中国科学: 信息科学 2016 年 第 46 卷 第 6 期: 698–713
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
论 文

算术表达式的一种可信计算算法及其软件 ISReal

赵世忠

华东师范大学上海市高可信计算重点实验室, 上海 200062
E-mail: szzhao@sei.ecnu.edu.cn

收稿日期: 2015–07–22; 接受日期: 2015–10–15; 网络出版日期: 2016–05–27
上海市高可信计算重点实验室开放课题 (批准号: 07dz22304201405) 和国家自然科学基金 (批准号: 11371143, 61321064, 61402537, 61572195) 资助项目

摘要 算术表达式的精确计算是确保各类复杂计算正确性的基础. 然而, 对于大多数主流软件来说, 计算过程中 "不正确的舍入误差" 与机器误差, 导致计算结果不稳定. 针对这种计算不稳定问题, 本文提出中间运算的精度自适应技术, 进而给出算术表达式的一种可信计算算法, 并用 C++ 实现软件 ISReal. 利用该算法 (软件), 不论计算模型是 "良态的" 还是 "病态的", 均可计算出其任意精度的值, 即计算结果始终与理论结果相一致.

关键词 误差可控计算 精确计算 算术表达式 可信计算 计算误差

1 引言

最近几十年, 随着科技的发展, 特别是计算机的广泛应用, 人类对计算的准确性与精确性要求越来越高 [1~10]. 但是, 对于由常数[1)]、加减乘除四则运算以及 (反) 三角、对数、指数等基本初等运算构成的算术表达式, 传统的 "有效位数" 计算模式存在一个问题: 同一个表达式, 不同 "有效位数" 下的计算结果不一致. 与此相对应, 许多软件留给用户 "太多" 的参数设置, 比如 Maple 的 Digits、Matlab 的 digits、Mathematica 的 $MaxExtraPrecision、Pari 的 \p 以及 Gmp 的 mpf_set_default_prec 等等. 对于同一表达式, 这些参数设置不同, 则会输出不同的结果. 由于在计算之前用户并不清楚如何设置它们 [11], 因此用户不得不对计算结果进行一些猜测工作 [12] ("The multiprecision ... in Mathematica and Maple is not very useful ..., because the working precision must be specified by the user and this naturally implies some guess work"). 于是, 误差的定量控制成为一个困难的问题 [13], 正如文献 [14] 的摘要中所说: "不幸的是, 绝大多数数值计算软件工具不能给用户提供任何精度保证 (Unfortunately, most numerical software tools do not provide any accuracy guarantee to the user)".

针对上述问题, 本文首先在第 2 节举例说明并剖析原因; 接着在第 3 节介绍算术表达式的一个可信计算算法及其软件 ISReal; 最后对全文进行总结.

1) 由 0~9 十个数字与小数点构成的位数有限的数.

引用格式: 赵世忠. 算术表达式的一种可信计算算法及其软件 ISReal. 中国科学: 信息科学, 2016, 46: 698–713, doi: 10.1360/N112015-00061

2 主流软件的错误计算

本节首先举例说明现有软件的计算不稳定性, 然后简述其原因.

例 2.1 已知

$$f(a,b)=333.75\,b^6+a^2\,(11\,a^2b^2-b^6-121\,b^4-2)+5.5\,b^8+\frac{a}{2\,b},\tag{1}$$

其中 $a=77617.0$, $b=33096.0$.

使用下列由 c 语言编制的程序计算上式的值 [15, 16].

```
#include < stdio.h >
int main(void)
{   double a = 77617.0;
    double b = 33096.0;
    double b2, b4, b6, b8, a2, firstexpr, f;
    b2 = b * b;
    b4 = b2 * b2;
    b6 = b4 * b2;
    b8 = b4 * b4;
    a2 = a * a;
    firstexpr = 11 * a2 * b2 - b6 - 121 * b4 - 2;
    f = 333.75 * b6 + a2 * firstexpr + 5.5 * b8 + (a/(2.0 * b));
    printf("Double precision result : $%1.17e\n", f);
}
```

若对上面程序变量采用不同的精度, 在不同的环境下运行, 则会得到完全不一致的结果.

在 IBM S/370 计算机的单精度、双精度、扩展精度[2)]下计算结果分别是 1.172603、1.1726039400531、1.172603940053178 [15]; 在 Pentium 4 工作站 (GCC, Linux) 上, 3 个不同类型下的计算结果依次为 2.0317×10^{29}、5.960604×10^{20}、-9.38724×10^{-323} [16]; 在 Sun 机器上 (Fortran 程序) 计算, 32 位、64 位、128 位精度下的结果分别为 -6.338253×10^{29}、$-1.1805916207174113\times10^{21}$、1.172603940053178631858834904520 1838 [17].

本文在 Windows 7 的 Maple 15 与 Matlab R2012a 环境下进行了验证. 计算过程与结果如下所述.

在 Maple 15 中执行下列语句: Digits := n : y := evalf[n](333.75 * 33096.0^6 + 77617.0^2 * (11 * 77617.0^2 * 33096.0^2 − 33096.0^6 − 121 * 33096.0^4 − 2) + 5.5 * 33096.0^8 + 77617.0/(2 * 33096.0)), 当 n 依次取 8、16、20 时, y 分别是 7×10^{29}、1×10^{21} 以及 $-9.9999999999999998827\times10^{16}$.

若在 Matlab R2012a 中直接进行计算: >> y = 333.75 * 33096.0^6 + 77617.0^2 * (11 * 77617.0^2 * 33096.0^2−33096.0^6−121*33096.0^4−2)+5.5*33096.0^8+77617.0/(2*33096.0), 结果是 -1.1806×10^{21}.

2) 此扩展精度为 128 位.

若使用其符号计算功能, 即输入 >> vpa(sym('333.75 * 33096.0^6 + 77617.0^2 * (11 * 77617.0^2 * 33096.0^2 − 33096.0^6 − 121 * 33096.0^4 − 2) + 5.5 * 33096.0^8 + 77617.0/(2 * 33096.0)'), digits), 则当 digits 取 8、16、20 时, 计算结果均为 −0.828125.

若将上式 sym 中的两个单引号去掉, 则结果分别为 -1.1805916×10^{21}、$-1.180591620717411\times10^{21}$ 以及 $-1.1805916207174113034\times10^{21}$.

事实上, 上述所有计算结果全部错误. $f(a,b)$ 的正确结果是 −0.8273960599... .

例 2.2 使用下列 C++ 程序

```
#include < iostream >
using std :: cout;
using std :: endl;
void main()
{  float a1 = 0.3333333333333333235, b1 = 0.33333333333333759, c1 = 0.008, d1 = 1e20;
   double a2 = 0.3333333333333333235, b2 = 0.33333333333333759, c2 = 0.008, d2 = 1e20;
   cout << (((float)1.0/(float)3 − a1) + ((float)1.0/(float)3 − b1) * c1) * d1 << endl;
   cout << (((double)1.0/(double)3 − a2) + ((double)1.0/(double)3 − b2) * c2) * d2 << endl;
}
```

计算

$$\left(\left(\frac{1}{3}-0.3333333333333333235\right)+\left(\frac{1}{3}-0.33333333333333759\right)\times0.008\right)\times10^{20} \tag{2}$$

的值.

若在 Windows 7 的 Visual Studio 2010 环境下执行上述程序, 则输出结果为 0 与 −355.271. 事实上, 式 (2) 的正确结果是 642.8. 程序的输出结果说明: 单精度计算的误差小于双精度计算的误差 [7, 18].

