数值分析

钱江(jqian104@gmail.com)

第一章　绪论

1　数值分析研究对象与特点

数值分析也称为计算方法，是计算数学的一个主要部分. 计算数学是数学科学的一个分支，主要研究用计算机求解各种数学问题的数值计算方法及其理论与软件实现.

数值分析的内容包括函数的数值逼近、数值微分与数值积分、非线性方程数值解、数值线性代数、常微和偏微数值解等.

虽然数值分析也是以数学问题为研究对象，但它不像纯数学那样只研究数学本身的理论，而是把理论与计算紧密结合，着重研究数学问题的数值方法及其理论.

数值分析不是各种数值方法的简单罗列和堆积，是一门内容丰富，研究方法深刻，有自身理论体系的课程.

数值分析既有纯数学的高度抽象性与严密科学性的特点，又有应用数学的广泛性与实际试验的高度技术性的特点，是一门与计算机使用密切结合的实用性很强的数学课程.

数值分析的特点：

1. 面向计算机，能根据计算机的特点提供切实可行的有效算法.
2. 有可靠的理论分析，能任意逼近并达到精度要求，对近似算法要保证收敛性和数值稳定性，还要对误差进行分析.
3. 要有好的计算复杂性，时间复杂性好是指节省时间，空间复杂性好是指节省存储量，这也是建立算法要研究的问题，它关系到算法能否在计算机上实现.
4. 要有数值实验，即任何一个算法除了从理论上要满足上述三点外，还要通过数值试验证明是行之有效的.

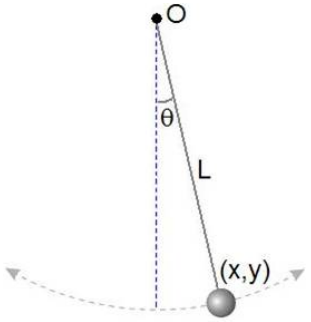
2　数值计算的误差

2.1　误差来源与分类

用计算机解决科学计算问题的过程如下：首先要建立数学模型，它是对被描述的实际问题进行抽象、简化而得到的，因而是近似的. 数学模型与实际问题之间出现的误差称为**模型误差**.

如单摆的振动方程为:

$$\theta'' = -\frac{g}{l}\sin(\theta)$$



当θ很小时, 我们往往会采用如下的更简单的数学模型:

$$\theta'' = -\frac{g}{l}\theta.$$

在数学模型中往往还有一些根据观测得到的物理量，如温度、长度、电压等等，这些量显然也包含误差. 这种由观测产生的误差称为**观测误差**.

以上两种误差不在“数值分析”的讨论范围. 数值分析只研究用数值方法求解数学模型产生的误差. 当数学模型不能得到精确解时，通常要用数值方法求它的近似解. 近似解与精确解之间的误差称为**截断误差**或**方法误差**.

由Taylor公式

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!} + \frac{1}{(n+1)!}x^{n+1}e^p,$$

其中$p \in (0, x)$. 如果我们用$\sum_{i=0}^{n} \frac{x^i}{i!}$作为$e^x$的近似, 则其截断误差为$\frac{1}{(n+1)!}x^{n+1}e^p$.

有了计算公式后，在用计算机做数值计算时，还要受计算机字长的限制，原始数据在计算机上表示会产生误差. 计算过程又可能产生新的误差，这种误差称为**舍入误差**. 例如用3.14159来近似π, 产生的误差$R = \pi - 3.14159 \approx 0.0000026...$ 就是舍入误差.

此外由原始数据或机器中的十进制数转化为二进制数产生的初始误差对数值计算也将造成影响. 分析初始数据的误差通常也归结为舍入误差.

研究计算结果的误差是否满足精度要求就是误差估计问题.

2.2 误差与有效数字

定义1: 设x为精确值, x^*是x的一个近似值, 则称

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

为近似值x^*的**绝对误差**, 简称**误差**.

通常准确值x未知, 因此误差也未知. 若能根据测量工具或计算情况估计出误差绝对值的一个上界，即

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

则称ε^*是近似值x^*的**误差限**, 它总是正数.

