$y(x_{n+1}) = y(x_n) + \frac{h}{2}[3f(x_n, y(x_n)) - f(x_{n-1}, y(x_{n-1}))] + T_{n+1}$$

截断 T_{n+1} 可得到 $y(x_{n+1})$ 近似值 y_{n+1} 的计算公式：

$$y_{n+1} = y_n + \frac{h}{2}[3f(x_n, y_n) - f(x_{n-1}, y_{n-1})] \tag{8.22}$$

上式称为二阶显式阿达姆斯公式。类似可得

三阶显式阿达姆斯公式：

$$y_{n+1} = y_n + \frac{h}{12}[23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})] \tag{8.23}$$

四阶显式阿达姆斯公式：

$$y_{n+1} = y_n + \frac{h}{24}[55f(x_n, y_n) - 59f(x_{n-1}, y_{n-1}) + 37f(x_{n-2}, y_{n-2}) - 9f(x_{n-3}, y_{n-3})] \tag{8.24}$$

若积分 $\int_{x_n}^{x_{n+1}} y'(x)dx$ 用关于积分节点 $x_{n+1}, x_n, \cdots, x_{n+1-q}$ 的数值积分近似，就可得到隐式的阿达姆斯公式。隐式格式需要通过迭代求解。它们分别为

二阶隐式阿达姆斯公式(也称为梯形公式)：

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})] \tag{8.25}$$

三阶隐式阿达姆斯公式：

$$y_{n+1} = y_n + \frac{h}{12}[5f(x_{n+1}, y_{n+1}) + 8f(x_n, y_n) - f(x_{n-1}, y_{n-1})] \tag{8.26}$$

四阶隐式阿达姆斯公式：

$$y_{n+1} = y_n + \frac{h}{24}[9f(x_{n+1}, y_{n+1}) + 19f(x_n, y_n) - 5f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2})] \tag{8.27}$$

例 8.5：建立 $p=1, q=2$ 显式公式。

解：

$$y_{n+1} = y_{n-1} + \int_{x_{n-1}}^{x_{n+1}} [l_0(x)f(x_n, y_n) + l_1(x)f(x_{n-1}, y_{n-1}) + l_2(x)f(x_{n-2}, y_{n-2})]dx$$

$$y_{n+1} = y_{n-1} + h[a_0 f(x_n, y_n) + a_1 f(x_{n-1}, y_{n-1}) + a_2 f(x_{n-2}, y_{n-2})]$$

$$a_0 h = \int_{x_{n-1}}^{x_{n+1}} \frac{(x - x_{n-1})(x - x_{n-2})}{(x_n - x_{n-1})(x_n - x_{n-2})}dx = \frac{7}{3}h$$

$$a_1 h = \int_{x_{n-1}}^{x_{n+1}} \frac{(x - x_n)(x - x_{n-2})}{(x_{n-1} - x_n)(x_{n-1} - x_{n-2})}dx = -\frac{2}{3}h$$

$$a_2 h = \int_{x_{n-1}}^{x_{n+1}} \frac{(x - x_n)(x - x_{n-1})}{(x_{n-2} - x_n)(x_{n-2} - x_{n-1})}dx = \frac{1}{3}h$$

计算格式：

$$y_{n+1} = y_{n-1} + \frac{h}{3}[7f(x_n, y_n) - 2f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2})], n = 2,3,4,\cdots,m-1$$

其中：$h = \frac{b-a}{m}$，并用龙格-库塔公式计算 y_1, y_2。

截断误差：

$$R(x) = hT_{n+1} = \int_{x_{n-1}}^{x_{n+1}} \frac{f^{(3)}(\xi)}{3!}(x - x_{n-2})(x - x_{n-1})(x - x_n)dx$$

$$= \frac{1}{6}\int_{x_{n-1}}^{x_{n+1}} f^{(3)}(\xi)(x - x_{n-2})(x - x_{n-1})(x - x_n)dx$$

$$(f^{(3)}(x, y) = y^{(4)}(x))$$

这是一个三步三阶的显式格式。

例 8.6：构造 $p=2, q=2$ 的隐式格式。

解：以 $[x_{n-2}, x_{n+1}]$ 做积分区间，以 $(x_{n+1}, f(x_{n+1}))$，$(x_n, f(x_n))$，$(x_{n-1}, f(x_{n-1}))$ 构造拉格朗日插值多项式。

$$y_{n+1} = y_{n-2} + h[\beta_0 f(x_{n-1}, y_{n+1}) + \beta_1 f(x_n, y_n) + \beta_2 f(x_{n-1}, y_{n-1})]$$

则

$$\beta_0 h = \int_{x_{n-2}}^{x_{n+1}} \frac{(x - x_n)(x - x_{n-1})}{(x_{n+1} - x_n)(x_{n+1} - x_{n-1})} dx = \frac{3}{4}h$$

$$\beta_1 h = \int_{x_{n-2}}^{x_{n+1}} \frac{(x - x_{n+1})(x - x_{n-1})}{(x_n - x_{n+1})(x_n - x_{n-1})} dx = 0$$

$$\beta_2 h = \int_{x_{n-2}}^{x_{n+1}} \frac{(x - x_{n+1})(x - x_n)}{(x_{n-1} - x_{n+1})(x_{n-1} - x_n)} dx = \frac{9}{4}h$$

计算格式：

$$y_{n+1} = y_{n-2} + \frac{h}{4}[3f(x_{n+1}, y_{n+1}) + 9f(x_{n-1}, y_{n-1})]$$

截断误差：

$$hT_{n+1} = \frac{1}{6}\int_{x_{n-2}}^{x_{n+1}} y^{(4)}(\xi)(x - x_{n+1})(x - x_n)(x - x_{n-1})dx$$

为了避免迭代，可用预估-校正公式替换上面的计算公式。

$$\begin{cases} \overline{y}_{n+1} = y_{n-2} + \dfrac{h}{3}[7f(x_n, y_n) - 2f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2})] \\ y_{n+1} = y_{n-2} + \dfrac{h}{4}[3f(x_{n+1}, \overline{y}_{n+1}) + 9f(x_{n-1}, y_{n-1})] \end{cases}$$

8.4 常微分方程组的数值解法

8.4.1 一阶常微分方程组的数值解法

将由 m 个一阶方程组成的常微分方程初值问题：

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2, \cdots, y_m) \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2, \cdots, y_m) \\ \cdots \\ \dfrac{dy_m}{dx} = f_m(x, y_1, y_2, \cdots, y_m) \qquad (a \leq x \leq b) \\ y_1(a) = \eta_1 \\ y_2(a) = \eta_2 \\ \cdots \\ y_m(a) = \eta_m \end{cases} \tag{8.28}$$

写成向量形式：

$$\begin{cases} \dfrac{dY}{dx} = F(x, y) \\ Y(a) = \eta \end{cases} \tag{8.29}$$

其中：

$$Y(x)=\begin{pmatrix}y_1(x)\\y_2(x)\\\cdots\\y_m(x)\end{pmatrix},F(x,y)=\begin{pmatrix}f_1(x,y_1,\cdots,y_m)\\f_2(x,y_1,\cdots,y_m)\\\cdots\\f_m(x,y_1,\cdots,y_m)\end{pmatrix},\eta=\begin{pmatrix}\eta_1\\\eta_2\\\cdots\\\eta_m\end{pmatrix}。$$

前面对常微分方程所用的欧拉方法、龙格-库塔方法等各种方法，都可以平行地应用到常微分方程组的数值解中。为了叙述方便，下面以两个方程组为例，给出相应的计算公式。

常微分方程组：

$$\begin{cases}\dfrac{dy}{dx}=f(x,y,z)\\[2ex]\dfrac{dz}{dx}=g(x,y,z)\qquad(a\leq x\leq b)\\[2ex]y(a)=y_0\\z(a)=z_0\end{cases}$$

欧拉公式：

$$\begin{cases}y_{n+1}=y_n+hf(x_n,y_n,z_n)\\z_{n+1}=z_n+hg(x_n,y_n,z_n)\end{cases}\tag{8.30}$$

预估-校正公式：

$$\begin{pmatrix}\bar{y}_{n+1}\\\bar{z}_{n+1}\end{pmatrix}=\begin{pmatrix}y_n\\z_n\end{pmatrix}+h\begin{pmatrix}f(x_n,y_n,z_n)\\g(x_n,y_n,z_n)\end{pmatrix}$$

$$\begin{pmatrix}y_{n+1}\\z_{n+1}\end{pmatrix}=\begin{pmatrix}y_n\\z_n\end{pmatrix}+\frac{h}{2}\left[\begin{pmatrix}f(x_n,y_n,z_n)\\g(x_n,y_n,z_n)\end{pmatrix}+\begin{pmatrix}f(x_{n+1},\bar{y}_{n+1},\bar{z}_{n+1}\\g(x_{n+1},\bar{y}_{n+1},\bar{z}_{n+1})\end{pmatrix}\right]\tag{8.31}$$

