Numerical Analysis

NOTEBOOK

1907402030

熊　雄[1]

授课老师：张亚楠

2022 年 6 月 2 日

[1]mrxiongx@foxmail.com　苏州大学数学科学学院本科生

目录

Chapter1. 绪论

1.1 绝对误差

1. 准确数与近似数之差, 即 $e = x - x^*$.
2. 绝对误差限即为绝对误差的上界, 即 $|e| = |x - x^*| \leq \varepsilon$.
3. 对于 x 的近似值 $x^* = \pm 0.a_1a_2 \ldots a_n \times 10^m$, 若误差 $|x - x^*| \leq \frac{1}{2} \times 10^{m-l}$, 则 x^* 有 l 位有效数字. 例如 π 的近似值 3.1416 有五位有效数字.

Remark 1.1. 绝对误差不是误差的绝对值, 可能是正的, 也可能是负的. 由于精确值通常是不知道的, 因此绝对误差一般也是不可知的. 在做误差估计时, 我们所求的通常是误差限, 误差限不唯一, 越小越好, 一般是指所能找到的最小上界.

1.2 相对误差

Definition 1.1 (相对误差)**.** 记 $e^r = \frac{x-x^*}{x}$ 为 x^* 的相对误差, 记相对误差的上限为相对误差限, 即

$$|e_r| = \frac{|x - x^*|}{|x|} \leq \varepsilon_r^*. \tag{1.1}$$

因为近似值的精确程度通常取决于**相对误差**的大小, 因此近似值的精确程度不能仅仅看绝对误差, 更要看相对误差.

Example 1.1. 设近似值 $x^* = \pm 0.a_1a_2 \ldots a_n \times 10^m$ 有 n 位有效数字，则其相对误差限为：

$$\frac{1}{2a_1} \times 10^{-n+1}.$$

1.3 数值运算的误差估计

1.3.1 四则运算后的误差限

设近似数 x_1^* 与 x_2^* 的误差限分别为 $\varepsilon(x_1^*)$ 与 $\varepsilon(x_2^*)$, 则他们的四则运算后的误差限为:

$$\begin{cases} \varepsilon\left(x_1^* \pm x_2^*\right) \approx \varepsilon\left(x_1^*\right) + \varepsilon\left(x_2^*\right), \\ \varepsilon\left(x_1^* \cdot x_2^*\right) \approx \left|x_2^*\right| \varepsilon\left(x_1^*\right) + \left|x_1^*\right| \varepsilon\left(x_2^*\right), \\ \varepsilon\left(x_1^*/x_2^*\right) \approx \frac{\left|x_2^*\right| \varepsilon\left(x_1^*\right) + \left|x_1^*\right| \varepsilon\left(x_2^*\right)}{\left|x_2^*\right|^2}. \end{cases} \tag{1.2}$$

1.3.2 一元函数的误差限

设 x^* 为 x 的近似值, 若 $f(x)$ 可导, 则有

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

由于 $x^* - x$ 相对较小, 所以当 $|f''(x)|$ 与 $|f'(x)|$ 的比值不是很大时, 我们可以忽略二阶项, 即

$$|f(x^*) - f(x)| \approx |f'(x)| \cdot |x^* - x|.$$

因此, 可得一元函数的误差限

$$\varepsilon(f(x^*)) \approx |f'(x)|\,\varepsilon(x^*) \approx |f'(x^*)|\,\varepsilon(x^*) \tag{1.3}$$

1.3.3 多元函数的误差限

Theorem 1.1. *对于可微函数 $A = f(x_1, x_2, \ldots, x_n)$, 计算 $A^* = f(x_1^*, x_2^*, \ldots, x_n^*)$ 时的误差限为:*

$$\varepsilon(A^*) = \sum_{k=1}^{n} \left| \left(\frac{\partial f}{\partial x_k} \right) \right| \varepsilon\left(x_k^*\right) \tag{1.4}$$

证明. 由 Taylor 展开得函数值 A^* 的误差 $e(A^*)$ 为

$$\begin{aligned} e(A^*) &= A^* - A = f\left(x_1^*, x_2^*, \cdots, x_n^*\right) - f\left(x_1, x_2, \cdots, x_n\right) \\ &\approx \sum_{k=1}^{n} \left(\frac{\partial f\left(x_1^*, x_2^*, \cdots, x_n^*\right)}{\partial x_k} \right) \left(x_k^* - x_k\right) = \sum_{k=1}^{n} \left(\frac{\partial f}{\partial x_k} \right)^* e_k^* \end{aligned} \tag{1.5}$$

于是误差限

$$\varepsilon(A^*) \approx \sum_{k=1}^{n} \left| \left(\frac{\partial f}{\partial x_k} \right)^* \right| \varepsilon\left(x_k^*\right), \tag{1.6}$$

而 A^* 的相对误差限为

$$\varepsilon_r^* = \varepsilon_r(A^*) = \frac{\varepsilon(A^*)}{|A^*|} \approx \sum_{k=1}^{n} \left| \left(\frac{\partial f}{\partial x_k} \right)^* \right| \frac{\varepsilon\left(x_k^*\right)}{|A^*|}.$$

□

1.4 数值计算稳定性

若误差在计算过程中越来越大, 则算法不稳定, 即初始误差在计算中传播导致误差增长很快. 否则算法是稳定的.

Example 1.2. 要计算 $I_n = \int_0^1 \frac{x^n}{x+5} dx$, 则我们有

$$I_n = \frac{1}{n} - 5I_{n-1}, \quad I_{n-1} = \frac{1}{5}\left(\frac{1}{n} - I_n\right).$$

第一个算法是不稳定的, 因为误差 $e_n = -5e_{n-1} = (-5)^n e_0$, 误差随迭代次数而增加; 第二个算法是稳定的, 因为误差 $e_n = -\frac{1}{5}e_{n-1} = \left(-\frac{1}{5}\right)^n e_0$, 误差会逐渐减小.

Example 1.3 (课本 P20 Ex.11)**.** 序列 $\{y_n\}$ 满足:

$$y_n = 10y_{n-1} - 1, \quad n = 1, 2, \ldots,$$

取 $y_0 = \sqrt{2} \approx 1.41$, 计算得到 y_n 的误差有多大? 计算过程是否稳定?

证明. 计算 y_0 的误差为

$$e_0 = |y_0 - y_0^*| = \sqrt{2} - 1.41 < \frac{1}{2} \times 10^{-2} = \delta_0.$$

计算到 y_n 时的误差为

$$e_n = |y_n^* - y_n| = \left|(10y_{n-1}^* - 1) - (10y_{n-1} - 1)\right| = 10\left|y_{n-1}^* - y_{n-1}\right| = 10e_{n-1}.$$

因此

$$e_n = 10e_{n-1} = 10^2 e_{n-2} = \cdots = 10^n e_0 < 10^n \delta_0,$$

计算到 y_n 时的误差为 $10^n e_0$, 比计算 y_0 时的误差放大了 10^n 倍, 因此计算过程是不稳定的. □

1.5 数值计算的原则

1. 避免除数绝对值远小于被除数绝对值;
2. 避免相近数相减;
3. 避免大数吃小数;
4. 简化计算.

1.6 数值分析的主要任务

- 算法设计: 构造求解各种数学问题的高效可靠的数值方法.
- 算法分析: 研究数值方法的收敛性、稳定性、计算复杂性、计算精度等.
- 算法实现: 编程实现与优化、软件开发和维护等.

1.7 好的数值方法

一个好的数值方法一般需满足以下几点:

- 可靠：有可靠的理论分析, 即收敛性、稳定性等有数学理论保证.
- 高效：有良好的计算复杂性 (时间和空间).
- 可用：可以并易于在计算机上编程实现.
- 实用：要通过数值试验来证明是行之有效的.

Remark 1.2. 需要指出的是, 数值方法是近似计算, 因此求出的解是有误差的. 对于同一问题, 不同的算法在计算性能上可能相差千百倍或者更多.

因此, 在学习的过程中, 我会尽量注意以下几点, 这几点也会在后续的笔记中着重进行强调.

- 注意掌握数值方法的基本思想和原理;
- 注意数值方法的设计和分析的一些常用技巧;
- 重视误差分析、收敛性和稳定性的基本理论;
- 适量的数值计算训练 (包括编程实践和手工推导).

Chapter2. 插值法

笔者在学习过程中产生了一个疑问: 我们为何不利用 Taylor 公式, 而要另外建立一套插值方法呢? 根据之前分析课程我们知道, Taylor 公式会有余项, 而余项产生的误差往往在远离展开点时很难被控制, 例如下面这个例子.

Example 2.1. 设 $f(x) = e^x$, 则在 0 处展开的一阶至四阶 *Taylor* 公式分别为

$$P_1(x) = 1 + x + o(x),$$

$$P_2(x) = 1 + x + \frac{x^2}{2} + o(x^2),$$

$$P_3(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + o(x^3),$$

$$P_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + o(x^4),$$

我们容易发现, 在 x 远离 0 时, 估计的误差虽然会随着阶数的增加而减小, 但是仍然会非常大.

可能又会有读者提出疑问: Taylor 级数虽然有着如上缺陷, 但是利用 Taylor 级数的收敛性不是也可以吗? 这确实是一个很好的想法, 我们知道 $e^x = \sum_{x=0}^{\infty} \frac{x^n}{n!}$ 对任意的 $x \in \mathbb{R}$ 均成立, 可是在数值分析中, 我们要考虑到实际的方面: 计算机的计算资源是有限的, 真正意义上的无穷不可能达到, 所以 Taylor 级数在实现中仍然非常受限.

Taylor 公式的缺陷就在于: 其余项为 $(x - x_0)^n$ 这样的形式, 而这样的形式将估计的误差集中在了展开点处. 我们之后介绍的 Lagrange 插值方法则会将误差均匀地分布在插值点处.

2.1 Lagrange 插值

2.1.1 利用 Lagrange 基函数求插值多项式

Lagrange 插值的想法很朴素, 完全是一种构造性的想法. 要使得 $P(x_i) = y_i,\ i = 1, \dots, n$, 那么我只需要构造如下形式

$$P(x) = L_0(x)y_0 + L_1(x)y_1 + \dots + L_n(x)y_n,$$

使得这些 $L_i(x)$ 能起到类似于示性函数的作用即可.

Definition 2.1. 已知 $f(x_i) = y_i,\ i = 0, 1, 2, \dots, n$, 由 *Lagrange* 插值法可得插值多项式:

$$L_n(x) = \sum_{k=0}^{n} l_k(x)y_k \tag{2.1}$$

其中 $l_k(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}$, 则我们有

$$l_i\left(x_j\right) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad i,j = 0,1,\ldots,n. \tag{2.2}$$

我们称上式中 $l_i(x)$ 为**插值基函数**.

除具体实现方法外, 我们很关心的还有一点: Lagrange 插值方法的余项是什么? 要知道, Taylor 公式就是因为余项导致的估计误差全部集中在一个点处而被我们抛弃.

2.1.2 Lagrange 插值的余项

Theorem 2.1. 设 $f \in C^{n+1}([a,b];\mathbb{R})$, 在 $x_0, x_1, \ldots, x_n$ 处进行插值, 则 $\forall x \in [a,b]$, *Lagrange* 插值的截断误差/插值余项为:

$$R_n(x) = f(x) - L_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w_{n+1}(x), \ \xi \in (a,b) \tag{2.3}$$

其中 $w_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$.

我们可以看到: Lagrange 插值公式的余项与 Taylor 公式不同. 误差是均匀分布分布在所有插值点附近的, 而非集中在一点处. 这就是为何 Lagrange 插值在一个区间上进行函数的估值效果更优.

Remark 2.1.

- 余项公式只有当 $f(x)$ 的高阶导数存在时才能使用;
- 余项中的 ξ_x 与 x 是相关的, 通常无法确定, 因此在实际应用中通常是估计其上界, 即如果有 $M_{n+1} = \max_{a\leq x\leq b}\left|f^{(n+1)}\right|$, 则

$$|R_n(x)| \leq \frac{M_{n+1}}{(n+1)!}\left|\omega_{n+1}(x)\right|.$$

- 在利用插值方法计算插值点 x 上的近似值时, 应该尽量选取与 x 相近的插值节点.

2.1.3 Lagrange 基函数的两个重要性质

如果 $f(x)$ 是一个次数不超过 n 的多项式, 则 $f^{(n+1)}(x) \equiv 0$, 因此

$$R_n(x) = \frac{f^{(n+1)}\left(\xi_x\right)}{(n+1)!}\omega_{n+1}(x) \equiv 0 \tag{2.4}$$

因此我们有下面的性质.

Proposition 2.2. 当 $f(x)$ 为一个次数不超过 n 的的多项式时, 其 n 次插值多项式是精确的.

Proposition 2.3 (课本 P48 Ex.4)**.** 设 x_j 为互异节点 $(j = 0, 1, \dots, n)$, 则我们有

1. $\sum_{j=0}^{n} x_j^k l_j(x) \equiv x^k, \quad k = 0, 1, \cdots, n;$

2. $\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x) \equiv 0, \quad k = 0, 1, \cdots, n.$

证明. 1. 先看这个恒等式的形式, 左边是一个类似于 Lagrange 多项式的形式, 右边是一个关于 x 的函数. 因此, 我们可以对 x^k 构造 Lagrange 多项式. 令

$$f(x) = x^k, \quad k = 0, 1, \dots, n.$$

因为给出了 n 个互异节点, 因此由这 n 个互异节点构造 Lagrange 多项式, 得

$$L_n(x) = \sum_{j=0}^{n} x_j^k l_j(x).$$

因为 $k \le n$, 所以 $f^{(n+1)}(x) = 0$. 因此误差为

$$|R(x)| = |f(x) - L_n(x)| = \left|\frac{f^{(n+1)}(x)}{(n+1)!} w_{n+1}(x)\right| \equiv 0.$$

从而

$$f(x) \equiv L_n(x),$$

即

$$\sum_{j=0}^{n} x_j^k l_j(x) \equiv x^k.$$

2. 可以先将括号展开:

$$\left(x_j - x\right)^k = \sum_{i=0}^{k} \binom{k}{i} x_j^i x^{k-i}.$$

于是我们有

$$\begin{aligned}
\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x) &= \sum_{j=0}^{n} \sum_{i=0}^{k} \binom{k}{i} x_j^i (-x)^{k-i} l_j(x) \\
&= \sum_{i=0}^{k} \sum_{j=0}^{n} \binom{k}{i} x_j^i (-x)^{k-i} l_j(x) \\
&= \sum_{i=0}^{k} \left(\sum_{j=0}^{n} \binom{k}{i} x_j^i (-x)^{k-i} l_j(x) \right) \\
&= \sum_{i=0}^{k} \left(\binom{k}{i} (-x)^{k-i} \sum_{j=0}^{n} x_j^i l_j(x) \right).
\end{aligned}$$

由 a) 的结果可知,$\sum_{j=0}^{n} x_j^k l_j(x) \equiv x^k$, 因此

$$\begin{aligned}\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x) &= \sum_{i=0}^{k} \left(\binom{k}{i} (-x)^{k-i} \sum_{j=0}^{n} x_j^i l_j(x) \right) \\ &= \sum_{i=0}^{k} \binom{k}{i} (-x)^{k-i} x^i \\ &= (-x + x)^k \\ &= 0.\end{aligned}$$

因此

$$\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x) \equiv 0.$$

虽然这样展开是可以证明的, 但是再对比一下 1), 这两个恒等式其实很像. 如果我们把 1) 的思路用在这里, 也就是说要先找到一个函数, 用它构造 Lagrange 插值函数我们可以得到 $\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x)$. 对比一下 Lagrange 插值函数的一般形式, 构造出来的函数应该是

$$f(t) = (t - x)^k.$$

对 $f(t)$ 构造 Lagrange 插值函数, 得到

$$L_n(t) = \sum_{j=0}^{n} \left(x_j - x\right)^k l_j(t).$$

很明显与待证等式左边的部分极其类似. 由 1) 可知

$$f(t) = (t - x)^k \equiv L_n(t) = \sum_{j=0}^{n} \left(x_j - x\right)^k l_j(t),$$

该等式对任意的 t 均成立. 要令 $f(t) = 0$, 只需令 $t = x$ 即可, 得到的等式很显然为

$$f(x) = (x - x)^k = 0 \equiv L_n(x) = \sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x),$$

即

$$\sum_{j=0}^{n} \left(x_j - x\right)^k l_j(x) \equiv 0.$$

□

Example 2.2. 令 x_i $(i = 1, 2, 3, 4, 5)$ 是互异节点, $l_i(x)$ 为对应的 5 次 *Lagrange* 插值基函数. 求

1. $\sum_{i=0}^{5} x_i^5 l_i(0)$;
2. $\sum_{i=0}^{5} \left(x_i^5 + x_i^4 + 4x_i^2 - x_i + 4\right) l_i(x)$.

Solve. 由 Proposition 2.3,

$$\sum_{i=0}^{5} x_i^5 l_i(0) = 0^5 = 0,$$

$$\sum_{i=0}^{5} \left(x_i^5 + x_i^4 + 4x_i^2 - x_i + 4\right) l_i(x) = x^5 + x^4 + 4x^2 - x + 4.$$

□

2.1.4 Runge 现象

Definition 2.2 (龙格现象)**.** 在科学计算领域, **龙格现象** *(Runge)* 指的是对于某些函数, 使用均匀节点构造高次多项式差值时, 在插值区间的边缘的误差可能很大的现象.

在数值分析中, 高次插值会产生龙格现象. 即在两端处波动极大, 产生明显的震荡. 它是由 Runge 在研究多项式差值的误差时发现的, 这一发现很重要, 因为它表明, 并不是插值多项式的阶数越高, 效果就会越好. 具体实验将会在 Section 2.5以 $f(x) = \frac{1}{1+25x^2}$ $(-1 \le x \le 1)$ 进行代码演示.

我们数学分析中学过的魏尔斯特拉斯定理,

Theorem 2.4 (Weierstrass Approximation Theorem)**.** 设 $f \in C([a, b]; \mathbb{R})$, 则 $\forall \varepsilon > 0$, $\exists$ 多项式 $P(x) \in C([a, b])$ 使得

$$\max_{a \le x \le b} |f(x) - P(x)| < \varepsilon.$$

Remark 2.2. *Weierstrass* 定理表明, 任意一个闭区间上的连续函数都可以用多项式来一致逼近, 即实系数多项式构成的集合在 $C[a, b]$ 内是处处稠密的.

Theorem 2.4说的是任何一个函数都能用一个次数足够高的多项式进行逼近, 这是不是和 Runge 现象矛盾了呢? 实际上并没有, 因为 Theorem 2.4中并没有要求多项式的值在给定点上与函数值相同, 事实上我们有下面两个定理结果

Theorem 2.5 (Faber Theorem)**.** 通过任意规则给定插值节点

$$a \le x_0^{(n)} \le x_1^{(n)} \ldots \le x_n^{(n)} \le b, \quad n \ge 0,$$

则存在一个连续函数, 它在这组节点上插值多项式不收敛于它.

Theorem 2.6 (插值收敛性定理)**.** *给定连续函数, 则存在某种规则下得到的一组插值节点*

$$a \le x_0^{(n)} \le x_1^{(n)} \dots \le x_n^{(n)} \le b, \quad n \ge 0,$$

使得这个函数在这组插值节点上的插值多项式收敛于它.

其中 Faber 定理 Theorem 2.5告诉我们, 使用一种特定的规则 (如取等距节点), 盲目地增加插值多项式的次数是不合理的; 而插值收敛性定理 Theorem 2.6虽然保证对于任何连续函数都有一组插值节点能很好地收敛, 但也没有给出求节点的方法. 为了保证插值多项式在区间划分足够精细后, 能够收敛于目标函数, 我们将在后面引进分段插值.