例如，用毫米刻度的米尺测量一长度x, 读出和该长度接近的刻度x^*, 它是x的近似值, 其误差限是$0.5mm$, 即

$$|x^* - x| \leq 0.5mm.$$

如读出的长度为$100mm$, 则$|100 - x| \leq 0.5mm$. 虽然从这个不等式不能知道准确的x是多少，但可知

$$99.5mm \leq x \leq 100.5mm.$$

对于一般的$|x^* - x| \leq \varepsilon^*$, 有$x^* - \varepsilon^* \leq x \leq x^* + \varepsilon^*$, 也可表示为

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

但要注意的是，误差限的大小并不能完全表示近似值的好坏. 除考虑误差的大小外，还应考虑准确值x本身的大小.

定义2: 称近似值x^*的误差e^*与准确值x的比值

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

为近似值x^*的**相对误差**, 记作e_r^*.

实际计算中, 由于真值x未知, 通常取

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

作为x^*的相对误差. 它的绝对值的上界叫做**相对误差限**, 记作ε_r^*.

对于任意给定的自然数$N \geq 2$, 可以证明任意正实数a都可以表示为

$$a = a_m N^m + a_{m-1} N^{m-1} + ... + a_1 N^1 + a_0 N^0 + a_{-1} N^{-1} + a_{-2} N^{-2} + ...$$

其中

$$0 \leq a_i \leq N - 1, \quad a_m \neq 0,$$

或等价的表示为:

$$a = \sum_{i=0}^{\infty} a_{m-i} N^{m-i}$$

其中

$$0 < a_m < N, \quad 0 \leq a_{m-i} < N, \quad \forall i \geq 1.$$

而计算机只能处理有限位的表示, 即

$$\alpha = \sum_{i=0}^{n-1} \alpha_{m-i} N^{m-i},$$

其中$0 < \alpha_m < N$, $0 \leq \alpha_{m-i} < N, (i = 1, 2, ..., n-1)$. 当$N = 10$时, 这就是十进制表示,数$\alpha_{m-i}$, $0 \leq i \leq n-1$ 称作**有效数字**, 并称实数α有n位有效数字.

例1 数52000.25有7位有效数字.

$$0.0000235 = 2(10)^{-5} + 3(10)^{-6} + 5(10)^{-7}$$

只有3位有效数字, 而

$$\begin{aligned} 0.2350000 &= 2(10)^{-1}+3(10)^{-2}+5(10)^{-3}+0(10)^{-4} \\ &\quad +0(10)^{-5}+0(10)^{-6}+0(10)^{-7} \end{aligned}$$

有7位有效数字.

如果实数a的有效数字的位数超过实际处理能力或需要, 我们可以用一个更少有效位数,比如说n位, 的数a^*来近似a. 一般来说有两种方式: **截断**或**四舍五入**.

截断是指只保留数a的前n位有效数字.

例2 对

$$e^2 = 7.38905609...$$

进行截断, 分别保留二,三,五,七位有效数字, 则得到

$$7.3,\ \ 7.38,\ \ 7.3890 \ \text{ and } \ 7.389056.$$

四舍五入是指保留前n位有效数字, 如果第$n+1$位有效数字小于5, 则第n位有效数字不变, 否则第n位有效数字加一.

例3 对数

$$e^2 = 7.38905609...$$

进行四舍五入, 分别保留二,三,五,七位有效数字, 则得到

$$7.4,\ \ 7.39,\ \ 7.3891 \ \text{ and } \ 7.389056.$$

2.3 数值计算的误差估计

两个近似数x_1^*, X_2^*, 其误差限分别为$\varepsilon(x_1^*), \varepsilon(x_2^*)$, 它们进行加减乘除的误差限分别为:

$$\begin{aligned} &\varepsilon(x_1^* \pm x_2^*) = \varepsilon(x_1^*) + \varepsilon(x_2^*), \\ &\varepsilon(x_1^* x_2^*) \approx |x_1^*|\varepsilon(x_2^*) + |x_2^*|\varepsilon(x_1^*), \\ &\varepsilon(x_1^*/x_2^*) \approx \frac{|x_1^*|\varepsilon(x_2^*) + |x_2^*|\varepsilon(x_1^*)}{|x_2^*|^2} \quad (x_2^* \neq 0). \end{aligned}$$

一般情况下, 当自变量有误差时函数值也产生误差, 其误差限可利用函数的泰勒展开式进行估计. 设$f(x)$是一元函数, x的近似值为x^*, 用$f(x^*)$来近似$f(x)$, 其误差界记为$\varepsilon(f(x^*))$. 利用Taylor展开

