软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(12):3685−3699 [doi: 10.13328/j.cnki.jos.005883]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

循环迭代程序的一种可信计算算法*

赵世忠[1,2], 陈冬火[3], 刘 静[1,2]

[1](华东师范大学 软件工程学院,上海 200062)

[2](上海市高可信计算重点实验室(华东师范大学),上海 200062)

[3](苏州大学 计算机科学与技术学院,江苏 苏州 215006)

通讯作者: 赵世忠, E-mail: szzhao@sei.ecnu.edu.cn

摘 要: 循环迭代程序作为软件的基本组成部分,其正确运行具有重要意义.然而,有时(比如其相关错数大于 0 时)计算时的舍入误差(或表示误差)会导致循环迭代的计算结果不稳定.基于“中间计算精度自动动态调整”的计算技术,给出了循环迭代程序的一种可信计算算法.利用该算法,可获得循环迭代程序任意次迭代的任意位的正确有效数字.目前,通过 C++语言该算法已被编程实现于 ISReal 中.

关键词: 循环迭代;误差可控计算;可信计算;可靠计算;错数

中图法分类号: TP311

中文引用格式: 赵世忠,陈冬火,刘静.循环迭代程序的一种可信计算算法.软件学报,2020,31(12):3685−3699. http://www.jos.org.cn/1000-9825/5883.htm

英文引用格式: Zhao SZ, Chen DH, Liu J. Reliable algorithm for computing cyclic iterative program. Ruan Jian Xue Bao/ Journal of Software, 2020,31(12):3685−3699 (in Chinese). http://www.jos.org.cn/1000-9825/5883.htm

Reliable Algorithm for Computing Cyclic Iterative Program

ZHAO Shi-Zhong[1,2], CHEN Dong-Huo[3], LIU Jing[1,2]

[1](Software Engineering Institute, East China Normal University, Shanghai 200062, China)

[2](Shanghai Key Laboratory for Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[3](School of Computer Science and Technology, Soochow University, Suzhou 215006, China)

Abstract: As a basic component of software, the correct running of cyclic iteration program is of great significance. However, sometimes (e.g., when its NID is greater than 0) the rounding error (or representation error) in the calculation can lead to unstable results of the cyclic iteration. Based on the computing technology of “automatic dynamic adjustment of intermediate calculation accuracy”, a reliable calculation algorithm for cyclic iteration is presented in this paper. By using this algorithm, the value of arbitrary precision of cyclic iteration can be obtained. At present, the algorithm has been programmed and implemented in ISReal through C++ language.

Key words: cyclic iteration; error-controlled calculation; trusted computing; reliable computing; NID

1989 年,Muller 在其著作《计算机算术》[1]中给出了一个循环迭代:

$$\begin{cases} u_1 = 2 \\ u_2 = -4 \\ u_n = 111 - \dfrac{1130}{u_{n-1}} + \dfrac{3000}{u_{n-1}u_{n-2}} \end{cases} \tag{1}$$

* 基金项目: 国家重点研发项目(2017YFB1001800); 国家自然科学基金(61772203, 61972150, 61876034)

Foundation item: National Key Research and Development Program of China (2017YFB1001800); National Natural Science Foundation of China (61772203, 61972150, 61876034)

收稿时间: 2019-04-17; 修改时间: 2019-05-30, 2019-07-21; 采用时间: 2019-08-05

它收敛于 6.然而,“它在任何系统任何精度下均出错(and yet, on any system with any precision, the sequence will seem to go to 100[2])”.即它总是趋向 100,而不是正确答案 6.比如,图 1 中的程序在一个机器(基于 Pentium 4 的工作站,Linux 系统,Gcc)上的几个输出结果为 u_{17}=7.2356654170119432,u_{20}=98.350416551346285, u_{30}=99.999999999998948.它表明:从 u_{17} 开始,程序的输出结果中不再含有正确有效数字,并且从 u_{20} 开始,结果就开始非常接近于 100[2].

```
#include ⟨stdio.h⟩
int main(void)
{
   double u, v, w;
   int i, max;

   printf(“n=”);
   scanf(“%d”,&max);
   printf(“u1=”);
   scanf(“%lf”,&u);
   printf(“u2=”);
   scanf(“%lf”,&v);
   printf(“Computation from 3 to n:\n”);
   for (i=3; i<=max; i++)
      {
        w=111.−1130./v+3000./(v*u);
        u=v;
        v=w;
        printf(“u%d=%1.17g\n”,i,v);
      }
   return θ;
}
```

Fig.1 A C program that is supposed to compute sequence u_n using double-precision arithmetic

图 1 一个 C 程序,它使用双精度计算 u_n 的值

15 年后(即 2004 年),被称为“浮点之父”的图灵奖获得者 Kahan 又给出了迭代的一个变形:

$$\begin{cases} y_1 = 4 \\ y_2 = 4.25 \\ y_n = 108 - \dfrac{815}{y_{n-1}} + \dfrac{1500}{y_{n-1}y_{n-2}} \end{cases} \tag{2}$$

对于该变形,同样,若直接编程,则任何机器均得不到正确结果 5.比如,若在 Intel 302(i386/387 IBM PC 的一种仿造机)上使用 FORTRAN 语言编程计算,则扩展精度下,从 y_{32} 开始,结果均为 100[3].