例 2.3 能从 Maple 的计算推出 $\sin(2^{100})$ 的符号吗 [14]?

在 Maple 15 中执行语句: Digits := n : evalf[n](sin(2^100)), 当 n 依次取 8、16、20 时, 计算结果分别是 0.44919995、−0.6955832137610442、−0.58645356896925826300.

事实上, 若保留 10 位有效数字, 正确结果是 −0.8721836054.

类似地, 讨论下面 Matlab 的计算结果.

例 2.4 能从 Matlab 的计算结果获得 $(e^{\pi\sqrt{163}}-262537412640768744)$ 1 位正确的有效数字吗 [14]?

若在 Matlab R2012a 中输入 >> exp(pi * sqrt(163)) − 262537412640768744, 则结果是 −480.

若使用其符号计算功能, 即输入 >> vpa(sym('exp(pi * sqrt(163)) − 262537412640768744'), digits), 则当 digits 取 8、16、20 时, 计算结果分别为 −480.0、0.000016212463378906 25、$-9.3132257461547851562\times10^{-10}$.

事实上, 上式的值是 $-7.499274028...\times10^{-13}$. Matlab 的 4 次运算均无 1 位正确的有效数字.

下面再讨论计算器的错误计算.

例 2.5 计算下式的值:

$$20^{65}-e^{65\times\ln(20)}. \tag{3}$$

显然正确结果是 0. 但是, 若使用 Casio(fx-82ES PLUS)、微软 (Windows 7)、Google [3] 等计算器进行计算, 结果分别为 2.79×10^{72}、$-7.3005772780336394753872461081808\times10^{48}$、$1.0007532\times10^{71}$.

产生上述错误计算的原因是: (1) 现有 "相同的有效位数计算模式" 会产生不正确的舍入误差; (2) 十进制数与二进制数之间的转换会导致机器误差. 对于前者, 由于所有中间运算与计算结果保留相同的有效位数, 因此计算结果不一定正确. 比如, 要计算 $(10^{100}+123-10^{100})$ 的值, 结果保留两位有效数字. 若中间运算均保留两位有效数字, 那么计算结果为 0. 事实上, 对于例 2.1 中式 (1) 来说, 在计算过程中, 若保留 8 位、16 位以及 20 位有效数字, 则计算结果分别为 7.0000000×10^{29}、$1.000000000000000\times10^{21}$、$-9.9999999999999998827\times10^{16}$ (见补充材料中的式 (S1)、(S2)、(S3)). 由于 32 位的单精度表示具有 24 位二进制 "有效位数", 而 2^{24} 是 8 位的十进制数, 这样单精度下的计算大致相当于十进制的 8 位 "有效位数" 模式下的计算. 同理, 64 位的双精度、80 位的扩展精度下的计算分别相当于十进制的 16 位、20 位 "有效位数" 模式下的计算. 因此, 若不考虑后者, 即机器误差, 那么软件在 3 个精度下的计算结果应该接近上述 3 个 "理论" 结果. 从前面的内容可知, 只有 Maple 的 3 个计算结果与上面的结果一致; 而其余软件的运行结果与上述错误 "理论" 结果相差甚远.

3 可信计算算法及其软件 ISReal

上节讨论了现有主流软件的计算不稳定问题. 本节介绍一种定量控制误差的计算算法及其软件 ISReal. 对于由常数、加减乘除四则运算以及对数指数等基本初等运算构成的算术表达式, 通过自动调节每一个中间运算的精度, 可计算算术表达式任意精度的值.

3.1 算法

设 a 是由常数、加减乘除四则运算以及对数运算、幂运算、指数运算、三角运算和反三角运算等构成的算术表达式, 它的准确值是一个实数. 我们用 $\tilde{a}$ 或 (y,ε) 来表示 a 的计算值, 其中后者说明 $|y-a|<\varepsilon$, 这时 $\varepsilon>0$ 代表运算时的误差限. 另外, $\bar{a}$ 与 $\underline{a}$ 分别表示 a 的一个上界与下界; 若 $a>0$, 定义函数 $\mathrm{Cons}(a)=\max\{\frac{1}{10^n}|\frac{1}{10^n}\leqslant a, n\in Z\}$, 例如, Cons(0.5)=0.1, Cons(0.00234)=0.001.

本节的算法是一个递归算法. 首先, 对于两个常数的四则运算, 假设可以获得它们任意精度 (误差限 $\varepsilon>0$) 的值, 而对于它们的和、差或乘积, 也可以获得准确值; 其次, 对于 "简单" 运算, 如 $\sin(0.5)$, $\cos(0.034)$, $\mathrm{e}^{3.54}$ 等, 可以利用 Taylor 展式等进行计算; 对于 "复合" 运算, 如 $\sin(0.5)\times\cos(0.034)$, $\mathrm{e}^{3.54+233\times\sin(0.5)}$ 等, 则可以利用前两个步骤递归获得计算值.

下面首先给出主函数 $\mathrm{Main}(a,\varepsilon)$ 的内容, 然后在后面详细讨论其中的每一个子函数或算法.

容易看出, 算法 1 是一个递归算法; 它既利用已有公式进行计算, 也调用子算法. 接下来讨论各个子算法.

3.1.1 乘法

对于两个表达式相乘, 有下面的定理.

定理 3.1 已知两个表达式 a_1 与 a_2. 假设存在算法, 可以求得 a_1 和 a_2 任意精度的值, 则存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $a_1\times a_2$ 的一个计算值 y, 以使 $|y-a_1\times a_2|<\varepsilon$.

3) http://www.google.com/ig/directory?type=gadgets&url=www.google.com/ig/modules/calculator.xml&hl=zh-cn.