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

取绝对值有

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

假定$f'(x^*)$和$f''(x^*)$的比值不大, 忽略$\varepsilon(x^*)$的高阶项, 则有

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

当f是多元函数时, 用$f(x_1^*,\ldots,x_n^*)$来近似$f(x_1,\ldots,x_n)$的误差$e(f(x_1^*,\ldots,x_n^*))$为

$$e(f(x_1^*,\ldots,x_n^*)) = f(x_1^*,\ldots,x_n^*) - f(x_1,\ldots,x_n) \approx \sum_{k=1}^{n}\left(\frac{\partial f(x_1^*,\ldots,x_n^*)}{\partial x_k}\right),$$

由此可以得到一个近似误差限.

3 误差分析和避免误差伤害

Some disasters caused by numerical errors

(1) Patriot Missile Failure

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks and killed 28 soldiers.

The cause was an inaccurate calculation of the time since boot due to computer arithmetic errors. Specifically, the time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds. This calculation was performed using a 24 bit fixed point register. In particular, the value 1/10, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point. The small chopping error, when multiplied by the large number giving the time in tenths of a second, lead to a significant error.

The binary expansion of 1/10 is

0.00011001100110011001100110011001100....

Now the 24 bit register in the Patriot stored instead

0.00011001100110011001100

introducing an error of 0.00000000000000000000000011001100... binary, or about 0.000000095 decimal. The Patriot battery had been up around 100 hours. Multiplying by the number of tenths of a second in 100 hours gives 0.000000095100606010=0.34 seconds. A Scud travels at about 1,676 meters per second, and so travels more than half a kilometer in this time. This was far enough that the incoming Scud was outside the "range gate" that the Patriot tracked.

(2) Explosion of the Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after lift-off. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million.

The cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,768, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.

(3) The Vancouver Stock Exchange

In 1982 the Vancouver Stock Exchange instituted a new index initialized to a value of 1000.000. The index was updated after each transaction. Twenty two months later it had fallen to 520. The cause was that the updated value was truncated rather than rounded. The rounded calculation gave a value of 1098.892.

(4) The sinking of the Sleipner A offshore platform

The Sleipner A platform produces oil and gas in the North Sea and is supported on the

seabed at a water depth of 82 m. It is a Condeep type platform with a concrete gravity base structure consisting of 24 cells and with a total base area of 16000m2. Four cells are elongated to shafts supporting the platform deck. The first concrete base structure for Sleipner A sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway on 23 August 1991.

The top deck weighs 57,000 tons, and provides accommodation for about 200 people and support for drilling equipment weighing about 40,000 tons. When the first model sank in August 1991, the crash caused a seismic event registering 3.0 on the Richter scale, and left nothing but a pile of debris at 220m of depth. The failure involved a total economic loss of about $700 million.

The 24 cells and 4 shafts referred to above are shown in Figure 1 while at the sea surface.

Figure 1

The cells are 12m in diameter. The cell wall failure was traced to a tricell, a triangular concrete frame placed where the cells meet, as indicated in the Figure 2 below.

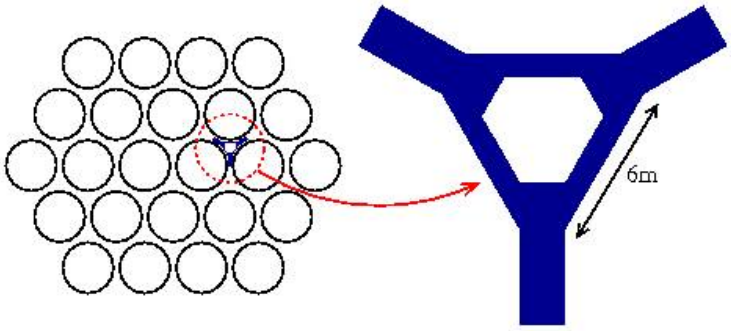


Figure 1

The post accident investigation traced the error to inaccurate finite element approximation of the linear elastic model of the tricell (using the popular finite element program NASTRAN). The shear stresses were underestimated by 47%, leading to insufficient design. In particular, certain concrete walls were not thick enough. More careful finite element analysis, made after the accident, predicted that failure would occur with this design at a depth of 62m, which matches well with the actual occurrence at 65m.

假设输入数据为$x_1^*, \ldots, x_n^*$, 它们是真实输入数据$x_1, \ldots, x_n$的近似, 通过某种算法计算出的输出数据为y^*, 把它作为真实输出数据y的近似.