Remark 2.3. *除了 Runge 现象以外, 俄罗斯数学家 C. H. Bernstein 在 1916 年还给出了以下结论: 函数 $f(x) = |x|$ 在 $[-1,1]$ 上取 $n+1$ 个等距节点进行插值, 其中 $x_0 = -1, x_n = 1$, 构造出的 n 插值多项式记为 $P_n(x)$. 当 n 不断增大时, 除了 $-1, 0, 1$ 这三个点外, 在 $[-1,1]$ 中任何点处 $P_n(x)$ 都不收敛于 $f(x)$. 该结论的证明可以参见《函数构造论》(下册, 何旭初和唐述剑翻译, 科学出版社,1959).*

最后, 值得一提的是, 在进行 Lagrange 插值的过程中, 我们往往先选取距离要估计点最靠近的插值点进行多项式构造, 并确保要估计点落在插值点构成的区间内部, 这样得到的估计往往会更精确. 然而从更加实际的方面来看待 Lagrange 插值公式, 我们发现其计算过程非常繁杂. 具体体现在: 我们在计算高阶 Lagrange 插值公式时并无法用到我们先前计算过的低阶 Lagrange 插值公式的结果, 即当插值节点增减时, 计算要全部重新进行. 我们为了改进这一缺陷, 想要找到低阶与高阶公式之间的联系, 我们将在下一节引入 Newton 插值.

2.2 差商与 Newton 插值

Newton 插值方法也源于一个朴素的想法: 使得多项式具有如下形式

$$P(x) = a_0 + a_1(x - x_0) + \dots + a_n(x - x_0) \dots (x - x_{n-1}). \tag{2.5}$$

这样一种形式的好处是, 当我们要加入插值节点 x_{n+1} 时, 我们只需要在末尾加上一项 $a_{n+1}(x - x_0) \dots (x - x_n)$ 即可, 使得计算十分地方便. 那么我们如何确定这些 a_i 呢? 首先我们先给出 k 阶差商的概念.

2.2.1 差商及其性质

Definition 2.3. *称*

$$f[x_0, x_1, \dots, x_k] = \frac{f[x_0, x_1, \dots, x_{k-1}] - f[x_1, x_2, \dots, x_k]}{x_0 - x_k} \tag{2.6}$$

为 $f(x)$ 的 k 阶差商.

由概念可以很容易证明差商的一些性质:

- k 阶差商可表示为 $f(x_0),f(x_1),\dots,f(x_n)$ 的线性组合, 即

$$f\left[x_0,x_1,\dots,x_k\right]=\sum_{j=0}^{k}\frac{f\left(x_j\right)}{w_{k+1}'\left(x_j\right)}. \tag{2.7}$$

- 差商具有对称性, 即

$$f[x_0,x_1,\dots,x_n]=f[x_1,x_0,x_2,\dots,x_n]=\cdots=f[x_1,\dots,x_n,x_0]. \tag{2.8}$$

- 若 $\{x_i\}_{i=0}^{n}\in[a,b]$ 且 $f\in C^n[a,b]$. 则 $\exists\xi_x\in(a,b)$ 满足

$$f[x_0,\dots,x_n]=\frac{1}{n!}f^{(n)}(\xi). \tag{2.9}$$

差商的极限为导数, 那么我们有很多与微积分类似的结果, 比如若 $f(x)$ 是 n 次多项式, 则 $n+1$ 阶差商 $f[x,x_0,x_1,\dots,x_n]=0$. 另外计算差商时, 可以作差商表方便计算.

x_i	$f(x_i)$	$f[*,*]$	$f[*,*,*]$	$\cdots$	$f[*,\cdots,*]$
x_0	$f(x_0)$				
		$f[x_0,x_1]$			
x_1	$f(x_1)$		$f[x_0,x_1,x_2]$		
		$f[x_1,x_2]$		$\ddots$	
x_2	$f(x_2)$		$f[x_2,x_3,x_4]$	$\cdots$	$f[x_0,x_1,\cdots,x_n]$
		$f[x_2,x_3]$		$\iddots$	
x_3	$f(x_3)$		$\iddots$		
$\vdots$	$\vdots$	$\iddots$			
x_n	$f(x_n)$				

2.2.2 Newton 插值多项式

我们可以发现 $a_0=P(x_0)=f(x_0)$, 由 $P_1=f(x_1)$ 我们可以解出, $a_1=\frac{f(x_1)-f(x_0)}{x_1-x_0}$, 我们发现 a_i 与差商的形式恰好吻合. 事实上, 利用数学归纳法我们容易证明 $a_i=f\left[x_0,x_1,\dots,x_i\right]$ 满足所有插值条件, 故我们得到了 Newton 插值公式.

Newton 插值多项式为:

$$N_n(x)=f\left(x_0\right)+f\left[x_0,x_1\right]\left(x-x_0\right)+\cdots+f\left[x_0,x_1,\dots,x_n\right]\left(x-x_0\right)\left(x-x_1\right)\dots\left(x-x_{n-1}\right) \tag{2.10}$$

实际上, 由插值多项式的存在唯一性可知, 用 Newton 插值法和用 Lagrange 插值法得到的

同次插值多项式是完全相同的, 即

$$N_n(x) \equiv L_n(x)$$

因此截断误差也是完全一致的, 即

$$f[x, x_0, \ldots, x_n] \prod_{i=0}^{n} (x - x_i) \equiv \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \quad \xi_x \in (a, b) \tag{2.11}$$

由此, 我们立即可以得到下面的结论, 这就是我们前面提到的差商与导数之间的关系(2.9)式.

Proposition 2.7. *设 $f \in C^n([a,b]; \mathbb{R})$, 且 $f^{(n+1)}(x)$ 在 (a,b) 内存在. 则 $\forall x \in [a,b]$, 存在 $\xi_x \in (a,b)$, 使得*

$$f[x, x_0, \ldots, x_n] = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}. \tag{2.12}$$

Remark 2.4. *Newton 插值余项更具实用性, 因为它仅涉及插值点与插值节点的差商, 而不需要计算导数, 因此在导数不存在的情况下仍然可以使用. 但在计算差商 $f[x, x_0, \ldots, x_n]$ 时, 由于 $f(x)$ 未知, 只能使用插值得到的近似值, 因此得到的差商可能具有一定的偏差.*

2.2.3 Newton 插值法的代码实现

使用 Newton 插值法计算的核心在于求解差商表, 在程序中用矩阵 diff 表示, 为符合编程习惯, 将课本上的差商表取转置存储, 即

$$\text{diff} = \begin{bmatrix} f[x_1] & f[x_2] & f[x_3] & \cdots & f[x_n] \\ & f[x_1, x_2] & f[x_2, x_3] & \cdots & f[x_{n-1}, x_n] \\ & & f[x_1, x_2, x_3] & \cdots & f[x_{n-2}, x_{n-1}, x_n] \\ & & \ddots & \vdots & \\ & & & f[x_1, x_2, \cdots, x_n] & \end{bmatrix} \tag{2.13}$$

利用差商的性质, 从上至下递推求解差商表, 递推式如下 (记 $\text{diff}_{ij} = a_{ij}$):

$$a_{ij} = \begin{cases} f(x_j), & i = 1 \\ \frac{a_{i-1,j} - a_{i-1,j-1}}{x_j - x_{j-i+1}}, & i \geqslant 2 \end{cases} \tag{2.14}$$

再利用差商表点的对角线上的元素即可容易地求出 Newton 插值多项式, 即

$$\begin{aligned} N_n(x) =& f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) + \cdots \\ &+ f[x_1, x_2, \cdots, x_n](x - x_1)(x - x_2) \cdots (x - x_{n-1}) \\ =& \sum_{i=1}^{n} f[x_1, x_2, \cdots x_i] \pi_{i-1}(x) \end{aligned} \tag{2.15}$$

则具体函数代码 Newton.m 如下

```
% Newton 插值法,a 为插值点列向量,b 为插值点对应的函数值列向量
function N = Newton(a, b)
    [n, ~] = size(a);
    diff = zeros(n, n); % 初始化差商表
    diff(1, :) = b';
    syms x % 定义参数 x
    now = 1; % 对应上文中的 pi(x)
    N = diff(1, 1);
    % 递推计算差商表，并同时计算出插值多项式 N(x)
    for i = 2 : n
        for j = i : n
            diff(i, j) = (diff(i-1, j)-diff(i-1, j-1)) / (a(j) - a(j-i+1));
        end
        now = now * (x-a(i-1));
        N = N + diff(i, i) * now;
    end
    N = expand(N); % 展开插值多项式
    N = vpa(N, 5); % 将分式系数转化为小数系数
end
```

Remark 2.5. 本笔记全部使用 *MATLAB* 进行编程，使用 *MATLAB* 编程的好处在于可以容易地绘制函数图像，可实现带参数的运算，可化简/展开多项式，除了这几个功能外的更高级的功能，如解方程组，求三次样条插值等功能均由本人实现.

2.3 Hermite 插值

给定的函数关系中含有导数的插值即称为 Hermite 插值，Hermite 插值进一步要求插值多项式不仅在插值点处的函数值和原函数一致，还要求插值多项式在插值点处的一阶导数值与原函数的一阶导数一致.

Definition 2.4. 设置插值节点为 $x_0, x_1, \dots, x_n$，若要求插值多项式满足

$$P(x_i) = f(x_i), \quad , P'(x_i) = f'(x_i), P''(x_i) = f''(x_i), \dots, P^{(m)}(x_i) = f^{(m)}(x_i).$$

则计算这类插值多项式的方法就称为 ***Hermite* 插值**.

Remark 2.6. 在 *Hermite* 插值中，并不一定需要在所有插值节点上的导数都相等，在有些情况下，可能只需要在部分插值点上的导数值相等即可.

我们更习惯于利用带重节点的差商表进行计算，利用此法要比书上所说的待定系数求得插值多项式的方法简单很多.

2.3.1 重节点差商与 Taylor 插值

首先介绍差商的一个重要性质.

Theorem 2.8. *设 $x_0, x_1, \dots, x_n$ 为 $[a,b]$ 上的互异节点, $f(x) \in C^n[a,b]$, 则 $f[x0, x_1, \dots, x_n]$ 是其变量的连续函数.*

Definition 2.5. *定义 $f(x) \in C^n[a,b]$ 在点 $x \in [a,b]$ 的n* **阶重节点差商**为

$$f[\underbrace{x, x, \dots, x}_{n+1 \text{ 个}}] \triangleq \lim_{x_i \to x} f[x, x_1, x_2, \dots, x_n] = \frac{1}{n!} f^{(n)}(x). \tag{2.16}$$

在 Newton 插值公式(2.10)式中, 令 $x_i \to x_0,\ i = 1, 2, \dots, n$, 则

$$\begin{aligned} N_n(x) =& f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &+ f[x_0, x_1, \dots, x_n](x - x_0) \cdots (x - x_{n-1}) \\ =& f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n. \end{aligned}$$

这就是 **Taylor 插值**, 也即是 $f(x)$ 在 x_0 点的 Taylor 展开式的前 $n+1$ 项之和. 余项为

$$R_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x - x_0)^{n+1} \tag{2.17}$$

Remark 2.7. *Taylor 插值就是在一个插值点 x_0 的 n 次 Hermite 插值.*

2.4 三次样条插值

2.4.1 三次样条插值理论简介

三次样条插值的思想和前三者有较大的不同. 前三种方法, 无论 Lagrange, Newton, 或者是 Hermite 方法, 都属于基函数插值. 它们的特点是都有一个基函数, 这些基函数在不同的插值点往往取值不同且有规律. 我们通常先构造出使我们满意的基函数, 接着再利用基函数进行线性组合来得到插值多项式. 三次样条的基本思想则是**利用分段的思想来进行近似**, 与其让多项式次数变得非常高, 不如在相邻的两个插值点间利用低次 (3 次) 多项式进行估计, 之后再想办法让多项式在插值点处具备光滑性, 从而将一段段光滑地“连接”在一起.

2.4.2 求解三对角方程组的追赶法

此算法在三次样条插值法中将会用到, 所以先实现此算法.

求解三对角方程组 $Ax = d$, 即

$$\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} & \\ & & & a_{n,n-1} & a_{nn} & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{2.18}$$

对系数矩阵 A 进行 LU 分解即可 (第五章线性方程组的内容), 再令 $Ux = y$, 转化原方程组问题为

$$Ly = d, \quad Ux = y.$$

在计算 LU 分解的同时就可以将 y 一并求出, 且由于三对角矩阵的性质, 使得 LU 分解计算量大幅降低, 所以该算法具有线性的复杂度. 具体函数代码如下

```
function x = chase(A, d) % A 为系数矩阵,d 为方程组右侧列向量
    [n, ~] = size(A);
    % 预分配内存, 提高运算速度
    u = zeros(n, 1);
    l = zeros(n, 1);
    y = zeros(n, 1);
    u(1) = A(1, 1);
    y(1) = d(1);
    for i = 2 : n
        l(i) = A(i, i-1) / u(i-1);
        u(i) = A(i, i) - l(i) * A(i-1, i);
        y(i) = d(i) - l(i) * y(i-1);
    end
    x(n) = y(n) / u(n);
    for i = n-1 : -1 : 1
        x(i) = (y(i) - A(i, i+1) * x(i+1)) / u(i);
    end
end
```

2.4.3 三次样条插值的代码实现

三次样条插值法中有三种边界条件, 分别为: 二阶导数条件, 一阶导数条件, 周期条件. 下面的代码实现中所使用的的边界条件为第一种, 即边界点处的二阶导数作为已知条件.

三次样条插值的具体推导课本上已有详细介绍, 这里不再阐述, 这里主要介绍如何将课本上复杂的运算, 转化为矩阵运算的方式, 这样不仅能加快运算速度同时还能减少代码量. 下面定义

所用的参数名和课本基本保持一致, 由 $S(x)$ 的定义可知

$$S(x) = \begin{cases} S_2(x), & x \in [x_1, x_2] \\ S_3(x), & x \in (x_2, x_3] \\ \vdots \\ S_n(x), & x \in (x_{n-1}, x_n] \end{cases} \tag{2.19}$$

参数名	定义
$f(x)$	被插函数
$[a,b]$	插值区间
$\{x_1, x_2, \cdots, x_n\}$	插值点集合$x_i \in [a,b]$ $(i = 1, 2, \cdots, n)$
y_i	在x_i 处的函数值$f(x_i)$ $(i = 1, 2, \cdots, n)$
$S(x)$	n 个插值节点对应的三次样条插值函数
$S_i(x)$	$S(x)$ 在区间 $[x_{i-1}, x_i]$ 上的限制 $(i = 2, 3, \cdots, n)$
M_i	在x_i 处的插值函数的二阶导函数值$S''(x_i)$
h_i	第i 段插值区间长度$x_i - x_{i-1}$ $(i = 2, 3, \cdots, n)$
μ_i	中间变量$h_i/(h_i + h_{i+1})$ $(i = 2, 3, \cdots, n-1)$
λ_i	中间变量$1 - \mu_i$ $(i = 2, 3, \cdots, n-1)$
d_i	中间变量$6f[x_{i-1}, x_i, x_{i+1}]$ $(i = 2, 3, \cdots, n-1)$

为了简便运算, 定义一些向量如下 (其中 a, b, M_1, M_n 由函数传入参数直接给出)

$$a = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$$

$$b = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^T$$

$$S(x) = \begin{bmatrix} S_2(x) & S_3(x) & \cdots & S_n(x) \end{bmatrix}^T$$

$$M = \begin{bmatrix} M_1 & M_2 & \cdots & M_n \end{bmatrix}^T$$

$$h = \begin{bmatrix} h_2 & h_3 & \cdots & h_n \end{bmatrix}^T$$

$$\mu = \begin{bmatrix} \mu_2 & \mu_3 & \cdots & \mu_{n-1} \end{bmatrix}^T$$

$$\lambda = \begin{bmatrix} \lambda_2 & \lambda_3 & \cdots & \lambda_{n-1} \end{bmatrix}^T$$

$$d = \begin{bmatrix} d_2 & d_3 & \cdots & d_{n-1} \end{bmatrix}^T$$

根据课本上的推导, 计算 $S(x)$ 应分为如下三步:

- 计算定义的向量 h, μ, λ, d(预处理);

- 构造求解 M 的方程组, 利用追赶法求解该方程组.
- 解出 $S(x)$.

下面根据该过程进行逐步解析.

2.4.3.1 预处理

根据各向量的定义可得

$$
\begin{aligned}
h &= \begin{bmatrix} h_2 & h_3 & \cdots & h_n \end{bmatrix}^T = \begin{bmatrix} x_2 - x_1 & x_3 - x_2 & \cdots & x_n - x_{n-1} \end{bmatrix}^T \\
\mu_i &= \frac{h_i}{h_i + h_{i+1}} (i = 2, 3, \cdots, n) \\
\lambda &= 1 - \mu \\
d_i &= 6f\left[x_{i-1}, x_i, x_{i+1}\right] = \frac{6}{h_{i+1} + h_i} \left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \right)
\end{aligned}
$$

由于 d_i 的计算比较复杂, 所以将由 h_i, h_{i+1} 构成的向量分别定义为 h_1, h_2, 详细的代码如下

```
% 向量定义
  h = a(2:n) − a(1:n−1); % 求解向量 h
  mu = h(1:n−2) ./ (h(1:n−2) + h(2:n−1)); % 求解向量 mu
  la = 1 − mu; % 求解向量 lambda
  h1 = h(1:n−2); % 定义中间变量 h1
  h2 = h(2:n−1); % 定义中间变量 h2
  % 求解向量 d
  d = 6∗((b(3:n)−b(2:n−1))./h2−(b(2:n−1)−b(1:n−2))./h1)./(h1+h2);
```

2.4.3.2 构造并求解方程组

这里直接使用第一种边界条件所构造的方程组 $Ax = d$ 如下:

$$
\begin{bmatrix}
2 & \lambda_2 & & & & \\
\mu_3 & 2 & \lambda_3 & & & \\
& \ddots & \ddots & \ddots & & \\
& & \ddots & \ddots & \ddots & \\
& & & \mu_{n-2} & 2 & \lambda_{n-2} \\
& & & & \mu_{n-1} & 2
\end{bmatrix}
\begin{bmatrix} M_2 \\ M_3 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{bmatrix}
=
\begin{bmatrix} d_2 - \mu_2 M_1 \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} - \lambda_{n-1} M_n \end{bmatrix}
\tag{2.20}
$$

利用循环的方式即可生成系数矩阵 A, 右侧列向量 d 可以直接修改在**预处理**步骤中所得到的 d, 使用**追赶法**即可解出 $M_2, M_3, \ldots, M_{n-1}$, 最后将边界条件 M_1, M_n 代入即可求出 M. 具体代码实现如下

```
% 生成系数矩阵 A，修改 d 向量，利用追赶法求解 M
    A = zeros(n-2, n-2); % 初始化系数矩阵 A
    for i = 1 : n-2 % 循环构造系数矩阵 A
        if (i > 1)
            A(i, i-1) = mu(i); % 对角线下方
        end
        A(i, i) = 2; % 对角线
        if (i < n-2)
            A(i, i+1) = la(i); % 对角线上方
        end
    end
    d(1) = d(1) - mu(1) * M1; % 修改向量 d
    d(n-2) = d(n-2) - la(n-2) * Mn; % 修改向量 d
    M = [M1; chase(A, d)'; Mn]; % 利用追赶法求解方程组，并解出向量 M
```

2.4.3.3 求出插值函数 $S(x)$

这里直接给出插值函数表达式如下

$$\begin{aligned} S_i(x) =& \frac{(x_i - x)^3}{6h_i} M_{i-1} + \frac{(x - x_{i-1})^3}{6h_i} M_i \\ &+ \left(y_{i-1} - \frac{h_i^2}{6} M_{i-1}\right) \frac{x_i - x}{h_i} + \left(y_i - \frac{h_i^2}{6} M_i\right) \frac{x_i - x_{i-1}}{h_i} \quad (i = 2, 3, \cdots, n) \end{aligned} \tag{2.21}$$

如果直接计算 $S(x)$ 要使用复杂的双层循环，且每一个 $S_i(x)$ 都十分复杂，容易出错，于是做出一些代换，利用矩阵对应项之间的运算，直接求出 $S(x)$，高效且便于书写，定义如下 (注意：下面所定义的均为 $n-1$ 维列向量)

$$\begin{aligned} x_{\mathbf{1}} &= x_i - x \\ x_{\mathbf{2}} &= x - x_{i-1} \\ m_{\mathbf{1}} &= M_{i-1} \quad (i = 2, 3, \cdots, n) \\ m_{\mathbf{2}} &= M_i \\ y_{\mathbf{1}} &= y_{i-1} \\ y_{\mathbf{2}} &= y_i \end{aligned}$$

于是可以给出化简以后的 $S(x)$ 计算公式 (下面的 h 已在**预处理**中求得)

$$S(x) = \frac{x_1^3 m_1}{6h} + \frac{x_2^3 m_2}{6h} + \left(y_1 - \frac{h^2 m_1}{6}\right)\frac{x_1}{h} + \left(y_2 - \frac{h^2 m_2}{6}\right)\frac{x_2}{h} \tag{2.22}$$

Remark 2.8. 上面公式中 $n-1$ 维向量之间的运算全部为**对应项**之间的运算, 使用的求幂运算均是对矩阵**逐个元素分别求幂**.