算法 1 Main(a, ε)— 计算 a 的值

Require: a, $\varepsilon > 0$;
Ensure: $y \approx a$, $|y - a| < \varepsilon$;
  if a 是常数 **then**
    $y \Leftarrow a$; //直接返回 a 或在精度 ε 的条件下将其截断.
  else if a 是两个常数的四则运算 **then**
    $y \Leftarrow$ 两个常数的四则运算的结果; //由假设知, 可以获得两个常数的四则运算任意精度的值.
  else if $a = -a_1$ **then**
    $y \Leftarrow -\mathrm{Main}(a_1, \varepsilon)$; //若 a 表示为 a_1 的相反数, 则先获得 a_1 的值.
  else if $a = \sum_{i=1}^{n} a_i$ **then**
    $y \Leftarrow \sum_{i=1}^{n} \mathrm{Main}(a_i, \mathrm{Cons}(\frac{\varepsilon}{n}))$; //若 a 是 n 个表达式的和, 则首先平均分配误差并计算各项的值, 然后求和.
  else if $a = a_1 \times a_2$ **then**
    $y \Leftarrow \mathrm{Mul}(a_1, a_2, \varepsilon)$; //若 a 是两个表达式的积, 则调用算法 2.
  else if $a = \frac{a_1}{a_2}$ **then**
    $y \Leftarrow \mathrm{Div}(a_1, a_2, \varepsilon)$; //若 a 是两个表达式的商, 则调用算法 3.
  else if $a = \sin(a_1)$ **then**
    $y \Leftarrow \mathrm{Sin}(a_1, \varepsilon)$; //若 a 表示为 a_1 的正弦, 则调用算法 5.
  else if $a = \cos(a_1)$ **then**
    $y \Leftarrow \mathrm{Sin}(\frac{\pi}{2} - a_1, \varepsilon)$; //若 a 表示为 a_1 的余弦, 则利用算法 5.
  else if $a = \tan(a_1)$ **then**
    $y \Leftarrow \mathrm{Div}(\sin(a_1), \cos(a_1), \varepsilon)$; //若 a 表示为 a_1 的正切, 则利用算法 3.
  else if $a = \cot(a_1)$ **then**
    $y \Leftarrow \mathrm{Div}(\cos(a_1), \sin(a_1), \varepsilon)$; //若 a 表示为 a_1 的余切, 则利用算法 3.
  else if $a = \sec(a_1)$ **then**
    $y \Leftarrow \mathrm{Div}(1, \cos(a_1), \varepsilon)$; //若 a 表示为 a_1 的正割, 则利用算法 3.
  else if $a = \csc(a_1)$ **then**
    $y \Leftarrow \mathrm{Div}(1, \sin(a_1), \varepsilon)$; //若 a 表示为 a_1 的余割, 则利用算法 3.
  else if $a = \arcsin(a_1)$ **then**
    $y \Leftarrow \mathrm{Main}(2\arctan(\frac{a_1}{(1+\sqrt{1-a_1^2})}), \varepsilon)$; //若 a 表示为 a_1 的反正弦, 则利用公式 [19] 变换后再计算.
  else if $a = \arccos(a_1)$ **then**
    $y \Leftarrow \mathrm{Main}(\frac{\pi}{2} - \arcsin(a_1), \varepsilon)$; //若 a 表示为 a_1 的反余弦, 则变换后再计算.
  else if $a = \arctan(a_1)$ **then**
    $y \Leftarrow \mathrm{ArcTan}(a_1, \varepsilon)$; //若 a 表示为 a_1 的反正切, 则调用算法 7.
  else if $a = \mathrm{arccot}(a_1)$ **then**
    $y \Leftarrow \mathrm{Main}(\frac{\pi}{2} - \arctan(a_1), \varepsilon)$; //若 a 表示为 a_1 的反余切, 则变换后再计算.
  else if $a = \ln(a_1)$ **then**
    $y \Leftarrow \mathrm{Ln}(a_1, \varepsilon)$; //若 a 表示为 a_1 的自然对数, 则调用算法 9.
  else if $a = \log_{a_1} a_2$ **then**
    $y \Leftarrow \mathrm{Main}(\frac{\ln(a_2)}{\ln(a_1)}, \varepsilon)$; //若 a 表示为以 a_1 为底, a_2 的对数 (a_1、$a_2 > 0$, $a_1 \neq 1$), 则变换后再计算.
  else if $a = \mathrm{e}^{a_1}$ **then**
    $y \Leftarrow \mathrm{Exp}(a_1, \varepsilon)$; //若 a 表示为以自然对数底为底, a_1 的幂次, 则调用算法 11.
  else if $a = {a_1}^{a_2}$ **then**
    $y \Leftarrow \mathrm{Exp}(a_1, a_2, \varepsilon)$; //若 a 表示为以 a_1 为底, a_2 的幂次, 则调用算法 12.
  else if $a = \sinh(a_1)$ **then**
    $y \Leftarrow \mathrm{Main}(\frac{\mathrm{e}^{a_1} - \mathrm{e}^{-a_1}}{2}, \varepsilon)$; //若 a 表示为 a_1 的双曲正弦, 则利用定义进行计算.
  else if $a = \cosh(a_1)$ **then**
    $y \Leftarrow \mathrm{Main}(\frac{\mathrm{e}^{a_1} + \mathrm{e}^{-a_1}}{2}, \varepsilon)$; //若 a 表示为 a_1 的双曲余弦, 则利用定义进行计算.
  else if $a = \pi$ **then**
    $y \Leftarrow \mathrm{Main}(16\arctan(\frac{1}{5}) - 4\arctan(\frac{1}{239}), \varepsilon)$; //若 a 等于 π, 则利用公式 [20] 计算 π.
  end if

证明 设 $\overline{|a_2|}$ 是 $|a_2|$ 的一个上界. 若 a_1 和 a_2 的计算值依次取 $(\tilde{a}_1, \frac{\varepsilon}{2\times\overline{|a_2|}})$ 和 $(\tilde{a}_2, \frac{\varepsilon}{2\times\overline{|\tilde{a}_1|}})$, 则有

$$|\tilde{a}_1\times\tilde{a}_2 - a_1\times a_2| = |(\tilde{a}_1-a_1)\times a_2 + (\tilde{a}_2-a_2)\times\tilde{a}_1| \leqslant |\tilde{a}_1-a_1|\times\overline{|a_2|} + |\tilde{a}_2-a_2|\times|\tilde{a}_1| < \frac{\varepsilon}{2}+\frac{\varepsilon}{2} = \varepsilon.$$

令 $y = \tilde{a}_1\times\tilde{a}_2$, 则有结论成立.

由上面的定理可得乘法子算法 (算法 2).

算法 2 Mul(a_1, a_2, ε)— 计算 $a_1\times a_2$ 的值

Require: a_1, a_2, $\varepsilon > 0$;

Ensure: $y \approx a_1\times a_2$, $|y - a_1\times a_2| < \varepsilon$;

$\tilde{a}_2 \Leftarrow$ Main(a_2, 0.1); //调用主算法, 获取满足条件 $|\tilde{a}_2 - a_2| < 0.1$ 的一个 $\tilde{a}_2$.

$\varepsilon_1 \Leftarrow$ Cons($\frac{\varepsilon}{2\times(|\tilde{a}_2|+0.1)}$); //选取不大于 $\frac{\varepsilon}{2\times(|\tilde{a}_2|+0.1)}$ 的一个正数.

$\tilde{a}_1 \Leftarrow$ Main(a_1, ε_1); //调用主算法, 获取满足条件 $|\tilde{a}_1 - a_1| < \varepsilon_1$ 的一个 $\tilde{a}_1$.

$\varepsilon_2 \Leftarrow$ Cons($\frac{\varepsilon}{2\times|\tilde{a}_1|}$); //选取不大于 $\frac{\varepsilon}{2\times|\tilde{a}_1|}$ 的一个正数.

$\tilde{a}_2 \Leftarrow$ Main(a_2, ε_2); //调用主算法, 获得满足条件 $|\tilde{a}_2 - a_2| < \varepsilon_2$ 的一个 $\tilde{a}_2$.

$y \Leftarrow \tilde{a}_1\times\tilde{a}_2$; //返回 $\tilde{a}_1$ 与 $\tilde{a}_2$ 的乘积 (准确值).

若 a_2 为非零常数, 则 a_1 只要取 $(\tilde{a}_1, \frac{\varepsilon}{|a_2|})$, 则有

$$|\tilde{a}_1\times a_2 - a_1\times a_2| = |a_2|\times|\tilde{a}_1 - a_1| < |a_2|\times\frac{\varepsilon}{|a_2|} = \varepsilon.$$

3.1.2 除法

对于两个表达式相除, 由下面定理可获得除法子算法.

定理 3.2 已知两个表达式 a_1 与 a_2 ($\neq 0$). 假设存在算法, 可以求得 a_1 和 a_2 任意精度的值, 则存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $\frac{a_1}{a_2}$ 的一个计算值 y, 以使 $|y - \frac{a_1}{a_2}| < \varepsilon$.