四阶龙格-库塔公式：

$$Y_{n+1}=Y_n+\frac{h}{6}[K_1+2K_2+2K_3+K_4]$$

$$\begin{pmatrix}y_{n+1}\\z_{n+1}\end{pmatrix}=\begin{pmatrix}y_n\\z_n\end{pmatrix}+\frac{h}{6}\left[\begin{pmatrix}k_1^{(1)}\\k_1^{(2)}\end{pmatrix}+2\begin{pmatrix}k_2^{(1)}\\k_2^{(2)}\end{pmatrix}+2\begin{pmatrix}k_3^{(1)}\\k_3^{(2)}\end{pmatrix}+\begin{pmatrix}k_4^{(1)}\\k_4^{(2)}\end{pmatrix}\right]\tag{8.32}$$

$$K_1=\begin{pmatrix}k_1^{(1)}\\k_1^{(2)}\end{pmatrix}=\begin{pmatrix}f(x_n,y_n,z_n)\\g(x_n,y_n,z_n)\end{pmatrix}$$

$$K_2=\begin{pmatrix}k_2^{(1)}\\k_2^{(2)}\end{pmatrix}=\begin{pmatrix}f\left(x_n+\frac{h}{2},y_n+\frac{h}{2}k_1^{(1)},z_n+\frac{h}{2}k_1^{(2)}\right)\\g\left(x_n+\frac{h}{2},y_n+\frac{h}{2}k_1^{(1)},z_n+\frac{h}{2}k_1^{(2)}\right)\end{pmatrix}$$

$$K_3=\begin{pmatrix}k_3^{(1)}\\k_3^{(2)}\end{pmatrix}=\begin{pmatrix}f\left(x_n+\frac{h}{2},y_n+\frac{h}{2}k_2^{(1)},z_n+\frac{h}{2}k_2^{(2)}\right)\\g\left(x_n+\frac{h}{2},y_n+\frac{h}{2}k_2^{(1)},z_n+\frac{h}{2}k_2^{(2)}\right)\end{pmatrix}$$

$$K_4 = \begin{pmatrix} k_4^{(1)} \\ k_4^{(2)} \end{pmatrix} = \begin{pmatrix} f(x_n + h, y_n + hk_3^{(1)}, z_n + hk_3^{(2)}) \\ g(x_n + h, y_n + hk_3^{(1)}, z_n + hk_3^{(2)}) \end{pmatrix}$$

例 8.7:两种果树寄生虫,其数量分别是 $u = u(t), v = v(t)$,其中一种寄生虫以吃另一种寄生虫为生,两种寄生虫的增长函数如下列常微分方程组所示,预测 3 年后这一对寄生虫的数量。

$$\begin{cases} \dfrac{du}{dt} = 0.09u\left(1 - \dfrac{u}{20}\right) - 0.45uv \\ \dfrac{dv}{dt} = 0.06v\left(1 - \dfrac{v}{15}\right) - 0.001uv \\ u(0) = 1.6 \\ v(0) = 1.2 \end{cases}$$

解:记 $\begin{cases} f(u,v) = 0.09u\left(1 - \dfrac{u}{20}\right) - 0.45uv \\ g(u,v) = 0.06v\left(1 - \dfrac{v}{15}\right) - 0.001uv \end{cases}$

在本题中 $f(t,u,v) = f(u,v), g(t,u,v) = g(u,v)$。用欧拉预估-校正公式(8.31):

$$\begin{pmatrix} \overline{u}_{n+1} \\ \overline{v}_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} + h\begin{pmatrix} f(u_n, v_n) \\ g(u_n, v_n) \end{pmatrix}$$

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} + \frac{h}{2}\begin{pmatrix} f(u_n, v_n) \\ g(u_n, v_n) \end{pmatrix} + \begin{pmatrix} f(u_{n+1}, \overline{v}_{n+1}) \\ g(u_{n+1}, \overline{v}_{n-1}) \end{pmatrix}$$

取 $h = 1$,计算结果见表 8.4。

表 8.4

t(年)	$u(t)$	$v(t)$
1	1.6	1.2
2	0.28766	1.33103
3	0.0291308	1.47776
4	0.00021727	1.63947

8.4.2 高阶常微分方程数值方法

下面以三阶常微分方程为例说明高阶常微分方程的数值计算步骤。

$$\begin{cases} \dfrac{d^3y(x)}{dx} = f(x, y, y', y'') \\ y(a) = \eta^{(0)} \\ y'(a) = \eta^{(1)} \\ y''(a) = \eta^{(2)} \end{cases} \qquad (a \leq x \leq b) \tag{8.33}$$

将三阶方程化为一阶方程组。令

$$y(x) = y_1(x)$$

$$\frac{dy_1(x)}{dx} = y_2(x)$$

$$\frac{dy_2(x)}{dx} = y_3(x) \tag{8.34}$$

得到一阶方程组：

$$\begin{cases} \dfrac{dy_1(x)}{dx} = y_2(x) \\ \dfrac{dy_2(x)}{dx} = y_3(x) \\ \dfrac{dy_3(x)}{dx} = f(x, y_1(x), y_2(x), y_3(x)) \\ y_1(a) = \eta^{(0)} \\ y_2(a) = \eta^{(1)} \\ y_3(a) = \eta^{(2)} \end{cases}$$

已将高阶方程化为一阶方程组了。

8.5* 常微分方程的稳定性

用欧拉公式 $y_{n+1} = y_n + hf(x_n, y_n)$ 计算 y_{n+1} 时，假设计算中的某一步有误差，而以后的计算是精确的，那么这一步的误差对以后的计算有何影响？如果随着计算的进程这一步的误差对以后的影响逐步消失，我们称方法是绝对稳定的；如果这一步的误差在以后的计算中恶性放大，则称方法是不稳定的。还有其它的一些稳定性定义。在选用方法时，方法的绝对稳定性是最为重要的指标。不绝对稳定的方法是不能采用的。讨论绝对稳定性是把方法用到最为典型的稳定微分方程(8.35)上进行的。

$$\begin{cases} \dfrac{dy}{dx} = \lambda y \\ y(a) = y_0 \end{cases} \quad (a \leq x \leq b, \lambda \text{ 是复数}, \mathrm{Re}\lambda < 0) \tag{8.35}$$

例 8.8：讨论欧拉方法的稳定性。

解：用欧拉方法计算方程(8.35)的公式：$y_{n+1} = y_n + \lambda h y_n$

若 y_n 有误差 ρ_n，记为 $y_n^* = y_n + \rho_n$，那么 y_{n+1} 就有误差 ρ_{n+1}，记为 $y_{n+1}^* = y_{n+1} + \rho_{n+1}$。$y_{n+1}^*$ 满足