向前误差分析是指估计y^*作为y的近似的相对或绝对误差限, 把它们表示为输入数据的相对或绝对误差限的函数, 即直接估计输出数据的误差大小.

向后误差分析则是基于如下的看法: 计算出的输出数据不是原问题的精确解, 但它可以看成是一些与原问题很接近的问题的精确解. 向前误差分析就是要估计计算出的输出数据是一个跟原问题多接近的问题的精确解.

3.1 病态问题与条件数

对一个数值问题本身, 如果输入数据有微小扰动（即误差）, 引起输出数据相对误差很大, 这就是病态问题.

例如计算函数值$f(x)$时, 如果x有扰动$\Delta x = x - x^*$, 其相对误差为$\frac{\Delta x}{x}$, 函数值$f(x^*)$的相对误差为$\frac{f(x)-f(x^*)}{f(x)}$. 相对误差的比值

$$\left|\frac{f(x) - f(x^*)}{f(x)}\right| / \left|\frac{\Delta x}{x}\right| \approx \left|\frac{xf'(x)}{f(x)}\right| = C_p.$$

C_p成为计算函数值问题的**条件数**.

自变量相对误差一般不会太大, 如果条件数C_p 很大, 将引起函数值相对误差很大, 出现这种情况的问题就是病态问题. 例如对于$f(x) = x^n$, 则有

$$C_p = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x\dot{n}x^{n-1}}{x^n}\right| = n,$$

它表示相对误差可能会被放大n倍.

如$n=10$, 取$x=1, x^*=1.02$, 则有$f(1)=1, f(1.02)\approx 1.24$. 自变量的相对误差为2%, 而函数值的相对误差为24%.

条件数C_p如果很大就认为问题是病态的, C_p越大病态越严重.

再例如, 对于线性方程组

$$\begin{bmatrix} 1.0001 & 0.9999 \\ 0.9999 & 1.0001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

其真解为$[1,1]^T$. 若右端项变为

$$\begin{bmatrix} 2.0001 \\ 1.9999 \end{bmatrix},$$

则真解变为$[1.5, 0.5]^T$. 若系数矩阵变为

$$\begin{bmatrix} 1.0001 & 1 \\ 0.9999 & 1 \end{bmatrix},$$

则真解变为$[0,2]^T$. 注意这里的真解是指数学意义上的精确解, 与使用的算法无关. 可以看出对于这个问题, 系数矩阵或右端项的微小扰动都有可能引起解的巨大变化, 这表明这个问题是病态的.

问题是否病态是问题本身的固有特性, 与求解这个问题的算法无关.

3.2 算法的数值稳定性

用一个算法进行计算，如果初始数据误差在计算中传播使计算结果的误差增长很快，这个算法就是**数值不稳定**的.

例4 下面两个函数显然数学上是等价的:

$$f(x) = x(\sqrt{x+1} - \sqrt{x}) \quad \text{and } g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

取$x=1000$, 则$f(1000)$的真实值15.807437...假设使用的计算机只有4位有效数字, 则有

$$\begin{aligned} f(1000) &= 1000(\sqrt{1001} - \sqrt{1000}) \\ &= 1000(31.6386 - 31.6228) = 1000(0.0158) = 15.8000, \\ g(1000) &= \frac{1000}{31.6386 - 31.6228} = \frac{1000}{63.2614} = 15.8074. \end{aligned}$$

例5 考虑计算积分:

$$y_n = \int_0^1 \frac{x^n}{a+x}\,\mathrm{d}x \qquad a > 1.$$

由

$$\begin{aligned} y_n &= \textstyle\int_0^1 \frac{x^n}{a+x}\,\mathrm{d}x \\ &= \textstyle\int_0^1 \frac{x^{n-1}(x+a-a)}{x+a}\,\mathrm{d}x \\ &= \textstyle\int_0^1 x^{n-1}\,\mathrm{d}x - a\int_0^1 \frac{x^{n-1}}{x+a}\,\mathrm{d}x, \end{aligned}$$

可以得到一个递推式

$$y_n \;=\; \frac{1}{n} \;-\; a y_{n-1} \qquad n = 1, 2, \ldots .$$

$y_0 \;=\; \ln\frac{1+a}{a}$容易算出, 因此理论上可以用上式递推的计算出所有的$y_1, y_2, y_3, \ldots$. 但实际上, 上面的递推关系式是非常不稳定的.