具体代码实现如下

```
% 求解三次样条插值函数 S(x)
    syms x % 定义参数 x
    % 根据上述定义, 计算出以下 6 个中间变量
    x1 = a(2:n) - x;
    x2 = x - a(1:n-1);
    m1 = M(1:n-1);
    m2 = M(2:n);
    y1 = b(1:n-1);
    y2 = b(2:n);
    % 求解三次样条插值函数
    S = (x1.^3).*m1./(6*h)+(x2.^3).*m2./(6*h)+(y1-h.^2.*m1/6).*x1./h+(y2-h.^2.*m2/6)
        .*x2./h;
    S = expand(S); % 展开插值多项式
    S = vpa(S, 5); % 将分式系数转化为小数系数
end
```

2.4.3.4 三次样条插值的函数代码

有了以上的铺垫, 我们便可以有如下的三次样条插值的函数代码

```
% 三次样条插值法
% a 为传入的插值点构成的向量,b 为插值点对应函数值构成的向量,M1,Mn 为边界条件
function S = mySpline(a, b, M1, Mn)
    [n, ~] = size(a);
    % 向量定义
    h = a(2:n) - a(1:n-1); % 求解向量 h
    mu = h(1:n-2) ./ (h(1:n-2) + h(2:n-1)); % 求解向量 mu
    la = 1 - mu; % 求解向量 lambda
    h1 = h(1:n-2); % 定义中间变量 h1
    h2 = h(2:n-1); % 定义中间变量 h2
    % 求解向量 d
    d = 6*((b(3:n)-b(2:n-1))./h2-(b(2:n-1)-b(1:n-2))./h1)./(h1+h2);

    % 生成系数矩阵 A, 修改 d 向量, 利用追赶法求解 M
```

```
    A = zeros(n-2, n-2); % 初始化系数矩阵 A
    for i = 1 : n-2 % 循环构造系数矩阵 A
        if (i > 1)
            A(i, i-1) = mu(i); % 对角线下方
        end
        A(i, i) = 2; % 对角线
        if (i < n-2)
            A(i, i+1) = la(i); % 对角线上方
        end
    end
    d(1) = d(1) - mu(1) * M1; % 修改向量 d
    d(n-2) = d(n-2) - la(n-2) * Mn; % 修改向量 d
    M = [M1; chase(A, d)'; Mn]; % 利用追赶法求解方程组, 并解出向量 M

    % 求解三次样条插值函数 S(x)
    syms x % 定义参数 x
    % 根据上述定义, 计算出以下 6 个中间变量
    x1 = a(2:n) - x;
    x2 = x - a(1:n-1);
    m1 = M(1:n-1);
    m2 = M(2:n);
    y1 = b(1:n-1);
    y2 = b(2:n);
    % 求解三次样条插值函数
    S = (x1.^3).*m1./(6*h)+(x2.^3).*m2./(6*h)+(y1-h.^2.*m1/6).*x1./h+(y2-h.^2.*m2/6)
        .*x2./h;
    S = expand(S); % 展开插值多项式
    S = vpa(S, 5); % 将分式系数转化为小数系数
end
```

2.5 代码实验: 观察高次插值多项式的龙格现象

2.5.1 问题描述

给定函数

$$f(x) = \frac{1}{1+25x^2} \quad (-1 \le x \le 1),$$

取等距节点, 构造牛顿插值多项式 $N_5(x)$ 和 $N_{10}(x)$ 及三次样条插值函数 $S_{10}(x)$. 分别将三种插值多项式与 $f(x)$ 的曲线画在同一个坐标系上进行比较.

2.5.2 代码实现

利用前面给出的 Newton 插值的代码与三次样条插值的代码进行实验, 代码如下

```
function main
```

```
    format short
    a = -1;
    b = 1;
    x = (a:0.01:b)';
    plot(x, f(x), 'r', 'linewidth', 1); % f(x) 图像
    hold on
    % Newton 插值法求 N5,N10
    xx = (a:2/5:b)';
    y = Newton(xx, f(xx));
    class(y)
    disp("N5 = ")
    disp(y) % 输出 N5 插值结果
    plot(x, subs(y, x), 'color', '#43A047', 'linewidth', 1); % N5 图像
    hold on
    xx = (a:2/10:b)';
    y = Newton(xx, f(xx));
    disp("N10 = ")
    disp(y) % 输出 N10 插值结果
    plot(x, subs(y, x), 'linewidth', 1); % N10 图像
    hold on
    % 三次样条插值
    xx = (a:2/10:b)';
    y = mySpline(xx, f(xx), fff(a), fff(b));
    disp("S10 = ")
    disp(y)
    for i = 1 : 10
        l = xx(i);
        r = l + 2/10;
        x = l:0.01:r;
        plot(x, subs(y(i), x), 'b', 'linewidth', 1) % S10 图像
        hold on
    end
    legend(["f(x)" "N5" "N10" "S10"]);
    set(gca, 'FontName', 'Times New Roman', 'FontSize', 16);
end

function y = f(x)
    y = 1 ./ (1 + 25 * x .* x);
end
function y = fff(x)
    y = 50 * (75*x.*x-1) ./ (1+25*x.*x)^3;
end
```

2.5.3 结果分析

对 $f(x) = \frac{1}{1+25x^2}$ 在 $[-1,1]$ 上的均匀分布的插值点进行插值, 根据程序的输出, 得到如下插值多项式

$$N_5(x) = 1.2019 * x^4 - 1.7308 * x^2 + 0.56731$$

$$N_{10}(x) = -220.94 * x^{10} + 494.91 * x^8 - 381.43 * x^6 + 123.36 * x^4 - 16.855 * x^2 + 1.0$$

$$S_{10}(x) = \begin{cases} 0.11927 * x^3 + 0.46308 * x^2 + 0.64433 * x + 0.33898, & x \in [-1, -0.8]; \\ 0.95287 * x^3 + 2.4637 * x^2 + 2.2448 * x + 0.76578, & x \in (-0.8, -0.6]; \\ 0.82039 * x^3 + 2.2252 * x^2 + 2.1018 * x + 0.73717, & x \in (-0.6, -0.4]; \\ 13.413 * x^3 + 17.336 * x^2 + 8.146 * x + 1.5431, & x \in (-0.4, -0.2]; \\ -54.471 * x^3 - 23.394 * x^2 + 1.0, & x \in (-0.2, 0]; \\ 54.471 * x^3 - 23.394 * x^2 + 1.0, & x \in (0, 0.2]; \\ -13.413 * x^3 + 17.336 * x^2 - 8.146 * x + 1.5431, & x \in (0.2, 0.4]; \\ -0.82039 * x^3 + 2.2252 * x^2 - 2.1018 * x + 0.73717, & x \in (0.4, 0.6]; \\ -0.95287 * x^3 + 2.4637 * x^2 - 2.2448 * x + 0.76578, & x \in (0.6, 0.8]; \\ -0.11927 * x^3 + 0.46308 * x^2 - 0.64433 * x + 0.33898, & x \in (0.8, 1] \end{cases}$$

利用 MATLAB 将三个插值多项式图像绘制在同一个 plot 上, 效果如下图

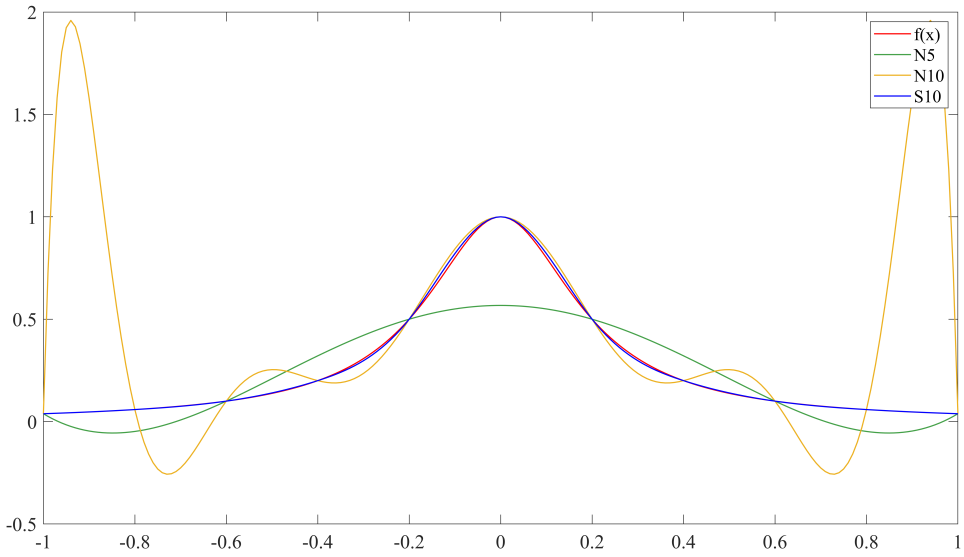


图 2.1: 龙格现象

通过观察图像, 我们可以发现 S_{10} 几乎和 $f(x)$ 完全重合, 插值效果最好, 而 N_5 由于插值点

个数太少截断误差太大, N_{10} 在 $[-0.2, 0.2]$ 之间有很好的差值效果, 但是由于插值多项式次数过高, 在 $[-1, -0.2)$, $(0.2, 1]$ 发生了明显的**龙格现象**. 所以, 使用多段低次插值多项式 (三次样条插值法), 可以有效避免高次插值多项式所产生的的龙格现象.

Remark 2.9. 由于三次样条插值法涉及的变量较多, 首先要分清楚变量之间的推导关系, 进而设定每个向量或矩阵的内容和大小, 再写出化简后的推导式, 有了简化的关系式代码就很容易写出了.

Chapter3. 函数逼近

函数逼近的基本思想就是使用简单易算的函数去近似表达式复杂的函数. 最简单的函数莫过于多项式函数, 因此, 我们首先考虑用多项式函数去做函数逼近. 对于闭区间上任意连续函数 $f(x)$, 由泛函分析中的 Weierstrass 定理, 即 Theorem 2.4可知, 存在一个多项式序列一致收敛到 $f(x)$. 这就是用多项式函数逼近一般连续函数的理论基础.

3.1 基本概念与预备知识

3.1.1 函数逼近与曲线拟合

对于一个给定的复杂函数 $f(x)$, 在某个表达式较简单的函数类 $\boldsymbol{\Phi}$ 中寻找一个函数 $P^*(x)$, 使其在某种度量下距离 $f(x)$ 最近, 即**最佳逼近**. 这就是**函数逼近**.

Remark 3.1.

- 函数 $f(x)$ 通常较复杂, 但一般是连续的. 我们主要考虑 $f(x) \in C[a,b]$.
- 函数类 $\boldsymbol{\Phi}$ 通常由简单函数构成, 比如多项式, 分段多项式, 有理函数, 或者三角函数等
- 在不同的度量下, $f(x)$ 的最佳逼近可能不一样;
- 函数逼近通常采用基函数法.

如果只知道函数在部分节点上的值, 且这些数值带有一定的误差, 需要在函数类 $\boldsymbol{\Phi}$ 中寻找一个函数 $P(x)$, 使其在某种度量下是这些数据的最佳逼近, 这就是**曲线拟合**, 也称为**数据拟合**, 可以看作是离散情况下的函数逼近.

3.1.2 最佳逼近多项式

Definition 3.1 (最佳逼近函数)**.** 设 $\boldsymbol{\Phi}$ 为某个函数空间, 给定函数 $f(x) \in C[a,b]$, 若存在 $g^*(x) \in \boldsymbol{\Phi}$, 使得

$$\|f(x) - g^*(x)\| = \min_{g(x)\in\boldsymbol{\Phi}} \|f(x) - g(x)\|,$$

则称 $g^*(x)$ 为 $f(x)$ 在 $\boldsymbol{\Phi}$ 中的**最佳逼近函数**.

Remark 3.2. $g^*(x)$ 与函数空间 $\boldsymbol{\Phi}$, 范数 $\|\cdot\|$ 和区间 $[a,b]$ 有关.

Definition 3.2. 设 H_n 为所有次数不超过 n 的多项式组成的函数空间, 给定函数 $f(x) \in C[a,b]$, 若存在 $P^*(x) \in H_n$, 使得

$$\|f(x) - P^*(x)\| = \min_{P(x)\in H_n} \|f(x) - P(x),\|$$

则称 $P^*(x)$ 为 $f(x)$ 在 $[a,b]$ 上的n **次最佳逼近多项式**.

若使用的范数为 $\|\cdot\|_\infty$, 则称 $P^*(x)$ 为n **次最优一致逼近多项式**;

若使用的范数为 $\|\cdot\|_2$, 则称 $P^*(x)$ 为n **次最佳平方逼近多项式**.

Definition 3.3. 如果只知道 $f(x)$ 在某些节点上的函数值 $f(x_i) = y_i$ $(i = 0, 1, 2, \ldots, m)$, 在某个函数空间 $\boldsymbol{\Phi}$ 中寻找 $P^*(x)$ 使得

$$\sum_{i=1}^{m} |y_i - P^*(x_i)|^2 = \min_{g(x)\in\boldsymbol{\Phi}} \sum_{i=1}^{m} |y_i - P(x_i)|^2, \tag{3.1}$$

则称 $P^*(x)$ 为 $f(x)$ 的**最小二乘拟合**. 若 $\boldsymbol{\Phi}$ 取 H_n, 则称 $P^*(x)$ 为 $f(x)$ 的n **次最小二乘拟合**. 这里一般有 $m > n$.

我们在求解最佳逼近多项式需要用到一类重要的多项式: 正交多项式.

3.2 正交多项式

3.2.1 正交函数族与正交多项式

Definition 3.4 (正交函数)**.** 设 $f(x), g(x) \in C[a,b]$, $\rho(x)$ 为 $[a,b]$ 上的权函数, 若

$$(f,g) = \int_a^b \rho(x) d(x) g(x) dx = 0, \tag{3.2}$$

则称 $f(x)$ 与 $g(x)$ 在 $[a,b]$ 上**带权** $\rho(x)$ **正交**.

Remark 3.3. 正交与所使用的内积和权函数有关.

Definition 3.5 (正交函数族)**.** 设 $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x), \ldots \in C[a,b]$, $\rho(x)$ 为 $[a,b]$ 上的权函数, 若

$$\left(\varphi_i, \varphi_j\right) = \int_a^b \rho(x)\varphi_i(x)\varphi_j(x)dx = \begin{cases} 0, & i \neq j \\ A_i > 0, & i = j \end{cases} \quad i,j = 0,1,2,\ldots, \tag{3.3}$$

则称 $\{\varphi_n(x)\}_{n=0}^{\infty}$ 是 $[a,b]$ 上带权 $\rho(x)$ 的**正交函数族**; 若 $A_i \equiv 1$, 则称为**标准正交函数族**.

Definition 3.6 (正交多项式)**.** 设 $P_n(x)$ 是首项系数不为零的 n 次多项式, $n = 0, 1, 2, \ldots$, $\rho(x)$ 为 $[a,b]$ 上的权函数, 若

$$\left(P_i, P_j\right) = \int_a^b \rho(x) P_i(x) P_j(x) dx = \begin{cases} 0, & i \neq j \\ A_i > 0, & i = j \end{cases} \quad i,j = 0,1,2,\ldots, \tag{3.4}$$

则称 $\{P_n(x)\}_{n=0}^{\infty}$ 为 $[a,b]$ 上带权 $\rho(x)$ 正交, 并称 $P_n(x)$ 为n **次正交多项式**.

设 $\{P_n(x)\}_{n=0}^{\infty}$ 为 $[a,b]$ 上带权 $\rho(x)$ 正交多项式, 则显然 $P_0(x), P_1(x), P_2(x), \ldots, P_n(x)$ 线性无关, 所以它们构成 H_n 的一组**正交基**.

由于 $P_n(x)$ 与 $P_0(x), P_1(x), P_2(x), \ldots, P_{n-1}(x)$ 都正交, 故 $P_n(x)$ 与 H_{n-1} 中的任意多项式都正交, 即

$$(P_n(x), P(x)) = \int_a^b \rho(x) P_n(x) P(x) \mathrm{d}x = 0, \quad \forall P(x) \in H_{n-1}. \tag{3.5}$$

下面我们给出一个正交多项式的三项递推公式.

Theorem 3.1. 设 $\{\varphi_n(x)\}_0^\infty$ 为 $[a,b]$ 上带权 $\rho(x)$ 的正交多项式, 对 $n \geq 0$ 成立递推关系

$$\varphi_{n+1}(x) = (x-\alpha_n)\varphi_n(x) - \beta_n\varphi_{n-1}(x), \quad n = 0,1,\dots, \tag{3.6}$$

其中 $\varphi_0(x) = 1$, $\varphi_{-1}(x) = 0$,

$$\alpha_n = \frac{(x\varphi_n, \varphi_n)}{(\varphi_n, \varphi_n)}, n = 0,1,2,\dots, \quad \beta_n = \frac{(\varphi_n, \varphi_n)}{(\varphi_{n-1}, \varphi_{n-1})}, n = 1,2,\dots.$$

Remark 3.4. 所有首项系数为 1 的正交多项式族都满足这个公式, 该公式也给出了正交多项式的一个递推计算方法.

Theorem 3.2. 设 $\{P_n(x)\}_{n=0}^\infty$ $n = 0$ 是 $[a,b]$ 上带权 $\rho(x)$ 的正交多项式, 则当 $n \geq 1$ 时, $p_n(x)$ 在 (a,b) 内有 n 个不同零点.

证明. 假设 $P_n(x)$ 在 (a,b) 内没有零点, 则 $P_n(x)$ 在 (a,b) 内不变号, 故

$$\left|\int_a^b \rho(x)P_n(x)dx\right| > 0.$$

另一方面, 由式(3.2)可知

$$0 = (P_n, 1) = \int_a^b \rho(x)P_n(x)dx.$$

矛盾, 因此 $P_n(x)$ 在 (a,b) 内至少有一个零点. 假设 $P_n(x)$ 在 (a,b) 内没有奇数重零点, 则 $P_n(x)$ 在 (a,b) 内不变号, 同样可以导出矛盾. 因此 $P_n(x)$ 在 (a,b) 内至少有一个奇数重零点. 设 $P_n(x)$ 在 (a,b) 的所有奇数重零点为 $x_1, x_2, \dots, x_l$ $(1 \leq l \leq n)$. 构造多项式 $P_l(x) = (x-x_1)(x-x_2)\dots(x-x_l)$. 则 $P_l(x)P_n(x)$ 在 (a,b) 内只有偶数重零点, 因此在 (a,b) 内不变号. 于是

$$|(P_n, P_l)| = \left|\int_a^b \rho(x)P_n(x)P_l(x)dx\right| > 0.$$

如果 $l < n$, 则由式(3.2)可知, $(P_n, P_l) = 0$, 矛盾. 因此 $l = n$. □

3.2.2 Legendre 多项式

在区间 $[-1,1]$ 上权函数取常数 1, 由 $1, x, x^2, \dots$ 正交化得到的即是 **Legendre 多项式**, 记为

$$P_0(x), P_1(x), P_2(x), \dots.$$

Theorem 3.3. *Legendre* 多项式的递推公式为

$$\begin{cases} P_{n+1}(x) = \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x) \\ P_0(x) = 1, \quad P_1(x) = x \end{cases} \tag{3.7}$$

Theorem 3.4. *Legendre* 多项式的一般形式为

$$P_0(x) = 1, \quad P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left(x^2-1\right)^n, \quad x \in [-1,1], \quad n = 1,2,\ldots \tag{3.8}$$

- 显然, $P_n(x)$ 的首项系数为 $\frac{(2n)!}{2^n(n!)^2}$.
- 若令 $\tilde{P}_n(x) = \frac{2^n(n!)^2}{(2n)!}P_n(x)$, 则称 $\tilde{P}_n(x)$ 为**首项系数为 1 的 Legendre 多项式**.

Theorem 3.5. $P_n(x)$ 在 $(-1,1)$ 内有 n 个不同的零点.