虽然上述案例均是学者们构造的,但是谁能保证类似错误计算的循环迭代模型不会在实践中(特别是攸关安全的领域)出现呢?更为可怕的是,这种错误很难被发现.首先,程序没有逻辑、语法等静态错误;其次,程序在运行中也没有越界、溢出等动态错误;最后,即使使用高精度计算,也往往得不到正确结果.因此,该类错误计算仅得到个别学者的关注:他们均通过变换公式将循环迭代表示成关于 n 的函数,然后利用该函数分析迭代出错的原因[3,4].有鉴于此,研究循环迭代程序正确的、可信的计算算法是迫切的、必要的.

接下来,第 1 节阐明迭代的错误计算原因.然后,其余部分研究下列循环迭代的可信计算算法:

$$\begin{cases} y_1 = f_1(X), & X = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m) \\ y_i = f_i(y_1, y_2, ..., y_{i-1}), & \text{if } 2 \leqslant i \leqslant n \end{cases} \tag{3}$$

即,讨论如何获得 y_n 误差可控的结果.

1 错误计算原因

造成上述错误迭代计算的原因是,对应函数的错数[5,6]大于 0.下面简介错数的概念以及计算方法.

已知 $f(x)$为一单变元函数.任给一个自变量的取值 x_0,其二进制表示可能导致自变量的实际取值中包含错误

数字.这时,自变量的误差可能传播到函数值 $f(x_0)$,从而使得后者也含有错误有效数字.不妨设 N_x 与 N_f 分别为 x_0 与 $f(x_0)$各自近似值中含有的正确有效数字个数.我们称 N_x-N_f 为 $f(x_0)$(或 $f(x)$关于 x_0)的错数.一般情形下,若错数大于 0,则它代表计算结果中错误有效数字的个数.

比如,设:

$$f(x)=(x-1)^4,x_0=0.9993 \tag{4}$$

在 53 位的双精度下,x_0 变成了二进制下的 0.1111111111010010000111111111001011100100100011101000 1,对应的十进制数为 0.99929999999999996607158436745521612465381622314453125. 这样,N_x=16(理论上,3= $2.\dot{9}=2.999...$.因此,与 3 相比,3.000023 与 2.999923 均具有 5 位正确有效数字.即连续 9 或 0 均为正确数字).即 x_0 的表示值具有 16 位正确有效数字.对于 $f(x_0)$来说,若在 Ubuntu 15.04 的 Gcc 4.9.2 或 Windows 7 的 Visual Studio 2010 下执行语句 “printf(“%1.28f”,pow((double)0.9993−1,4));”, 则可得相同结果,为 2.401000000000466e−013.这样,与正确结果 2.401e−013 相比,它有 N_f=13 位正确有效数字.因此,$(x-1)^4$ 在 0.9993 处的错数为

$$N_x-N_f=16-13=3 \tag{5}$$

由于 53 位的双精度运算相当于十进制下的 16 位有效位数下的运算,因此实质上,上述错数 3 表示 16 位(或双精度下)的结果中包含的错误有效数字的个数.

已知实数 $x\neq0$.若使用科学记数法,不妨设 $x=\pm0.t_1t_2\ldots\times10^m(t_1\neq0,t_i$ 为 0~9 之一),则整数 m 为其指数.比如,0.9993 的指数为 0,2.401e−013 的指数为−12.

若 $f(x)$可导,不妨设 x_0,$f(x_0)$以及 $f'(x_0)$的指数分别为 m_1,m_2 与 m_0,则 $f(x_0)$的错数为 $m_1-m_2+m_0$.其中,误差为 1.

对于式(4)中的函数与自变量来说,不难算得 $m_0=-8$.因此,对应错数为 0−(−12)+(−8)=4,它表示函数值的错误数字比自变量的错误数字多 4 位或 3 位.或更进一步,它表示双精度下,函数值的 16 位计算结果中含有 4 位或 3 位错误数字(实际是 3 位).这也正是式(5)中错数为 3 的根本原因.

对于一个迭代来说,若其具有收敛值(即极限值),则迭代可看作为一个函数,并且存在一个自然数 N,当 $n>N$ 时,$m_1=m_2$.这时,错数等于 $m_1-m_2+m_0=m_0$.比如,对于迭代式(1)来说,它可被近似看作为函数 $111-1130/x+3000/x^2$,并且当自变量取 6 时,函数值也为 6.因此 $m_1=m_2=1$.不难验证,上述函数在 6 的导数值约为 3.6.因此,错数为 $m_0=1$.这样,16 次迭代就可能将 16 位正确有效位数消耗殆尽,这也正是图 1 中程序的 u_{17} 不再含有正确有效数字的原因.另外,容易验证:不论是在 Gcc 4.9.2 下还是 Visual Studio 2010 下,均是从 u_{17} 开始,不再含有正确有效的数字.

同样,类似的,容易求得式(2)对应函数的错数也为 1.因此,几乎任何系统任何精度下,迭代计算均出错.

由上可知,构造这种计算机计算错误的循环迭代并不是件难事.首先,给出一个函数 $f(x)$,然后令 $x=f(x)$,再进一步获得满足上式的一个解 x_0,最后选择合适的初值(以便使得极限值是 x_0).这样,只要 $f'(x_0)$的指数大于 0,那么迭代必定会出错.

