

附录 A IEEE 浮点运算标准

在数值计算中, 小数在内存中是以浮点数格式表示和参与运算的. 浮点数是数字 (或者说数值) 在内存中的一种存储格式, 它和定点数是相对的.

A.1 浮点数与定点数

浮点数和定点数中的“点”指的是小数点.

所谓定点数, 就是指小数点的位置是固定的, 不会向前或者向后移动. 比如我们用 4 个字节 (32 位字长) 来存储无符号的定点数 x (通常用 4 个字节存储单精度数), 并且约定: 前 16 位表示整数部分, 后 16 位表示小数部分, 如下图所示:



这种表示方法的优点是: 整数部分和小数部分一目了然, 非常直观.

 对于整数, 所有位都用来存储整数部分, 所以一般采用定点数格式存储.

但定点数格式有个很大的缺点, 就是所能表示的数的取值范围比较小. 比如在前面的例子中 (4 个字节), 所能表示的最大值和最小值 (非零) 是

$$x_{\max} = 1111111111111111.1111111111111111_2 \approx 2^{16} = 65536,$$

$$x_{\min} = 0000000000000000.0000000000000001_2 = 2^{-16} \approx 1.5 \times 10^{-5}.$$

这在科学计算中显然是远远不够的. 比如电子的质量大约是 9×10^{-28} 克, 用 4 位定点数格式就无法表示. 即使是采用 8 个字节 (通常用 8 个字节存储双精度数), 假定前 32 位表示整数, 后 32 位表示小数, 则所能表示的的数的范围为 (0 除外)

$$x_{\max} \approx 2^{32} \approx 4.3 \times 10^9, \quad x_{\min} = 2^{-32} \approx 2.3 \times 10^{-10}.$$

为了克服这个缺点, 人们发明了一种更加科学的存储格式, 即浮点数格式, 也就是通常所说的科学计数法. 该格式以指数形式存储数字, 不但节省内存, 也非常直观, 而且所能表示的数的范围也大大增加.

A.2 IEEE 中的浮点数的表示方法

自计算机发明以来, 曾出现许多中不同的浮点数表示方式, 但目前最通用的是 IEEE 二进制浮点数算术标准 (IEEE Standard for Binary Floating-Point Arithmetic, 简称 IEEE 754 标准).

IEEE 754 标准的主要起草者是加州大学伯克利分校数学系的 William Kahan 教授, 他帮助 Intel 公司设计了 8087 浮点处理器, 并以此为基础形成了 IEEE 754 标准, Kahan 教授也因此获得了 1989 年的图灵奖.

通常一个浮点数由符号、尾数、基和指数组成, 如:

$$-0.31415926_{10} \times 10^2, \quad 0.10101_2 \times 2^3.$$

这里要求小数点前面为零, 小数点后面的数称为**尾数**. 若尾数的首位数字不为 0 时, 我们称其为**正规数**(或**规范化数**), 否则称为**次正规数**(或**非规范化数**). 如 $0.314_{10} \times 10^2$ 是正规数, 而 $0.00314_{10} \times 10^4$ 是次正规数. 正规化表示方法可以使得每个浮点数的表示方式唯一, 而且可以空出一个位置, 使得表示精度更高.

- IEEE 754 标准中定义了表示浮点数的四种格式:
 - 两种基本的浮点数: **单精度** (32 位字长) 和 **双精度** (64 位字长).
其中单精度格式具有 24 位有效数字 (二进制), 而双精度格式具有 53 位有效数字 (二进制), 相对于十进制来说, 分别是 7 位 ($2^{24} \approx 10^7$) 和 16 位 ($2^{53} \approx 10^{16}$) 有效数字.
 - 两种扩展的浮点数: **单精度扩展** 和 **双精度扩展**.
IEEE 754 标准中并未规定扩展格式的精度和大小, 但它指定了最小精度和字长: 单精度扩展需 43 位字长以上, 双精度扩展需 79 位字长以上 (64 位有效数字). 单精度扩展很少使用, 而对于双精度扩展, 不同的机器架构中有着不同的规定, 有的为 80 位字长 (如 X86), 有的为 128 位字长 (如 SPARC).
- 一般来说, 描述一个浮点数的三个基本要素为:
 - **基**: 计算机一般都以 2 为基;
 - **尾数**的位数: 确定有效数字的位数, 即精度;
 - **指数**的位数: 确定所能表示的数的范围.
- 在 IEEE 754 标准中, 浮点数是用二进制表示的, 由三部分组成: 符号 (sign, 其值用 s 表示), 指数 (exponent, 其值用 e 表示) 和尾数 (fraction, 其值用 f 表示), 见图 A.1. 单精度数占 32 位字长 (4 个字节), 第 1 位是符号位, 第 2 至 9 位 (8 位字长) 是指数位, 最后 23 位是尾数. 双精度数占 64 位字长 (8 个字节), 第 1 位是符号位, 第 2 至 12 位 (11 位字长) 是指数位, 最后 52 位是尾数.