$$y_{n+1}^* = y_n^* + \lambda h y_n^* \tag{8.36}$$

两式相减得到 $\rho_{n+1} = \rho_n + \lambda h \rho_n = (1 + \lambda h)\rho_n$

或

$$\frac{|\rho_{n+1}|}{|\rho_n|} = |1 + \lambda h| \tag{8.37}$$

当 $|1 + \lambda h| < 1$ 时，即 λh 落在图 8.3 中的单位圆上时有

$$\frac{|\rho_{n+1}|}{|\rho_n|} < 1$$

即这时误差逐次衰减，方法是绝对稳定的。

图 8.3 上的单位圆，称为绝对稳定区域。

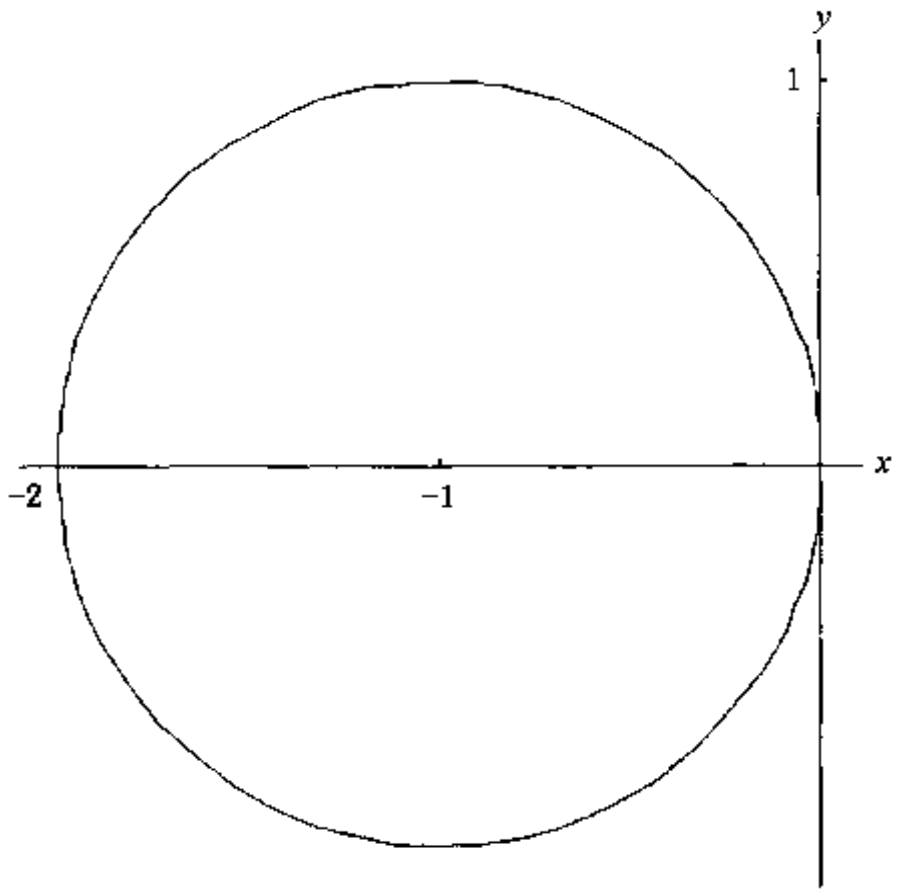


图 8.3　欧拉方法的绝对稳定区域

例 8.9:讨论向后欧拉方法的稳定性。

解:用向后欧拉方法计算方程(8.35)的公式:

$$y_{n+1} = y_n + \lambda h y_{n+1}$$

误差方程:

$$y_{n+1} - y_{n+1}^* = y_n - y_n^* + \lambda h(y_{n+1} - y_{n+1}^*)$$

$$\rho_{n+1} = \rho_n + \lambda h \rho_{n+1}$$

计算相邻两步误差的比值:

$$\left| \frac{\rho_{n+1}}{\rho_n} \right| = \frac{1}{|1 - \lambda h|}$$

当 $\mathrm{Re}(\lambda) < 0$,恒有 $\left| \frac{\rho_{n+1}}{\rho_n} \right| = \frac{1}{|1 - \lambda h|} < 1$,当绝对稳定区域是左半平面时,则称在这个区域数值方法是 A 稳定的,或称无条件绝对稳定。

例 8.10:讨论中心差分方法的稳定性。

解:用中心差分方法计算方程(8.35)的公式:

$$y_{n+1} = y_{n-1} + 2\lambda h y_n$$

误差方程:

$$\rho_{n+1} = \rho_{n-1} + 2\lambda h \rho_n \tag{8.38}$$

不失一般性,我们考虑由 ρ_0, ρ_1 对以后 ρ_n 的影响,差分方程(8.36)的特征方程为:

$$\xi^2 - 2\lambda h \xi - 1 = 0$$

它的两个根为:

$$\xi_1 = \lambda h + \sqrt{1 + (\lambda h)^2}, \xi_2 = \lambda h - \sqrt{1 + (\lambda h)^2}$$

由差分方程理论(略),ρ_n 的一般解可由下式表达:

$$\rho_n = a(\lambda h + \sqrt{(1+\lambda h)^2})^n + b(\lambda h - \sqrt{(1+\lambda h)^2})^n \tag{8.39}$$

其中 a, b 可由 ρ_0, ρ_1 决定。

$$\lambda h - \sqrt{(1+\lambda h)^2} = \lambda h - 1 + O((\lambda h)^2)$$

由于 $\mathrm{Re}(\lambda h) < 0$,故 $|\lambda h - 1| > 1$

$b(\lambda h - \sqrt{(1+\lambda h)^2})^n$ 因 n 增大而恶性发展,所以方法为不稳定的。

8.6 程序示例

程序 15 用改进的欧拉公式求解常微分方程初值问题:

$$\begin{cases} y'(x) = f(x,y) \\ y(a) = y_0 \end{cases} (a \le x \le b)$$

算法描述

对给定的 $f(x,y)$,用改进的欧拉公式

$$\begin{cases} \bar{y}_{n+1} = y_n + hf(x_n + y_n) \\ y_{n+1} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})] \end{cases}$$

求解常微分方程初值问题的解。

程序源码

```
//////////////////////////////////////////////
//      Purpose: 改进的欧拉公式          //
//////////////////////////////////////////////
# include <stdio.h>
# include <math.h>
# define f(x,y) (y * y)                    //dy/dx = f(x,y)
int main()
{   int m;
    int i;
    double a, b, y0;
    double xn, yn, xn1, yn1, yn1b;
    double h;
    printf(" \ nInput the begin and end of x:");
    scanf("%lf%lf", &a, &b);
    printf("Input the y value at %f:", a);
    scanf("%lf", &y0);
    printf("Input m value[divide (%f, %f)]:", a, b);
    scanf("%d", &m);
    if (m <= 0)
    {   printf("Please input a number larger then 1. \ n");
        return 1;
    }
```

```
    h = (b-a)/m;
    xn = a; yn = y0;
    for (i = 1; i < = m; i + + )
    {   xn1 = xn + h;
        yn1b = yn + h * f(xn, yn);
        yn1 = yn + h/2 * (f(xn, yn) + f(xn1 . yn1b));
        printf("x%d = %f, y%d = %f \ n", i, xn1, i, yn1);
        xn = xn1; yn = yn1;
    }
    return 0;   }
```

计算实例

用改进的欧拉公式,求解常微分方程初值问题的解:

$$\begin{cases} \dfrac{dy}{dx} = y^2 \\ y(0) = 1 \end{cases} \quad 0.1 \leq x \leq 0.4$$

程序输入输出

```
Input the begin and end of x: 0 0.4
Input the y value at 0.000000: 1
Input m value[divide (0.000000,0.800000)]: 4
x1 = 0.100000, y1 = 1.110500
x2 = 0.200000, y2 = 1.248276
x3 = 0.300000, y3 = 1.424760
x4 = 0.400000, y4 = 1.658736
```

程序 16 用四阶龙格-库塔方法求解常微分方程初值问题:

$$\begin{cases} y'(x) = f(x,y) \\ y(a) = y_0 \end{cases} (a \leq x \leq b)$$

算法描述

对给定的 $f(x,y)$,用四阶龙格-库塔方法求解常微分方程初值问题。

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 = f(x_n, y_n) \\ k_2 = f\left(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}hk_1\right) \\ k_3 = f\left(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}hk_2\right) \\ k_4 = f(x_n + h, y_n + hk_3) \end{cases}$$

程序源码

```
////////////////////////////////////////////////////////////////////////
//      Purpose: 四阶龙格-库塔法求解常微分方程初值问题      //
////////////////////////////////////////////////////////////////////////
# include < stdio.h >
# include < math.h >
```

```
# define f(x,y) (x/y)        //dy/dx = f(x,y)
int main()
{   int m;
    int i;
    double a, b, y0;
    double xn, yn, yn1;
    double k1, k2, k3, k4;
    double h;
    printf(" \ nInput the begin and end of x:");
    scanf("%lf%lf", &a, &b);
    printf("Input the y value at %f:", a);
    scanf("%lf", &y0);
    printf("Input m value[divide (%f,%f)]:", a, b);
    scanf("%d", &m);
    if (m< =0)
    {
      printf("Please input a number larger then 1. \ n");
      return 1;
    }
    h = (b-a)/m;
    xn = a; yn = y0;
    for (i = 1; i< = m; i + + )
    {
      k1 = f(xn, yn);
      k2 = f((xn + h/2), (yn + h * k1/2));
      k3 = f((xn + h/2), (yn + h * k2/2));
      k4 = f((xn + h), (yn + h * k3));
      yn1 = yn + h/6 * (k1 + 2 * k2 + 2 * k3 + k4);
      xn + = h;
      printf("x%d = %f, y%d = %f \ n", i, xn, i, yn1);
      yn = yn1;
    }
    return 0;
}
```

计算实例

用四阶龙格-库塔公式解初值问题：

$$\begin{cases} \dfrac{dy}{dx} = x/y \\ y(2.0) = 1 \end{cases} \quad 2.0 \leq x \leq 2.6, \text{取 } h = 0.2$$

程序输入输出

```
Input the begin and end of x: 2.0 2.6
Input the y value at 0.000000: 1
Input m value[divide (0.000000,0.800000)]: 3
```

x1 = 2.200000, y1 = 1.356505

x2 = 2.400000, y2 = 1.661361

x3 = 2.600000, y3 = 1.939104

习题 8

8.1 用向前欧拉公式解初值问题：

$$\begin{cases} \dfrac{dy}{dx} = x + y^2 \\ y(0) = 1 \end{cases} 0.1 \leq x \leq 0.5, \text{取 } h = 0.1$$