证明. 直接由 Theorem 3.2 即得. □

3.2.3 Chebyshev 多项式

在区间 $[-1,1]$ 上权函数取 $\rho(x) = \frac{1}{\sqrt{1-x^2}}$, 由 $1, x, x^2, \ldots$ 正交化得到的即是 **Chebyshev 多项式**, 记为

$$T_0(x), T_1(x), T_2(x), \ldots.$$

Theorem 3.6. *Chebyshev* 多项式的递推公式为

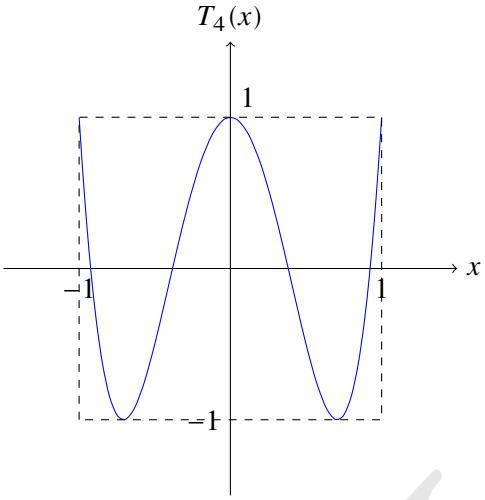
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n = 1,2,\ldots. \tag{3.9}$$

Remark 3.5. 由递推公式直接可得 $T_{2n}(x)$ 只含偶次幂, $T_{2n+1}(x)$ 只含奇次幂, 故

$$T_n(-x) = (-1)^n T_n(x). \tag{3.10}$$

Theorem 3.7. *Chebyshev* 多项式的一般形式为

$$T_n(x) = \cos(n\arccos x), \quad x \in [-1,1], \quad n = 1,2,\ldots. \tag{3.11}$$



- 显然 $|T_n(x)| \le 1$.
- 令 $x = \cos\theta$, $\theta \in [0, \pi]$, 则 $\theta = \arccos x$, 所以

$$\begin{aligned} T_n(x) = \cos(n\theta) &= \cos^n\theta - C_n^2\cos^{n-2}\theta\sin^2\theta + C_n^4\cos^{n-4}\theta\sin^4\theta + \cdots \\ &= x^n - C_n^2 x^{n-2}\left(1-x^2\right) + C_n^4 x^{n-4}\left(1-x^2\right)^2 + \cdots. \end{aligned}$$

  故 $T_n(x)$ 是 n 次多项式.
- 显然 $T_n(x)$ 的首项系数为 2^{n-1}.
- 若令 $\tilde{T}_n(x) = \frac{1}{2^{n-1}}T_n(x)$, 则称 $\tilde{T}_n(x)$ 为**首项系数为 1 的 Chebyshev 多项式.**

Theorem 3.8. *$T_n(x)$ 在 $(-1,1)$ 内有 n 个不同的零点:*

$$x_k = \cos\frac{2k-1}{2n}\pi, \quad k = 1, 2, \ldots, n. \tag{3.12}$$

$T_n(x)$ 在 $[-1,1]$ 内有 $n+1$ 个极值点 (含两个端点):

$$\tilde{x}_k = \cos\frac{k\pi}{n}, \quad k = 0, 1, 2, \ldots, n. \tag{3.13}$$

3.2.4 Chebyshev 多项式零点插值

我们首先介绍 Chebyshev 多项式的一个重要性质.

Theorem 3.9. *设 $\tilde{P}_n(x)$ 是首项系数为 1 的 Chebyshev 多项式, 则*

$$\max_{-1\le x\le 1}\left|\tilde{T}_n(x)\right| \le \max_{-1\le x\le 1}|p(x)|, \quad \forall p(x) \in \tilde{\mathbb{H}}_n, \tag{3.14}$$

其中 $\tilde{\mathbb{H}}_n$ 表示次数不超过 n 的所有首项系数为 1 的多项式组成的集合. 且

$$\max_{-1\le x\le 1}\left|\tilde{T}_n(x)\right| = \frac{1}{2^{n-1}}. \tag{3.15}$$

Remark 3.6. 这个性质表明, 在次数不超过 n 的所有首项系数为 1 的多项式中, $\tilde{P}_n(x)$ 在 $[-1,1]$ 上与零的偏差是最小的 (在无穷范数意义下).

利用 Theorem 3.9, 我们可以采用 Chebyshev 多项式的零点作为节点进行多项式插值, 以使得插值的总体误差达到最小化.

Theorem 3.10. 设 $f(x)\in C^{n+1}[-1,1]$, 插值节点为 $T_{n+1}(x)$ 的零点, 即

$$x_k = \cos\frac{2k+1}{2(n+1)}\pi, \quad k=0,1,2,\dots,n. \tag{3.16}$$

令 $L_n(x)$ 是 $f(x)$ 在 $[0,1]$ 上的 n 次插值多项式, 则插值误差满足

$$\left\|f(x)-L_n(x)\right\|_\infty \le \frac{1}{2^n(n+1)!}\left\|f^{(n+1)}(x)\right\|_\infty. \tag{3.17}$$

Remark 3.7. 对于一般区间 $[a,b]$, 可通过伸缩平移变换得到任意区间的 *Chebyshev* 点:

$$x_k = \frac{b-a}{2}\cos\left(\frac{2k+1}{2n+2}\pi\right)+\frac{a+b}{2}.$$

此时总体的插值误差为

$$\begin{aligned}\max_{a\le x\le b}\left|f(x)-L_n(x)\right| &\le \frac{1}{2^n(n+1)!}\max_{-1\le t\le 1}\left|\frac{d^{n+1}f}{dt^{n+1}}\right| \\ &= \frac{1}{2^n(n+1)!}\frac{(b-a)^{n+1}}{2^{n+1}}\max_{-1\le t\le 1}\left|f^{(n+1)}(x(t))\right| \\ &= \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!}\max_{a\le x\le b}\left|f^{(n+1)}(x)\right|.\end{aligned}$$

为了尽可能地减小插值误差, 在可以自由选取插值节点时, 我们尽量使用 Chebyshev 多项式零点.

3.2.5 Chebyshev 插值的代码实现

选择目标函数 $f(x)=e^{-x}$ 为例, 使用 Chebyshev 多项式零点进行 Newton 插值的代码示例:

```
function main
    format short
```

```
    n = 2; % 插值点个数
    rg = [-1 1]; % 插值区间
    x = cos((1:2:(2*n-1))*pi/(2*n))'; % [-1, 1]
  % 做线性变化 [-1,1]->[a,b]
    x = (rg(1)+rg(2)) / 2 + (rg(2)-rg(1)) * x / 2; % [a, b]
    y = Newton(x, f(x));
    syms x
  % 变化回去
    vpa(subs(y, x, (2*x-rg(1)-rg(2)) / (rg(2)-rg(1))), 5)
end
function y = f(x) % 目标函数
    y = exp(-x);
end
```

输出的结果为

$$y = 1.2606 - 1.0854 * x$$

紧接着 Section 2.5的实验, 我们将补充 Chebyshev 插值法的代码, 并对 **Runge 现象**的处理效果进行比较. 只需要将 Newton 插值法中插值点从均匀点改成 Chebyshev 多项式的零点即可, 代码不难实现:

```
% Chebyshev 插值法与均匀插值点作比较
xx = cos((1:2:21)*pi/22)';
y = Newton(xx, f(xx));
disp("T10 = ")
disp(y) % 输出 T10 的结果
plot(x, subs(y, x), 'linewidth', 1); % T10 图像
```

完整代码如下:

```
function main
    format short
    a = -1;
    b = 1;
    x = (a:0.01:b)';
    plot(x, f(x), 'r', 'linewidth', 1); % f(x) 图像
    hold on
    % Newton 插值法求 N5,N10
    xx = (a:2/5:b)';
    y = Newton(xx, f(xx));
    class(y)
    disp("N5 = ")
    disp(y) % 输出 N5 插值结果
    plot(x, subs(y, x), 'color', '#43A047', 'linewidth', 1); % N5 图像
    hold on
    xx = (a:2/10:b)';
    y = Newton(xx, f(xx));
```

```
    disp("N10 = ")
    disp(y) % 输出 N10 插值结果
    plot(x, subs(y, x), 'linewidth', 1); % N10 图像
    hold on
    % 三次样条插值
    xx = (a:2/10:b)';
    y = mySpline(xx, f(xx), fff(a), fff(b));
    disp("S10 = ")
    disp(y)
    for i = 1 : 10
        l = xx(i);
        r = l + 2/10;
        x = l:0.01:r;
        plot(x, subs(y(i), x), 'b', 'linewidth', 1) % S10 图像
        hold on
    end
    legend(["f(x)" "N5" "N10" "S10"]);
    set(gca, 'FontName', 'Times New Roman', 'FontSize', 16);
end

function y = f(x)
    y = 1 ./ (1 + 25 * x .* x);
end
function y = fff(x)
    y = 50 * (75*x.*x-1) ./ (1+25*x.*x)^3;
end
```

将 11 个均匀点的插值函数 N10，和 11, 21 次 Chebyshev 多项式零点作为插值点所确定的 10, 20 次插值函数 T10, T20 作比较，可得如下结果

$$T_{10} = -46.633 * x^{10} + 130.11 * x^8 - 133.44 * x^6 + 61.443 * x^4 - 12.477 * x^2 + 1.0$$

$$\begin{aligned} T_{20} =& 6466.6 * x^{20} - 34208.0 * x^{18} + 77754.0 * x^{16} - 99300.0 * x^{14} + 78236.0 * x^{12} \\ & - 39333.0 * x^{10} + 12636.0 * x^8 - 2537.3 * x^6 + 306.63 * x^4 - 21.762 * x^2 + 1.0 \end{aligned}$$

运行后输出的图像:

不难看出, Chebyshev 插值法有效的消除了高次插值多项式的 **Runge 现象**, 是非常有效的改进算法.

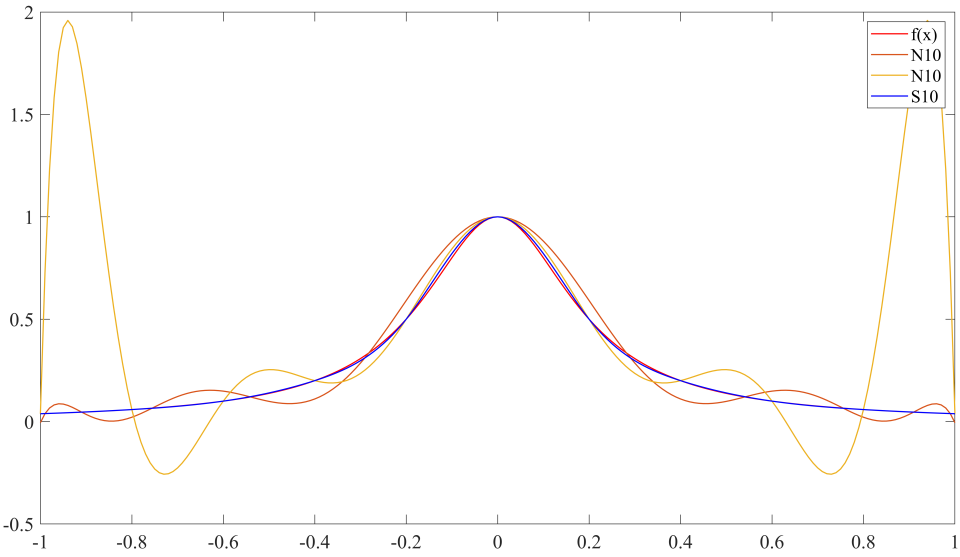


图 3.1: 龙格现象

3.2.6 其他正交多项式

在实际的应用过程中, 我们更多采用 Chebyshev 极值点插值, 也称为**第二类 Chebyshev 点**, 或者 **Chebyshev-Labatto 点**:

$$x_j = \cos\frac{j\pi}{N}, \quad 0 \le j \le N.$$

Type	Range	Weight
Legendre	$[-1,1]$	1
Chebyshev	$[-1,1]$	$\frac{1}{\sqrt{1-x^2}}$
Laguerre	$[0,\infty)$	e^{-x}
Hermite	$(-\infty,\infty)$	e^{-x^2}

3.3 最佳平方逼近

3.3.1 求最佳平方逼近

Definition 3.7 (最佳平方逼近函数)**.** *对 $f(x) \in C[a,b]$, 以及 $C[a,b]$ 的一个子集 (子空间、线性空间)*

$$\boldsymbol{\Phi} = \operatorname{span}\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\},$$

若存在 $S^*(x) \in \boldsymbol{\Phi}$, 使得

$$\|f(x) - S^*(x)\| = \min_{s\in\boldsymbol{\Phi}} \|f(x) - S(x)\|,$$

则称 $S^*(x)$ 是 $f(x)$ 在 $\boldsymbol{\Phi}$ 中的**最佳平方逼近函数**.

记 $S(x)=\sum_j a_j\varphi_j(x)$, 最佳平方逼近问题等价于多元函数

$$I\left(a_0,a_1,\cdots,a_n\right)=\int_a^b \rho(x)\left[f(x)-\sum_j a_j\varphi_j(x)\right]^2 dx$$

求极小值. 由 Fermat 引理, 我们有

$$\frac{\partial I}{\partial a_k}=2\int_a^b \rho(x)\left[f(x)-\sum_j a_j\varphi_j(x)\right]\varphi_k(x)dx=0, \tag{3.18}$$

即

$$\sum_{j=0}^{n}\left(\varphi_k,\varphi_j\right)a_j=\left(f,\varphi_k\right),\quad k=0,1,\cdots,n. \tag{3.19}$$

若 φ_j **线性无关**, 则上述线性方程组**解存在唯一**.

Theorem 3.11. 最佳平方逼近误差为

$$\|f-S^*\|^2=(f-S^*,f-S^*)=(f,f-S^*)=\|f\|^2-(f,S^*). \tag{3.20}$$

Remark 3.8. 第二个等号参考式(3.18).

取定一组多项式基函数 $\{\varphi_j=x^j,0\le j\le n\}$, 权函数 $\rho(x)=1$ 可通过上述过程解得最佳平方逼近多项式. 只需计算相应的系数矩阵和右端项即可.

3.4 最小二乘拟合 (离散的最佳平方逼近)

3.4.1 曲线拟合

曲线拟合 (curve fitting) 是指选择适当的曲线来拟合通过观测或实验所获得的数据. 科学和工程中遇到的很多问题, 往往只能通过诸如采样、实验等方法获得若干离散的数据. 根据这些数据, 如果能够找到一个连续的函数（即曲线）或者更加密集的离散方程, 使得实验数据与方程的曲线能够在最大程度上近似吻合, 就可以根据曲线方程对数据进行理论分析和数值预测, 对某些不具备测量条件的位置的结果进行估算.

3.4.2 最小二乘拟合的方法

假设我们已知离散点处的函数值 $f(x_j)=y_j,\quad j=1,2,\ldots,m$, 即

x	x_0	x_1	x_2	$\cdots$	x_m
y	y_0	y_1	y_2	$\cdots$	y_m

我们希望能在集合 $\varphi = \operatorname{span}\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\}$ 中找一个函数 $S^*(x) = \sum_{l=0}^{n} a_l \varphi_l(x)$, s.t.

$$\sum_{j=0}^{m} \left|S^*\left(x_j\right) - y_j\right|^2 = \min_{S \in \varphi} \sum_{j=0}^{m} \left|S\left(x_j\right) - y_j\right|^2. \tag{3.21}$$

这里的 n 通常远远小于 m.

若引入权函数 $\omega(x)$, 更一般的问题提法为寻求极小

$$\min_{s \in \varphi} \sum_{j=0}^{m} w\left(x_j\right) \left|S\left(x_j\right) - y_j\right|^2, \tag{3.22}$$

相应带权内积为 $\langle \varphi_l, \varphi_k \rangle = \sum_{j=0}^{m} w\left(x_j\right) \varphi_l\left(x_j\right) \varphi_k\left(x_j\right)$.

Remark 3.9. 离散数据拟合的最小二乘问题实质上可以看作是最佳平方逼近问题的离散形式, 因此, 可以将求连续函数的最佳平方逼近函数的方法直接用于求解该问题.

我们可以将上述问题归结为多元函数求极值问题

$$I\left(a_0, \ldots, a_n\right) = \sum_{j=0}^{m} w_j \left|\sum_{l=0}^{n} a_l \varphi_l\left(x_j\right) - y_j\right|^2,$$

求偏导可得

$$\frac{\partial I}{\partial a_k} = 2 \sum_{j=0}^{m} w_j \left(\sum_{l=0}^{n} a_l \varphi_l\left(x_j\right) - y_j\right) \varphi_k\left(x_j\right) = 0,$$

从而

$$\sum_{l=0}^{n} a_l \left\langle \varphi_l, \varphi_k \right\rangle = \left\langle y, \varphi_k \right\rangle, \quad k = 0, 1, \ldots, n. \tag{3.23}$$

形式上, 离散的最佳平方逼近与连续函数情形所得方程组形式上一样。解的存在唯一性不但要求所选基函数线性无关, 还需要 **Haar 条件**.

Theorem 3.12. 设 $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x) \in C[a, b]$ 线性无关. 如果 $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x)$ 的任意 (非零) 线性组合在点集 $\{x_0, x_1, \ldots, x_m\}$ 上至多只有 n 个不同的零点. 该定理中的条件就被称为 ***Haar* 条件**.

3.4.3 最小二乘法的线性代数处理

- 根据逼近函数空间基函数写出函数表达式 $S(x) = \sum_{l-0}^{n} a_l \varphi_l(x)$;
- 根据 $m+1$ 组离散数据 $S(x_j) = y_j$ 写出对应线性方程组 $Aa = b$

$$a_0 \varphi_0\left(x_j\right) + a_1 \varphi_1\left(x_j\right) + \ldots + a_n \varphi_n\left(x_j\right) = y_j, \quad 0 \le j \le m;$$

- 根据权函数改写等价方程组

$$w_j\left[a_0\varphi_0\left(x_j\right)+a_1\varphi_1\left(x_j\right)+\ldots+a_n\varphi_n\left(x_j\right)\right]=w_j*y_j,\quad 0\leq j\leq m;$$

- 上述系数矩阵 A 的阶是 $(m+1)\times(n+1)$, 记权函数 $W=\mathrm{Diag}\left(\omega_0,\omega_1,\cdots,\omega_m\right)\quad(m>n)$, 此时最小二乘拟合等价于

$$A^{\mathrm{T}}*W*Aa=A^{\mathrm{T}}*W*b. \tag{3.24}$$

Remark 3.10. 对于曲线拟合问题, 如何选择数学模型很重要 (即函数空间, 通常需要根据物理意义或所给数据的分布情况来确定.

Chapter4. 数值积分

Remark 4.1. 由于前面的内容写得过于繁琐, 因此从本章开始, 将主要关注重点内容与个人学习历程的想法, 不会完全按照自学的顺序进行总结, 并且部分书中已有的定义等便不再赘述.

4.1 数值积分的关注点

- 问题背景: 求未知函数的积分.
- 基本思想: 用函数值的线性组合来近似定积分, 有时也会用到导数值.
- 数值积分主要研究的问题:
  - 求积公式的构造;
  - 精确程度的衡量;
  - 余项估计/误差估计.

4.2 求积公式精确程度的衡量标准

4.2.1 代数精度

根据 Weierstrass 逼近定理 Theorem 2.4可知, 任意一个连续函数都可以通过多项式来一致逼近, 因此, 如果一个求积公式能对次数较高的多项式精确成立, 那么我们就认为该求积公式具有较高的精度. 基于这样的想法, 我们给出代数精度的概念.

Definition 4.1 (代数精度)**.** 如果一个求积公式对所有次数不超过 m 的多项式精度成立, 但对 $m+1$ 次多项式不精确成立, 则称该求积公式具有 m 次**代数精度**.

这给出了计算一个求积公式的代数精度的方法. 由定义可知, 一个求积公式具有 m 次代数精度当且仅当求积公式

1. 对 $f(x) = 1, x, x^2, \dots, x^m$ 精确成立;
2. 对 $f(x) = x^{m+1}$ 不精确成立.

4.2.2 收敛性

Definition 4.2. 设求积公式的余项为 $R[f]$, 若

$$\lim_{h\to 0} R[f] = 0, \quad \text{其中} h = \max_{0\le i\le n-1}\{x_{i-1} - x_i\},$$

则称求积公式是**收敛的**.