比如,下面这两个迭代:

$$\begin{cases}x_1=12.3\\ x_n=212.3-2460/x_{n-1}\end{cases}\text{与}\begin{cases}y_1=0.5\\ y_n=\sin(121*\arcsin(y_{n-1}))\end{cases} \tag{6}$$

前者沿用 Muller 与 Kahan 的思路,是一个更为简单的迭代.容易验证,其每一次迭代的值均等于初值 x_1= 12.3.然而容易算得,其对应函数的错数为 2.因此,双精度下,前 7 次迭代结果(从 x_2 到 x_8)一般含有正确有效数字,而第 16 次迭代的结果(即 x_{17})一定没有正确有效数字.事实上,Gcc 4.9.2 与 Visual Studio 2010 的结果中,分别从 x_{14} 与 x_{15} 开始,不再含有正确有效的数字.

对于后者来说,同样容易知道,其每一个 y_n 均为 0.5,对应错数为 3.这样,每次迭代的结果中,即使仅有 2 位错误数字,那么含有正确有效数字的也仅仅是 y_1 到 y_8.因此,y_9 中一定没有正确有效数字.事实上,无论 Gcc 4.9.2 还是 Visual Studio 2010,均是从 y_9 开始,结果中不再含有正确有效数字.

2 向后精度分析

本节讨论函数求值的反问题:已知 $f(x_1,x_2,\ldots,x_n)$是关于 n 个变元的函数,$\tilde{y}_i$ 是实数 y_i 满足精度$\varepsilon_i(>0)$的近似值,即 $|\tilde{y}_i-y_i|<\varepsilon_i(1\leqslant i\leqslant n)$,而 y 是 $f(\tilde{y}_1,\tilde{y}_2,...,\tilde{y}_n)$ 的一个计算值.对于任意的一个误差限$\varepsilon(>0)$,如何判断$|y-f(y_1,y_2,\ldots,y_n)|<\varepsilon$是否成立?若不成立,需要提高哪些 $\tilde{y}_i$ 的精度?即缩小哪些ε_i?并且如何缩小?

比如,已知函数 $f(x_1,x_2)=\sin(1000x_1)+x_1x_2$ 与自变量值 $Y=(y_1,y_2)=(\pi,\sqrt{2})$.虽然我们可以获得 $f(y_1,y_2)$任意精度的值,但是在计算过程中,我们并没有使用到真正的 y_1 与 y_2.因为它们均是无理数,即,我们只是使用了它们的近似值.反过来,任给类似的一对近似值,我们能够判断其是否足够精确,以至利用它们可以获得函数误差可控的值(参见第 2.2 节).

2.1 算 法

设 f 是一个算术表达式,它的准确值是一个实数.我们用 $\tilde{f}$ 或 $(\tilde{f},\varepsilon)$ 以及$[\![f]\!]_\varepsilon$来表示 f 的计算值,其中,后二者分别说明 $|\tilde{f}-f|<\varepsilon$ 与$|[\![f]\!]_\varepsilon-f|<\varepsilon$.这时,$\varepsilon>0$ 代表运算时的误差限(有时也称作精度).若 $f>0$,定义函数$\delta(f)=\{0.1^n|0.1^n\leqslant f<0.1^{n-1},n\in Z\}$,例如,$\delta(0.5)=0.1,\delta(0.00234)=0.001$.最后,设 $\vec{D}$ 是一个四元组,我们用函数 $Fst(\vec{D}),Snd(\vec{D})$,$Thd(\vec{D}),Fth(\vec{D})$ 分别获取它的第 1~第 4 个分量.

对于一个算术表达式,假设可以获得其任意精度的值.而对于两个常数(位数有限的实数,下同)的和、差或乘积,也可以获得准确值.

根据这些定义与假设,我们给出主算法(算法 0).它检验 $Y(Y$代表$(y_1,y_2,\ldots,y_n)$,下同)的计算值 $\tilde{Y}$ 的精度是否精确到当用 $\tilde{Y}$ 代替 Y 后,也能获得 $f(Y)$的一个计算值 y,使得$|y-f(Y)|<\varepsilon$.

算法 0. *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)|1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon$).

Require: $\{(\tilde{y}_i,\varepsilon_i)|\tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f(x_1,x_2,\ldots,x_n),\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 *enough*=true,则$|y-f(y_1,y_2,\ldots,y_n)|<\varepsilon$;否则,当 *enough*=false 时,$\tilde{y}_k$ 的精度ε_k不能满足计算的要求,其精度需要提高至ε_0.

1: **if** f 是算术表达式(可以包含π与 e),或若干常数与若干变元的和,或变元与常数相乘或相除,或两个变元相乘或相除 **then**

2: $\quad\vec{D}\Leftarrow Sub1\_AccurateEnough(\{(\tilde{y}_i,\varepsilon_i)|1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon)$;

3: **else if** $f=f_1\times f_2$, f_1/f,$\sin(f_1)$,$\arctan(f_1)$,$\ln(f_1)$,e^{f_1} 之一 **then**

4: $\quad\vec{D}\Leftarrow Sub2\_AccurateEnough(\{(\tilde{y}_i,\varepsilon_i)|1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon)$;

5: **else if** $f=-f_1$,$\sum_{i=1}^{n_1}f_i$, $f_1^{f_2}$或公式(34)中之一 **then**

6: $\quad\vec{D}\Leftarrow Sub3\_AccurateEnough(\{(\tilde{y}_i,\varepsilon_i)|1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon)$;

7: **end if**

8: **return** $\vec{D}$;

这个主算法调用了 3 个子算法 *Sub*1_,*Sub*2_,*Sub*3_*AccurateEnough*,分别代表递归计算的“出口”算法、利用特定子算法进行的计算以及利用公式进行的递归计算.由于它们与上述主算法拥有相同的输入(require)、输出(ensure)以及返回(return $\vec{D}$)项,因此为了节省篇幅,在后面的算法中,将省去此 3 项内容.