8.2 用向后欧拉公式解初值问题：

$$\begin{cases} \dfrac{dy}{dx} = x^2 + y \\ y(1.0) = 1 \end{cases} 1.0 \leq x \leq 1.5, \text{取 } h = 0.1$$

8.3 $p(t)$是关于 t 年时的人口数，若平均出生率 b 为常数，平均死亡率与人口数量成正比，用向前欧拉公式解初值问题，则人口生长率满足

$$\frac{dp(t)}{dt} = bp(t) - kp^2(t)$$

其中：$p(0) = 50976, b = 2.9 \cdot 10^{-2}, k = 1.4 \cdot 10^{-7}$。

用二阶龙格-库塔公式计算 5 年后的人口数。

8.4 用四阶龙格-库塔公式解初值问题：

$$\begin{cases} \dfrac{dy}{dx} = x/y \\ y(2.0) = 1 \end{cases} 2.0 \leq x \leq 2.6, \text{取 } h = 0.2$$

8.5 用线性多步法解初值问题，取 $p = 2, q = 2$。

$$\begin{cases} \dfrac{dy}{dx} = xy \\ y(3.0) = 1 \end{cases} 3.0 \leq x \leq 3.6, \text{取 } h = 0.2$$

8.6 一对物种称为 A.fisheri 和 A.melinus，其数量分别为 $u = u(t), v = v(t)$，其中 A.fisheri 以吃 A.melinus 为生，预测 3 年后这一对物种的数量？（方法自选）

$$\begin{cases} \dfrac{du}{dt} = 0.05u\left(1 - \dfrac{u}{20}\right) - 0.002uv \\ \dfrac{dv}{dt} = 0.09v\left(1 - \dfrac{v}{15}\right) - 0.15uv \\ u(0) = 0.193 \\ v(0) = 0.083 \end{cases}$$

第 9 章* 在 Mathematica 中做题

学习任何课程，做题都是巩固和检验学习效果的重要步骤之一。怎样做计算方法的题目？计算题型是计算方法的主要题型。传统的做题方法是套用公式手工计算一些简单的题目，例如，用简单迭代法解一个三元方程组。掌握了解题的方法和步骤后，做算术运算就是解题的主要工作，难免与繁琐和枯燥同行。如果用 C 语言等高级语言编程解题，即可减少繁琐，又可提高计算机技术的应用能力。但用高级语言编程也有新的工作量，使得所做题型不能太多，训练范围受到制约。

建议用某种符号计算系统做计算方法的题目，它是一种雅俗共赏的做数学题的环境。在符号计算系统中做题，既可免除繁琐的算术运算，又可在宽松环境中编程解题，使学生将精力主要放在理解方法和演示方法的过程中。

9.1 符号计算系统 Mathematica 基本操作

符号计算系统是一种集成化的数学软件系统，它主要包括数值计算、符号计算、图形生成和程序设计四个方面。符号计算系统含有种类丰富的功能强大的内部函数，用户也可以自由地定义自己的函数并扩充到系统函数中。

目前，较流行的符号计算系统有（按字母顺序排列）：Axiom，Macsyma，Maple，Mathematica，Reduce 和 Derive。

Mathematica 是优秀的符号计算系统之一。下面简要给出 Mathematica 系统的交互式的操作步骤，列出有关数值计算方法的函数，以供使用参考。要进一步掌握和了解 Mathematica 的功能，请查阅 Mathematica 系统的在线帮助和有关书籍。

符号计算系统有两种运行方式，一种是交互式，用户键入一个函数，系统执行相应的计算，例如，计算定积分 $\int_{-1}^{1} x^2(11-\sin x)\,dx$ 的函数命令是：

```
NIntegrate [x^2(11 - Sin[x]), {x, -1, 1}];
```

另一种是写一段程序，系统连续执行一个计算序列，程序使用的是符号计算系统自己的语言，对于已掌握一门高级语言的用户来说，语句的形式大同小异，容易掌握。

进入 Mathematica

在 Windows 3.x 环境中，在装好 Mathematica 2.2 的窗口上双击 Mathematica 的图标 。在 Windows 95 下，在“开始”菜单的“程序”中单击“Mathematica 2.2”。屏幕显示有随机图形的启动图面，稍停片断，系统装好内核后（也有运行第一条命令时装载内核），则显示如图 9.1 的工作屏幕。

Mathematica 的工作窗口有主菜单和若干工具按钮。大部分工具按钮和菜单的图标和意义与 Windows 环境中的大多数软件系统的图标和意义是相同的。例如：菜单名称的意

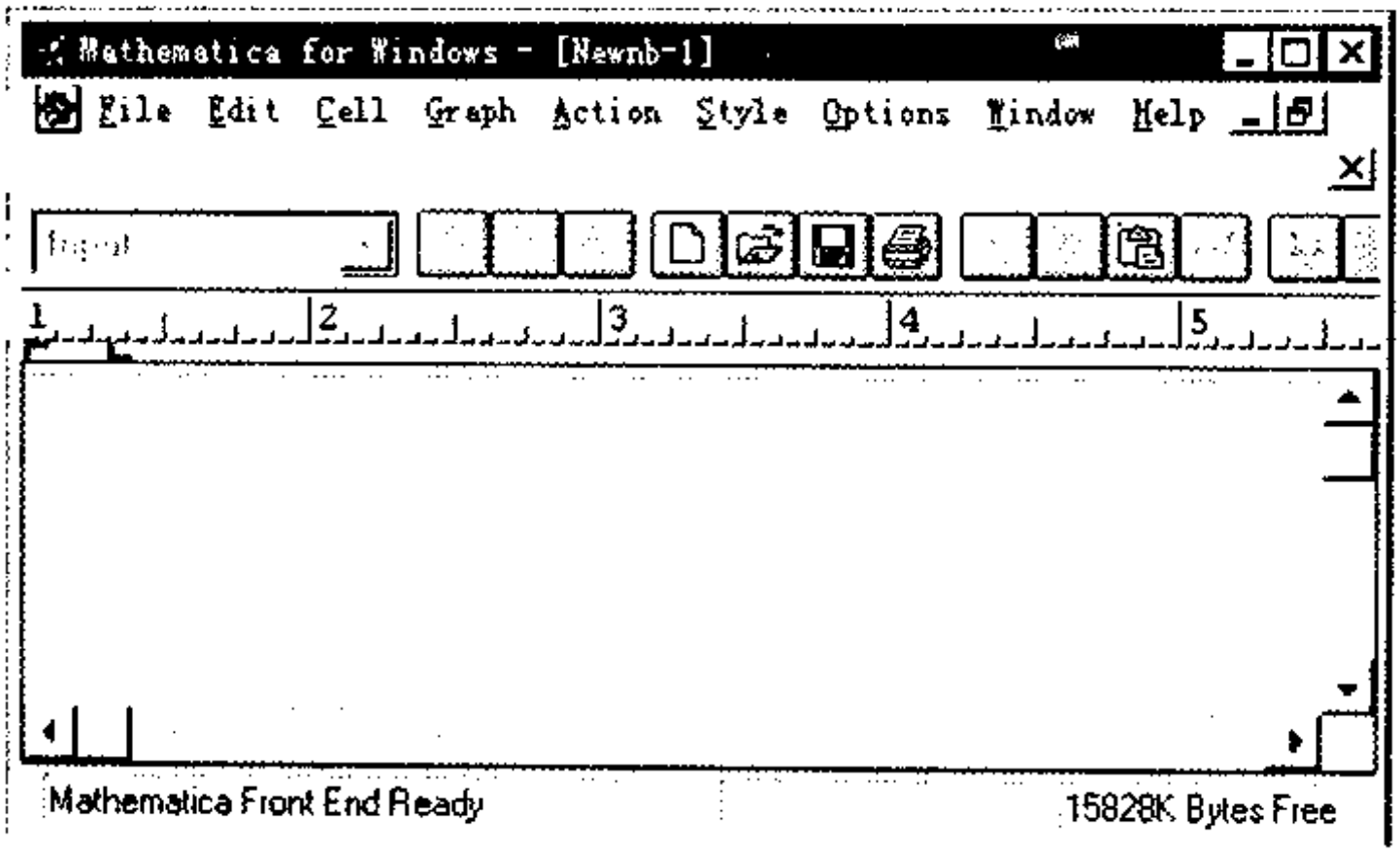
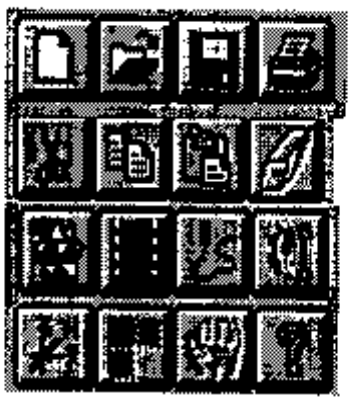