由于$y_0 \;=\; \ln\frac{1+a}{a}$不能精确计算, 假设$y_0$有一个误差$\varepsilon_0$ (比如舍入误差). 记$\tilde{y}_n$为计算出的y_n, 则有

$$\begin{aligned} \tilde{y}_0 \quad &= \quad y_0 \;+\; \varepsilon_0 \\ \\ \tilde{y}_n \quad &= \quad -a\tilde{y}_{n-1} \;+\; \tfrac{1}{n} \qquad n = 1, 2, \ldots . \end{aligned}$$

因此若记$e_n \;=\; y_n - \tilde{y}_n$, 则它满足

$$\begin{aligned} e_n \;&=\; -a e_{n-1} \qquad\qquad n = 1, 2, \ldots , \\ \\ \Rightarrow \quad e_n \;&=\; (-a)^n \, \varepsilon_0 \qquad n = 0, 1, 2, \ldots . \end{aligned}$$

由于$a > 1$, 从上式可以看出在递推的每一步, 计算值与真实值之间的误差都会被放大a倍, 这回带来灾难性的后果.

事实上可以把上面的递推式换一种写法:

$$y_{n-1} \;=\; \frac{1}{a}\left(\frac{1}{n} \;-\; y_n\right) .$$

利用上式可以从y_n开始递推的计算$y_{n-1}, y_{n-2}, \ldots, y_1, y_0$. 这个递推式是稳定的, 因为如果假设$y_n$有$\varepsilon$的误差, 则递推的每一步会把误差缩小$1/a$倍. 这样经过足够多步的递推后, 原来的$\varepsilon$的误差就几乎可以忽略不计了.

注意到$\lim\limits_{n\to\infty} y_n \;=\; 0$, 我们从$y_{20} \;=\; 0$开始, 假设$a \;=\; 10$,则10步后有

$$\begin{aligned} y_{10} \quad &= \quad 0.008327966 \\ y_9 \quad &= \quad 0.009167203 \\ y_8 \quad &= \quad 0.010194391 \\ y_7 \quad &= \quad 0.011480561 \\ y_6 \quad &= \quad 0.013137658 \\ y_5 \quad &= \quad 0.015352901 \\ y_4 \quad &= \quad 0.018464710 \\ y_3 \quad &= \quad 0.023153529 \\ y_2 \quad &= \quad 0.031017980 \\ y_1 \quad &= \quad 0.046898202 \\ y_0 \quad &= \quad 0.095310180 \end{aligned}$$

显然开始的y_{20}包含误差, 但这些计算值与精确值完全相同.

例6 考虑计算$\sqrt{2} \approx 1.41421$.

一种方法是著名的Babylonian方法, 其迭代式为:

$$x_{k+1} = \frac{x_k}{2} + \frac{1}{x_k}.$$

另一种迭代式为:

$$x_{k+1} = (x_k^2 - 1)^2 + x_k.$$

我们称之为X方法. 下表给出了分别取初值$x_1 = 1.4$和$x_1 = 1.42$时用两种方法的计算结果:

Babylonian方法	Babylonian方法	X方法	X方法
$x_1 = 1.4$	$x_1 = 1.42$	$x_1 = 1.4$	$x_1 = 1.42$
$x_2 = 1.4142857$	$x_2 = 1.41422535$	$x_2 = 1.4016$	$x_2 = 1.42026896$
$x_3 = 1.414213564$	$x_3 = 1.4142135624$	$x_3 = 1.4028614$	$x_3 = 1.42056$
		$\cdots$	$\cdots$
		$x_{1000000} = 1.41421$	$x_{28} = 7280.2284$

一个算法是否数值稳定是算法本身的固有特性, 与问题是否病态无关. 一般来说, 只有用一个数值稳定的算法去计算一个良态的问题才能期望得到可靠的计算结果.

3.3 避免误差危害的若干原则

数值计算中首先要分清问题是否病态和算法是否数值稳定，计算时还应尽量避免误差危害，防止有效数字的损失，有下面若干原则.

(1) 要避免大数除以小数.

用很大的数去除以很小的数, 由于计算结果很大, 这样很容易带来很大的绝对误差.