Remark 4.2. 由定义可以, 求积公式的收敛性与定积分的存在性是类似的, 即当分割足够细时极限存在, 并且两者的值相等.

4.2.3 稳定性

在利用机械求积公式计算定积分时, 需要计算函数值. 由于存在一定的舍入误差, 因此最后的结果也会带有一定的误差. 求积公式的稳定性就是用来表示这些舍入误差对计算结果的影响.

Definition 4.3. 考虑机械求积公式(4.2). 设 f_k 是计算 $\tilde{f}_k$ 时得到的近似值. 若 $\forall \varepsilon > 0$, $\exists \delta > 0$, 使得当 $|f(x_k) - \tilde{f}_k| < \delta$ 对 $k = 0, 1, 2, \ldots, n$ 都成立时, 有

$$\left| \sum_{k=0}^{n} A_k f(x_k) - \sum_{k=0}^{n} A_k \tilde{f}_k \right| < \varepsilon, \tag{4.1}$$

则称求积公式(4.2)是**稳定的**.

下面给出一个判别机械求积公式稳定性的充分条件.

Theorem 4.1. 若机械求积公式(4.2)中的求积系数 A_i 非负, 则求积公式是稳定的.

4.3 机械求积

4.3.1 机械求积方法

进行机械求积的思想: 转化为函数值的加权平均, 即

$$\int_a^b f(x) = \sum_{i=1}^{n} A_i f(x_i) dx \triangleq I_n(f). \tag{4.2}$$

这样便避开了需要寻求原函数的困难, 很适合在计算机上使用.

- 梯形公式: $T(f) = \int_a^b f(x)dx = \frac{b-a}{2}(f(a) + f(b))$;
- (中) 矩形公式: $\int_a^b f(x)dx = (b-a)f(\frac{a+b}{2})$;
- Simpson 公式: $S(f) = \int_a^b f(x)dx = \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$.

Remark 4.3. 梯形公式与矩形公式在当 $f(x) = 1$, $f(x) = x$ 时, 都精确成立, 但对 $f(x) = x^2$ 就不成立, 因此它们的代数精度为 *1*. 同样可以验证 *Simpson* 公式的代数精度为 *3*.

Remark 4.4. 机械求积公式只包含函数值, 但求积公式并不局限于机械求积公式, 有些求积公式可能会包含其它信息, 如导数值等.

4.3.2 构造机械求积

对机械求积公式而言, 重点在于怎么确定式(4.2)中各个 A_i 的值. 这里使用两种方法:

1. **解方程组**

   有几个未知量, 就列几个方程, 假设求积公式对前 n 次多项式都精确成立. 但缺点是方程组过大时难以求解.

2. **插值**

   使用插值法找到 $f(x)$ 的近似函数 $P_n(x)$(这个近似函数是一个多项式函数), 对 $P_n(x)$ 求积作为近似值.

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x \approx \int_a^b P_n(x)\mathrm{d}x &= \int_a^b \sum_{k=0}^{n} L_k(x) f(x_k)\,\mathrm{d}x \\ &= \sum_{k=0}^{n}\left(\int_a^b L_k(x)\mathrm{d}x\right) f(x_k) \triangleq \sum_{k=0}^{n} A_k f(x_k)\end{aligned} \tag{4.3}$$

即 $A_i = \int_a^b L_i(x)dx$, 其中 $L_i(x)$ 是插值基函数. 这样构造出来的求积公式称为**插值型**公式. 由插值余项公式可知, 插值型数值积分公式(4.3)的余项为

$$R[f] = \int_a^b (f(x) - P_n(x))\,dx = \int_a^b R_n(x)dx = \int_a^b \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega_{n+1}(x)dx \tag{4.4}$$

其中 $\omega_{n+1}(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$.

Remark 4.5. 由于 ξ_x 是关于 x 的未知函数, 上面(4.4)的误差值是无法得到的, 因此我们通常用下面的方法来估计误差

$$|R[f]| = \left|\int_a^b \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega_{n+1}(x)dx\right| \le \frac{M_{n+1}}{(n+1)!}\int_a^b |\omega_{n+1}(x)|\,dx \tag{4.5}$$

其中 $M_{n+1} = \max_{a\le x\le b}\left|f^{(n+1)}(x)\right|$.

注意到, 由于插值得到的函数 $P_n(x)$ 就是一个多项式, 而代数精度也是通过令 $f(x)$ 为多项式函数来进行计算, 因此插值法得到的代数精度至少为 n. 我们用一个定理来表述:

Theorem 4.2. 一个机械求积公式 $\sum_{i=1}^{n} A_i f(x_i)dx$ 是**插值型**公式当且仅当其代数精度为 n.(这里注意 n 是机械求积公式的项数)

Remark 4.6. 当机械求积公式具有尽可能高的代数精度时, 它总是插值型的.

4.4 Newton-Cotes 公式

由插值性求积公式的定义可知: 一旦求积节点 x_j 给定, 则求积系数 A_k 唯一确定, 进而求积公式唯一确定. 设计插值性求积公式的唯一目标即是: **选择合适的求积节点!**

Newton-cotes 公式给出了最简单的节点选择方式: 在积分区间均匀选择 $(n+1)$ 个点 (包含端点).

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} f(x_k) A_k = \sum_{k=0}^{n} f(x_k) \int_a^b l_k(x)dx, \quad x_k = a + k\frac{b-a}{n}, \tag{4.6}$$

其中 l_k 为插值基函数.

Remark 4.7. *当 $n = 1, 2$ 时, 式(4.6)分别为梯形公式和 Simpson 公式. 梯形公式和 Simpson 公式简单易用, 因此很受欢迎.*

作为插值型的求积公式, n 阶 Newton-cotes 公式(4.6)至少有 n 次代数精度. 实际的代数精度在 n 为偶数时能进一步提高到 $n+1$. 下面给出定理:

Theorem 4.3. *Newton-cotes 公式(4.6)至少有 n 次代数精度. 当 n 为偶数时, 至少有 $n+1$ 次代数精度.*

4.5 Newton-Cotes 公式的余项推导

计算求积公式余项小结 (三步曲)

1. 计算求积公式的代数精度, 设为 m;
2. 构造 m 次插值多项式, 写出插值条件和插值余项 (除导数部分外, 要确保不变号);
3. 利用积分中值定理, 计算出求积公式的余项.

4.5.1 梯形公式的余项

Theorem 4.4 (梯形公式的余项)**.** *设 $f \in C^2[a,b]$, 则梯形求积公式 $T(f)$ 的余项为*

$$R[f] = -\frac{(b-a)^3}{12} f''(\eta), \quad \eta \in (a,b). \tag{4.7}$$

所以, 带余项的梯形公式可写为

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2}(f(a)+f(b)) - \frac{(b-a)^3}{12} f''(\eta), \quad \eta \in (a,b). \tag{4.8}$$

证明. 前面我们推导了插值型数值积分公式的余项(4.4), $n = 1$ 时即对应梯形公式, 如果 $f''(\eta)$ 关于 x 是连续的, 则由于 $(x-a)(x-b)$ 在 $[a,b]$ 上不变号, 于是根据**积分第一中值定理**我们有

$$I(f) - T(f) = \int_a^b \frac{f''(\xi_x)}{2}(x-a)(x-b)dx = \frac{f''(\eta)}{2}\int_a^b (x-a)(x-b)dx = -\frac{f''(\eta)}{2}\frac{(b-a)^3}{6}.$$

需要指出的是, 这里要证明 $f''(\eta)$ 关于 x 是连续的. 事实上, 由 Lagrange 插值余项公式可知

$$f(x) - L_1(x) = \frac{1}{2!} f''(\xi_x)(x-a)(x-b),$$

即

$$f''(\xi_x) = \frac{2!\,(f(x) - L_1(x))}{(x-a)(x-b)} \triangleq g(x).$$

由于 $f \in C^2[a,b]$, 则上式右端 (即 $g(x)$) 显然在除插值节点以外的所有点都连续. 应用 L'Hôpital 法则, 我们可以求得 $g(x)$ 在插值节点处的极限 (在两端点处为右极限或左极限). 而根据余项公式, 在插值节点处, $f''(\xi_x)$ 可以取任意值. 因此, 我们可以将 $g(x)$ 的极限设为 $f''(\xi_x)$ 在节点处的值, 这样 $f''(\xi_x)$ 就在整个区间上连续. □

Remark 4.8. 上面证明 $f''(\eta)$ 关于 x 是连续的过程可推广到一般情形, 即: 设 $f \in C^{n+1}[a,b]$, 可得 n 次多项式插值余项

$$f(x) - L_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x)(x-x_0)(x-x_1)\cdots(x-x_n), \quad \xi_x \in [a,b], \tag{4.9}$$

通过定义 $f^{(n+1)}(\xi_x)$ 在插值节点处的值, 可使得 $f^{(n+1)}(\xi_x)$ 关于 x 是连续的.

4.5.2 Simpson 公式的余项

Theorem 4.5 (Simpson 公式的余项)**.** 设 $f \in C^4[a,b]$, 则 *Simpson* 求积公式 $S(f)$ 的余项为

$$R[f] = -\frac{(b-a)^5}{2880} f^{(4)}(\eta), \quad \eta \in (a,b). \tag{4.10}$$

所以, 带余项的 *Simpson* 公式可写为

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) - \frac{(b-a)^5}{2880} f^{(4)}(\eta), \quad \eta \in (a,b). \tag{4.11}$$

证明. 由于 Simpson 公式具有 3 次代数精度, 构造 $f(x)$ 关于点 $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$ 的三点三次 Hermite 插值多项式 $H_3(x)$, 满足

$$H_3(x_k) = f(x_k), k = 0,1,2, \quad H_3'(x_1) = f'(x_1).$$

于是有

$$\int_a^b H_3(x)\mathrm{d}x = \frac{b-a}{6}\left(H_3(x_0) + 4H_3(x_1) + H_3(x_2)\right), \tag{4.12}$$

且插值余项为

$$R(x) = \frac{f^{(4)}(\xi_x)}{4!}(x-x_0)(x-x_1)^2(x-x_2).$$

所以

$$\begin{aligned} R[f] &= \int_a^b f(x)dxx - \frac{b-a}{6}\left(f\left(x_0\right) + 4f\left(x_1\right) + f\left(x_2\right)\right) \\ &= \int_a^b f(x)dx - \frac{b-a}{6}\left(H_3\left(x_0\right) + 4H_3\left(x_1\right) + H_3\left(x_2\right)\right) \quad \text{(插值条件)} \\ &= \int_a^b f(x)dx - \int_a^b H_3(x)dx \quad \text{(由(4.12))} \\ &= \int_a^b \left(f(x) - H_3(x)\right)dx \\ &= \int_a^b \frac{f^{(4)}\left(\xi_x\right)}{4!}\left(x-x_0\right)\left(x-x_1\right)^2\left(x-x_2\right)dx. \end{aligned}$$

由于 $f^{(4)}\left(\xi_x\right)$ 是 x 的连续函数, 且 $\left(x-x_0\right)\left(x-x_1\right)^2\left(x-x_2\right)$ 在 $[a,b]$ 内不变号, 所以由积分中值定理可知, $\exists \eta \in (a,b)$, 使得

$$R[f] = \int_a^b \frac{f^{(4)}\left(\xi_x\right)}{4!}\left(x-x_0\right)\left(x-x_1\right)^2\left(x-x_2\right)dx = \frac{f^{(4)}(\eta)}{4!}\int_a^b \left(x-x_0\right)\left(x-x_1\right)^2\left(x-x_2\right)dx$$

将 $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$ 代入, 可求得

$$\int_a^b \left(x-x_0\right)\left(x-x_1\right)^2\left(x-x_2\right)dx = -\frac{(b-a)^5}{120}.$$

因此, Simpson 公式的余项为

$$R[f] = -\frac{f^{(4)}(\eta)}{4!}\cdot\frac{(b-a)^5}{120} = -\frac{(b-a)^5}{2880}f^{(4)}(\eta), \quad \eta \in (a,b).$$

□

4.5.3 Newton-Cotes 公式余项的一般形式

Theorem 4.6. *当 n 是奇数时, 若 $f \in C^{n+1}[a,b]$, 则 Newton-Cotes 公式的余项为*

$$R[f] = \frac{h^{n+2}f^{(n+1)}(\eta)}{(n+1)!}\int_0^n t(t-1)(t-2)\cdots(t-n)dt, \quad \eta \in (a,b). \tag{4.13}$$

当 n 是偶数时, 若 $f \in C^{n+2}[a,b]$, 则 Newton-Cotes 公式的余项为

$$R[f] = \frac{h^{n+3}f^{(n+2)}(\eta)}{(n+2)!}\int_0^n t^2(t-1)(t-2)\cdots(t-n)dt, \quad \eta \in (a,b). \tag{4.14}$$

4.6 复化求积公式

思想: 将区间 $[a,b]$ 分成 n 个小区间, 在每个区间上构造低阶求积公式, 并累加得到结果. 其中令小区间宽度为 $h=\frac{a-b}{n}$.

- **复化梯形公式**:

$$T_n=\int_a^b f(x)dx=\frac{h}{2}[f(a)+f(b)+2\sum_{k=1}^{n-1}f(x_k)].$$

- **复化 Simpson 公式**: 要求 n 为偶数, 令 $n=2m$, 则

$$S_n=\int_a^b f(x)dx=\frac{h}{3}[f(a)+f(b)+4\sum_{k=1}^{m}f(x_{2k-1})+2\sum_{k=1}^{m-1}f(x_{2k})].$$

它们的余项 (截断误差) 分别为

$$\begin{cases} R_T=-\frac{b-a}{12}h^2f''(\eta),\eta\in(a,b) \\ R_S=-\frac{b-a}{180}\left(\frac{h}{2}\right)^4f^{(4)}(\eta),\eta\in(a,b) \end{cases}$$

要注意, 使用复化梯形余项公式和复化 Simpson 余项公式分别要满足 $f(x)$ 在 $[a,b]$ 上分别有连续的 2, 3 阶导函数. 这两个复化公式中, 复化 Simpson 公式的效率最高.

Remark 4.9. 由余项公式可以看出, 当 $n\to\infty$, 上述两个余项公式均收敛到 0, 所以复化梯形公式和复化 *Simpson* 公式是收敛的. 易知公式中的求积系数都是正的, 因此两个公式是稳定的.

Remark 4.10. 复合梯形公式看似精度不高, 但如果被积函数是以 $b-a$ 为周期的周期函数, 则其具有 n 阶三角多项式精度:

$$\int_a^b f(x)\mathrm{d}x-h\sum_{k=0}^{n-1}f(a+kh)=\begin{cases}-(b-a), & 若m\neq 0\ 被n\ 整除,\\ 0, & 其他,\end{cases}$$

其中 $f(x)=\exp\left(\frac{2\pi\mathbf{i}mx}{b-a}\right)$, $\mathbf{i}$ 表示虚数单位, $h=\frac{b-a}{n}$.

4.7 Romberg 求积

4.7.1 Romberg 求积的推导

作为 Richardson 外推法的一个重要例子, Romberg 积分在计算光滑函数积分时可以达到极好的效果 (大优于 Simpson). 由于其容易理解, 算法实现简单, Romberg 积分也是广泛使用的一个方法.

思想: 使用渐进的方式, 先取 $n = 1$, 使用复化求积计算积分, 如果精度不达要求, 再取 $n = 2, 4, 8 \ldots$, 直到精度达到要求.

这就要求找到从 $n = n_1$ 到 $n = 2n_1$ 的递推关系, 以便可以利用上一步的计算结果进行计算. 推导可得, 梯形复化积分的递推公式分别为

$$T_{2n} = \frac{1}{2}T_n + \frac{h}{2}\sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right). \tag{4.15}$$

为了加速收敛, 我们在此基础上, 利用梯形公式的余项推导得出 T_{2n} 的误差, 并用这个误差修正 T_{2n}, 得到修正后的公式

$$\bar{T} = \frac{4}{3}T_{2n} - \frac{1}{3}T_n. \tag{4.16}$$

这里经过验证发现, 这个修正后的梯形公式, 竟然和 Simpson 公式, 也就是 S_n 的结果完全相等, 即 $\bar{T} = S_n$. 类似地, S_{2n} 用同样的方法修正, 在这个过程中, 精度越来越高. 因此, 我们把精度最高的 S_n 变步长 (渐进) 求积, 并用误差修正后的公式作为 Romberg 公式的最终值, 也就是

$$R_n(f) = \frac{16}{15}S_{2n}(f) - \frac{1}{15}S_n(f) = S_{2n}(f) + \frac{1}{15}\left[S_{2n}(f) - S_n(f)\right]. \tag{4.17}$$

Remark 4.11. *可以看到, Romberg 公式将之前所讲的方法都集于一身. 在计算 Romberg 公式时, 一般就按照 $T \to S \to R$ 的过程一步一步求积, 直到精度满足要求.*

4.7.2 Romberg 求积的算法

Romberg 公式的计算过程如下:

1. 令 $k = 0$, $h = b - a$
2. 计算 $T_0^{(0)} = \frac{h}{2}\left(f(a) + f(b)\right)h$
3. 令 $k = 1$
4. 利用梯形复化积分的递推公式(4.15)计算 $T_0^{(k)} = T\left(\frac{h}{2^k}\right)$
5. **for** i = 1,2,...,k **do**
6. 计算 $T_i^{(k-i)} = \frac{4^i T_{i-1}^{(k-i+1)} - T_{i-1}^{(k-i)}}{4^i - 1}$
7. **end for**
8. 若 $\left|T_k^{(0)} - T_{k-1}^{(0)}\right| < \varepsilon$, 则终止计算, 取 $T_k^{(0)}$ 为定积分近似值
9. 令 $k = k + 1$, 转到第 4 步

Romberg 算法的计算过程也可以用下面的表格来描述.

k	$h^{(k)}$	$T_0^{(k)}$	$T_1^{(k)}$	$T_2^{(k)}$	$T_3^{(k)}$	$T_4^{(k)}$	$\cdots$
0	$b-a$	$T_0^{(0)}$					
1	$\frac{b-a}{2}$	$T_0^{(1)}$	$T_1^{(0)}$				
2	$\frac{b-a}{2^2}$	$T_0^{(2)}$	$T_1^{(1)}$	$T_2^{(0)}$			
3	$\frac{b-a}{2^3}$	$T_0^{(3)}$	$T_1^{(2)}$	$T_2^{(1)}$	$T_3^{(0)}$		
4	$\frac{b-a}{2^4}$	$T_0^{(4)}$	$T_1^{(3)}$	$T_2^{(2)}$	$T_3^{(1)}$	$T_4^{(0)}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

4.8 Gauss 求积公式

在 Newton-Cotes 公式中, 我们选取的是等距节点, 这样做的好处就是计算方便. 但等距节点不一定是最好的选择, 事实上, 我们可以更好地选取节点, 使得求积公式具有更高的代数精度.

4.8.1 一般 Gauss 求积公式

Definition 4.4. 设 $\rho(x)$ 是 $[a,b]$ 上的权函数, 若求积公式

$$\int_a^b \rho(x)f(x)dx \approx \sum_{i=0}^{n} A_i f(x_i) \tag{4.18}$$

具有 $2n+1$ 次代数精度, 则称该公式为 ***Gauss* 求积公式**, 节点 x_i 称为 ***Gauss* 点**, A_i 称为 ***Gauss* 系数**.

Remark 4.12. 需要指出的是, 求积公式(4.4)的右端只包含 $f(x)$ 的函数值, 与权函数无关.

求积公式(4.4)中含有 $2n+2$ 的待定参数, 即 A_i 和 x_i, $i=0,1,2,\ldots,n$. 我们可以将 $f(x)=1,x,x^2,\ldots,x^{2n+1}$ 代入, 并令求积公式(4.4)精确成立, 然后解出 A_i 和 x_i . 这样就可以使得求积公式至少具有 $2n+1$ 次代数精度, 所以, Gauss 求积公式总是存在的.

Remark 4.13. 事实上, 求积公式(4.4)的代数精度不可能超过 $2n+1$. 取 $2n+2$ 次多项式 $f(x)=\prod_{j=0}^{n}(x-x_j)^2$ 则

$$0<\int_a^b \rho(x)f(x)dx \neq \sum_{k=0}^{n} A_k f(x_k)=0.$$

即求积公式 (4.4)对 $2n+2$ 次多项式 $f(x)$ 不精确成立, 所以它的代数精度小于 $2n+2$.