图 9.1　Mathematica 的工作窗口

义文如其名，File 表示文件，Edit 表示编辑，Cell 表示对 Cell(单元)的处理。大部分工具按钮的图标也是“图”如文意，例如：用剪刀表示剪切。图 9.2 是部分常用按钮及其意义。



新建，打开，保存，打印Notebooks

剪切，复制，粘贴，取消

计算表达式，动画，图形的Postscript，播放声音

三维图形视点，颜色选择，中断，帮助

图 9.2　部分常用工具按钮

输入和计算表达式

用户在工作屏幕上输入函数命令，系统接受计算命令后显示计算结果。工作屏幕像一张长长的草稿纸，称为 Notebook，它是后缀为 .ma 的文件类型。像其它的计算机文件一样，可以对 Notebook 进行新建、保存、打开、修改和打印等操作，这些操作命令都放在“File”菜单中。

在 DOS 界面下，屏幕上先显示输入的行标记“In[1]:=”，用户在输入一个表达式后，按回车键，系统立即开始计算输入的表达式。

在 Windows 环境中，用户先输入一行或多行表达式，再发计算表达式命令，系统在完成计算后输出计算结果时，再显示输入的行标记“In[1]:=”。表达式的输入形式在 Windows 界面和 DOS 界面是相同的。需要注意的是：在 Windows 界面中回车键仅表示命令之间的分隔标记。

计算表达式的命令方式有：

(1) 单击运行按钮  。

(2) 按 Shift + 回车键。

(3) 按键盘右边的小数字键盘上的“5”。

例如：启动 Mathematica 后，输入 Plot[x Sin[x] - 1, {x, -20 ,20}]，单击运行按钮  ，则绘出函数 $f(x) = x\sin x - 1$ 在区间 $[-20,20]$ 上的图形。屏幕显示如图 9.3 所示。

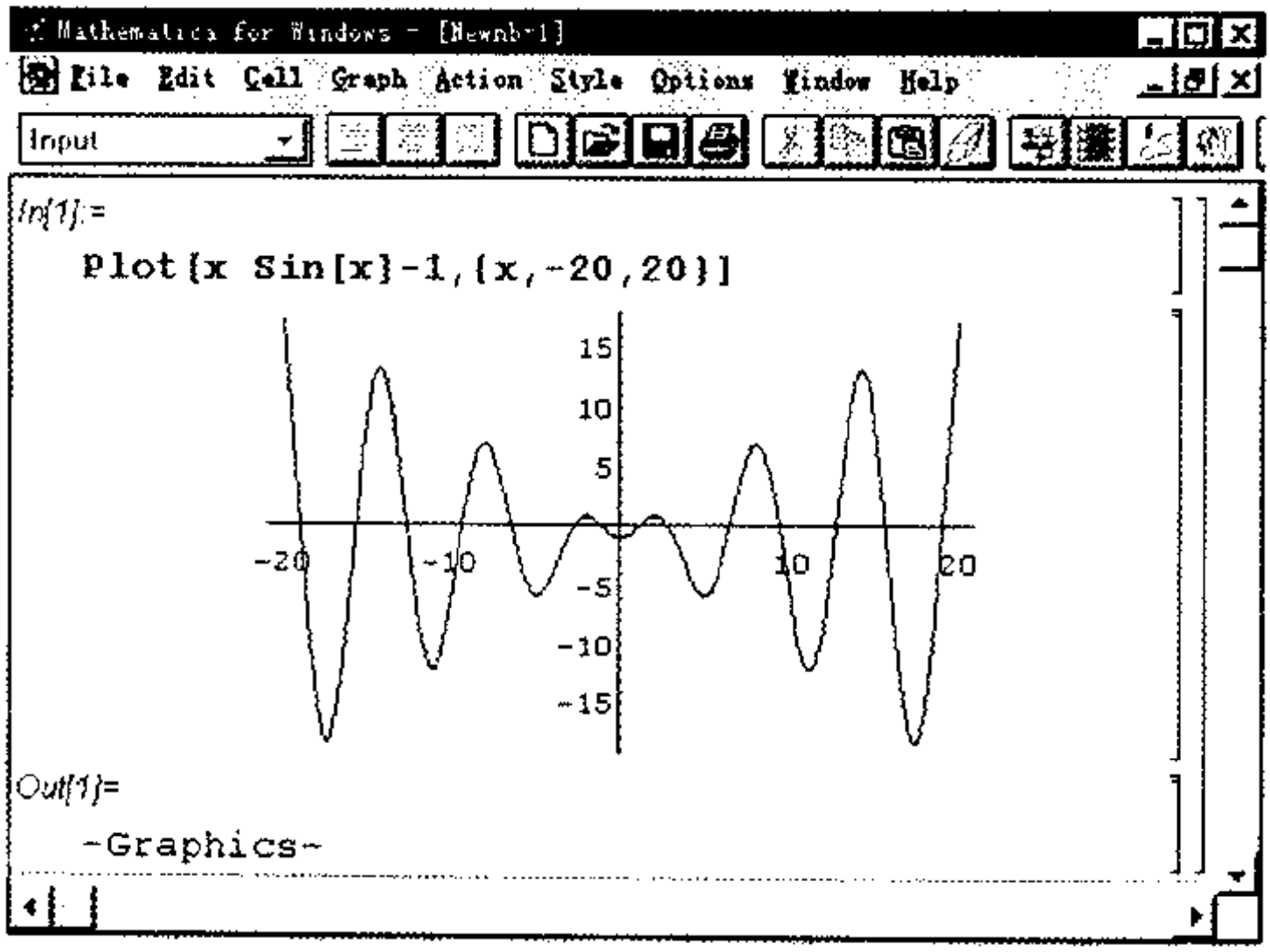


图 9.3　执行一条命令后的窗口

退出 Mathematica

在退出 Mathematica 窗口前，应关闭所有的文件(Notebook 等)。

在“File”菜单中单击“Exit”，如果窗口中还有未关闭的文件(Notebook)，则出现如图

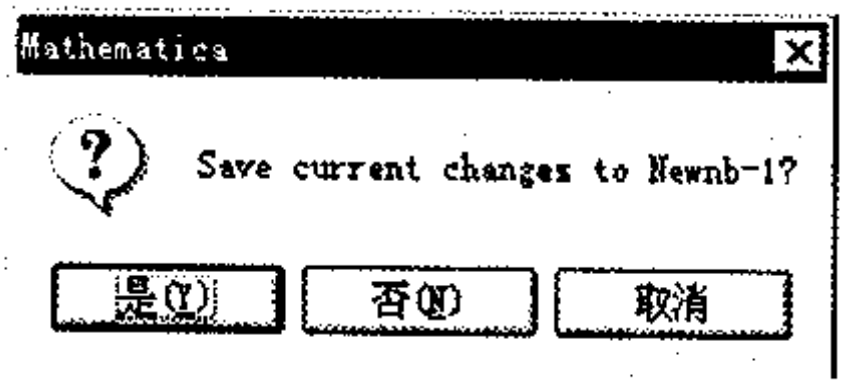


图 9.4　对话框

9.4所示的对话框,问你是否保存文件,单击“否(N)”命令按钮则系统关闭文件后退出Mathematica;单击“是(Y)”命令按钮则系统保存文件后再退出 Mathematica 对话框。保存文件的方式与 Windows 环境下的大多数应用软件类似,首次保存时,系统调出“Save as (保存)”对话框,用户在对话框的“File name”文本框中填入文件名,并可设置存放文件的位置;再次保存文件时,系统按原文件名和原位置存放。

在 Mathematica 中获取帮助

除了单击“Help(帮助)”菜单获取帮助外,在行文中输入“ ? 函数名”,可得到有关这个函数调用形式;输入“ ?? 函数名”,可得到有关这个函数更多的信息。例如,

```
输入:? NIntegrate          (* 单击运行按钮,则显示定积分函数的意义 *)
输出:NIntegrate[f, {x, xmin, xmax}] gives a numerical approximation to
     the integral of f with respect to x over the interval xmin to xmax.
```

9.2 插　　值

插值函数 InterpolatingPolynomial [data, var]

做出以 data 为插值点数据,以 var 为变量名的插值多项式。插值点数据是以点列的形式排列,所有点列{{x0,y0},{x1,y},...}组成一个表。运行后,系统显示所构造的用最少乘法次数的牛顿插值函数形式。

例 9.1:给出函数表,

x	1	2	4	5
y	16	12	8	9

计算 $f(1.2)$。

```
In[1]:= data = Table[{{1.,16}, {2.,12},{4.,8},{5.,9}}];
    f[x_] = InterpolatingPolynomial [data, x]
Out[1]= 16 + (-4. + (0.666667 + 0.0833333 (-4 + x)) (-2 + x)) (-1. + x)
In[2]:= f[1.2]
Out[2]= 15.1307
```