2.1.1 “出口”子算法

如果 f 不含变元,即它是一个算术表达式,那么由假设,算法直接返回它的计算值.如果 f 表示为若干常数与变元的和,那么若对应自变量计算值的误差的和小于预先设定的误差限,则返回和的结果;否则,平均分配误差,并返回误差最大者的下标.如果 f 等于一个变元乘以或除以一个常数,那么若对应计算值的精度与此常数的积或商的值小于预先设定的误差限,则返回它们的值;否则,在精度为“对应计算值的精度与常数的商或积”的前提下,重新计算对应计算值.

命题 2.1. 已知 y_i,y_j 是两个未知实数,$\varepsilon>0$.存在算法,可以获得 y_i,y_j 任意精度的计算值.

定理 2.1. 若命题 2.1 成立,则存在算法,对于 y_i,y_j 任意给定的一对计算值 $(\tilde{y}_i,\varepsilon_i),(\tilde{y}_j,\varepsilon_j)$,或者可以判定:

$$|\tilde{y}_i\times\tilde{y}_j-y_i\times y_j|<\varepsilon \tag{7}$$

成立,或者根据这一对计算值可以给出一对新的计算值,该计算值满足式(7).

证明:令:

$$\varepsilon'=\varepsilon_i\times(|\tilde{y}_j|+\varepsilon_j) \tag{8}$$

$$\varepsilon''=\varepsilon_j\times(|\tilde{y}_i|+\varepsilon_i) \tag{9}$$

$$Mul\_bound=\varepsilon'+\varepsilon''=\varepsilon_i\times(|\tilde{y}_j|+\varepsilon_j)+\varepsilon_j\times(|\tilde{y}_i|+\varepsilon_i) \tag{10}$$

则有:

$$|\tilde{y}_i\times\tilde{y}_j-y_i\times y_j|\leqslant|(\tilde{y}_i-y_i)\times y_j|+|(\tilde{y}_j-y_j)\times\tilde{y}_i|\leqslant\varepsilon_i\times|y_j|+\varepsilon_j\times|\tilde{y}_i|\leqslant\varepsilon'+\varepsilon''=Mul\_bound \tag{11}$$

这样,若:

$$Mul\_bound<\varepsilon \tag{12}$$

则式(7)成立.

若式(12)不成立,下面分两种情形来讨论:

(1) $\varepsilon'>\varepsilon''$

若$\varepsilon'>\varepsilon''$,则$\varepsilon'>\varepsilon/2$(否则式(12)成立).这时,令 $\varepsilon_i=\delta(\varepsilon/(2(|\tilde{y}_j|+\varepsilon_j)))$,并在此新的更高精度下计算 y_i 新的值,则这个新值(不依赖于 y_j 的计算值而始终)满足下列不等式:

$$|(\tilde{y}_i-y_i)\times y_j|\leqslant\varepsilon/(2(|\tilde{y}_j|+\varepsilon_j))\times y_j<\varepsilon/2 \tag{13}$$

如果将 y_i 新的值代入式(9)后有$\varepsilon''\leqslant\varepsilon/2$,则结论成立;否则,利用新的 $(\tilde{y}_i,\varepsilon_i)$,令 $\varepsilon_j=\delta(\varepsilon/(2(|\tilde{y}_i|+\varepsilon_i)))$,并在此新的更高精度下计算一个 y_j 新的值,则这个新值满足下列不等式:

$$|(\tilde{y}_j-y_j)\times\tilde{y}_i|<\varepsilon/(2(|\tilde{y}_i|+\varepsilon_i))\times\tilde{y}_i<\varepsilon/2 \tag{14}$$

这样,由式(11)、式(13)以及式(14)知,式(7)成立.

(2) $\varepsilon'\leqslant\varepsilon''$

由于两个变量相乘具有对称性,即 $x_i\times x_j=x_j\times x_i$.因此,只要将式(11)改写为:

$$|\tilde{y}_i\times\tilde{y}_j-y_i\times y_j|\leqslant|(\tilde{y}_i-y_i)\times\tilde{y}_j|+|(\tilde{y}_j-y_j)\times y_i|,$$

则类似于第 1 种情形,可以获得一对新的计算值,满足式(7).证明完毕. □

定理 2.2. 若命题 2.1 成立,并且 $y_j\neq0$,则存在算法,对于 y_i,y_j 任意给定的一对计算值 $(\tilde{y}_i,\varepsilon_i),(\tilde{y}_j,\varepsilon_j)$,或者可以判定:

$$|[\![\tilde{y}_i/\tilde{y}_j]\!]_{\varepsilon/2}-y_i/y_j|<\varepsilon \tag{15}$$

成立,或者可以给出一对新的计算值,该计算值满足式(15).