图 A.1. IEEE 754 中单精度格式与双精度格式的位模式

- **单精度格式**: 用 8 位字长的二进制数来表示指数, 因此 e 的取值范围为 $[0, 255]$. 当 $0 \leq e < 255$ 时, 按单精度格式存储的数, 其对应的值是使用以下方法得到的:

将二进制基数点 (小数点) 插入到尾数 f 最高有效位的左侧, 并将一个**隐含位**插入到二进制基数点的左侧, 从而得到的是一个二进制带分数 (整数加小数).

由此构成的带分数就是**单精度格式有效数字**. 隐含位的值并不是显式指定的 (即不存储), 而是通过指数 e 的值来隐式指定:

- 当 $0 < e < 255$ 时, 表示该数为**二进制正规数**, 此时隐含位设为 1.
- 当 $e = 0$ 时, 表示该数为**二进制次正规数**, 隐含位设为 0.

 由于引入了隐含位 (为了尽可能地增加所能表示的数的精度), 这里的正规数概念与前面的定义有点区别, 因此我们加上“二进制”三个字.

单精度格式位模式中的尾数只有 23 位, 但由于使用了隐含位, 所以能提供 24 位有效数字 (二进制). 因此, 在 IEEE 中, 单精度数的表示方法为

$$\begin{aligned} &(-1)^s \times 1.f \times 2^{e-127} \quad (\text{二进制正规数}) \\ &(-1)^s \times 0.f \times 2^{-126} \quad (\text{二进制次正规数}) \end{aligned}$$

完整的对应关系是

单精度格式位模式	值
$0 < e < 255$	$(-1)^s \times 1.f \times 2^{e-127}$ (二进制正规数)
$e = 0, f \neq 0$	$(-1)^s \times 0.f \times 2^{-126}$ (二进制次正规数)
$e = 0, f = 0$	$(-1)^s \times 0.0$ (有符号的零)
$e = 255, f = 0, s = 0$	+inf (正无穷大)
$e = 255, f = 0, s = 1$	-inf (负无穷大)
$e = 255, f \neq 0$	NaN (非数、非确定值)

其中 127 是单精度格式的**指数偏移值** (exponent bias), 在 IEEE 标准中, 这个值定义为 $2^{(\text{指数位长}-1)} - 1$. 所以对于单精度格式, 指数偏移值就是 $2^{8-1} - 1 = 127$, 而对于双精度格式, 这个值为 $2^{11-1} - 1 = 1023$.

- **双精度格式**: 与单精度格式类似, 完整的对应关系是

双精度格式位模式	值
$0 < e < 2047$	$(-1)^s \times 1.f \times 2^{e-1023}$ (二进制正规数)
$e = 0, f \neq 0$	$(-1)^s \times 0.f \times 2^{-1022}$ (二进制次正规数)
$e = 0, f = 0$	$(-1)^s \times 0.0$ (有符号的零)
$e = 2047, f = 0, s = 0$	+inf (正无穷大)
$e = 2047, f = 0, s = 1$	-inf (负无穷大)
$e = 2047, f \neq 0$	NaN (非数、非确定值)




例 A.1 单精度格式所能表示的十进制数范围.

- 最大二进制正规数为 $7F7FFFFF_{16} = 3.40282347 \times 10^{38}$
- 最小(正的)二进制正规数为 $00800000_{16} = 1.17549435 \times 10^{-38}$
- 最大二进制次正规数为 $007FFFFFFF_{16} = 1.17549421 \times 10^{-38}$
- 最小(正的)二进制次正规数为 $00000001_{16} = 1.40129846 \times 10^{-45}$

由此可见, 浮点数所能表示的数的范围比定点数要大很多.

例 A.2 双精度格式所能表示的十进制数范围.