证明 假设 $(\tilde{a}_1, \varepsilon_1)$ 和 $(\tilde{a}_2, \varepsilon_2)$ 分别是 a_1 和 a_2 的任意一个计算值, $\overline{|a_1|}$ 和 $\underline{|a_2|}$ 分别为 $|a_1|$ 和 $|a_2|$ 的一个上界与一个下界, 并且 $\underline{|a_2|} > 0$. 由 $|y - \frac{a_1}{a_2}| \leqslant |y - \frac{\tilde{a}_1}{\tilde{a}_2}| + |\frac{\tilde{a}_1}{\tilde{a}_2} - \frac{a_1}{a_2}|$ 知, 若

$$\left|y - \frac{\tilde{a}_1}{\tilde{a}_2}\right| < \frac{\varepsilon}{2}, \tag{4}$$

$$\left|\frac{\tilde{a}_1}{\tilde{a}_2} - \frac{a_1}{a_2}\right| < \frac{\varepsilon}{2}, \tag{5}$$

则有 $|y - \frac{a_1}{a_2}| < \varepsilon$.

对于式 (4) 中两个常数 $\tilde{a}_1$ 和 $\tilde{a}_2$ 相除, 由前面的假设知, 可以获得满足上式的 y. 下面仅讨论如何获得 $\tilde{a}_1$ 与 $\tilde{a}_2$, 以便使得式 (5) 成立.

首先, 我们有

$$\left|\frac{\tilde{a}_1}{\tilde{a}_2} - \frac{a_1}{a_2}\right| = \left|\frac{\tilde{a}_1\times a_2 - a_1\times\tilde{a}_2}{\tilde{a}_2\times a_2}\right| < \left|\frac{\tilde{a}_1\times a_2 - a_1\times a_2}{\tilde{a}_2\times a_2}\right| + \left|\frac{a_1\times a_2 - a_1\times\tilde{a}_2}{\tilde{a}_2\times a_2}\right| < \left|\frac{\tilde{a}_1 - a_1}{\tilde{a}_2}\right| + \left|\frac{\overline{|a_1|}\times(\tilde{a}_2 - a_2)}{\tilde{a}_2\times\underline{|a_2|}}\right|.$$

若存在 a_2 的一个计算值 $(\tilde{a}_2, \varepsilon_2)$ 满足

$$\left|\frac{\overline{|a_1|}\times\varepsilon_2}{\tilde{a}_2\times\underline{|a_2|}}\right| < \frac{\varepsilon}{4}, \tag{6}$$

则有

$$\left|\frac{\overline{|a_1|}\times(\tilde{a}_2-a_2)}{\tilde{a}_2\times\underline{|a_2|}}\right|<\frac{\varepsilon}{4}. \tag{7}$$

最后, 令 $|\tilde{a}_1-a_1|<\frac{|\tilde{a}_2|\times\varepsilon}{4}$, 则有 $|\frac{\tilde{a}_1-a_1}{\tilde{a}_2}|<\frac{\varepsilon}{4}$, 从而式 (5) 成立.

于是有算法 3.

算法 3 Div(a_1, a_2, ε)— 计算 $\frac{a_1}{a_2}$ 的值

Require: a_1, $a_2\neq 0$, $\varepsilon>0$;
Ensure: $y\approx\frac{a_1}{a_2}$, $|y-\frac{a_1}{a_2}|<\varepsilon$;
  $\tilde{a}_1\Leftarrow$ Main(a_1, 0.1);
  $\overline{|a_1|}\Leftarrow|\tilde{a}_1|+0.1$; //获得 $|a_1|$ 的一个上界.
  $\varepsilon_2\Leftarrow 0.1$;
  $\tilde{a}_2\Leftarrow$ Main(a_2, ε_2);
  while ($|\tilde{a}_2|\leqslant 2\times\varepsilon_2$) **do**
    $\varepsilon_2\Leftarrow 0.1\times\varepsilon_2$;
    $\tilde{a}_2\Leftarrow$ Main(a_2, ε_2);
  end while
  $\underline{|\tilde{a}_2|}\Leftarrow|\tilde{a}_2|-\varepsilon_2$; //若 $|\tilde{a}_2|>2\times\varepsilon_2$, 则 $(|\tilde{a}_2|-\varepsilon_2)$ 是 $|a_2|$ 的一个正的下界.
  while ($|4\times\overline{|a_1|}\times\varepsilon_2|\geqslant|\tilde{a}_2\times\underline{|a_2|}\times\varepsilon|$) **do**
    $\varepsilon_2\Leftarrow 0.1\times\varepsilon_2$;
    $\tilde{a}_2\Leftarrow$ Main(a_2, ε_2); //获得 a_2 的满足式 (6) 的一个计算值 $(\tilde{a}_2,\varepsilon_2)$.
  end while
  $\tilde{a}_1\Leftarrow$ Main(a_1, Cons($\frac{|\tilde{a}_2|}{4}\times\varepsilon$)); //计算 a_1 的一个值, 其中误差限为 $\frac{|\tilde{a}_2|}{4}\times\varepsilon$.
  $y\Leftarrow$ Main($\frac{\tilde{a}_1}{\tilde{a}_2}$, $\frac{\varepsilon}{2}$); //由假设知, 对于两个常数相除, 可以获得任意精度的值.

类似于乘法, 若 a_2 为非零常数, 则只要 a_1 取 $(\tilde{a}_1,\frac{|a_2|\times\varepsilon}{2})$, $\frac{\tilde{a}_1}{a_2}$ 取 $(y,\frac{\varepsilon}{2})$, 就有

$$\left|y-\frac{a_1}{a_2}\right|\leqslant\left|y-\frac{\tilde{a}_1}{a_2}\right|+\left|\frac{\tilde{a}_1}{a_2}-\frac{a_1}{a_2}\right|<\frac{\varepsilon}{2}+\frac{\frac{|a_2|\times\varepsilon}{2}}{|a_2|}=\varepsilon.$$

3.1.3 正弦运算

对于三角运算来说, 由于余弦等运算均可用正弦运算来表示, 因此本小节首先讨论如何计算一个常数的正弦值, 然后讨论如何计算表达式的正弦值.

引理 3.1 已知 c 是一个常数. 存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $\sin(c)$ 的一个计算值 y, 以使 $|y-\sin(c)|<\varepsilon$.

证明 下面利用 Taylor 展式 [21], 讨论 $\sin(c)$ 的可信计算.

首先, $\sin(c)$ 可以表示成

$$\sin(c)\;=\;c-\frac{c^3}{3!}+\frac{c^5}{5!}+\cdots+(-1)^n\frac{c^{2n+1}}{(2n+1)!}+\cdots. \tag{8}$$

记上式前 n 项的和为 α, 则有

$$|\alpha-\sin(c)|\;<\;\frac{c^{2n+1}}{(2n+1)!}. \tag{9}$$

若令

$$\left|\frac{c^{2n+1}}{(2n+1)!}\right|\;<\;\frac{\varepsilon}{2},\;\;|\tilde{\alpha}-\alpha|\;<\;\frac{\varepsilon}{2}, \tag{10}$$

其中 $\tilde{\alpha}$ 是 α 的计算值, 则由式 (9) 和 (10) 可得

$$|\tilde{\alpha}-\sin(c)|<|\tilde{\alpha}-\alpha|+|\alpha-\sin(c)| \ < \ \varepsilon.$$

故只要令 $y=\tilde{\alpha}$, 就有本结论成立.