Theorem 4.7. *Gauss* 求积公式是具有最高代数精度的插值型求积公式.

4.8.2 Gauss 点的计算

我们所关心的是如何构造 Gauss 求积公式. 直观的想法是我们可以将 $f(x)=1,x,x^2,\ldots,x^{2n+1}$ 代入, 并令求积公式(4.4)精确成立, 这样就能解出 A_i 和 x_i. 但这时需要解一个非线性方程组, 而

一般情况下, 求解非线性方程组是非常困难的. 因此, 当 $n > 2$ 时, 这种方法是不可行的.

一个比较可行的方法是将 x_i 和 A_i 分开计算, 即先通过特殊的方法求出 Gauss 点 x_i, 然后再用待定系数法解出 A_i. 这也是目前构造 Gauss 公式的通用方法.

Theorem 4.8. *插值型求积公式的求积节点是 Gauss 点的充要条件是多项式*

$$\omega_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$$

与所有次数不超过 n 的多项式带权 $\rho(x)$ 正交, 即

$$\int_a^b \rho(x)\omega_{n+1}(x)P(x)dx = 0, \quad \forall P(x) \in H_n.$$

从而我们有计算 Gauss 点的一般方法:

1. 设 $\omega_{n+1}(x) = x^{n+1} + a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$;
2. 利用 $\omega_{n+1}(x)$ 与 $P(x) = 1, x, x^2, \ldots, x^n$ 带权正交的性质, 得到 $n+1$ 个线性方程, 解出 a_k, 这样就能确定多项式 $\omega_{n+1}(x)$;
3. 求出多项式 $\omega_{n+1}(x)$ 的 $n+1$ 个零点, 这就是 Gauss 点.

4.8.3 Gauss 求积公式的余项和收敛性与稳定性

Theorem 4.9. *Gauss 求积公式(4.4)的余项为*

$$R_n[f] = \frac{f^{(2n+2)}(\eta)}{(2n+2)!}\int_a^b \rho(x)\omega_{n+1}^2 \, \mathrm{d}x \tag{4.19}$$

Theorem 4.10. *设 $f(x) \in C[a,b]$, 则 Gauss 求积公式是收敛的, 即*

$$\lim_{n\to\infty}\sum_{i=0}^{n} A_i f(x_i) = \int_a^b \rho(x)f(x)dx$$

Theorem 4.11. *Gauss 求积公式中的系数 A_i 全是正数, 因此 Gauss 求积公式是稳定的.*

证明. 令 $f(x) = L_k^2(x) = \left(\prod_{i=0, i\neq k}^{n} \frac{x-x_i}{x_k-x_i}\right)^2 \in H_{2n}$, 由于 Gauss 公式具有 $2n+1$ 次代数精度, 所以

$$\int_a^b \rho(x)L_k(x)dx = \sum_{i=0}^{n} A_i l_k(x_i) = A_k,$$

上式左边显然大于 0, 故 $A_k > 0$, 所以结论成立. □

Chapter5. 解线性方程组的直接方法

本章和下一章主要介绍如何求解下面的线性方程组

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n. \tag{5.1}$$

目前求解线性方程组的方法可以分为两大类: 直接法和迭代法. 本章主要介绍直接法. 直接法具有良好的稳定性和健壮性, 是当前求解中小规模线性方程组的首选方法, 同时也是求解某些具有特殊结构的大规模线性方程组的主要方法. 在本章中, 我们总是假定系数矩阵 A 是非奇异的, 即线性方程组(5.1)的解存在且唯一.

5.1 Gauss 消去法

5.1.1 Gauss 消去法的计算过程

- Gauss 消去法的基本思想: 消元.
- Gauss 消去法的主要思路: 将系数矩阵 A 化为上三角矩阵, 然后回代求解.

Remark 5.1. 高斯消去法是求解线性方程组的经典算法，它在当代数学中有着非常重要的地位和价值，是线性代数的重要组成部分。高斯消去法除了用于线性方程组求解外，还用于计算矩阵行列式、求矩阵的秩、计算矩阵的逆等.

由高等代数的知识知, Gauss 消去法能顺利进行下去的充要条件是 $a_{kk}^{(k)} \neq 0, k = 1, 2, \dots, n$, 这些元素被称为**主元**.

Theorem 5.1. 设 $A \in \mathbb{R}^{n\times n}$, 则所有主元 $a_{kk}^{(k)}$ 都不为零的充要条件是 A 的所有顺序主子式都不为零, 即

$$D_1 \triangleq a_{11} \neq 0, \quad D_k \triangleq \begin{vmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kk} \end{vmatrix} \neq 0, \quad k = 2, 3, \dots, n.$$

Remark 5.2. 事实上, 如果 A 的所有顺序主子式都不为零, 则主元为

$$a_{11}^{(1)} = D_1, \quad a_{kk}^{(k)} = \frac{D_k}{D_{k-1}}, \quad k = 2, 3, \dots, n.$$

Proposition 5.2. *Gauss* 消去法能顺利完成的充要条件是 A 的所有顺序主子式都不为零.

5.1.2 Gauss 消去法的运算量

评价算法的一个重要指标是执行时间, 但这依赖于计算机硬件和编程技巧等, 因此直接给出算法执行时间是不太现实的. 所以我们通常是统计算法中算术运算 (加减乘除) 的次数. 在数值

算法中, 大多仅仅涉及加减乘除和开方运算. 一般地, 加减运算次数与乘法运算次数具有相同的量级, 而除法运算和开方运算次数具有更低的量级.

下面统计整个 Gauss 消去法的乘除运算的次数. 在第 k 步中, 我们需要计算

$$\begin{aligned} m_{ik} = a_{ik}^{(k)}/a_{kk}^{(k)}, i = k+1, \dots, n \quad &\to \quad n-k \text{ 次} \\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}, i = k+1, \dots, n \quad &\to \quad (n-k)^2 \text{ 次} \\ b_i^{(k+1)} = b_i^{(k)} - m_{ik}b_k^{(k)}, i = k+1, \dots, n \quad &\to n-k \text{ 次} \end{aligned}$$

所以整个消去过程的乘除运算为

$$\sum_{k=1}^{n-1} 2(n-k) + (n-k)^2 = \sum_{\ell=1}^{n-1} 2\ell + \ell^2 = n(n-1) + \frac{n(n-1)(2n-3)}{6}.$$

易知, 回代求解过程的乘除运算为

$$1 + \sum_{i=1}^{n-1} n - i + 1 = \frac{n(n+1)}{2}.$$

所以整个 Gauss 消去法的乘除运算为

$$\frac{n^3}{3} + n^2 - \frac{n}{3}.$$

同理, 也可统计出, Gauss 消去法中的加减运算次数为

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}.$$

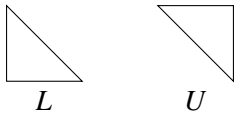
5.1.3 Gauss 消去法与 LU 分解

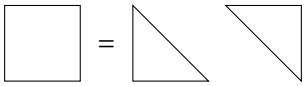
换个角度看 Gauss 消去过程: 每次都是做矩阵初等变换, 因此也可理解为不断地左乘初等矩阵. 将所有这些初等矩阵的乘积记为 $\tilde{L}$, 则可得 $\tilde{L}A = U$, 其中 U 是一个上三角矩阵. 记 $L \triangleq \tilde{L}^{-1}$, 则

$$A = LU,$$

这就是**矩阵 LU 分解**.

Theorem 5.3 (LU 分解的存在性和唯一性)**.** *设 $A \in \mathbb{R}^{n\times n}$, 则存在唯一的单位下三角矩阵 L 和非奇异上三角矩阵 U, 使得 $A = LU$ 的充要条件是 A 的所有顺序主子矩阵 $A_k = A(1:k, 1:k)$ 都非奇异, $k = 1, 2, \dots, n$.*





如果 A 存在 LU 分解, 则 $Ax = b$ 可写为 $LUx = b$, 因此就等价于求解下面两个方程组

$$\begin{cases} Ly = b \\ Ux = y \end{cases} \tag{5.2}$$

由于 L 是单位下三角, U 是非奇异上三角, 因此上面的两个方程都很容易求解.

将上面的 LU 分解过程写成算法如下:

1. Set $L = I$, $U = 0$ % 将 L 设为单位矩阵, U 设为零矩阵
2. **for** $k = 1$ to $n - 1$ **do**
3. **for** $i = k + 1$ to n **do**
4. $m_{ik} = a_{ik}/a_{kk}$ % 计算 L 的第 k 列
5. **end for**
6. **for** $i = k$ to n **do**
7. $u_{ki} = a_{ki}$ % 计算 U 的第 k 行
8. **end for**
9. **for** $i = k + 1$ to n **do**
10. **for** $j = k + 1$ to n **do**
11. $a_{ij} = a_{ij} - m_{ik}u_{kj}$ % 更新 $A(k + 1 : n, k + 1 : n)$
12. **end for**
13. **end for**
14. **end for**

Remark 5.3. *Gauss 消去法何时好用?*

- 带状矩阵
- 三对角矩阵 *(D-对角矩阵的三角形分解的计算量为 $O(d^2N)$)*

5.1.4 列主元 Gauss 消去法

我们知道, 只要系数矩阵 A 非奇异, 则线性方程组就存在唯一解. 但 Gauss 消去法却不一定有效, 即可能会出现主元为零的情形, 此时算法就进行不下去. 在实际计算中, 即使主元都不为零, 但如果主元的值很小, 由于舍入误差的原因, 也可能会给计算结果带来很大的误差.

解决上面问题的一个有效方法就是选主元. 具体做法就是, 在执行 Gauss 消去过程的第 k 步之前, 插入下面的选主元过程.

1. 选取**列主元**: $\left|a_{i_k,k}^{(k)}\right| = \max_{k\le i\le n}\left\{\left|a_{i,k}^{(k)}\right|\right\}$;
2. 交换: 如果 $i_k \neq k$, 则交换第 k 行与第 i_k 行.

上面选出的 $a_{i_k,k}^{(k)}$ 就称为**列主元**. 加入这个选主元过程后, 就不会出现主元为零的情形 (除非 A 是奇异的). 因此 Gauss 消去法就不会失效. 这种带选主元的 Gauss 消去法就称为**列主元 Gauss 消去法**.

Theorem 5.4 (列主元 LU 分解)**.** *若矩阵 A 非奇异, 则存在置换矩阵 P, 使得*

$$PA = LU, \tag{5.3}$$

其中 L 为单位下三角矩阵, U 为上三角矩阵.

Remark 5.4. *列主元 Gauss 消去法比普通 Gauss 消去法要多做一些比较运算, 但列主元 Gauss 消去法*

- *对系数矩阵要求低, 只需非奇异即可;*
- *比普通 Gauss 消去法更稳定.*

Remark 5.5. *列主元 Gauss 消去法是当前求解线性方程组的直接法中的首选算法. 类似地, 我们可以衍生出***全主元 *Gauss* 消去法***, 全主元 Gauss 消去法具有更好的稳定性, 但很费时, 在实际计算中一般很少采用.*

5.2 对称正定矩阵的三角分解法

前面讨论的是一般线性方程组, 并没有考虑系数矩阵的特殊结构. 如果 A 是对称正定的, 则可以得到更加简洁高效的方法.

5.2.1 Cholesky 分解

Theorem 5.5. *若 A 对称, 且所有顺序主子式都不为 0, 则 A 可以唯一分解为*

$$A = LDL^T,$$

其中 L 为单位下三角矩阵, D 为对角矩阵.

Theorem 5.6. 设 A 对称正定, 则存在唯一的对角线元素全为正的下三角矩阵 L, 使得

$$A = LL^T,$$

该分解称为 ***Cholesky* 分解**.

5.2.2 计算 Cholesky 分解

我们仍然使用待定系数法. 设

$$A = LL^{\mathrm{T}}, \quad \text{即} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n,2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{n,n} \end{bmatrix}.$$

类似于 LU 分解, 直接比较等式两边的元素, 可得到一般公式

$$a_{ij} = \sum_{k=1}^{n} l_{ik} l_{jk} = \sum_{k=1}^{j-1} l_{ik} l_{jk} + l_{jj} l_{ij}, \quad j = 1, 2, \ldots, n, \quad i = j, j+1, \ldots, n. \tag{5.4}$$

根据这个计算公式即可得下面的 **Cholesky 分解算法** (这里我们将 L 存放在 A 的下三角部分).

1. **for** $j = 1$ to n **do**
2. **for** $j = 1$ to n **do**
3. $a_{jj} = a_{jj} - a_{jk}^2$
4. **end for**
5. $a_{jj} = \sqrt{a_{jj}}$
6. **for** $i = j + 1$ to n **do**
7. **for** $k = 1$ to $j - 1$ **do**
8. $a_{ij} = a_{ij} - a_{ik} a_{jk}$
9. **end for**
10. $a_{ij} = a_{ij} / a_{jj}$
11. **end for**
12. **end for**

Remark 5.6. *Cholesky 分解算法的乘除运算量为 $\frac{1}{3}n^3 + O(n^2)$, 大约为 LU 分解的一半. 另外, Cholesky 分解算法是稳定的 (稳定性与全主元 Gauss 消去法相当), 因此不需要选主元.*

利用 Cholesky 分解求解线性方程组的方法就称为**平方根法**. 具体的 **Cholesky 分解求解线性方程组**算法如下:

1. 计算 Cholesky 分解 (此处省略, 参见 Cholesky 分解算法)
2. $y_1 = b_1/a_{11}$ % 向前回代求解 $Ly = b$
3. **for** $i = 2$ to n **do**
4. **for** $j = 1$ to $i - 1$ **do**
5. $b_i = b_i - a_{ij}y_j$
6. **end for**
7. $y_i = b_i/a_{ii}$
8. **end for**
9. $x_n = b_n/a_{nn}$ % 向后回代求解 $L^{\mathrm{T}}x = y$
10. **for** $i = n - 1$ to 1 **do**
11. **for** $j = i + 1$ to n **do**
12. $y_i = y_i - a_{ji}x_j$
13. **end for**
14. $x_i = y_i/a_{ii}$
15. **end for**

5.2.3 改进的平方根法

在 Cholesky 分解中, 需要计算平方根. 当矩阵为负定或不定矩阵时, Cholesky 分解不可直接使用, 为了避免计算平方根, 我们可以采用改进的 Cholesky 分解, 即

$$A = LDL^{\mathrm{T}},$$

其中 L 为单位下三角矩阵, D 为对角矩阵. 这个分解也称为 LDL^{T} 分解. 由待定系数法, 设

$$A = LDL^{\top} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} 1 & l_{21} & \cdots & l_{n1} \\ & 1 & \ddots & \vdots \\ & & \ddots & l_{n,n-1} \\ & & & 1 \end{bmatrix}$$

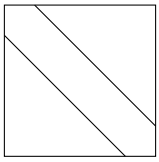
比较等式两边的元素, 可得

$$a_{ij} = d_j l_{ij} + \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}, \quad j = 1, 2, \dots, n, \quad i = j+1, j+2, \dots, n. \tag{5.5}$$

基于以上分解的求解对称正定线性方程组的算法就称为**改进的平方根法**, 描述如下 (将 L 存放在 A 的下三角部分, D 存放在 A 的对角部分).

5.3 三对角线性方程组的追赶法

这里考虑一类具有简单结构的线性方程组, 即三对角线性方程组.



在计算样条插值时会遇到这类线性方程组. 由于系数矩阵的特殊结构, 我们可以设计更加简洁高效的求解方法.

考虑线性方程组 $Ax = f$, 其中 A 是三对角矩阵:

$$A = \begin{bmatrix} b_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix}.$$

我们假定

$$|b_1| > |c_1| > 0, \quad |b_n| > |a_{n-1}| > 0, \tag{5.6}$$

且

$$|b_i| \geq |a_{i-1}| + |c_i|, \quad a_i c_i \neq 0, \quad i = 1, \dots, n-1. \tag{5.7}$$

即 A 是不可约弱对角占优的 (定义见下一章), 对上述方程进行 LU 分解, 得到如下形式

$$A = \begin{bmatrix} b_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix} = \begin{bmatrix} \alpha_1 & & & \\ a_1 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & a_{n-1} & \alpha_n \end{bmatrix} \begin{bmatrix} 1 & \beta_1 & & \\ & 1 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & & 1 \end{bmatrix} \triangleq LU \tag{5.8}$$

由待定系数法, 我们可以得到递推公式:

$$\begin{aligned}
&\alpha_1 = b_1, \quad \beta_1 = c_1/\alpha_1 = c_1/b_1 \\
&\begin{cases} \alpha_i = b_i - a_{i-1}\beta_{i-1} \\ \beta_i = c_i/\alpha_i = c_i/\left(b_i - a_{i-1}\beta_{i-1}\right), \end{cases} \quad i = 2,3,\ldots,n-1 \\
&\alpha_n = b_n - a_{n-1}\beta_{n-1}.
\end{aligned}$$

为了使得算法能够顺利进行下去, 我们需要证明 $\alpha_i \neq 0$.

Theorem 5.7. 设三对角矩阵 A 满足条件(5.6)和(5.7), 则 A 非奇异, 且

(1) $|\alpha_1| = |b_1| > 0$;

(2) $0 < |\beta_i| < 1, \quad i = 1,2,\ldots,n-1$;

(3) $0 < |c_i| \leq |b_i| - |a_{i-1}| < |\alpha_i| < |b_i| + |a_{i-1}|, \quad i = 2,3,\ldots,n.$

证明. 由于 A 是不可约且弱对角占优, 所以 A 非奇异.

结论 (1) 是显然的, 下面证明 (2) 和 (3).

由于 $0 < |c_1| < |b_1|$, 且 $\beta_1 = c_1/b_1$, 所以 $0 < |\beta_1| < 1$. 又 $\alpha_2 = b_2 - a_1\beta_1$, 所以

$$|\alpha_2| \geq |b_2| - |a_1| \cdot |\beta_1| > |b_2| - |a_1| \geq |c_2| > 0, \tag{5.9}$$

$$|\alpha_2| \leq |b_2| + |a_1| \cdot |\beta_1| < |b_2| + |a_1|. \tag{5.10}$$

再由结论(5.9)和 β_2 的计算公式可知 $0 < |\beta_2| < 1$. 类似于(5.9)和(5.10), 我们可以得到

$$|\alpha_3| \geq |b_3| - |a_2| \cdot |\beta_2| > |b_3| - |a_2| \geq |c_3| > 0,$$

$$|\alpha_3| \leq |b_3| + |a_2| \cdot |\beta_2| < |b_3| + |a_2|.$$

依此类推, 我们就可以证明结论 (2) 和 (3). □

由 Theorem 5.7可知, 分解(5.8)是存在的. 因此, 原方程就转化为求解 $Ly = f$ 和$Ux = y$. 由此便可得求解三对角线性方程组的**追赶法**, 也称为 **Thomas 算法**. 其乘除运算量大约为 $5n$, 加减运算大约为 $3n$.

Chapter6. 解线性方程组的迭代法

我们需要清楚的是: 向量范数是对于向量长度的一种度量, 矩阵范数是对于矩阵作为线性变换作用在向量上的效果的度量. 由向量与矩阵范数, 我们可以定义向量间的距离, 并将微积分 (极限概念) 扩展到向量空间与矩阵空间上. 与上一章一样, 我们都将集中精力在线性方程组(5.1)的数值求解方法上. 我们都很熟悉上一章的 Gauss 方法, 因为其消元的操作非常易于我们理解. 但是现在, 我们要跳出这样的思维局限. 容易发现求解方程从本质上来说与求解零点是相同的, 只不过现在我们要求解的是一个更复杂的矩阵函数 $Ax - b$ 的零点.

Remark 6.1. 我们在下一章会介绍零点求解方法, 我们可以从那些方法中得到一些启发. 对于零点求解, 主要的方法有二分法与不动点迭代法两大类. 二分法显然不适合用来处理复杂的线性方程组求解问题, 我们想是否可以类比**不动点迭代**法创造出一些全新的线性方程组求解方法. 在不动点迭代中, 我们将 $f(x) = 0$ 的问题化为 $g(x) = x$ 求解, 并且保证其等解性. 而在矩阵处理中, 我们可以类似地, 将对角线上的元素进行变量分离, 来起到不动点迭代中 x 的效果. 由这样的一个基本想法, 我们可以得到最基础的迭代求解方法:*Jacobi* 迭代法.