- 最大二进制正规数为 $7FEFFFFFFF_{16} = 1.7976931348623157 \times 10^{308}$
- 最小(正的)二进制正规数为 $0010000000000000_{16} = 2.2250738585072014 \times 10^{-308}$
- 最大二进制次正规数为 $000FFFFFFF_{16} = 2.2250738585072009 \times 10^{-308}$
- 最小(正的)二进制次正规数为 $0000000000000001_{16} = 4.9406564584124654 \times 10^{-324}$
- $3FF0000000000000_{16} = 1$ (补码)

 在 MATLAB 中, `hex2num` 可以将一个由 16 个 16 进制数组成的字符串转化为其所对应的浮点数(根据 IEEE 标准), 类似的命令有 `hex2dec`, `bin2dec`, `base2dec`, `num2hex`, `dec2bin`, ...

例 A.3 把二进制数 $(1001.0101)_2$ 转换成十进制数.

$$\begin{aligned} (1001.0101)_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &\quad + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-2} \\ &= 9.3125_{10} \end{aligned}$$

例 A.4 把十进制数 13.125_{10} 转换成二进制数.

整数部分: $13_{10} = 1101_2$ (辗转相除法)

小数部分:

- $0.125 \times 2 = 0.25$, 整数位是 $0 \rightarrow .0$;
- $0.25 \times 2 = 0.5$, 整数位是 $0 \rightarrow .00$;
- $0.5 \times 2 = 1$, 整数位是 $1 \rightarrow .001$; (小数部分已变为 0, 运算结束)

所以 $13.125_{10} = 1101.001_2$

一个十进制数能否用二进制浮点数精确表示, 关键在于小数部分.

例 A.5 十进制数 0.1_{10} 能否用二进制数精确表示?

- $0.1 \times 2 = 0.2$, 整数位是 $0 \rightarrow .0$;



- $0.2 \times 2 = 0.4$, 整数位是 0 $\rightarrow .00$;
- $0.4 \times 2 = 0.8$, 整数位是 0 $\rightarrow .000$;
- $0.8 \times 2 = 1.6$, 整数位是 1 $\rightarrow .0001$;
- $0.6 \times 2 = 1.2$, 整数位是 1 $\rightarrow .00011$;
- $0.2 \times 2 = 0.4$, 整数位是 0 $\rightarrow .000110$;
-

得到一个无限循环的二进制小数, 显然用有限位字长是无法表示的, 因此 0.1_{10} 无法用 IEEE 754 浮点数精确表示.

同理可知 $0.2, 0.4, 0.6, 0.8, 0.3, 0.7, 0.9$ 也是无法精确表示的. 故 0.1 至 0.9 的 9 个小数中, 只有 0.5 可以精确表示.

例 A.6 能用二进制数精确表示的十进制小数. 易知

$$0.1_2 = 2_{10}^{-1} = 0.5$$

$$0.01_2 = 2_{10}^{-2} = 0.25$$

$$0.001_2 = 2_{10}^{-3} = 0.125$$

$$0.0001_2 = 2_{10}^{-4} = 0.0625$$

$$0.00001_2 = 2_{10}^{-5} = 0.03125$$

$$0.000001_2 = 2_{10}^{-6} = 0.015625$$

$$0.0000001_2 = 2_{10}^{-7} = 0.0078125$$

$$0.00000001_2 = 2_{10}^{-8} = 0.00390625$$

... ..

由此可知, 一个十进制小数要能用浮点数精确表示, 最后一位必须是 5. 当然这是必要条件, 并非充分条件. 如 0.35 就无法精确表示.

例 A.7 N 位二进制小数能精确表示的非零十进制小数总共有多少个?

- 1 位二进制小数能精确表示的有 $2^0 = 1$ 个 ($0.1_2 = 0.5_{10}$);
- 2 位二进制小数能精确表示的有 $2^1 = 2$ 个 ($0.01_2 = 0.25_{10}, 0.11_2 = 0.75_{10}$);
- 3 位二进制小数能精确表示的有 $2^2 = 4$ 个
-
- N 位二进制小数能精确表示的有 2^{N-1} 个

所以 N 位二进制小数能精确表示的十进制小数总共有 2^{N-1} 个.