为了节省版面,人为地将“In[1]:=”和“data = {{1,16},{2,46},{4,88},{5,0}};”放在一行,后面的例子也采取这种方式。

在数据中还可以包括插值点处的导数,其数值按{{x0,{y0,dy0}},{x1,{y1,dy1}},...}形式存放数据。

例 9.2:给定数据 $f(0)=1.0, f(1)=0.75, f(3)=0.25, f'(3)=0.56$。作出三次插值多项式。

```
In[1]:= d ={{0,1.0},{1,0.75},{3,{0.25,0.56}}}
In[2]:= InterpolatingPolynomial[d,x]
Out[2]= 1. + (-0.25)x + 0.135(-3 + x)(-1 + x))x(* 做出牛顿插值多项式 *)
In[3]:= Simplify [%]     (* 化简插值多项式 *)
```

Out[3] = 1. + 0.155 x - 0.54x^2 + 0.135 x^3

插值函数 Interpolation [data, InterpolationOrder - > n],

对数据 data 进行插值运算,并可设置插值多项式的次数 n,默认值为 3,Interpolation 生成一个 InterpolatingFunction [插值范围, < >]目标(如 Out[2]),系统不显示所构造的插值函数。因此,用户直接用生成的插值函数计算函数的近似值。

例 9.3:按例 9.1 给出的数据,用 Interpolation 函数计算 $x = 1.2$ 处函数的近似值。

```
In[1]: = data = Table[{{1.,16}, {2.,12}, {4.,8}, {5.,9}}];
         g = Interpolation[data, InterpolationOrder - > 3]
Out[1] = InterpolatingFunction[{1, 5}, < > ]
In[2]: = g[1.2]
Out[2] = 15.1307
```

例 9.4:按例 9.1 给出的数据构造拉格朗日插值多项式,计算 $x = 1.2$ 处函数的近似值。

```
In[1]: = la[x_,x0_,a_,b_,c_]: = (x-a)(x-b)(x-c)/((x0-a)(x0-b)(x0-c))
P[x_] = la[x,1.0,2.,4.,5.]16 + la[x,2.0,1.,4.,5.]12 + la[x,4.,1,2.,5]8
        + la[x,5.0,1,2,4]9.
Out[1] = - 1.33333 (- 5. + x) (- 4. + x) (- 2. + x) + 0.75 (- 4 + x)(- 2 + x)
         (- 1 + x) - 1.33333 (- 5 + x) (- 2. + x) (- 1 + x) + 2. (- 5. + x)
         (- 4. + x) (- 1. + x)
In[2]: = P[1.2]
Out[2] = 15.1307
```

9.3 数值积分

定积分函数 NIntegrate

NIntegrate [f, {x, a, b}]　　计算定积分 $\int_a^b f(x)$。

NIntegrate [f, {x, a, b}, {y, c, d}]　　计算定积分 $\int_a^b dx\int_c^d f(x,y)\,dy$。

例 9.5:计算 $\int_1^2 \sin(\cos x^2)\,dx$。

例 9.6:计算 $\int_0^1\int_1^2\int_2^3\int_3^4 \frac{\sqrt{uv}}{\sin x + y}dxdydudv$。

```
In[1]: = NIntegrate [sin[cos[x^2]], {x,1,2}]
Out[1] = - 0.386289
In[2]: = NIntegrate [Sqrt[u v]/(sin[x] + y), {x,0,1}, {y,1,2}, {u,2,3}, {v,3,4}]
Out[2] = 1.56892
```

对于被积函数的奇点,可在积分区域上列出。

例 9.7:计算 $\int_{-1}^{1} \frac{1}{\sqrt{|x|}}dx$,0 是奇点。

例 9.8:计算复平面上的线积分,积分曲线从 -1 起到 -i,1,i 再回到 -1 的闭曲线。

```
In[1]:= NIntegrate[1/Sqrt[Abs[x]], {x, -1,0,1}]
Out[1]= 4.
In[2]:= NIntegrate[1/x, {x, -1, i, 1, i, -1}]
Out[2]= 1.11022 10^-16 + 6.28319 I   (* 由高斯公式得准确解 2πi      *)
```

例 9.9:分别取步长 0.02, 0.002 和 0.0002 计算积分 $\int_0^{0.6}(1-x^2)\,dx$。

```
In[1]:= f[x_]:=1.0-x^2 ; a= 0;b= 0.6;
In[2]:= t3=(f[a]/2+f[b]/2+NSum[f[k],{k,a,b,0.02}])0.02
Out[2]= 0.56076
In[3]:= t2= (f[a]/2+f[b]/2+NSum[f[k],{k,a,b,0.002}])0.002
Out[3]= 0.53128
In[4]:= t1= (f[a]/2+f[b]/2+NSum[f[k],{k,a,b,0.0002}])0.0002
Out[4]= 0.528328      (* 准确值 ∫_0^0.6 (1-x^2)dx = 0.528    *)
```

9.4 曲线拟合

拟合函数 Fit[data,fun,var]

用数据 data,以 var 为变量,按拟合的基函数 fun 的形式构造拟合函数。数据的表示方法与插值函数中的表示方式相同,都是使用点列形式。

例 9.10:给出下列数据,用二次多项式拟合这些数据。

x_i	-1.00	-0.50	0.00	0.25	0.75
y_i	0.2200	0.800	2.000	2.5000	3.8000

```
In[1]:= dd ={{-.1,0.22},{-0.5,0.8},{0,2.0},{0.25,2.5},{0.75,3.8}};
In[2]:= Fit[dd,{1,x,x^2},x]
Out[2]= 1.90736 + 2.20445 x + 0.472238 x^2
```

例 9.11:(不调用系统函数)按例 9.10 的数据做出线性拟合函数。

```
In[1]:= a ={{1.,-1.00},{1,-0.50},{1.,0.00},{1,0.25},{1,0.75}};
    MatrixForm[A=Transpose[a].a]     (* 用矩阵形式输出法方程 *)
    b=Transpose[a].{0.22,0.8,2.0,2.5,3.8}
    LinearSolve[A,b]                 (* 解法方程 *)
Out[1]= MatrixForm=      5.      -0.5   (* 法方程 *)
                        -0.5      1.875
    {9.32, 2.855}                    (* 常数项 b *)
    {2. 07151, 2.07507}
```

得到拟合函数 $p(x)=2.07151+2.07507x$。

例 9.12:给出下列数据:

x	-3	-2	-1	2	4
y	14.3	8.36	4.72	8.34	22.75

(1) 用最小二乘法求形如 $y = a + bx^2$ 的经验公式,并计算最小平方误差。

(2) 用三次多项式拟合这些数据。

```
In[1]:= ee = {{-3,14.3},{-2,8.3},{-1,4.7},{2,8.3},{4,22.7}};
In[2]:= Fit[ee, {1, x^2},x]
Out[2]= 3.53086 + 1.20046x^2
In[3]:= f[x_]:=3.5+1.2 x^2
In[4]= Sum[(d[[i,2]]-f[d[[i,1]]])^2,{i,1,5}]   (* d[[i,1]]表示 x_i,d[[i,2]]
表示 y_i *)
Out[4]= 0.0081          (* 最小平方误差 *)
In[5]:= Fit[ee, {1, x, x^2, x^3},x]
Out[5]= 3.54179-0.00292566x-1.19796x^2+0.000845875x^3
```

样条拟合函数 SplineFit[插值点序列,类型]

用插值点序列生成特定类型样条函数,其中样条类型的选项有:Cubic, Bezier 和 CompositeBezier。Cubic 表示分段三次样条函数,一阶连续可导,首尾端点的二阶导数为 0;Bezier 表示 Bezier 样条函数,函数仅在起点和终点过插值点,是一条凸函数;CompositeBezier 表示由一组 Bezier 样条函数复合而成的函数。

```
In[1]:= << NumericalMath'SplineFit'  (* 在 NumericalMath 中调出 SplineFit *)
```

在系统的子程序包 NumericalMath 中,还有一些数值计算方法的函数。

```
In[2]:= d = {{0.,-9.},{1.,-10.},{2.,-11.},{3.,-6.},{4.,11.}}
In[3]:= s = SplineFit[d,Cubic]    (*对数据 d 调用三次样条函数*)
Out[3]:= SplineFunction[Cubic,{0.,4.},<>]
In[4]:= s[3.7]
Out[4]= {3.7, 5.1395}                    (*s(3.7)=5.1395*)
```

9.5 非线性方程

非线性方程求根函数 FindRoot

FindRoot[方程,{x, x0}]　　用牛顿迭代计算方程在 x0 附近的一个近似根。

FindRoot[方程,{x, {x0, x1}}]