6.1 Jacobi 迭代法

6.1.1 Jacobi 迭代法的计算公式

我们对线性方程组(5.1)第 i 个方程分离变量得到

$$a_{ii}x_i = b_i - \left(a_{i1}x_1 + \ldots + a_{i,i-1}x_{i-1} + a_{i,i+1}x_{i+1} + a_{in}x_n\right),$$

进一步我们有

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

在变形完成之后, 我们继续模仿求解零点的不动点方法. 在求解零点时, 我们先选取初值 p_0, 并且生成 $p_1 = g(p_0), p_2 = g(p_1), \ldots$, 在求解线性方程组中, 我们类似地, 利用上一次的估计值来生成下一次的估计. 我们用 $x^{(k)}$ 表示第 k 次迭代中生成的对方程组解的估计. 给定迭代初值 $x^{(0)}$, 则我们得到如下 **Jacobi 迭代法的计算式**:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n. \tag{6.1}$$

其实就是每行的 b 减去 a 和 x 乘积的和 (除了对角元素), 然后除以对角元素:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j} a_{ij}x_j^{(k)} - a_{ii}x_i^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

我们可能仍旧会疑惑: Jacobi 方法与不动点迭代在形式上完全不同, 为何却说它是不动点迭代在矩阵代数中的推广呢? 我们下面尝试着将 Jacobi 方法的估计式写成矩阵形式即可立马清晰地看出其本质. 我们在后文都假设对于矩阵 A, D 为其对角元素组成的对角矩阵, L 为其下三角部分 (不含主对角线) 元素的相反数组成的下三角矩阵, U 为其上三角部分 (不含主对角线) 元素的相反数组成的上三角矩阵. 即进行一个简单的分解使得

$$A = D - L - U.$$

则式(6.1)可以写成

$$Dx^{(k+1)} = (L + U)x^{(k)} + b. \tag{6.2}$$

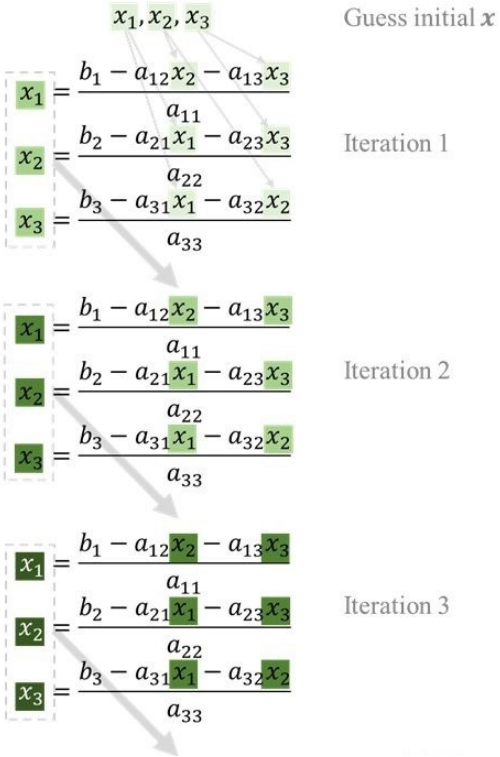
当 D 可逆时, 我们有

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b. \tag{6.3}$$

这就是 Jacobi 迭代法的矩阵形式. 回顾刚刚做的事情, 容易发现我们将线性方程组(5.1)转化为了等解的矩阵迭代形式(6.3). 进一步抽象的话, Jacobi 迭代法给出了线性方程组求解的 $x^{(k+1)} = Tx^{(k)} + c$ 的迭代形式, 其中 $T = D^{-1}(L + U)$, $c = D^{-1}b$. 因此在学习完下一章的知识后, 我们可以说 **Jacobi 迭代法为不动点迭代的推广**.

6.1.2 Jacobi 迭代的求解过程

下图演示了 $n = 3$ 时 Jacobi 迭代求解的过程. 首先随便猜测初始值 (一般设为 0), 然后代入每一行的式(6.1). 得到所有新的 x 后进入下一轮迭代.

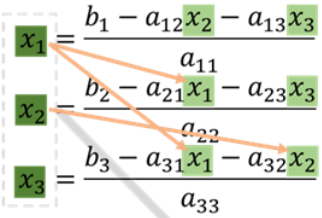


6.2 Gauss-Seidel 迭代法

6.2.1 Gauss-Seidel 迭代法的计算公式

在实际应用中, Jacobi 迭代法收敛的速度是不够快的, 那么问题是要如何改进 Jacobi 迭代法呢? 我们再观察一下 Jacobi 迭代法的估计式(6.1), 我们发现这样的一个估计式仍旧有改进的余地: 注意到在第 $k+1$ 次迭代中估计第 i 个分量时, 我们已经完成了第 $k+1$ 次迭代中对于 $x_1, x_2, \dots, x_{i-1}$ 的估计. 故我们可以用第 $k+1$ 次迭代中的值而不是第 k 次迭代中的值, 这样会显著地提高收敛速度.

Remark 6.2. 通俗来说, *Jacobi* 迭代法每次迭代总是使用上一步的结果, 每行全部独立计算完后才进入下一轮迭代. 其实每当计算 x_2 时, 本轮迭代的 x_1 已经更新了, 我们却还在使用上一轮提供的 x_1 . 同样计算 x_3 时, 本轮迭代的 x_1 和 x_2 已经更新了, 我们还在用上一轮提供的 x_1 和 x_2.



如果能利用最新的信息及时更新, 就会提高运算效率. 经过如此改进后的方法就是 **Gauss-Seidel 迭代法**.

Gauss-Seidel 迭代法计算公式:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, 2 \dots, n. \tag{6.4}$$

可见 Gauss-Seidel 迭代法利用了当前步已经求出的结果 $\{x_1, x_2, \dots, x_{i-1}\}$, 也即式(6.4)中 $\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)}$ 的部分.

对于 Gauss-Seidel 迭代法, 我们先如同前面一样给出其矩阵形式. 我们可以将计算式(6.4)写成

$$Dx^{(k+1)} = Lx^{(k+1)} + Ux^{(k)} + b,$$

从而得到

$$(D - L)x^{(k+1)} = Ux^{(k)} + b,$$

当 $D - L$ 可逆时, 可以得到 Gauss-Seidel 迭代法的矩阵形式

$$x^{(k+1)} = (D - L)^{-1}Ux^{(k)} + (D - L)^{-1}b. \tag{6.5}$$

同样地, 我们容易看出其也是不动点迭代方法的推广, 而区别仅仅在于其进行了与 Jacobi 方法不同的等解变换, 利用了不同的不动点迭代函数.

Remark 6.3. *Jacobi 迭代和 Gauss-Seidel 迭代形式几乎一样, 但 Gauss-Seidel 及时利用了最新的迭代结果, 直观上比 Jacobi 效率更高. 大量实践都证明了 Gauss-Seidel 迭代法在大多数情况下收敛速度快于 Jacobi 方法.*

6.2.2 Gauss-Seidel 迭代法的求解过程

我们依然用图示来演示 Gauss-Seidel 迭代的流程.

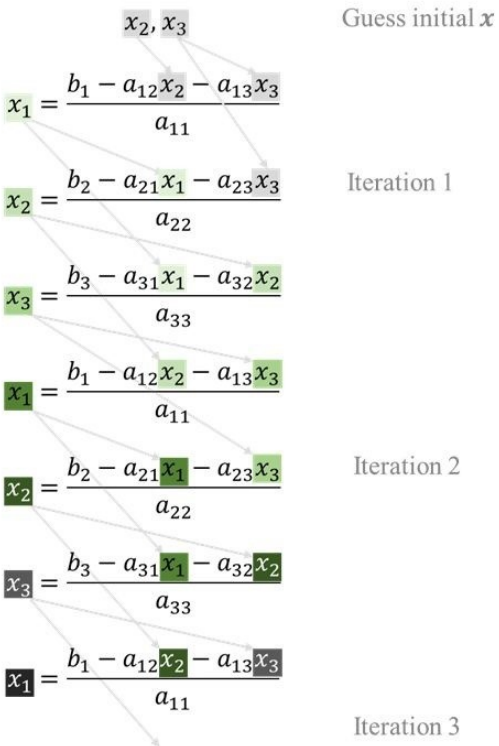


图 6.1: Gauss-Seidel 迭代的流程

6.3 Jacobi 迭代和 Gauss-Seidel 迭代的收敛性

我们已经得到了求解线性方程组的两种迭代方法, 但是我们之前一直没有考虑一个十分重要的问题: 这两种方法对任意矩阵都收敛吗? 如若不然, 收敛条件是什么呢?

下面我们对于这类利用不动点迭代进行推广得到的迭代方法进行收敛条件的分析. 我们如同之前一样, 将一个迭代方法抽象为 $x^{(k+1)} = Tx^{(k)} + c$, 而 Jacobi 与 Gauss-Seidel 方法均只是 T 和 c 取一定值条件下的特例. 则我们有如下收敛条件

Theorem 6.1. *迭代方法 $x^{(k+1)} = Tx^{(k)} + c$ 收敛到 $x = Tx + c$ 的解当且仅当 $\rho(T) < 1$.*

证明.

- 一方面若 $\rho(T) < 1$, 则 $I - T$ 可逆, 且 $\sum_{j=0}^{\infty} T^j = (I - T)^{-1}$, 又因为

$$x^{(k)} = Tx^{(k-1)} + c = T^2x^{(k-2)} + (T + I)c = \ldots = T^kx^{(0)} + \left(I + T + \ldots + T^{k-1}\right)c,$$

故

$$\lim_{k\to\infty} x^{(k)} = x^{(0)} \lim_{k\to\infty} T^k + (I - T)^{-1}c.$$

由于一个矩阵收敛当且仅当其谱半径小于 1, 故

$$\lim_{k\to\infty} T^k = 0,$$

从而 $\lim_{k\to\infty} x^{(k)} = (I - T)^{-1}c$ 即为 $x = Tx + c$ 的解.

- 另一方面, 要证明 $\rho(T) < 1$, 即等价于证明 $\forall z$, $\lim_{k\to\infty} T^kz = 0$. 我们利用 Lagrange 中值定理的形式进行估计. 我们发现

$$\lim_{k\to\infty} T^kz = 0x - x^{(k)} = T\left(x - x^{(k-1)}\right) = \ldots = T^k\left(x - x^{(0)}\right),$$

则很自然地, 取 $x^{(0)} = x - z$, 则 $x - x^{(k)} = T^kz$. 由于方法收敛, 我们有

$$\lim_{k\to\infty} x - x^{(k)} = 0,$$

故

$$\lim_{k\to\infty} T^kz = 0.$$

□

事实上, 线性方程组的迭代方法作为不动点迭代的推广, 在性质上与不动点迭代非常类似. 我们同样可以得到不动点迭代形式的误差估计. 要注意的是, $\rho(T)$ 在某种程度上代替了导数在不动点迭代之中的作用. $\rho(T)$ 越小, 方法收敛速度越快. 但是由于计算谱半径比较复杂, 直接给出收敛性判定的一般结论:

Theorem 6.2. 设 $Ax = b$, 则

1. 当 A 为严格对角占优矩阵时, *Jacobi* 迭代法与 *Gauss-Seidel* 迭代法均收敛;
2. 当 A 对称正定, *Gauss-Seidel* 迭代法收敛;
3. 当 A 对称正定且 $2D - A$ 对称正定, 则 *Jacobi* 迭代也收敛.

Remark 6.4. 直观解释:

1. 严格对角占优, 对角阵就记录了 A 大部分信息;
2. A 对称正定, *Gauss-Seidel* 利用下三角和对角来近似 A , 实际上已经包含了 A 的所有信息;

3. 如果 A 对称正定且 $2D - A$ 对称正定也是为了保证对角线元素比重更大.

6.4 Jacobi 迭代和 Gauss-Seidel 迭代对比

我们可以看到 Jacobi 迭代只利用了对角信息, 表现为每轮迭代只利用上一轮的结果; Gauss-Seidel 迭代利用了对角 + 下三角的信息, 表现为及时利用本轮已经更新的结果.

虽然我们一直在说 Gauss-Seidel 方法是 Jacobi 方法的改进, 且大多数情况下能使得求解更快收敛, 这却并不是绝对的. 那么 Jacobi 方法就会被 Gauss-Seidel 方法淘汰吗?

- 我们在计算 Jacobi 迭代过程的时候有一个细节: 每行全部独立计算完后进入下一轮迭代. 这意味着 **Jacobi 计算式天然适用于并行计算的**! 在每轮迭代中, 每个进程可以独立计算每一行, 而 **Gauss-Seidel 迭代法为了利用最近更新的结果只能按序执行**. 可以从二者的流程图中体会.
- 有些时候 Jacobi 迭代法不收敛但是 Gauss-Seidel 迭代法收敛, 有些时候 Jacobi 迭代法收敛但是 Gauss-Seidel 迭代法不收敛, 因此需要针对不同的情况选取不同的方法.

Example 6.1. 对于 $A = \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 4 \\ -5 \end{bmatrix}$, 容易验证 *Jacobi* 迭代法中 $\rho\left(T_j\right) = \frac{\sqrt{5}}{2}$, 而 *Gauss-Seidel* 方法中 $\rho\left(T_g\right) = \frac{1}{2}$, 这意味着 ***Jacobi* 迭代法不收敛**, ***Gauss-Seidel* 迭代法收敛**.

Example 6.2. 对于 $A = \begin{bmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}, b = \begin{bmatrix} 7 \\ 2 \\ 5 \end{bmatrix}$, 容易验证 *Jacobi* 迭代法中 $\rho\left(T_j\right) = 0$, 而 *Gauss-Seidel* 方法中 $\rho\left(T_g\right) = 2$, 这意味着 ***Jacobi* 迭代法收敛**, ***Gauss-Seidel* 迭代法不收敛**.

6.5 超松弛迭代法 (SOR)

接下来我们要介绍的超松弛方法 (SOR) 本质上是对于 G-S 方法的改进, 其引入了估计的权重以达到加速收敛与减小残差的目的. 该方法的基本思想是将 G-S 方法中的第 $k+1$ 步近似解与第 k 步近似解做一个加权平均, 从而给出一个新的近似解.

6.5.1 SOR 的计算公式

SOR 的计算公式:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right). \tag{6.6}$$

类比之前两个方法分析中的做法, 我们容易写出 SOR 估计式的矩阵形式:

$$(D-\omega L)x^{(k+1)} = ((1-\omega)D+\omega U)\,x^{(k)} + \omega b.$$

当 $D-\omega L$ 可逆时, 可写为

$$x^{(k+1)} = (D-\omega L)^{-1}\,((1-\omega)D+\omega U)\,x^{(k)} + \omega(D-\omega L)^{-1}b. \tag{6.7}$$

Remark 6.5.

- 容易看到, *SOR* 本质上也是不动点迭代方法的推广.
- 当 $\omega = 1$ 时, *SOR* 即为 *G-S* 迭代法.
- 当 $\omega > 1$ 时, 称为**超松弛法**; 当 $\omega < 1$ 时, 称为**低松弛法**

6.5.2 SOR 的收敛性

关于松弛参数 ω 的选取并不是任意的.

Theorem 6.3 (SOR 迭代法收敛的必要条件)**.** 设解线性方程组 $Ax = b$ 的 *SOR* 迭代法收敛, 则 $0 < \omega < 2$.

事实上, 对于**对称正定矩阵**而言, $0 < \omega < 2$ 的 SOR 方法对任意初值必定收敛. 对于对称正定的三**对角矩阵**而言, 我们甚至能够找到理论上的最佳 ω , 使得 $\rho(T_w) = w - 1$ 成立.

Chapter7. 非线性方程与方程组的数值解法

7.1 二分法

首先介绍一些基本概念:

- **解, 根, 零点**: 所有满足 $f(x^*) = 0$ 的数 x^* 都称为方程的解或根, 也称为 $f(x)$ 的零点.
- **根的重数**: 若 $f(x) = (x - x^*)^m g(x)$ 且 $g(x^*) \neq 0$, 则 x^* 为 $f(x) = 0$ 的 m 重根;
- **有解区间**: 若 $[a, b]$ 内至少存在 $f(x) = 0$ 的一个实数解, 则称 $[a, b]$ 为有解区间.

我们研究内容就是在**有解**的前提下求出方程 $f(x) = 0$ 的**近似根**.

Remark 7.1. 如果没有特别说明, 本讲我们只考虑实数解. 非线性方程求解通常比较困难, 而且可能存在多个或无穷多个解, 因此在求解时一般要强调求解区域, 即计算指定区域内的解.

7.1.1 二分法的数学思想

设 $f(x) = 0$ 在区间 $[a, b]$ 内至少有一个实数解. 二分法 (Bisection) 的基本思想就是将这个有解区间进行对分, 并找出解所在的小区间, 记为新的有解区间, 然后再对这个小区间进行对分. 依次类推, 直到有解区间的长度足够小为止, 此时有解区间内的任意一点都可以作为 $f(x) = 0$ 的近似解. 在实际计算中, 通常取中点. 二分法的数学思想是下面的介值定理:

Theorem 7.1 (介值定理)**.** 设 $f(x) \in C[a, b]$, 且 $f(a)f(b) < 0$, 则在 (a, b) 内至少存在一点 ξ, 使得 $f(\xi) = 0$.

7.1.2 二分法的收敛性

我们二分法的误差进行估计. 记算法在第 k 步时得到的有解区间为 $[a_k, b_k]$, 中点为 x_k. 则 $a_1 = a$, $b_1 = b$, 且

$$|x_k - x_*| = \left|\frac{1}{2}(a_k + b_k) - x_*\right| \leq \frac{1}{2}(b_k - a_k).$$

由于每次都是对分有解区间, 因此有

$$b_k - a_k = \frac{1}{2}(b_{k-1} - a_{k-1}) = \frac{1}{2^2}(b_{k-2} - a_{k-2}) = \cdots = \frac{1}{2^{k-1}}(b_1 - a_1).$$

因此

$$|x_k - x_*| \leq \frac{1}{2^k}(b_1 - a_1) = \frac{1}{2^k}(b - a).$$

所以

$$\lim_{k\to\infty} |x_k - x_*| = 0.$$

即算法收敛. 因此我们有下面的收敛性结论.

Theorem 7.2. 设 $f(x) \in C[a,b]$, 且 $f(a)f(b) < 0$, 则二分法收敛到 $f(x) = 0$ 的一个解.

Remark 7.2. **二分法适用范围**: 只适合求连续函数的单重实根或奇数重实根;

- **优点**: 简单易用, 只要满足介值定理的条件, 算法总是收敛的;
- **缺点**:
  1. 收敛速度较缓慢;
  2. 不能求复根和偶数重根;
  3. 一次只能求一个根.

总结: 二分法一般可先用来计算解的一个粗糙估计, 再用其他方法进行加速, 如 *Newton* 法.

7.1.3 二分法的代码实现

```
function [x, A, B, X, iter] = Bisection(f, a, b, eps)
% 二分法求解非线性方程
%
% 输入:
% f: 非线性方程对应函数
% [a,b]: 求根区间
% eps: 精度
%
% 输出:
% x: 根
% [A,B]: 求根区间序列
% M: 根序列
% iter: 迭代次数

if nargin < 4
    eps = 1e-6;
end

A = [];
B = [];
X = [];

iter = 0;

while abs(a - b) > eps
    iter = iter+1;
    x = (a + b) / 2;
    X = [X; x];
    A = [A; a];
    B = [B; b];
    if f(x) == 0
```

```
        break
    else
        if f(a) * f(x)<0
            b = x;
        else
            a = x;
        end
    end
end
end
```

7.2 不动点迭代法

7.2.1 不动点迭代法的数学思想

不动点迭代的基本思想是将原方程 $f(x) = 0$ 改写成一个等价的方程 $\varphi(x) - x = 0$ 或者 $x = \varphi(x)$, 然后就可以根据这个等价方程构造出一个迭代格式:

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \ldots. \tag{7.1}$$

其中 x_0 为迭代初始值, 可以任意选取. 这就是**不动点迭代法** (Fixed-Point iteration), 简称**迭代法**, $\varphi(x)$ 称为**迭代函数**.