于是有算法 4.

算法 4 Sin1(c, ε) — 计算 $\sin(c)$ 的值

Require: c 是常数, $\varepsilon>0$;
Ensure: $y\approx\sin(c)$, $|y-\sin(c)|<\varepsilon$;
  $n\Leftarrow 1$; //初始化 n.
  while $(2\times c^{2n+1}\geqslant(2n+1)!\times\varepsilon)$ **do**
    $n\Leftarrow n+1$;
  end while
  $y\Leftarrow$ Main(α, $\frac{\varepsilon}{2}$); //计算式 (8) 中前 n 项的和.

上面给出了常数的正弦的可信计算算法. 下面讨论表达式的正弦的可信计算算法.

定理 3.3 已知 a 是表达式. 假设存在算法, 可以计算 a 任意精度的值, 那么也存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $\sin(a)$ 的一个计算值 y, 以使 $|y-\sin(a)|<\varepsilon$.

证明 设 $(\tilde{a},\frac{\varepsilon}{2})$ 是 a 的一个计算值, $(y,\frac{\varepsilon}{2})$ 是 $\sin(\tilde{a})$ 的一个计算值, 则有

$$|y-\sin(a)|\leqslant|y-\sin(\tilde{a})|+|\sin(\tilde{a})-\sin(a)|<\frac{\varepsilon}{2}+|\tilde{a}-a|<\frac{\varepsilon}{2}+\frac{\varepsilon}{2} \ = \ \varepsilon.$$

综上所述, 可得算法 5.

算法 5 Sin(a, ε) — 计算 $\sin(a)$ 的值

Require: a, $\varepsilon>0$;
Ensure: $y\approx\sin(a)$, $|y-\sin(a)|<\varepsilon$;
  if a 是常数 **then**
    $y\Leftarrow$ Sin1(a,ε);
  else
    $\tilde{a}\Leftarrow$ Main(a, $\frac{\varepsilon}{2}$); //获得满足条件 $|\tilde{a}-a|<\frac{\varepsilon}{2}$ 的一个 $\tilde{a}$.
    $y\Leftarrow$ Sin1($\tilde{a}$, $\frac{\varepsilon}{2}$); //返回 $\sin(\tilde{a})$ 的精度为 $\frac{\varepsilon}{2}$ 的一个值.
  end if

3.1.4 反正切运算

前面讨论了三角运算的计算算法. 现在讨论反三角运算的可信计算算法.

由于反正弦运算的算法涉及到反正切运算, 因此本小节先讨论反正切运算的可信计算.

引理 3.2 已知常数 $c\in[-1,1]$. 存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $\arctan(c)$ 的一个计算值 y, 以使 $|y-\arctan(c)|<\varepsilon$.

证明 下面利用 Taylor 展式 [21], 讨论 $\arctan(c)$ 的可信计算.

首先 $\arctan(c)$ 可以写成

$$\arctan(c) \ = \ c-\frac{c^3}{3}+\frac{c^5}{5}+\cdots+(-1)^n\frac{c^{2n+1}}{2n+1}+\cdots. \tag{11}$$

记上式前 n 项的和为 α, 则有

$$|\alpha - \arctan(c)| \ < \ \frac{c^{2n+1}}{2n+1}. \tag{12}$$

令

$$\left|\frac{c^{2n+1}}{2n+1}\right| \ < \ \frac{\varepsilon}{2}, \ \ |\tilde{\alpha} - \alpha| < \frac{\varepsilon}{2}, \tag{13}$$

其中 $\tilde{\alpha}$ 是 α 的计算值, 则由式 (12) 和 (13) 可得

$$|\tilde{\alpha} - \arctan(c)| < |\tilde{\alpha} - \alpha| + |\alpha - \arctan(c)| \ < \ \frac{\varepsilon}{2} + \frac{\varepsilon}{2} \ = \ \varepsilon.$$

故只要令 $y = \tilde{\alpha}$, 则有本结论成立.

若 $c \notin [-1, 1]$, 那么利用公式 $\arctan(x) = 2\arctan(\frac{x}{1+\sqrt{1+x^2}})$ [19] 可将其变换到 $[-1, 1]$, 从而有算法 6.

算法 6 ArcTan1(c, ε) — 计算 arctan(c) 的值

Require: c 是常数, $\varepsilon > 0$;
Ensure: $y \approx \arctan(c)$, $|y - \arctan(c)| < \varepsilon$;
  if $c \notin [-1, 1]$ **then**
    $y \Leftarrow \text{Main}(2\arctan(\frac{c}{1+\sqrt{1+c^2}}), \varepsilon)$;
  else
    $n \Leftarrow 1$; //初始化 n.
    while $(2 \times c^{2n+1} \geqslant (2n+1) \times \varepsilon)$ **do**
      $n \Leftarrow n + 1$;
    end while
    $y \Leftarrow \text{Main}(\alpha, \frac{\varepsilon}{2})$; //计算式 (11) 中前 n 项的和.
  end if

若 a 不是常数, 那么可以利用前面的算法进行可信计算.

定理 3.4 已知 a 是表达式. 假设存在一个算法, 可以计算 a 任意精度的值, 那么也存在一个算法, 对于任意的误差限 ε (> 0), 可以获得 arctan(a) 的一个计算值 y, 以使 $|y - \arctan(a)| < \varepsilon$.

证明 设 $(\tilde{a}, \frac{\varepsilon}{2})$ 是 a 的一个计算值, $(y, \frac{\varepsilon}{2})$ 是 $\arctan(\tilde{a})$ 的一个计算值. 由 $\arctan(x)' = \frac{1}{1+x^2} \leqslant 1$ 可得

$$|y - \arctan(a)| \leqslant |y - \arctan(\tilde{a})| + |\arctan(\tilde{a}) - \arctan(a)| < \frac{\varepsilon}{2} + |\tilde{a} - a| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

于是有算法 7.

算法 7 ArcTan(a, ε) — 计算 arctan(a) 的值

Require: a, $\varepsilon > 0$;
Ensure: $y \approx \arctan(a)$, $|y - \arctan(a)| < \varepsilon$;
  if a 是常数 **then**
    $y \Leftarrow \text{ArcTan1}(a, \varepsilon)$;
  else
    $\tilde{a} \Leftarrow \text{Main}(a, \frac{\varepsilon}{2})$; //获得满足条件 $|\tilde{a} - a| < \frac{\varepsilon}{2}$ 的一个 $\tilde{a}$.
    $y \Leftarrow \text{Main}(\arctan(\tilde{a}), \frac{\varepsilon}{2})$; //返回 arctan($\tilde{a}$) 的满足精度为 $\frac{\varepsilon}{2}$ 的一个值.
  end if

3.1.5 自然对数运算

本小节讨论自然对数运算的可信计算算法.

引理 3.3 已知常数 $c>0$. 存在一个算法, 对于任意的误差限 ε (>0), 可以获得 $\ln(c)$ 的一个计算值 y, 以使 $|y-\ln(c)|<\varepsilon$.

证明 首先, 若 $x\in(-1,1)$, 则有 Taylor 展式 [21]

$$\ln\left(\frac{1+x}{1-x}\right)=2x+\frac{2}{3}x^3+\frac{2}{5}x^5+\cdots+\frac{2}{2n+1}x^{2n+1}+\cdots. \tag{14}$$

下面分两步来讨论.