A.3 IEEE 中的浮点数运算

- IEEE 754 标准也定义了浮点数的运算规则:



- **加、减、乘、除、平方根、余数、将浮点格式的数舍入为整数值、在不同浮点格式之间转换、在浮点和整数格式之间转换以及比较**: IEEE 对以上浮点运算的准确度作了规定: 求余和比较运算必须精确无误. 其他运算必须向其目标提供精确的结果, 除非没有此类结果, 或者该结果不满足目标格式, 此时运算必须按照下面介绍的舍入模式对精确结果进行最低限度的修改, 并将经过修改的结果提供给运算的目标.
- **在十进制字符串和两种基本浮点格式之一的二进制浮点数之间进行转换的准确度、单一性和一致性要求**: 对于在指定范围内的操作数, 这些转换必须生成精确的结果 (如果可能的话), 或者按照规定的舍入模式, 对此类精确结果进行最低限度的修改. 对于不在指定范围内的操作数, 这些转换生成的结果与精确结果之间的差值不得超过取决于舍入模式的指定误差.
- 五种类型的 IEEE 浮点异常: **无效运算 (如 $0/0$, ∞/∞ 等), 被零除, 上溢, 下溢和不精确**, 以及用于向用户指示发生这些类型异常的条件.
- 四种舍入模式: 设 x 是所要表示的数,
 - (1) **就近舍入**: 用最接近 x 的可表示的值来代替, 类似于整数的四舍五入. 如果 x 正好在两个相邻的可表示值的中间, 则首选二进制“偶数”(二进制最后一位为 0);
 - (2) **向下舍入**: 用不大于 x 的可表示的值来代替 (向负无穷大方向截断);
 - (3) **向上舍入**: 用不小于 x 的可表示的值来代替 (向正无穷大方向截断);
 - (4) **向 0 舍入**: 当 $x > 0$ 时采用向下舍入, 当 $x < 0$ 时采用向上舍入.

 我们将后面三种舍入模式统称为 **截断**.

 不同编译器对舍入可能有不同的处理方式.

- 下溢

当运算结果非常小时, 就会发生下溢. 下表是下溢阈值.

目标的精度	下溢阈值	
单精度	最小正规数	$1.17549435 \times 10^{-38}$
	最大次正规数	$1.17549421 \times 10^{-38}$
双精度	最小正规数	$2.2250738585072014 \times 10^{-308}$
	最大次正规数	$2.2250738585072009 \times 10^{-308}$

IEEE 算法处理下溢的方式是**渐进下溢**: 当生成的正确结果的数量级低于最小正正规数时, 就会生成次正规数, 而不是返回零.

- **机器精度**: 将 1.0 与大于 1.0 的最小浮点数之间的距离记为 ε_m . 它的一半称为 **unit roundoff**, 记为 ε_u , 它是计算机表示一个浮点数时的相对误差界,

$$\text{fl}(x) = x(1 + \delta) \quad \text{或} \quad \text{fl}(x) = \frac{x}{1 + \delta}, \quad |\delta| \leq \varepsilon_u.$$

这里 $\text{fl}(x)$ 表示 x 在计算机中实际存储的 IEEE 浮点数.

在 IEEE 标准下, 单精度和双精度浮点运算的最大相对误差 ε_u 分别为

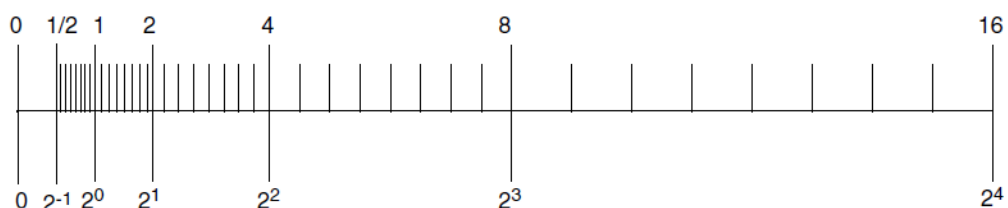


精度	最大相对误差
单精度	$2^{-24} \approx 5.960464 \times 10^{-8}$
双精度	$2^{-53} \approx 1.110223 \times 10^{-16}$

如果采用的不是就近舍入模式, 而是其他三种舍入模式 (即截断), 则最大相对误差为 ε_m .

有的文献中称 ε_u 为机器精度 (**machine epsilon**, **machine precision**, or **machepts**), 如 Demmel [30], LAPACK, Scilab, Wikipedia. 也有的文献称 ε_m 为机器精度, 如 Higham [68], MATLAB, Mathematica. 我们采用前面一种方式, 即“机器精度”指的是 ε_u .

例 A.8 假定要使用只有三个精度位的二进制算法. 那么, 最大相对误差为 2^{-3} . 在任意两个 2 的幂之间, 只有 $2^3 - 1 = 7$ 个可表示数字, 如下图所示.



数轴显示了数字之间的差距是随着指数增加而加倍增加的.