证明:下面验证,若计算值满足下面 3 个条件:

$$|\tilde{y}_j|>2\varepsilon_j,|\varepsilon_j\times(|\tilde{y}_i|+\varepsilon_i)/(\tilde{y}_j\times(|\tilde{y}_j|-\varepsilon_j))|<\varepsilon/4,\varepsilon_i/|\tilde{y}_j|<\varepsilon/4 \tag{16}$$

则有:

$$|\tilde{y}_i/\tilde{y}_j-y_i/y_j|<\varepsilon/2 \tag{17}$$

从而式(15)成立:$|[\![\tilde{y}_i/\tilde{y}_j]\!]_{\varepsilon/2}-y_i/y_j|\leqslant|[\![\tilde{y}_i/\tilde{y}_j]\!]_{\varepsilon/2}-\tilde{y}_i/\tilde{y}_j|+|\tilde{y}_i/\tilde{y}_j-y_i/y_j|<\varepsilon/2+\varepsilon/2=\varepsilon$.

首先,我们有:

$$|\tilde{y}_i/\tilde{y}_j-y_i/y_j|\leqslant|(\tilde{y}_i\times y_j-y_i\times y_j)/(\tilde{y}_j\times y_j)|+|(y_i\times y_j-y_i\times\tilde{y}_j)/(\tilde{y}_j\times y_j)|<\varepsilon_1+\varepsilon_2 \tag{18}$$

其中,$\varepsilon_1=|\varepsilon_i/\tilde{y}_j|,\varepsilon_2=|(y_i\times\varepsilon_j)/(\tilde{y}_j\times y_j)|$.然后,由式(16)中第 1 个不等式可以推出:$|y_j|>|\tilde{y}_j|-\varepsilon_j>0$.这样,由第 2 个不等式又能够得出:$\varepsilon_2<\varepsilon/4$.同时,由第 3 个不等式有$\varepsilon_1<\varepsilon/4$.因此,由式(18)知,式(17)成立.

若式(16)中第 1 个不等式不成立,那么由于$|y_j|>0$,因此,通过不停地提高精度,比如多次执行$\varepsilon_j=\delta(\varepsilon_j/10)$,我们总能找到一个满足条件的计算值;同样道理,对于式(16)中第 2 个不等式,只要不停地缩小ε_j 的值,则可得到满足这个不等式的一对新的计算值;最后,若第 3 个不等式不成立,那么令 $\varepsilon_i=\delta(|\tilde{y}_j|\varepsilon/4)$,则有不等式成立.证

明完毕. □

由上面的讨论,我们可得主算法的“出口”子算法.其中,f 是算术表达式(可以包含π与 e),或若干常数与若干变元的和,或变元与常数相乘或相除,或两个变元相乘或相除.

算法 1. *Sub*1_*AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon$).

1: $enough\Leftarrow$true;
2: **if** f 是算术表达式 **then**
3: $y\Leftarrow[\![f]\!]_\varepsilon$;
4: **else if** $f=\sum_{i=1}^{r}c_i+\sum_{i=1}^{l}x_{d_i}$,其中,$r\geqslant 0, l\geqslant 0, l+r>0, \{d_1,d_2,...,d_l\}\subseteq\{1,2,...,n\}$,每个$c_i$为常数 **then**
5: **if** $(l=0)\vee(\sum_{i=1}^{l}\varepsilon_{d_i}\leqslant\varepsilon)$ **then**
6: $y\Leftarrow\sum_{i=1}^{r}c_i+\sum_{i=1}^{l}\tilde{y}_{d_i}$;
7: **else**
8: $(k,\varepsilon_0,enough)\Leftarrow(\{\varepsilon_{d_1},\varepsilon_{d_2},...,\varepsilon_{d_l}\}$中最大者的下标$,\delta(\varepsilon/l),\text{false})$;
9: **end if**
10: **else if** f 表示为 x_i 与 d 的乘积(其中,$d\in\{c,1/c\}$,c 为非 0 常数) **then**
11: **if** $d\times\varepsilon_i<\varepsilon$ **then**
12: $y\Leftarrow[\![d\times\tilde{y}_i]\!]_\varepsilon$;
13: **else**
14: $(k,\varepsilon_0,enough)\Leftarrow(i,\delta(\varepsilon/d),\text{false})$;
15: **end if**
16: **else if** $f=x_i\times x_j$ **then**
17: **if** $\varepsilon_i\times(|\tilde{y}_j|+\varepsilon_j)+\varepsilon_j\times(|\tilde{y}_i|+\varepsilon_i)<\varepsilon$ **then**
18: $y\Leftarrow\tilde{y}_i\times\tilde{y}_j$;
19: **else**
20: $(k,\varepsilon_0,enough)\Leftarrow(d_1,\delta(\varepsilon/(2(|\tilde{y}_{d_2}|+\varepsilon_{d_2}))),\text{false})$,
其中:若 $\varepsilon_i(|\tilde{y}_j|+\varepsilon_j)>\varepsilon_j(|\tilde{y}_i|+\varepsilon_i)$,则 $d_1=i,d_2=j$;否则,$d_1=j,d_2=i$;
21: **end if**
22: **else if** $f=x_i/x_j$ **then**
23: // $\tilde{y}_i$或$\tilde{y}_j$的误差限可以为0.这时,y_i或y_j为常数 .
24: **if** $|\tilde{y}_j|>2\varepsilon_j$ **then**
25: **if** $4\varepsilon_j(|\tilde{y}_i|+\varepsilon_i)<\varepsilon|\tilde{y}_j|(|\tilde{y}_j|-\varepsilon_j)$ **then**
26: **if** $4\varepsilon_i\lessdot|\tilde{y}_j|\varepsilon$ **then**
27: $y\Leftarrow[\![\tilde{y}_i/\tilde{y}_j]\!]_{\varepsilon/2}$;
28: **else**
29: $(k,\varepsilon_0,enough)\Leftarrow(i,\delta(|\tilde{y}_j|\varepsilon/4),\text{false})$;
30: **end if**
31: **else**
32: $(k,\varepsilon_0,enough)\Leftarrow(j,\min\{\delta((\varepsilon|\tilde{y}_j|(|\tilde{y}_j|-\varepsilon_j))/(4(|\tilde{y}_i|+\varepsilon_i))),\delta(\varepsilon_j/10)\},\text{false})$;
33: **end if**
34: **else**
35: $(k,\varepsilon_0,enough)\Leftarrow(j,\min\{\delta(|\tilde{y}_j|/2),\delta(\varepsilon_j/10)\},\text{false})$;
36: **end if**

