数值计算的可靠性

Michal Křížek　张智民

谨以此文祝贺 Ivo Babuška 教授 90 岁寿辰

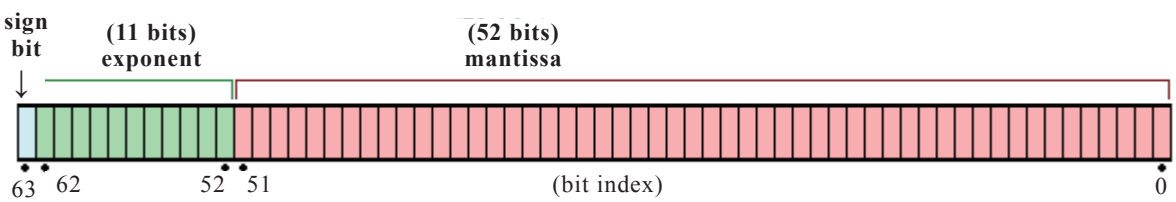
引言

首先，我们介绍一下使用计算机进行数值计算的几个基本术语。众所周知，计算机使用 2 进制表示数字，通常分为 32 位（bit）和 64 位两种。如果是 32 位，有一位表示符号，7 位用来记录方幂，剩下 24 位用于分数，这样一来，每个计算机表达的数和原先给的数之间的精度为 2 的负 25 次方，大约是 8 位十进制小数，这就是常说的“单精度”。如果是 64 位，有一位表示符号，11 位用来记录方幂，剩下 52 位用于分数，这样以来，精度为 2 的负 53 次方，大约是 16 位十进制小数，这就是“双精度”。所谓浮点运算，都是将每一个参与计算的数字表示成 32 或 64 位的浮点数来进行的。不难看出，使用单精度和双精度，计算过程产生的舍入误差是不一样的。

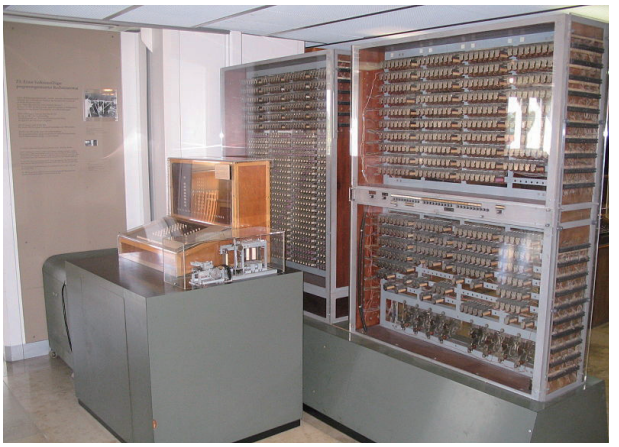
其次，读者还需了解目前广泛使用的两个商业数学和工程计算软件：MATLAB 和 Maple。MATLAB 是矩阵实验室（Matrix Laboratory）的简称，由美国 MathWorks 公司出品，它将数值分析、矩阵计算、科学数据可视化以及非线性动态系统的建模和仿真等诸多强大功能集成在一个易于使用的视窗环境中，在工业设计、科学计算以及课堂教学等方面都有非常广泛的应用。相比于 MATLAB，Maple 的特点是符号运算，其功能覆盖诸多的数学领域，如微积分、线性代数、微分方程、积分变换、概率论和数理统计、图论、张量分析、微分几何、线性规划、组合数学、抽象代数、泛函分析、数论、复分析和实分析、特殊函数、编码和密码理论、优化等。与历史更久的 Fortran 和 C 等计算机语言相比，MATLAB 和 Maple 是更高级、更加人性化的语言，其语法相对简单，经常是直接输入数学公式即可达到计算目的。