1. 设 $c>1$. 令 $\frac{1+x}{1-x}=c$, 则有

$$x=\frac{c-1}{c+1}. \tag{15}$$

显然, 对任意 $c>1$, 由上式得到的 x, 均有 $x\in(0,1)$, 从而能保证式 (14) 收敛.

记式 (14) 中前 n 项的和为 α, 即

$$\alpha=\frac{2}{1}\frac{(c-1)}{(c+1)}+\frac{2}{3}\frac{(c-1)^3}{(c+1)^3}+\frac{2}{5}\frac{(c-1)^5}{(c+1)^5}+\cdots+\frac{2}{(2n-1)}\frac{(c-1)^{2n-1}}{(c+1)^{2n-1}}, \tag{16}$$

则余项为

$$R_n(x)=\sum_{k=n}^{\infty}\frac{2}{2k+1}x^{2k+1}.$$

由于 $|x|<1$ 时,

$$\sum_{k=n}^{\infty}x^k=\frac{1}{1-x}-\sum_{k=0}^{n-1}x^k=\frac{1}{1-x}-\frac{1-x^n}{1-x}=\frac{x^n}{1-x}.$$

因此可将余项放大,

$$R_n(x)=\sum_{k=n}^{\infty}\frac{2}{2k+1}x^{2k+1}<\frac{x}{n}\sum_{k=n}^{\infty}(x^2)^k=\frac{x^{2n+1}}{n(1-x^2)}=\frac{(c-1)^{2n+1}}{4nc(c+1)^{2n-1}}.$$

令上式的右端小于 $\frac{\varepsilon}{2}$, 即

$$\frac{(c-1)^{2n+1}}{4nc(c+1)^{2n-1}}<\frac{\varepsilon}{2}, \tag{17}$$

则可获得满足上式的一个 n; 再令 α 的计算值为 $(y,\frac{\varepsilon}{2})$, 则有结论成立.

2. 接下来讨论 $c<1$ 的情形.

由于 $\frac{1}{c}>1$, 因此我们可以利用变换 $\ln(c)=-\ln(\frac{1}{c})$ 来计算. 这时容易验证, 上面的推导过程几乎不变: α 不需要作变换, 式 (17) 中的 $(c-1)$ 变为 $(1-c)$.

于是有算法 8.

算法 8 Ln1(c, ε)— 计算 $\ln(c)$ 的值

Require: 常数 $c > 0$, $\varepsilon > 0$;
Ensure: $y \approx \ln(c)$, $|y - \ln(c)| < \varepsilon$;
  if $c = 1$ **then**
    $y \Leftarrow 0$;
  else
    $n \Leftarrow 1$; //初始化 n.
    while $(2 \times |c-1|^{2n+1} \geqslant 4nc(c+1)^{2n-1} \times \varepsilon)$ **do**
      $n \Leftarrow n + 1$;
    end while
    $y \Leftarrow \mathrm{Main}(\alpha, \frac{\varepsilon}{2})$; //计算式 (16) 中 n 项的和.
  end if

对于表达式的自然对数, 有下面定理.

定理 3.5 已知表达式 $a > 0$. 假设存在一个算法, 可以计算 a 任意精度的值, 那么也存在一个算法, 对于任意的误差限 ε (> 0), 可以获得 $\ln(a)$ 的一个计算值 y, 以使 $|y - \ln(a)| < \varepsilon$.

证明 设 $(\tilde{a}, \varepsilon')$ 为 a 的一个计算值, $(y, \frac{\varepsilon}{2})$ 为 $\ln(\tilde{a})$ 的一个计算值. 由 Lagrange 中值定理知, 存在一个 ξ (介于 a 和 $\tilde{a}$ 之间), 使

$$|\ln(\tilde{a}) - \ln(a)| = \left|(\tilde{a} - a)\,\frac{1}{\xi}\right|.$$

若令

$$|\tilde{a}| > 2\varepsilon',\ \ \varepsilon' \leqslant \frac{(|\tilde{a}| - \varepsilon')\varepsilon}{2}, \tag{18}$$

则由 $|\ln(\tilde{a}) - \ln(a)| = |(\tilde{a} - a)\,\frac{1}{\xi}\,| \leqslant |(\tilde{a} - a)\,\frac{1}{|\tilde{a}|-\varepsilon'}| < \frac{\varepsilon'}{|\tilde{a}|-\varepsilon'} \leqslant \frac{\varepsilon}{2}$, 可得

$$|y - \ln(a)| \leqslant |y - \ln(\tilde{a})| + |\ln(\tilde{a}) - \ln(a)| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

于是有算法 9.

算法 9 Ln(a, ε)— 计算 $\ln(a)$ 的值

Require: $a > 0$, $\varepsilon > 0$;
Ensure: $y \approx \ln(a)$, $|y - \ln(a)| < \varepsilon$;
  if a 是常数 **then**
    $y \Leftarrow \mathrm{Ln1}(a, \varepsilon)$; //调用算法 8.
  else
    $\varepsilon' \Leftarrow 0.1$; //初始化.
    $\tilde{a} \Leftarrow \mathrm{Main}(a, \varepsilon')$;
    while $(|\tilde{a}| \leqslant 2\varepsilon' \ \vee\ 2\varepsilon' > (|\tilde{a}| - \varepsilon')\varepsilon)$ **do**
      $\varepsilon' \Leftarrow 0.1 \times \varepsilon'$;
      $\tilde{a} \Leftarrow \mathrm{Main}(a, \varepsilon')$;
    end while
    $y \Leftarrow \mathrm{Ln1}(\tilde{a}, \frac{\varepsilon}{2})$; //调用算法 8.
  end if

3.1.6 指数运算

本小节通过研究底为自然对数底的指数运算的计算算法, 给出一般指数运算的可信计算算法.

引理 3.4 已知 c 是一个常数. 存在一个算法, 对于任意的误差限 ε (>0), 可以获得 e^c 的一个计算值 y, 以使 $|y-\mathrm{e}^c|<\varepsilon$.

证明 下面利用 Taylor 展式 [21], 讨论 e^c 的可信计算.

首先, e^c 可以写成

$$\mathrm{e}^c = 1 + c + \frac{1}{2!}c^2 + \frac{1}{3!}c^3 + \cdots + \frac{1}{n!}c^n + \cdots . \tag{19}$$

记其前 n 项的和为 α, 则余项为

$$R_n(c) = \sum_{k=0}^{\infty} \frac{c^{n+k}}{(n+k)!} = \frac{c^n}{n!}\left[1 + \sum_{k=1}^{\infty} \frac{c^k}{(n+1)(n+2)\cdots(n+k)}\right].$$

将上式中的分母 $(n+1)(n+2)\cdots(n+k)$ 替换成较小值 n^k, 并设 $n>|c|$, 则可得

$$R_n(c) \leqslant \frac{c^n}{n!}\left[\sum_{k=0}^{\infty}(\frac{c}{n})^k\right] = \left(\frac{c^n}{n!}\right)\left(\frac{1}{1-c/n}\right) = \frac{c^n}{(n-1)!\,(n-c)}.$$

令上式右端小于 $\frac{\varepsilon}{2}$, 则可求得 n 的值, 再利用主算法可获得 α 的一个值 $(y, \frac{\varepsilon}{2})$.

于是有算法 10.