在 IEEE 单精度格式中, 两个最小正次正规数之间的差大约是 10^{-45} , 而两个最大有限数之间的数量级差大约是 10^{31} !

精确是偶然的, 误差是必然的. 做数值算法, 惟一能做的就是尽量使误差的传播和累积能够得到有效的控制.

A.4 浮点运算舍入误差分析

由于计算机无法精确表示所有的浮点数, 在做浮点运算时, 如果计算结果无法精确表示, 此时就会产生的误差, 这就是浮点运算的**舍入误差**. 根据 IEEE 浮点运算标准, 如果 $a \odot b$ 的结果无法精确表示, 则用一个最接近的浮点数来代替 (浮点运算时一般采用就近舍入模式), 记为 $\text{fl}(a \odot b)$. 这里的 \odot 表示加、减、乘、除四种运算符.

在不考虑溢出的情况下, 我们有

$$\text{fl}(a \odot b) = (a \odot b)(1 + \delta) \quad \text{或} \quad \text{fl}(a \odot b) = \frac{a \odot b}{1 + \delta}, \quad (\text{A.1})$$

其中 δ 表示浮点运算的相对误差, 满足 $|\delta| \leq \varepsilon_u$. 公式 (A.1) 是分析浮点运算舍入误差的基础 (标



准模型) [68, page 40]. IEEE 浮点运算标准同时也指出, 对于开根号运算, 产生的误差同样也满足 (A.1).

一个比较有趣的问题是何时两个浮点数能进行精确的相减运算.

定理 A.1 (Ferguson, 1995) [68] 设浮点数 x 和 y 满足

$$e(x - y) \leq \min\{e(x), e(y)\} \quad (\text{A.2})$$

其中 $e(\cdot)$ 表示一个正规浮点数的指数. 则 $\text{fl}(x - y) = x - y$. (假定 $x - y$ 不会下溢)

这里只考虑浮点运算误差, 所以假定 x, y 都是计算机可以精确表示的. 事实上, 由 (A.2) 可知, x 和 y 的指数至多差 1.

上面的定理对任何基都成立. 如果基是 2, 则有下面的结论.

推论 A.2 (Sterbenz, 1974) 设浮点数 x 和 y 满足

$$y/2 \leq x \leq 2y,$$

则 $\text{fl}(x - y) = x - y$. (假定 $x - y$ 不会下溢)

下面的结论对估计浮点运算的舍入误差非常有用 [68, page 63].

引理 A.3 设 $|\delta_i| \leq \varepsilon_u$ 且 $n\varepsilon_u < 1$, 则

$$\prod_{i=1}^n (1 + \delta_i)^{\rho_i} = 1 + \theta_n, \quad |\theta_n| \leq \gamma_n \triangleq \frac{n\varepsilon_u}{1 - n\varepsilon_u},$$

其中 $\rho_i = \pm 1$.

例 A.9 (多项式运算的舍入误差分析) : 已知多项式 $p(x) = \sum_{k=0}^n a_k x^k$. 对于给定的 x , 分析计算 $p(x)$ 的值时的舍入误差.

解. 在对多项式 $p(x) = a_n x^n + a_{n-1} x^{n-1} \dots + a_1 x + a_0$ 求值时, 我们通常采用 Horner 法则.

算法 A.1. Horner 法则

```

1:  $p = a_n$ 
2: for  $i = n - 1 : -1 : 0$  do
3:    $p = x * p + a_i$ 
4: end for

```

先看一个具体的例子. 设 $p(x) = (x - 2)^9$, 观测 $x = 2$ 附近的舍入误差. 我们通过画图来显示误差情况 (见图 A.2). 观察发现, 用 Horner 法则求值时会在 $x = 2$ 附近会出现“噪声带”.