以 x0 和 x1 为初始值,用弦截法计算方程的一个近似根。

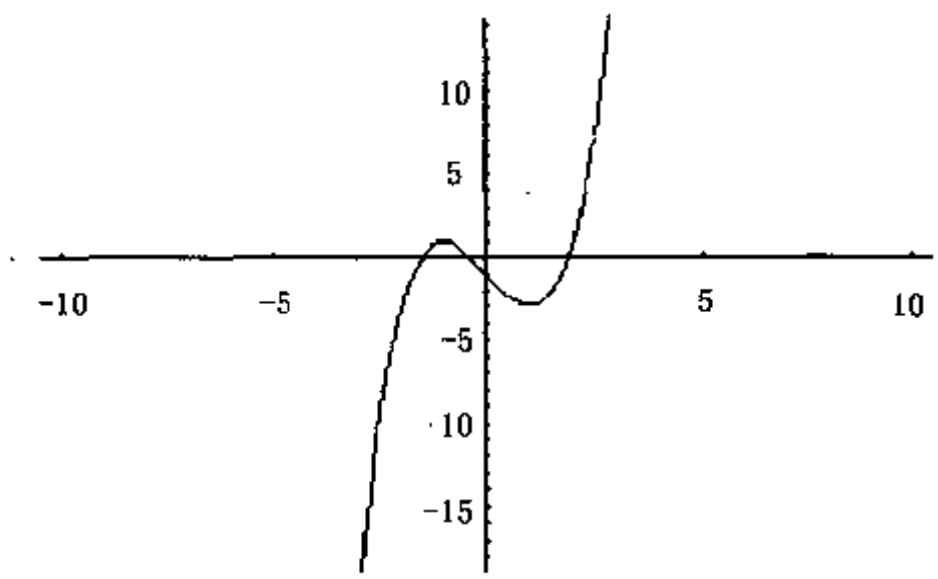
FindRoot[方程,{x,xstart,xmin,xmax}]

从 xstart 开始,在 xmin,xmax 范围内计算方程在 x 附近的一个数值解。

FindRoot[{方程组},{x,x0},{y,y0},...]　　计算联立非线性方程组的数值解。

例 9.13: $f(x) = x^2 - 3x - 1$,取 $\varepsilon = 10^{-4}$, $x_0 = 1.5$,用牛顿迭代法计算 $f(x)$ 的根。

```
In[1]:= Plot[x^3 - 3x - 1, {x, -10, 10}]
```



(* 图 9.5 $f(x) = x^3 - 3x - 1$ 的图示 *)

```
Out[1] = - Graphics -  (* 不妨先看一看函数的图示 *)
In[2]:= FindRoot[x^3 - 3x - 1 = =0, {x, 1.5}] (* 用两个等号表示方程的等号 *)
Out[2] = {x -> 1.87939}
```

例 9.14: $f(x) = x\sin x - 1$,用弦截法计算在 $x_0 = 18$, $x_1 = 21$ 附近的根。

```
In[3]:= FindRoot[x Sin[x] - 1 = =0, {x, 18, 21}]
Out[3] = {x -> 18.9025}
```

例 9.15: 用牛顿迭代法求解非线性方程组:

$$\begin{cases} x^2 + y^2 - 1 = 0 \\ x^3 - y = 0 \end{cases}, \quad 取 \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0.8 \\ 0.6 \end{pmatrix}。$$

```
In[4]:= FindRoot[{x^2 + y^2 - 1 = =0, x^3 - y = =0}, {x, 0.8}, {y, 0.6}]
                                        (* 用花括号将方程组括起来 *)
Out[4] = {x -> 0.826031, y -> 0.563624}
```

9.6 方程组求解

方程组求解函数 LinearSolve[A, B]

求解满足方程组 $AX = B$ 的一个解,A 为方阵。如果方程组有解,则输出一个解向量;如果方程组无解,则在屏幕上给出方程组无解的提示。

下列是常用矩阵计算符号和函数:

A + B	计算矩阵 A 和 B 的和
A·B	计算矩阵 A 和 B 的积
Det[M]	计算矩阵 M 的行列式
Inverse[M]	计算矩阵 M 的逆矩阵(M^{-1})
Transpose[M]	计算矩阵 M 的转置矩阵(M^T)

例 9.16: 求解方程组:

$$\begin{pmatrix} 2 & 4 & 6 \\ 1 & 4 & 7 \\ 3 & 8 & 12 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 26 \\ 25 \\ 46 \end{pmatrix}$$

```
In[1]:= A = {{2,4,6.0},{1,4,7},{3,8,12}}; b = {26,25,46}; LinearSolve[A,b]
Out[1]= {6.,-4.,5.}
```

例 9.17:用简单迭代法求解方程组:

$$\begin{pmatrix} 10 & -2 & -1 \\ -2 & 10 & -1 \\ -1 & -2 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 11 \\ -25 \\ 18 \end{pmatrix}$$

```
In[1]:= B = {{0,2./10,1./10},{2./10,0,1./10},{1./5,2./5,0}};
        g = {11./10,-25/10.,18./5}; x1 = {0,0,0};
        Do[x2 = B.x1 + g; x1 = x2; Print[x2],{k,6}]
Out[1]= (* 以{0,0,0}为初始值,迭代6步 *)
        {1.1, -2.5, 3.6}
        {0.96, -1.92, 2.82}
        {0.998, -2.026, 3.024}
        {0.9972, -1.998, 2.9892}
        {0.99932, -2.00164, 3.00024}
        {0.999696, -2.00011, 2.99921}
```

如果不想编程,可用平铺直叙的方式做题:

```
X1 = A·{0,0,0}
X2 = A·X1
  …
```

9.7 计算特征值和特征向量

计算特征值和特征向量函数

Eigenvalues [M]　　计算矩阵 *M* 的全部特征值

Eigenvectors [M]　　计算矩阵 *M* 的全部特征向量

Eigensystem [M]　　计算矩阵 *M* 的全部特征值和特征向量

例 9.18:计算矩阵 *A* 的特征值和特征向量。

$$A = \begin{pmatrix} 1 & 2 & 1 \\ -1 & 2 & 1 \\ 3 & 5 & 2 \end{pmatrix}$$

```
In[1]:= A = {{1., 2, 1}, {-1., 2, 1}, {3, 5, 2}};
c = Eigenvalues [A]  (* 计算全部特征值 *)
Out[2]= {4.44949, 1., -0.44949}
In[3]:= Eigenvectors [A]  (* 计算矩阵 A 的全部特征向量 *)
```

Out[3] = (* 输出略 *)

In[4]: = Eigensystem [A]　　(* 计算矩阵 A 的全部特征值和特征向量 *)

Out[4] = {{4.44949, 1., -0.44949}, {{0.382966, 0.210826, 0.899383}, {0.408248, -0.408248, 0.816497}, {-0.0738855, -0.402638, 0.912372}}}

(* 按形式 {{λ_1,λ_2,λ_3}, {{v_1}, {v_2}, {v_3}}输出特征值和特征向量 *)

不妨随机形成一个 100 阶的矩阵,再计算它的特征值和特征向量。对于阶数比较大的矩阵,可用 Max 函数挑出按模最大的特征值。

In[5]: = t = Max[Abs[c]]

Out[5] = 4.44949

9.8　常微分方程数值解

常微分方程数值解函数 NDSolve

NDSolve [{eqn1, eqn2,...}, y, {x, xmin, xmax}]

对常微分方程 eqn1,求函数 y 关于 x 在[xmin, xmax]范围内的数值解。

NDSolve [{eqn1, eqn2,...}, {y1, y2,...}, {x, xmin, xmax}]

对常微分方程组,求函数 y1, y2,... 关于 x 在 [xmin, xmax] 范围内的数值解。

因为要计算的是数值解,方程或方程组的初始条件也必须作为方程列出,并和方程放在一起。因此,要解 n 阶的常微分方程,必须同时给出 $n-1$ 个导数的初始值。

在系统的程序包(Packages)中有解常微分方程或方程组的龙格-库塔函数。

例 9.19:求解微分方程 $\ddot{x} + 2r\dot{x} + x + x^3 = f\cos(wt), 0 \leq t \leq 8$。

In[1]: = d = D[x[t], :t,2}] + 2 r D[x[t],t] + x[t]^3 - f Cos[w t]

ans = NDSolve[{d = = 0/. {r - > 1/10, f - > 1, w - > 4}, x'[0] = = 0, x[0] = = 0}, x, {t,0,8}]

Out[2] = {{x - > InterpolatingFunction[{0., 8.}, < >]}}

在求解过程中不妨画出解函数的图形。

In[3]: = Plot[x[t]/.ans, {t,0,8}, AxesLabel - > {"t","x"}, PlotLabel - > "x[t]"]