算法 10 Exp1(c, ε) — 计算 e^c 的值

Require: c 是常数, $\varepsilon>0$;
Ensure: $y \approx \mathrm{e}^c$, $|y-\mathrm{e}^c|<\varepsilon$;
  $n \Leftarrow \max\{\mathrm{Ceil}(|c|), 1\}$; //Ceil$(|c|)$ 表示取不小于 $|c|$ 的最小整数.
  while $(2\times c^n \geqslant (n-1)!\times(n-c)\times\varepsilon)$ **do**
    $n \Leftarrow n+1$;
  end while
  $y \Leftarrow \mathrm{Main}(\alpha, \frac{\varepsilon}{2})$; //计算式 (19) 中前 n 项的和.

对于非常数的表达式的以自然对数底为底的指数运算, 有下面定理.

定理 3.6 已知 a 是表达式. 假设存在一个算法, 可以计算 a 任意精度的值, 那么也存在一个算法, 对于任意的误差限 ε (>0), 可以获得 e^a 的一个计算值 y, 以使 $|y-\mathrm{e}^a|<\varepsilon$.

证明 设 $(\tilde{a}, \varepsilon')$ 为 a 的一个计算值. 根据 Lagrange 中值定理知, 存在一个 ξ (介于 a 和 $\tilde{a}$ 之间), 使得

$$|\mathrm{e}^{\tilde{a}} - \mathrm{e}^a| = |(\tilde{a}-a)\,\mathrm{e}^{\xi}|;$$

又由于 e^x 是增函数, 因此有

$$|\mathrm{e}^{\tilde{a}} - \mathrm{e}^a| = |(\tilde{a}-a)\,\mathrm{e}^{\xi}| \leqslant |(\tilde{a}-a)\,\mathrm{e}^{\tilde{a}+\varepsilon'}|;$$

再设 (y_1, ε'') 为 $\mathrm{e}^{\tilde{a}+\varepsilon'}$ 的一个计算值, 则有

$$|\mathrm{e}^{\tilde{a}} - \mathrm{e}^a| = |(\tilde{a}-a)\,\mathrm{e}^{\xi}| \leqslant |(\tilde{a}-a)\,\mathrm{e}^{\tilde{a}+\varepsilon'}| < |(\tilde{a}-a)\,(y_1+\varepsilon'')|;$$

最后令

$$|\tilde{a}-a| < \frac{\varepsilon}{2(y_1+\varepsilon'')},\ \ |y-\mathrm{e}^{\tilde{a}}| < \frac{\varepsilon}{2}, \tag{20}$$

则有

$$|y-\mathrm{e}^a| \leqslant |y-\mathrm{e}^{\tilde{a}}| + |\mathrm{e}^{\tilde{a}}-\mathrm{e}^a| < \frac{\varepsilon}{2}+\frac{\varepsilon}{2} = \varepsilon.$$

于是有算法 11.

算法 11 $\mathrm{Exp}(a,\varepsilon)$— 计算 e^a 的值

Require: a, $\varepsilon > 0$;
Ensure: $y \approx \mathrm{e}^a$, $|y-\mathrm{e}^a| < \varepsilon$;
  $\varepsilon' \Leftarrow 0.1$; //初始化.
  $\tilde{a} \Leftarrow \mathrm{Main}(a,\varepsilon')$;
  $\varepsilon'' \Leftarrow 0.2$; //初始化.
  $y_1 \Leftarrow \mathrm{Exp1}(\tilde{a}+\varepsilon', \varepsilon'')$;
  $\tilde{a} \Leftarrow \mathrm{Main}(a, \mathrm{Cons}(\frac{\varepsilon}{2(y_1+\varepsilon'')}))$;
  $y \Leftarrow \mathrm{Exp1}(\tilde{a}, \frac{\varepsilon}{2})$;

由上面的算法很容易推出一般指数运算的可信计算算法 (算法 12).

算法 12 $\mathrm{Exp}(a_1, a_2, \varepsilon)$— 计算 $a_1^{a_2}$ 的值

Require: a_1, a_2, $\varepsilon > 0$. 其中 a_1 可以确定其符号. 若 $a_1 = 0$, 则 $a_2 > 0$; 若 $a_1 < 0$, 则 a_2 可以表示成简分数 $\frac{n}{m}$, 并且 m 是奇数.
Ensure: $y \approx a_1^{a_2}$, $|y - a_1^{a_2}| < \varepsilon$;
  if $a_1 = 0$ **then**
    $y \Leftarrow 0$;
  else if $a_1 > 0$ **then**
    $y \Leftarrow \mathrm{Main}(\mathrm{e}^{a_2 \times \ln(a_1)}, \varepsilon)$;
  else if n 是偶数 **then**
    $y \Leftarrow \mathrm{Main}(\mathrm{e}^{a_2 \times \ln(-a_1)}, \varepsilon)$; //$a_1 < 0$, m 是奇数.
  else
    $y \Leftarrow -\mathrm{Main}(\mathrm{e}^{a_2 \times \ln(-a_1)}, \varepsilon)$; //$a_1 < 0$, n、m 均是奇数.
  end if

本节介绍了四则运算、(反) 三角运算、对数运算、指数运算等的可信计算算法. 事实上, 在每一步的计算过程中均可执行四舍五入运算, 以便减少计算量.

引理 3.5 设一个表达式的计算值是 $(\pm 0.t_1t_2\ldots t_k \times 10^{n\ 4)}, 0.1\times 10^m)$, 其中 $0 \leqslant t_i \leqslant 9$, $t_1 \neq 0$, $t_i \in N, 1 \leqslant i \leqslant k$, 并且 $t = \max\{n-m+2, 1\}$. 若 $k \geqslant t$, 那么将计算值从 t 位截断或四舍五入后, 结果仍然满足误差限.

例如, 若某一表达式的计算值是 (123.45678, 0.001). 由 $k=8$, $n=3$, $m=-2$ 可知 $t=7$, 因此将 123.45678 的后两位四舍五入后, 值 123.457 也是原表达式的计算值. 这种变换后值的 "保持性" 不难理解: 首先, (123.45678, 0.001) 说明, $(123.45678-0.001, 123.45678+0.001)$ 内的任意实数均是表达式的满足误差限为 0.001 的值; 其次, 显然将后面的有效数字舍入后, 值 123.457 仍然属于上述区间.

3.2 软件简介与举例说明

对于前面的算法, 使用 C++ 语言, 作者已编程实现可信软件 ISReal [5)]. 由于页数限制, 本节仅就

4) 对每个计算值, 符号只能有一个. 要么正号, 要么负号.
5) 调用方式见补充材料 (III) 或软件下载网址: http://www.zhaoshizhong.org/download.htm.

前面的一个错误计算案例, 详细说明可信计算的流程.

例 3.1 已知误差限 $\varepsilon = 0.0001$, 计算例 2.5 中式 (3) 的值.

首先平均分配误差: 20^{65} 与 $\mathrm{e}^{65\times\ln(20)}$ 的误差限各取 $\frac{\varepsilon}{2}$. 由于 20^{65} 可以通过乘法获得准确值, 因此我们仅讨论在误差限 $\frac{\varepsilon}{2}$ 的条件下, $\mathrm{e}^{65\times\ln(20)}$ 的计算过程.

对于函数 e^x 的计算, 由于其导数可能大于 1, 因此首先需要进行预处理, 以便获得导函数在 $65\times\ln(20)$ 附近的一个上界.

(1) 获得 $65\times\ln(20)$ 的一个上界.