37: **end if**

38: $\vec{D} \Leftarrow (y,k,\varepsilon_0,enough)$;

2.1.2 特定函数子算法

下面给出被主函数调用的第 2 个子算法.其中,f 为 $f_1\times f_2$,f_1/f_2,$\sin(f_1)$,$\arctan(f_1)$,$\ln(f_1)$或 e^{f_1}.

算法 2. *Sub2_AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon$).

1: **if** $f=f_1\times f_2$ **then**

2: $\vec{D} \Leftarrow AccurateEnough\_Mul(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1, f_2,\varepsilon)$;

3: **else if** $f=f_1/f_2$ **then**

4: $\vec{D} \Leftarrow AccurateEnough\_Div(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1, f_2,\varepsilon)$;

5: **else if** $f=\sin(f_1)$ **then**

6: $\vec{D} \Leftarrow AccurateEnough\_Sin(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1,\varepsilon)$;

7: **else if** $f=\arctan(f_1)$ **then**

8: $\vec{D} \Leftarrow AccurateEnough\_ArcTan(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1,\varepsilon)$;

9: **else if** $f=\ln(f_1)$ **then**

10: $\vec{D} \Leftarrow AccurateEnough\_Ln(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1,\varepsilon)$;

11: **else if** $f=e^{f_1}$ **then**

12: $\vec{D} \Leftarrow AccurateEnough\_Exp(\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1,\varepsilon)$;

13: **end if**

算法 2 中涉及到了 6 个子算法,分别为乘法运算子算法、除法运算子算法以及正弦函数、自然对数函数、反正切函数与指数函数子算法.

命题 2.2. 已知函数 $f(X)$,$f_1(X)$与 $f_2(X)$以及自变量 Y.存在一个算法,对于任意的误差限$\varepsilon>0$ 与 Y 的一个计算值 $\tilde{Y}$ 以及 $f(X)$或任意一个 $f_i(X)(1\leqslant i\leqslant 2)$,或者能够判定,利用此计算值可以获得一个值 z,使得:

$$|z-u|<\varepsilon(u\in\{f(Y),f_1(Y),f_2(Y)\}) \tag{19}$$

成立,或者可以获得一个更为精确的计算值,利用此新的计算值可以获得满足式(19)的一个新的 z.

(1) 乘法运算

定理 2.3. 若命题 2.2 成立,那么存在一个算法,对于 Y 的一个计算值 $\tilde{Y}$ 与ε,或者能够判定,利用此计算值可以获得一个值 y,使得:

$$|y-f_1(Y)\times f_2(Y)|<\varepsilon \tag{20}$$

成立,或者可以获得一个更为精确的计算值,使得利用此新的计算值,可以获得一个满足式(20)的 y.

证明:设$|z_1-f_1(Y)|<0.1$,$|z_2-f_2(Y)|<0.1$.这时:

$$|z_1|+0.1>|f_1(Y)|,\ |z_2|+0.1>|f_2(Y)| \tag{21}$$

令:

$$\varepsilon_1=\delta(\varepsilon/(2(|z_2|+0.1))),\ \varepsilon_2=\delta(\varepsilon/(2(|z_2|+0.1+\varepsilon_1))) \tag{22}$$

则存在 z_3 与 z_4,使得:

$$|z_3-f_1(Y)|<\varepsilon_1,\ |z_4-f_2(Y)|<\varepsilon_2 \tag{23}$$

其中,式(23)的第 1 个不等式又意味着:

$$|z_3|<|f_1(Y)|+\varepsilon_1 \tag{24}$$

由式(23)、式(22)、式(21)的第 2 个不等式、式(24)以及式(21)的第 1 个不等式,我们有下列不等式:

$$|z_3\times z_4-f_1(Y)\times f_2(Y)|\leqslant|z_3-f_1(Y)|\times|f_2(Y)|+|z_4-f_2(Y)|\times|z_3|<\varepsilon_1\times|f_2(Y)|+\varepsilon_2\times|z_3|\leqslant$$

$$\frac{\varepsilon}{2(|z_2|+0.1)}\times|f_2(Y)|+\frac{\varepsilon}{2(|z_1|+0.1+\varepsilon_1)}\times|z_3|<\frac{\varepsilon}{2}+\frac{\varepsilon}{2(|z_1|+0.1+\varepsilon_1)}\times(|f_1(Y)|+\varepsilon_1)<\frac{\varepsilon}{2}+\frac{\varepsilon}{2}=\varepsilon$$