Out[3] = (* 图略 *)

例 9.20:求解常微分方程

$$\begin{cases} y'''(x) + y''(x) + 7y'(x) + 10y(x) = 0 \\ y(0) = 6 \\ y'(0) = -20 \\ y''(0) = 20 \end{cases}, x \in [0,2]$$

计算 $y(1.36)$。

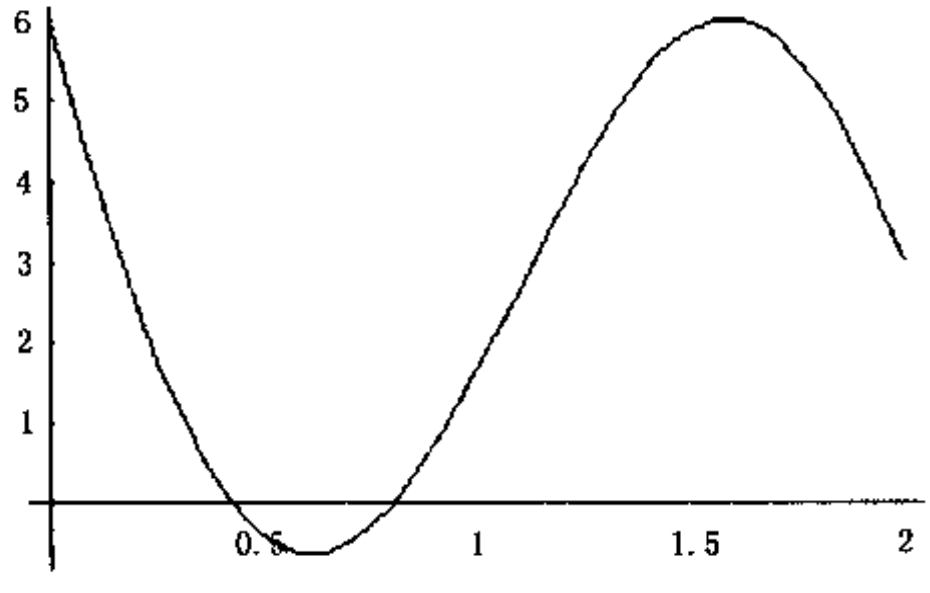
In[1]: = ss = NDSolve[{y'''[x] + y''[x] + 7 y'[x] + 10 y[x] = = 0, y[0] = = 6, y'[0] = = -20, y''[0] = = 20}, y, {x,0,2}]

Out[1] = {{y - > InterpolatingFunction[{0., 2.}, < >]}}

In[2]: = p = y/.First[%]　(* 取出近似函数的头部 *)

Out[2] = InterpolatingFunction[{0., 2.}, < >]

```
In[3]: = p[1.36]          (* 计算 y(1.36) *)
Out[3] = 5.09536
In[4]: = Plot [Evaluate [y[x]/.ss], {x,0,2}]  (* 画出解函数的图形 *)
```



(* 图 9.6 *)

例 9.21:求解常微分方程组

$$\begin{cases} x'(t) = -y(t) - x(t)^2 \\ y'(t) = 2x(t) - y(t), t \in [0,3] \\ x(0) = y(0) = 1 \end{cases}$$

计算 $x(1.2)$, $y(1.2)$ 的值。

```
In[5]: = NDSolve[{x'[t] = = -y[t] -x[t]^2, y'[t] = =2 x[t] -y[t],
          x[0] = = y[0] = =1}, {x, y}, {t, 0,3}]
Out[5] = {{x -> InterpolatingFunction[{0., 3.}, <>],
          {y -> InterpolatingFunction[{0., 3.}, <>]}}
In[6]: = {p = x/.First[%], q = y/.Last[%]}
Out[6] = {InterpolatingFunction[{0.,3.}, <>], InterpolatingFunction [{0.,3.}, <>]
In[7]: = {p[1.2], q [1.2]}  (* 计算 {x (1.2), y (1.2)}
Out[7] = {-0.301149, 0.432898}
```

例 9.22:用二阶龙格-库塔公式编程解初值问题,并画出近似解草图。

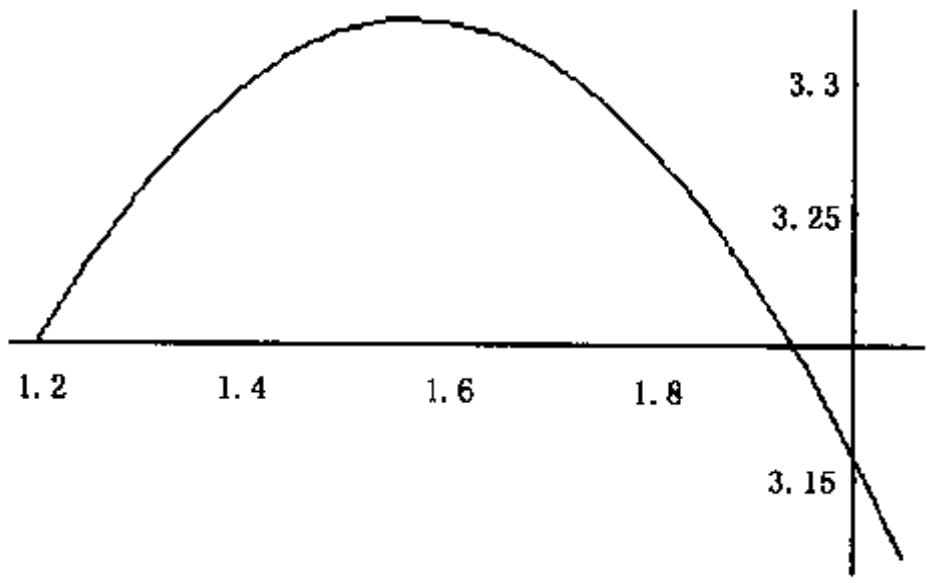
$$\begin{cases} \dfrac{dy}{dx} = \cos x\sqrt{y} \\ y(1.2) = 3.2 \end{cases}, 1.2 \leqslant x \leqslant 2.05, \text{取 } h = 0.05$$

解:计算公式:

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{2}(k_1 + k_2) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + h, y_n + hk_1) \end{cases}$$

```
In[1]: = f [x_,y_]: = Cos[x]Sqrt[y];      (* 定义函数 *)
    xylist = {{1.2,3.2}}; h = 0.05;       (* 赋初始值 *)
```

```
Do [xn = xylist[[n]][[1]]; yn = xylist[[n]][[2]];
    k1 = f [xn, yn];
    k2 = f [xn + h, yn + h k1];
    d = (k1 + k2) h/2;
    xylist = Append[xylist, {xn + h, yn + d}],        (* 将新点加入 xylist 序列中 *)
    {n, 1, 17}]                                       (* n 是循环控制量 *)
  xylist                                              (* 输出{x_n, y_n}, n = 1,2,…,17 *)
Out[1] = {{1.2, 3.2}, {1.25, 3.23038}, {1.3, 3.25662}, {1.35, 3.2786},
          {1.4, 3.29623}, {1.45, 3.30943}, {1.5, 3.31813}, {1.55, 3.3223},
          {1.6, 3.32192}, {1.65, 3.31699}, {1.7, 3.30752}, {1.75, 3.29358},
          {1.8, 3.27521}, {1.85, 3.2525}, {1.9, 3.22555},
          {1.95, 3.19449}, {2., 3.15945}, {2.05, 3.12059}}
In[2]:= ListPlot[xylist, PlotJoined -> True]          (* ListPlot[xylist] 画出点列 *)
```



(* 图 9.7 *)

```
Out[2] = -Graphics-
```

参 考 文 献

曹志浩，张玉德，李瑞忱编．1979．矩阵计算和方程求根．北京：人民教育出版社

冯康等．1978．数值计算方法．北京：国防工业出版社

何旭初，苏煜城，包雪松编．1980．计算数学简明教程．北京：人民教育出版社

胡同，邵勇，徐可等．1997．微积分．北京：高等教育出版社

奚梅成．1995．数值分析方法．合肥：中国科学技术大学出版社

徐萃薇．1985．计算方法引论．北京：高等教育出版社

张韵华．1998．Mathematica 符号计算系统实用教程．合肥：中国科学技术大学出版社

庄建南，林应举译．1989．计算数学例题与习题．北京：高等教育出版社

Ferenc Sidney．1982．数值分析的原理及过程．上海：上海科学技术文献出版社

Granville Sewell．1990．Computational Mathods of Linear Algebra．ELLIS HORWOOD

J．H．威尔金森著，石钟慈，邓建新译．1987．代数特征值问题．北京：科学出版社

Mount Holyoke College 编，白峰杉，蔡大用译．1998．数学实验室．北京：高等教育出版社