例7 考虑求解线性方程组

$$\begin{cases} 0.000001x_1 + x_2 = 1, \\ 2x_1 + x_2 = 2. \end{cases}$$

其精确解为

$$x_1 = \frac{200000}{399999} = 0.50000125, \quad x_2 = \frac{199998}{199999} = 0.999995.$$

假设计算机只有4位精度, 用消去法求解上述方程. 方程写为:

$$\begin{cases} 0.1000 \times 10^{-4} x_1 + 0.1000 \times 10^{1} x_2 = 0.1000 \times 10^{1}, \\ 0.2000 \times 10^{1} x_1 + 0.1000 \times 10^{1} x_2 = 0.2000 \times 10^{1}. \end{cases}$$

如果第一个方程除以$\frac{1}{2}(0.1000 \times 10^{-4})$再减去第二个方程, 则可得

$$\begin{cases} 0.1000 \times 10^{-4} x_1 + 0.1000 \times 10^{1} x_2 = 0.1000 \times 10^{1}, \\ 0.2000 \times 10^{6} x_2 = 0.2000 \times 10^{6}. \end{cases}$$

由此解出$x_2 = 1, x_1 = 0$. 显然与真解差别很大.

如果交换两个方程的位置, 再类似消去x_1可得

$$\begin{cases} 0.2000 \times 10^{1} x_1 + 0.1000 \times 10^{1} x_2 = 0.2000 \times 10^{1}, \\ 0.1000 \times 10^{6} x_2 = 0.1000 \times 10^{6}. \end{cases}$$

由此得到很到的近似解$x_2 = 1, x_1 = 0.5$.

(2) 要避免两个相近的数相减.

在数值计算时两个相近的数相减会使得有效数字严重损失. 例如$x = 532.65$和$y = 532.52$都有五位有效数字, 但是$x - y = 0.13$就只有两位有效数字. 此时最好是改变计算方法, 防止这种现象发生.

例8 考虑计算

$$\sqrt{9876} - \sqrt{9875}.$$

若直接计算, 则有

$$\begin{aligned}\sqrt{9876} &= 0.9937806599 \times 10^2\\ \sqrt{9875} &= 0.9937303457 \times 10^2\end{aligned}$$

$$\sqrt{9876} - \sqrt{9875} = 0.0000503142 \times 10^2 = 0.5031420000 \times 10^{-2}.$$

这里后面的4个零是不正确的.

我们可以换一种方式计算

$$\sqrt{9876} - \sqrt{9875} = \frac{1}{\sqrt{9876}+\sqrt{9875}} = 0.5031418679 \times 10^{-2},$$

这个结果的每个数字都是精确的.

一般来说, 当$|\varepsilon| \ll x$时, 应采用

$$\sqrt{x+\varepsilon} - \sqrt{x} =: \frac{x+\varepsilon-x}{\sqrt{x+\varepsilon}+\sqrt{x}} = \frac{\varepsilon}{\sqrt{x+\varepsilon}+\sqrt{x}}$$

中右边的式子来计算左边.

如果无法改变算式, 则可以采用增加有效位数进行计算, 但这要增加计算时间和存储空间.

(3) 要防止大数吃掉小数

在数值运算中参加运算的数有时数量级相差很大, 而计算机位数有限,如不注意运算次序就可能出现大数“吃掉”小数的现象,影响计算结果的可靠性.

例9 在五位十进制计算机上计算

$$A = 52492 + \sum_{i=1}^{1000} \delta_i,$$

其中$0.1 \le \delta_i \le 0.9$.

若把求和中每一个数依次加到52492, 则由于

$$52492 + 0.9 = 0.52494 \times 10^5 + 0.000009 \times 10^5,$$

后者在五位十进制计算机上会被看成0, 这样就被大数吃掉了.

可以先把数量级相同的一千个δ_i相加, 最后再加到52492上, 就可以避免这种情况.

(4) 注意简化计算步骤，减少运算次数.

同样一个计算问题, 如果能减少运算次数, 不但可节省计算机的计算时间, 还能减少舍入误差.

例10 计算多项式

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

的值, 若直接计算$a_k x^k$的值再相加, 一个需要$n(n+1)/2$次乘法和n次加法.

如果采用秦九韶算法

$$\begin{cases} S_n = a_n, \\ S_k = xS_{k+1} + a_k \ (k = n-1, n-2, \ldots, 1, 0), \\ P_n(x) = S_0, \end{cases}$$

则只需要n次乘法和n次加法.