这样,只要令 $y=z_3\times z_4$,则有式(20)成立.证明完毕. □

由上述定理,可得下列算法 3(与算法 4).它检验 Y 的计算值 $\tilde{Y}$ 的精度是否精确到当用 $\tilde{Y}$ 代替 Y 后,也能获得 $f_1(Y)\times f_2(Y)$的一个计算值 y,使得$|y-f_1(Y)\times f_2(Y)|<\varepsilon$.

算法 3. *AccurateEnough_Mul*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, $f_1(x_1,x_2,...,x_n)$, $f_2(x_1,x_2,...,x_n)$, ε).

Require: $\{(\tilde{y}_i,\varepsilon_i)\,|\,\tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$, $f_1(x_1,x_2,\ldots,x_n)$, $f_2(x_1,x_2,\ldots,x_n)$, $\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 *enough*=true,则$|y-f_1(Y)\times f_2(Y)|<\varepsilon$;否则,当 *enough*=false 时, $\tilde{y}_k$ 的精度ε_k 不能满足计算的要求,其精度需要提高至ε_0.

1: $\varepsilon'\Leftarrow 0.1$;
2: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f_1,ε') ;
3: $z_1\Leftarrow\tilde{f}_1$;
4: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f_2,ε') ;
5: $z_2\Leftarrow\tilde{f}_2$;
6: $\varepsilon_1\Leftarrow\delta(\varepsilon/(2(|z_2|+0.1)))$;
7: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f_1,ε_1) ;
8: $z_3\Leftarrow\tilde{f}_1$;
9: $\varepsilon_2\Leftarrow\delta(\varepsilon/(2(|z_2|+0.1+\varepsilon_1)))$;
10: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f_2,ε_2) ;
11: $z_4\Leftarrow\tilde{f}_2$;
12: $y\Leftarrow z_3\times z_4$;
13: **return** (y,0,0,true);

算法 4. *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, $f(x_1,x_2,...,x_n)$, ε)——这是一个内联函数,要嵌入到其他函数中.

Require: $\{(\tilde{y}_i,\varepsilon_i)\,|\,\tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$, $f(x_1,x_2,\ldots,x_n)$, $\varepsilon>0$;

Ensure:计算 *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f, ε) 的值.若其结果的第 4 个分量为 false,则返回该结果(4 元组);否则,将第 1 个分量赋给 $\tilde{f}$.

1: $\vec{D}\Leftarrow$ *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, f, ε);
2: **if** $Fth(\vec{D})=$ false **then**
3: **return** $\vec{D}$;
4: **else**
5: $\tilde{f}\Leftarrow Fst(\vec{D})$; //获取 $\vec{D}$ 的第 1 个分量.变量名是由函数的第 2 个参数名与波浪号组合而成.
6: **end if**

(2) 除法运算

定理 2.4. 若命题 2.2 成立,并且 $f_2(Y)\neq 0$,那么存在一个算法,对于 Y 的一个计算值 $\tilde{Y}$ 与ε,或者能够判定,利用此计算值可以获得一个值 y,使得:

$$|y-f_1(Y)/f_2(Y)|<\varepsilon \tag{25}$$

成立,或者可以获得一个更为精确的计算值,使得利用此新的计算值,可以获得一个满足式(25)的 y.

证明:设 $\bar{z}_1$ 是$|f_1(Y)|$的一个上界.类似于式(16),不难验证:对于 $f_i(Y)$的计算值$(z_i,\varepsilon_i)(1\leqslant i\leqslant 2)$来说,只要满足下列 3 个条件:

$$|z_2|>2\varepsilon_2,\ |\varepsilon_2\times\bar{z}_1/(z_2\times(|z_2|-\varepsilon_2))|<\varepsilon/4,\ \varepsilon_1/|z_2|<\varepsilon/4 \tag{26}$$

则有$|[\![z_1/z_2]\!]_{\varepsilon/2}-f_1(Y)/f_2(Y)|<\varepsilon$.这样,只要令 $y=[\![z_1/z_2]\!]_{\varepsilon/2}$,则有式(25)成立.证明完毕. □

算法 5. *AccurateEnough_Div*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}$, $f_1(x_1,x_2,...,x_n)$, $f_2(x_1,x_2,...,x_n)$, ε).

Require: $\{(\tilde{y}_i,\varepsilon_i)\mid \tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f_1(x_1,x_2,\ldots,x_n)$,$f_2(x_1,x_2,\ldots,x_n)$,$f(y_1,y_2,\ldots,y_n)\neq 0$,$\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 *enough*=true,则$|y-f_1(Y)/f_2(Y)|<\varepsilon$;否则,当 *enough*=false 时, $\tilde{y}_k$ 的精度ε_k 不能满足计算的要求,其精度需要提高至ε_0.