由于方程 $f(x) = 0$ 和 $x = \varphi(x)$ 是等价的, 因此 x^* 是 $f(x) = 0$ 的解当且仅当 $x^* = \varphi(x^*)$, 即它是 $\varphi(x)$ 的一个不动点.

Remark 7.3.

- 不动点迭代的一个非常重要的特征是将方程求解转化为函数求值, 后者显然要容易很多.
- 不动点的一个几何含义是曲线 $y = \varphi(x)$ 与直线 $y = x$ 的交点, 因此不动点迭代过程可以用几何图像来表示.

7.2.2 不动点迭代法的收敛性

7.2.2.1 全局收敛性

Theorem 7.3 (不动点的存在唯一性)**.** 设 $\varphi(x) \in C[a, b]$ 且满足

- $\forall x \in [a, b]$, 都有 $\varphi(x) \in [a, b]$,
- $\exists L \in (0, 1)$, 使得 $\forall x, y \in [a, b]$ 都有

$$|\varphi(x) - \varphi(y)| \leq L|x - y|, \tag{7.2}$$

则 $\varphi(x)$ 在 $[a, b]$ 上存在唯一的不动点.

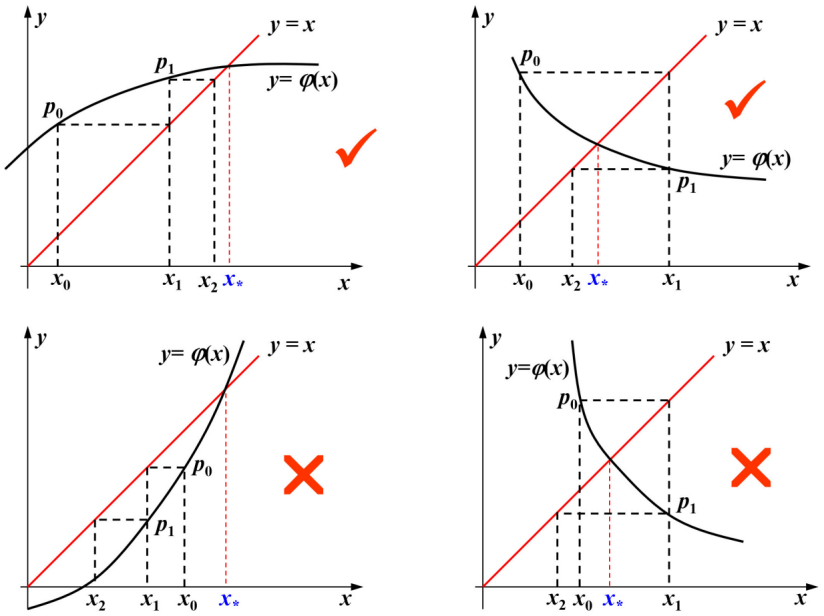


图 7.1: 不同迭代函数的不动点迭代过程几何表示

Remark 7.4. 条件(7.3)称为 ***Lipschitz* 条件**.

Theorem 7.4. 设 $\varphi(x) \in C[a,b]$ 且满足

- $\forall x \in [a,b]$, 都有 $\varphi(x) \in [a,b]$,
- $\exists L \in (0,1)$, 使得 $\forall x, y \in [a,b]$ 都有

$$|\varphi(x) - \varphi(y)| \le L|x - y|. \tag{7.3}$$

则对任意初始值 $x_0 \in [a,b]$, 不动点迭代(7.1)收敛, 且

$$|x_k - x^*| \le \frac{L}{1-L}|x_k - x_{k-1}| \le \frac{L^k}{1-L}|x_1 - x_0|,$$

其中 x^* 是 $\varphi(x)$ 在 $[a,b]$ 内存在唯一的不动点.

Remark 7.5. 由定理的结论可知, L 越小, 收敛越快!

如果 $\varphi(x)$ 在 $[a,b]$ 上连续, 在 (a,b) 内可导, 则由 Lagrange 中值定理可知, 对任意 $x, y \in [a,b]$, $\exists \xi \in (a,b)$ 使得

$$\varphi(y) - \varphi(x) = \varphi'(\xi)(y - x),$$

因此

$$|\varphi(y) - \varphi(x)| \le \max_{\xi \in [a,b]} |\varphi'(\xi)| \cdot |y - x|.$$

于是我们可以立即得到下面的定理.

Theorem 7.5. 设 $\varphi(x) \in C[a,b]$ 且 $\forall x \in [a,b]$, 都有 $\varphi(x) \in [a,b]$. 若 $\exists L \in (0,1)$, 使得

$$|\varphi'(x)| \leq L < 1, \quad \forall x \in [a,b],$$

则对任意初始值 $x_0 \in [a,b]$, 不动点迭代(7.1)收敛, 且

$$|x_k - x^*| \leq \frac{L}{1-L}|x_k - x_{k-1}| \leq \frac{L^k}{1-L}|x_1 - x_0|.$$

以上两个定理中, 迭代法的收敛性与迭代初值的选取无关, 这种收敛性称为**全局收敛性**.

7.2.2.2 局部收敛性

Definition 7.1. 设 x^* 是 $\varphi(x)$ 的不动点, 若存在 x^* 的某个 δ-邻域

$$U_\delta(x^*) \triangleq \{x \in \mathbb{R} : |x - x^*| < \delta\}$$

使得 $\forall x_0 \in U_\delta(x^*)$, 不动点迭代(7.1)均收敛, 则称该迭代是**局部收敛**的.

局部收敛意味着只有当初值离真解足够近时, 才能保证收敛. 由于真解是不知道的, 因此如果迭代法只具有局部收敛性, 则初值选取会比较困难, 很有可能无法保证算法的收敛. 这也是**局部收敛与全局收敛的最大区别**.

在实际计算中, 可以用其他具有全局收敛性的方法 (比如二分法) 获取一个近似解, 然后再进行迭代.

Theorem 7.6. 设 x^* 是 $\varphi(x)$ 的不动点, 若 $\varphi'(x)$ 在 x^* 的某个邻域内连续且 $|\varphi'(x^*)| < 1$, 则不动点迭代(7.1)局部 *(线性)* 收敛, 即 1 阶收敛.

Remark 7.6. 在前面的介绍的迭代法中, 计算 x_{k+1} 时, 只用到点 x_k 的值. 有时, 我们为了利用前面多个点的信息, 比如前面 $\ell \geq 2$ 个点, 可以设计下面的迭代法:

$$x_{k+1} = \varphi(x_k, x_{k-1}, \ldots, x_{k-\ell+1}), \quad k = \ell - 1, \ell, \ell+1, \ldots. \tag{7.4}$$

我们称之为**多点迭代法**. 比如后面会提到的割线法和抛物线法, 都是多点迭代. 显然, 多点迭代法一开始需要给定 ℓ 个初值 $x_0, x_1, \ldots, x_{\ell-1}$.

7.3 Newton 法

Newton 法是当前求解非线性方程 (组) 的最常用方法, 也是一般情况下的首选方法.

7.3.1 Newton 法的基本思想

Newton 法的基本思想是将**非线性方程线性化**.

设 x_k 是 $f(x)=0$ 的一个近似根, 在 x_k 附近我们用切线 $P(x)=f(x_k)+f'(x_k)(x-x_k)$ 来近似非线性函数 $f(x)$. 于是可以用 $P(x)$ 的零点来近似 $f(x)$ 的零点 (因此 Newton 法又称**切线法**), 并将其记为 x_{k+1}, 即

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}. \tag{7.5}$$

这就是 Newton 法的迭代格式, 其几何意义可以用下面的图像表示

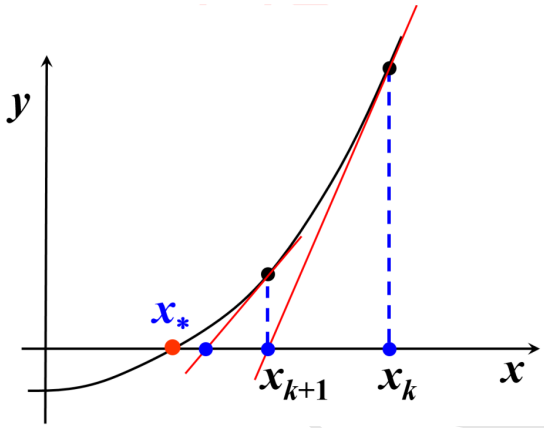


图 7.2: Newton 法的几何含义

Remark 7.7. *为了使得 Newton 法能顺利进行, 一般要求 $f'(x)\neq 0$.*

7.3.2 Newton 法的收敛性

Theorem 7.7. *设 x^* 是 $f(x)$ 的零点, 且 $f'(x^*)\neq 0$, 则 Newton 法至少二阶局部收敛.*

Remark 7.8. *Newton 法的优点是收敛速度快 (至少二阶局部收敛), 特别是当迭代点充分靠近精确解时. 但缺点是*

- *对重根收敛速度较慢, 只有线性收敛;*
- *对初值的选取很敏感, 要求初值相当接近真解, 因此在实际使用时, 可以先用其它方法获取一个近似解, 然后使用 Newton 法加速;*
- *每一次迭代都需要计算导数, 工作量大.*

7.3.3 Newton 法的代码实现

```
function [x, X, iter] = Newton(f, x0, eps)
% 牛顿法
%
% 输入:
% f: 方程对应函数
% x0: 迭代初值
```

```
% eps: 精度
%
% 输出:
% x: 根
% X: 根序列
% iter: 迭代次数

if nargin < 3
    eps = 1e-6;
end

syms t
y = f(t);
df = diff(y,t);
df = matlabFunction(df);
phi = t - y/diff(y,t);
phi = matlabFunction(phi);

x = x0;
x1 = x0 + 0.01;
X = [];
X = [X; x1; x0]; % 增加一个初始值只是为了 while 循环判断, 并不参与计算
iter = 0;

while abs(X(end) - X(end - 1)) > eps
    iter = iter+1;
    if df(x) == 0
        disp('计算失败! '); break
    else
        x = phi(x);
        X = [X;x];
    end
end

end
```

7.4 割线法与抛物线法

目的: 避免计算 Newton 法中的导数, 并且尽可能地保持较高的收敛性 (即超线性收敛).

7.4.1 割线法

割线法 (Secant Method) 也称**弦截法**, 主要思想是**用差商代替微商**, 即

$$f'(x_k) \approx f[x_{k-1}, x_k] = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

代入 Newton 法即可得**割线法的迭代格式**:

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \quad , \quad k = 1, 2, \ldots . \tag{7.6}$$

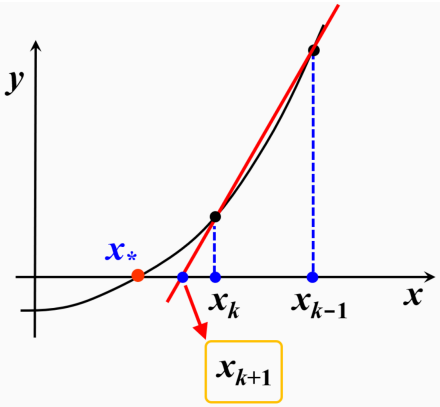


图 7.3: 割线法的几何含义

Remark 7.9. 割线法需要提供两个迭代初始值.

关于割线法超线性的收敛性, 我们有下面的结论.

Theorem 7.8. 假设 $f(x)$ 在根 x^* 的邻域 $U(x^*, \delta)$ 内二阶连续可导, 且 $f'(x) \neq 0$. 若初值 $x_0, x_1 \in U(x^*, \delta)$, 则当 δ 充分小时, 割线法具有 p 阶收敛性, 其中

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.618,$$

即 p 是 $p^2 - p - 1 = 0$ 的一个根.

下面是割线法的函数代码实现:

```
function [x,X,iter] = xianjie(f,x0,eps)
% 弦截法
%
% 输入:
% f: 方程对应函数
% x0: 迭代初值
% eps: 精度
%
% 输出:
% x: 根
% X: 根序列
% iter: 迭代次数

if nargin < 3
```

```
    eps = 1e-6;
end

x = x0;
x1 = x0+0.01;
X = [];
X = [X;x1;x0];
iter = 0;

while abs(X(end)-X(end-1)) > eps
    iter = iter+1;
    x = x - f(x)*(x-X(end-1))/(f(x)-f(X(end-1)));
    X = [X;x];
end

end
```

7.4.2 抛物线法

在 Newton 法和割线法中, 我们都是用直线来近似 $f(x)$. Newton 法可以看作是基于单个点的一次 Hermite 插值, 而割线法则是两点线性插值. 抛物线法 (Muller 法) 的主要思想则是用**基于三个点的二次插值多项式来近似** $f(x)$.

具体做法如下: 假定已知三个的迭代值 x_{k-2}, x_{k-1}, x_k, 构造过点 $(x_{k-2}, f(x_{k-2}))$, $(x_{k-1}, f(x_{k-1}))$, $(x_k, f(x_k))$ 的二次曲线 $p_2(x)$, 然后用 $p_2(x)$ 的零点作为下一步的迭代值 x_{k+1}. 在抛物线法中, 有

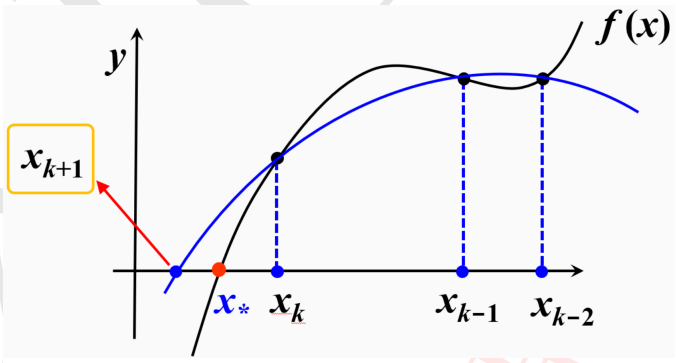


图 7.4: 抛物线法的几何含义

两个问题需要解决:

- **二次曲线 $p_2(x)$ 的构造**. 可以通过插值方法解决, 比如, 由 Newton 插值公式可得

$$p_2(x) = f(x_k) + f[x_k, x_{k-1}](x - x_k) + f[x_k, x_{k-1}, x_{k-2}](x - x_k)(x - x_{k-1}).$$

- **零点的选取**. 此时 $p_2(x)$ 有两个零点, 即

$$x_k - \frac{2f(x_k)}{\omega \pm \sqrt{\omega^2 - 4f(x_k) f[x_k, x_{k-1}, x_{k-2}]}},$$

其中

$$\omega = f[x_k, x_{k-1}] + f[x_k, x_{k-1}, x_{k-2}](x_k - x_{k-1}).$$

取哪个作为 x_{k+1} 呢? 通常的做法是取靠近 x_k 的那个零点.

Remark 7.10. 在一定条件下可以证明: 抛物线法的局部收敛阶为 $p \approx 1.840$. ($p^3 - p^2 - p - 1 = 0$)

Remark 7.11. 抛物线法需提供三个初始值. 抛物线法可能涉及复数运算, 有时可以用来求复根. 并且与割线法相比, 抛物线法具有更高的收敛阶.

7.5 代码实现: 二分法, Newton 法和割线法的对比

```
% 非线性方程求根
f = @(x) x.^3 - x - 1;
a = 1;
b = 2;
x0 = 1.5;
[x1, A, B, X1, iter1] = Bisection(f, a, b); %二分法
[x2,X2,iter2] = Newton(f, x0, eps); %Newton 法
[x3,X3,iter3] = xianjie(f, x0, eps); %弦截法

% 绘图
figure
subplot(3,1,1)
xx = a: 0.01: b;
plot(xx, f(xx), '-.', 'LineWidth', 2); hold on
plot(X1(:), f(X1(:)), 'o', 'LineWidth', 2)
xlabel('x');
ylabel('f(x)')
% xticks(a:0.05:b); yticks(-1:0.1:2); grid on
title('二分法')

subplot(3,1,2)
xx = a: 0.01: b;
plot(xx, f(xx), '-.', 'LineWidth', 2); hold on
plot(X2(:), f(X2(:)), 'o', 'LineWidth',2)
xlabel('x');
ylabel('f(x)')
title('牛顿法')

```

```
29 subplot(3,1,3)
30 xx = a: 0.01: b;
31 plot(xx,f(xx),'-.','LineWidth',2); hold on
32 plot(X3(:),f(X3(:)),'o','LineWidth',2)
33 xlabel('x'); ylabel('f(x)')
34 title('弦截法')
```

运行代码后输出图像如下:

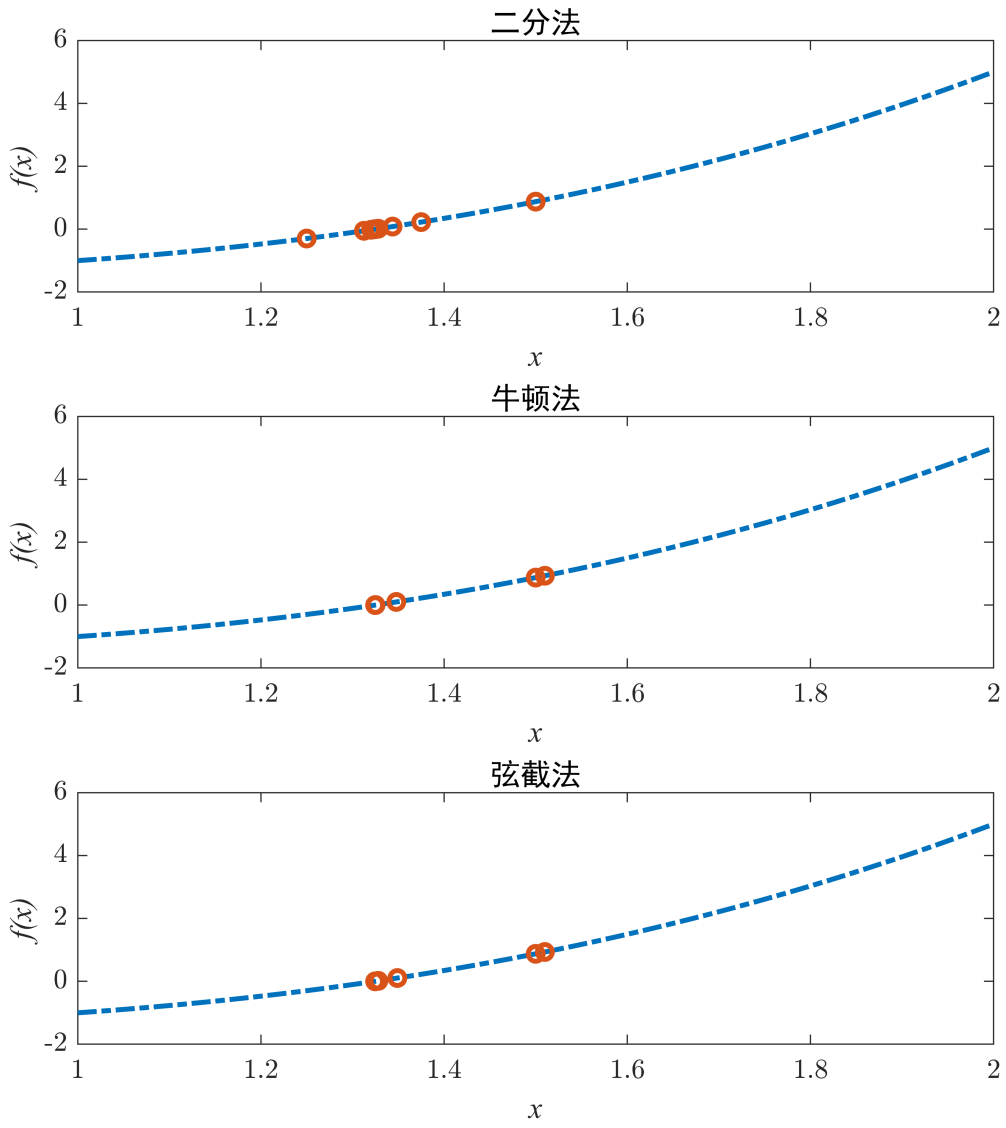


图 7.5: 三种方法的对比

可见对于本次求解过程, 对于相同精度要求, Newton 法和割线法的迭代速度要比二分法快近三倍, 说明牛顿法和弦截法的收敛速度是比较快的.

Chapter8. 矩阵特征值计算

To be continued ...

Chapter9. 常微分方程初值问题数值解法

To be continued ...