有了这些准备，现在言归正传。对下面这个定积分

$$I_n = \frac{1}{\mathrm{e}} \int_0^1 x^n \mathrm{e}^x dx$$



浮点数



第一台可编程计算机 Z3，包括浮点运算（在慕尼黑的德意志博物馆展出的复制品）

分步积分，我们可以得到以下的递推公式

$$I_n = 1 - nI_{n-1}, \quad n = 1, 2, \cdots$$

这里 $I_0 = 1 - \mathrm{e}^{-1}$。Renata Babušková[1] 早在 1964 年就曾讨论这个迭代的数值稳定性。以单精度计算，几步递推之后就出现了负值。在今天的标准 PC 上用 MATLAB 计算（双精度），我们观测到最初的一个递减序列直到 $I_{16} = 0.0374$。然而，随后的结果却令人惊愕：

$$I_{17} = 0.3259, \quad I_{18} = -5.1930, \quad I_{19} = 104.8628, \cdots,$$

我们看到一个迅速发散的交错序列。究其原因，最初的每一步，两个非常相近的数相减，只有很少几位有效数字；而这个数值误差又被其后的迭代以每步 n 倍的速度放大，从而产生了一个迅速发散的序列。

如果我们改变策略，用以下快速收敛的级数计算

$$I_n = \frac{1}{n+1} - \frac{1}{(n+1)(n+2)} + \frac{1}{(n+1)(n+2)(n+3)} - \cdots$$

则可以得到任意精度的逼近值。

类似的例子不胜枚举，我们可以在 1966 年的一本书中[2] 找到下面的例子：

$$\cdots (((((1 \div 2) \cdot 2) \div 3) \cdot 3) \div 4) \cdot 4 \cdots$$

不断进行先除后乘的运算。Karel Segeth 在不同的计算机上就这个例子进行了成千上万次乘除法运算，得到了许多有趣的结果。

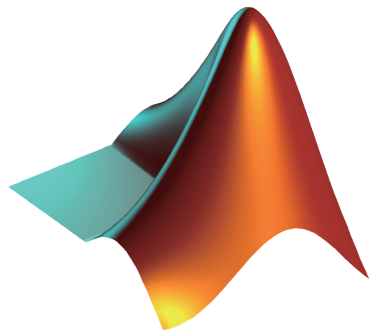
基于这样一个开场白，此文通过几个精选的例子警醒世人，在解读计算机得到的数值结果时，须要格外谨慎。

一个递减序列会被浮点运算变为递增序列吗？

[1] R. Babušková: *Über numerische Stabilität einiger Rekursionsformeln*. Apl. Mat. 9 (1964), 186–193.

[2] I. Babuška, M. Práger, E. Vitásek: *Numerical Processes in Differential Equations*. John Wiley & Sons, London, New York, 1966.

Maple 的标志



MATLAB 的标志

取 $a_0 = 1, a_1 = 11^{-1}$，开始以下递推：

$$a_{n+2} = \frac{34}{11}a_{n+1} - \frac{3}{11}a_n, \qquad n = 0,1,2,\cdots$$

准确解 $a_n = 11^{-n}$ 是一个单调递减序列。然而，使用 MATLAB 在标准 PC 上计算，我们只能看到递减序列到 $a_{12} = 0.2068 \cdot 10^{-11}$ 为止，随之序列开始递增，到 $a_{40} = 40.44$，然后迅速增长。这个例子说明即便很小的两个相近的数相减也会带来大麻烦。有趣的是，用以下稍加变化的递推公式，

$$a_{n+2} = \frac{1}{11}(34a_{n+1} - 3a_n), \quad n = 0,1,2,\cdots$$

再用 MATLAB 计算，可以看到序列从 a_{12} 开始变为负值，但绝对值迅速增长。

这里我们给出序列中第 50 个数在不同计算精度下的值：

$a_{50} \approx 2 \cdot 10^{11}$ 单精度，序列在 $n = 8$ 变为递增序列；

$a_{50} \approx 10^{10}$ 实值精度（相当于 12 位十进制精度），序列在 $n = 9$ 开始递增；

$a_{50} \approx 5 \cdot 10^{5}$ 双精度，序列在 $n = 12$ 开始递增；

$a_{50} \approx 10^{3}$ 扩展精度（相当于 20 位十进制精度），序列在 $n = 14$ 开始递增。

Babuška 的例子

考虑定义在 [0, 1] 区间上的狄利克雷函数 f：在有理点 $f = 0$，而在无理点 $f = 1$。由于这个区间上所有无理数组成的集合的勒贝格测度为 1，f 的勒贝格积分