下面我们基于舍入误差分析模型 (A.1) 来分析多项式求值的误差情况. 带舍入误差的 Horner 算法可写为

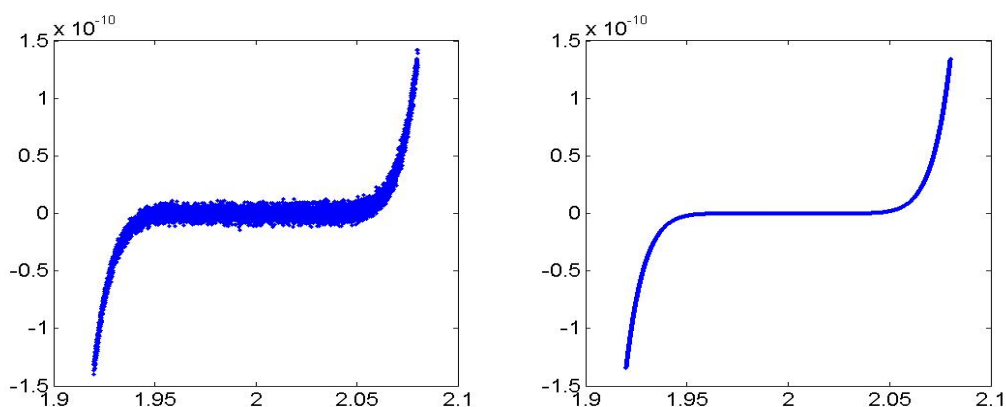


图 A.2. 分别利用 Horner 法则（左图）和 $y = (x - 2)^9$ （右图）在区间 $[1.92, 2.08]$ 上的 8000 个等分点上求值的结果

算法 A.2. 带舍入误差的 Horner 法则

```

1:  $p = a_n$ 
2: for  $i = n - 1 : -1 : 0$  do
3:    $p = ((x * p)(1 + \delta_i) + a_i)(1 + \tilde{\delta}_i)$    %  $|\delta_i| \leq \varepsilon_u, |\tilde{\delta}_i| \leq \varepsilon_u$ 
4: end for

```

所以得到的最终计算结果为

$$\text{fl}(p(x)) = \sum_{i=0}^{n-1} \left((1 + \tilde{\delta}_i) \prod_{j=0}^{i-1} (1 + \delta_j)(1 + \tilde{\delta}_j) \right) a_i x^i + \left(\prod_{j=0}^{n-1} (1 + \delta_j)(1 + \tilde{\delta}_j) \right) a_n x^n. \quad (\text{A.3})$$

假定 $j \cdot \varepsilon_u \ll 1$, 则有

$$\begin{aligned} (1 + \delta_1) \cdots (1 + \delta_j) &\leq (1 + \varepsilon_u)^j \lesssim 1 + j \cdot \varepsilon_u, \\ (1 + \delta_1) \cdots (1 + \delta_j) &\geq (1 - \varepsilon_u)^j \geq 1 - j \cdot \varepsilon_u. \end{aligned}$$

于是 (A.3) 可改写为

$$\text{fl}(p(x)) = \sum_{i=0}^n (1 + \hat{\delta}_i) a_i x^i \triangleq \sum_{i=0}^n \tilde{a}_i x^i, \quad \text{其中 } |\hat{\delta}_i| \leq 2n \cdot \varepsilon_u. \quad (\text{A.4})$$

所以 $\text{fl}(p(x))$ 可看作是另一个多项式的精确值, 且这个多项式的系数是原多项式系数的一个小的扰动. 这说明多项式计算的 Horner 算法是向后稳定的, 且系数的向后相对误差不超过 $2n \cdot \varepsilon_u$. 因此, 绝对误差为

$$\begin{aligned} |\text{fl}(p(x)) - p(x)| &= \left| \sum_{i=0}^n \tilde{a}_i x^i - \sum_{i=0}^n a_i x^i \right| \\ &= \left| \sum_{i=0}^n \hat{\delta}_i a_i x^i \right| \leq \sum_{i=0}^n |\hat{\delta}_i a_i x^i| \leq 2n \cdot \varepsilon_u \sum_{i=0}^n |a_i x^i|. \end{aligned}$$



相对误差为

$$\frac{|\text{fl}(p(x)) - p(x)|}{|p(x)|} \leq 2n \cdot \varepsilon_u \frac{\sum_{i=0}^n |a_i x^i|}{\left| \sum_{i=0}^n a_i x^i \right|}.$$

□

A.5 课后习题

练习 1.1 考虑抽样数据 x_1, x_2, \dots, x_n , 其均值为 $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$, 样本方差为 (无偏估计)

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

试证明:

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

对于数值计算, 上述两种计算方法哪一个更可靠, 为什么?

练习 1.2 根据浮点运算的舍入误差模型 (A.1), 在不考虑溢出的情况下, 证明 [68]

$$\text{fl} \left(\sum_{i=1}^n x_i y_i \right) = \sum_{i=1}^n x_i y_i (1 + \theta_i), \quad \text{其中 } |\theta_i| \leq \gamma_i \triangleq \frac{i\varepsilon_u}{1 - i\varepsilon_u}, \quad i = 1, 2, \dots, n.$$