设 $65\times\ln(20)$ 的误差限为 0.1. 这时 $\ln(20)$ 的误差限可取 $\mathrm{Cons}(\frac{0.1}{65}) = 0.001$, 并在此误差限条件下, 可以计算出 $\ln(20)$ 的值为 2.996. 从而 $65\times\ln(20)$ 有一个上界 $65\times 2.996 + 0.1 = 194.84$.

(2) 计算 $\mathrm{e}^{194.84}$ 的一个上界.

不妨误差限取 0.2, 可得 $\mathrm{e}^{194.84}$ 的值 $y_1 = 4148937131299313312230266926171994470257648576216071495027526126152981588365027182687.1$. 此值加上误差限 0.2, 则可获得 $\mathrm{e}^{194.84}$ 的一个上界 $b = y_1 + 0.2 = 4148937131299313312230266926171994470257648576216071495027526126152981588365027182687.3$.

经过上述两步, 已经获得 e^x 的导函数在 $65\times\ln(20)$ 附近的一个上界. 这时可以进行正式计算:

(1) 重新计算 $65\times\ln(20)$ 的一个值.

首先将 $\mathrm{Cons}(\frac{\varepsilon}{2\times b}) = 10^{-90}$ 作为 $65\times\ln(20)$ 的误差限. 这时 $\ln(20)$ 的误差限可取 $\mathrm{Cons}(\frac{10^{-90}}{65}) = 10^{-92}$, 并在此误差限下, 可以计算出 $\ln(20)$ 的一个值, 为 2.99573227355399099343522357614254077567660162298902823015400791046096623164704719584186053209, 此值乘以 65 后得出 $65\times\ln(20)$ 的一个计算值, 为 194.72259778100941457328953244926515041897910549428683496001051417996280505705806772972093458585, 将后面两位有效数字四舍五入后为 $x = 194.722597781009414573289532449265150418979105494286834960010514179962805057058067729720934586$.

(2) 计算 e^x 的一个值.

在误差限为 $\frac{\varepsilon}{2}$ 的条件下, 可得 e^x 的一个值 3689348814741910323200000000000000000000000000000000000000000000000000000000000000000.00000. 此值就是 $\mathrm{e}^{65\times\ln(20)}$ 的最终结果.

于是, 式 (3) 的结果为 0.

若利用现有计算模式, 那么无论计算精度多高, 均不会得到正确结果 0. 这可从补充材料中式 (S4)、(S5)、(S6) 的计算过程得到验证.

4 结束语

本文给出算术表达式的一种可信计算算法: (1) 它是一种定量控制误差的方法; (2) 每个中间运算的精度是自动获得的, 勿需人为设置; (3) 在不知算术表达式准确值的情况下, 可以获得其任意精度的计算值.

致谢 衷心感谢何积丰院士、杨路先生及其团队成员、周巢尘院士及其团队成员和陈冬火等的帮助.

补充材料 本文的补充材料见网络版 infocn.scichina.com. 补充材料为作者提供的原始数据, 作者对其学术质量和内容负责.

参考文献

1 Lorenz E N. Deterministic nonperiodic flow. J Atmos Sci, 1963, 20: 130–141
2 Quinn K. Even had problems rounding off figures? This stock exchange has. Wall Street J, 1983, 202: 37
3 Stroud R. Rounding error costs DHSS 100 million pounds. Risks Digest, 1987, 5
4 Robert S. Roundoff error and the patriot missile. SIAM News, 1992, 25: 11
5 Jakobsen B, Rosendahl F. The Sleipner platform accident. Struct Eng Int, 1994, 4: 190–193
6 Selby R G, Vecchio F J, Collins M P. The failure of an offshore platform. Concrete Int, 1997, 19: 28–35
7 McCullough B D, Vinod H D. The numerical reliability of econometric software. J Economic Literature, 1999, 37: 633–665
8 Yang L, Zhou C C, Zhan N J, et al. Recent advances in program verification through computer algebra. Front Comput Sci China, 2010, 4: 1–16
9 Tang E Y, Barr E, Su Z D, et al. Program instability detection based on systematically optimized numerical perturbation. Sci Sin Inform, 2014, 44: 1445–1466 [汤恩义, BARR Earl, 苏振东, 等. 程序数值误差的扰动检测与优化. 中国科学: 信息科学, 2014, 44: 1445–1466]
10 Zou D, Wang R, Xiong Y F, et al. A genetic algorithm for detecting significant floating-point inaccuracies. In: Proceedings of the 37th IEEE International Conference on Software Engineering. California: IEEE Computer Society, 2015. 529–539
11 Zhang J Z, Feng Y. Obtaining accurate values by approximate calculation. Sci China Math, 2007, 37: 809–816 [张景中, 冯勇. 采用近似计算获得准确值. 中国科学: 数学, 2007, 37: 809–816]
12 Cuyt A, Verdonk B, Becuwe S, et al. A remarkable example of catastrophic cancellation unraveled. Computing, 2001, 66: 309–320
13 Li Q Y, Wang N C, Yi D Y. Numerical Analysis. 5th ed. Beijing: Tsinghua University Press, 2008. 18 [李庆扬, 王能超, 易大义. 数值分析. 第 5 版. 北京: 清华大学出版社, 2008. 18]
14 Zimmermann P. Reliable computing with GNU MPFR. In: Mathematical Software–ICMS 2010. Berlin: Springer, 2010. 42–45
15 Rump S. Algorithms for verified inclusion: theory and practice. In: Reliability in Computing: the Role of Interval Methods in Scientific Computing. Boston: Academic Press, 1988. 109–126
16 Muller J M, Brisebarre N, Dinechin F D, et al. Handbook of Floating-Point Arithmetic. Boston: Birkhauser Boston, 2010. 12
17 Loh E, Walster G W. Rump's example revisited. Reliable Comput, 2002, 8: 245–248
18 Higham N J. Accuracy and Stability of Numerical Algorithms. 2nd ed. Philadelphia: SIAM, 2002. 17
19 Inverse trigonometric functions. Wikipedia. http://en.wikipedia.org/wiki/Inverse_trigonometric_function
20 John Machin. Wikipedia. https://en.wikipedia.org/wiki/John_Machin
21 Zhen X F. Applied Numerical Method. Beijing: Tsinghua University Press, 2006. 53–80 [甄西丰. 实用数值计算方法. 北京: 清华大学出版社, 2006. 53–80]

A reliable computing algorithm and its software application ISReal for arithmetic expressions

Shizhong ZHAO

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062*, China*
E-mail: szzhao@sei.ecnu.edu.cn

Abstract Precise evaluation of arithmetic expressions is fundamental for ensuring the accuracy of various complicated calculations. However, in most mainstream software, incorrect rounding computation error and machine error lead to results instability. To address the problem of unstable computation, this paper proposes an automated adaptive technique for determining the correct degree of precision in the intermediate steps of a calculation,

thereby an algorithm is presented for performing the reliable calculation of arithmetic expressions. In addition, we implement a software application, ISReal, using C++. Using this method, an arithmetic expression can be evaluated with any required degree of precision, regardless of whether the expression is well-conditioned or ill-conditioned, and the calculated value will always be consistent with the theoretical result.

Keywords error-controlled calculation, accurate calculation, arithmetic expressions, reliable computing, calculation error

Shizhong ZHAO received a Ph.D. degree in computer science from Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu, in 2006. He is a lecturer at East China Normal University, where he specializes in reliable computing, software engineering, and symbolic computation.