$$\int_0^1 f(x)dx = 1,$$

然而采用任何精度的数值积分都将得到

$$\sum_{i=1}^{n} w_i f(x_i) = 0,$$

因为由浮点运算产生的 x_i 值全部为有理数。

这个例子比较极端，数值积分的误差永远为 1，原因在于 f 不是一个光滑函数。而数值积分收敛理论总是要求被积函数 f 具有一定的光滑性。

Muller 的例子 [3]

取 $a_1 = e - 1 = 1.718281828\cdots$ 产生一个表面看来简单而又平常的序列：

$$a_n = n(a_{n-1} - 1), \quad n = 2, 3, \cdots \tag{1}$$

用数学归纳法可以得到：

$$a_n = n!\left(\frac{1}{n!} + \frac{1}{(n+1)!} + \frac{1}{(n+2)!} + \cdots\right).$$

显然 $\{an\}$ 是一个极限为 1 的递减序列。使用 MATLAB 数值实现 (1)，开始我们的确看到递减，但好景不长，从 $a_{15} = 1.0668$ 起，奇怪的事发生了：

$$a_{16} = 1.0685, \quad a_{17} = 1.1652, \quad a_{18} = 2.9731, \quad a_{19} = 37.4882, \quad a_{20} = 729.7637, \cdots$$

以后的序列数迅速膨胀。

下面我们记录了序列中第 25 号元素在不同运算精度的数值：

$a_{25} \approx -1.306946321 \cdot 10^{13}$ 实值精度，
$a_{25} \approx +1.201807248 \cdot 10^{9}$ 双精度，
$a_{25} \approx -7.3557319606 \cdot 10^{6}$ 扩展精度。

现在我们来看看提高计算有效位数 D（十进制）的效果：

D	$a_{25}(D)$
20	615990.413139
21	–4457.98859386
22	–4457.98859386
23	195.374419140
24	40.2623187072
25	–6.27131142281
26	1.48429359885

注意，对应于 $D = 21$ 和 $D = 22$ 的值相等，原因是欧拉数 e 的第 21 位是 0。当 D 增加到 26 以后，我们得到的计算值开始“靠谱”——接近真值 $a_{25} = 1.03993872967\cdots$ 比如 $a_{25}(30) = 1.039897\cdots$

让我们进一步探讨一下这一奇怪现象的原因。以 ε_i 代表第 i 步的截断误差，得到

$\tilde{a}_1 = e - 1 + \varepsilon_1 = a_1 + \varepsilon_1$,
$\tilde{a}_2 = 2(\tilde{a}_1 - 1) + \varepsilon_2 = 2(a_1 + \varepsilon_1 - 1) + \varepsilon_2 = a_2 + 2!\varepsilon_1 + \varepsilon_2$,
$\cdots$
$\tilde{a}_{25} = 25(\tilde{a}_{24} - 1) + \varepsilon_{25} = a_{25} + 25!\varepsilon_1 + \frac{25!}{2!}\varepsilon_2 + \frac{25!}{3!}\varepsilon_3 + \cdots + \varepsilon_{25}$.

因而，累计截断误差

$$a_{25} - \tilde{a}_{25} = -25!\left(\varepsilon_1 + \frac{1}{2!}\varepsilon_2 + \frac{1}{3!}\varepsilon_3 + \cdots + \frac{1}{25!}\varepsilon_{25}\right),$$

它的大小特别依赖于最初的几个截断误差 ε_1，ε_2，…这个例子再一次说明在数值计算中两个非常接近的数相减是多么的危险。

[3] J.-M. Muller et al.: *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2009.