1: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, f_1,0.1) ;

2: $\overline{z}_1\Leftarrow|\tilde{f}_1|+0.1$;

3: $\varepsilon_2\Leftarrow 0.1$;

4: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, f_2,ε_2) ;

5: **while** ($|\tilde{f}_2|\leqslant 2\times\varepsilon_2$) **do**;

6: $\varepsilon_2\Leftarrow 0.1\times\varepsilon_2$;

7: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, f_2,ε_2) ;

8: **end while**

9: **while** ($|4\times\overline{z}_1\times\varepsilon_2|\geqslant|\tilde{f}_2\times(|\tilde{f}_2|-\varepsilon_2)\times\varepsilon|$) **do**

10: $\varepsilon_2\Leftarrow 0.1\times\varepsilon_2$;

11: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, f_2,ε_2) ;

12: **end while**

13: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, $f_1,\delta(|\tilde{f}_2|/4\times\varepsilon)$) ;

14: $y\Leftarrow[\![\tilde{f}_1/\tilde{f}_2]\!]_{\varepsilon/2}$;

15: **return** (y,0,0,true);

(3) 正弦函数

定理 2.5. 若命题 2.2 成立,那么存在一个算法,对于 Y 的一个计算值 $\tilde{Y}$ 与ε,或者能够判定,利用此计算值可以获得一个值 y,使得:

$$|y-\sin(f(Y))|<\varepsilon \tag{27}$$

成立,或者可以获得一个更为精确的计算值,使得利用此新的计算值,可以获得一个满足式(27)的 y.

证明:首先,对于误差限$\varepsilon/2$ 来说,由假设知,我们或者可以判定,利用 $\tilde{Y}$ 可以获得一个值 z,满足:

$$|z-f(Y)|<\varepsilon/2 \tag{28}$$

或者可以获得一个更为精确的计算值,利用该计算值可以获得满足式(28)的一个 z.

若式(28)成立,则有:

$$|[\![\sin(z)]\!]_{\varepsilon/2}-\sin(f(Y))|\leqslant|[\![\sin(z)]\!]_{\varepsilon/2}-\sin(z)|+|\sin(z)-\sin(f(Y))|<\varepsilon/2+|z-f(Y)|<\varepsilon \tag{29}$$

这样,只要令 $y=[\![\sin(z)]\!]_{\varepsilon/2}$,则有式(27)成立.

若式(28)不成立,那么我们可以获得一个更为精确的计算值 $\tilde{Y}$,利用它可以获得一个新的 z,满足式(28).这样,式(29)也成立.因此,新的 z 的函数值$[\![\sin(z)]\!]_{\varepsilon/2}$ 使得式(27)成立.证明完毕. □

算法 6. *AccurateEnough_Sin*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, $f(x_1,x_2,...,x_n),\varepsilon$).

Require: $\{(\tilde{y}_i,\varepsilon_i)\mid \tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f(x_1,x_2,\ldots,x_n)$,$\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 *enough*=true,则$|y-\sin(f(y_1,y_2,\ldots,y_n))|<\varepsilon$;否则,当 *enough*=false 时, $\tilde{y}_k$ 的精度ε_k 不能满足计算的要求,其精度需要提高至ε_0.

1: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, $f,\varepsilon/2$) ;

2: $y\Leftarrow[\![\sin(\tilde{f})]\!]_{\varepsilon/2}$;

3: **return** (y,0,0,true);

(4) 反正切函数

由于 $\arctan(x)'=1/(1+x^2)\leqslant 1$,因此类似于正弦函数的算法 6,我们有下列算法 7.

算法 7. *AccurateEnough_ArcTan*($\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\}$, $f(x_1,x_2,...,x_n),\varepsilon$).

Require: $\{(\tilde{y}_i,\varepsilon_i)\mid \tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f(x_1,x_2,\ldots,x_n)$,$\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 $enough$=true,则$|y-\arctan(f(y_1,y_2,\ldots,y_n))|<\varepsilon$;否则,当 $enough$=false 时, $\tilde{y}_k$ 的精度ε_k不能满足计算的要求,其精度需要提高至ε_0.

1: $Calculate\_and\_Judge(\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\},f,\varepsilon/2)$;

2: $y\Leftarrow[\![\arctan(\tilde{f})]\!]_{\varepsilon/2}$;

3: **return** $(y,0,0,\text{true})$;

(5) 自然对数函数

定理 2.6. 若命题 2.2 成立,并且 $f(Y)>0$,那么存在一个算法,对于 Y 的一个计算值 $\tilde{Y}$ 与ε,或者能够判定,利用此计算值可以获得一个值 y,使得:

$$|y-\ln(f(Y))|<\varepsilon \tag{30}$$

成立,或者可以获得一个更为精确的计算值,使得利用此新的计算值,可以获得一个满足式(30)的 y.

证明:首先,由拉格朗日中值定理,在 z 与 $f(Y)$之间存在一个 c,使得$|\ln(z)-\ln(f(Y))|=|(z-f(Y))/c|$.

已知不等式组:

$$|z-f(Y)|<\varepsilon',z>2\varepsilon'\leqslant(z-\varepsilon')\varepsilon \tag{31}$$

其中,$\varepsilon'=\varepsilon$.

若式(31)成立,则有$|[\![\ln(z)]\!]_{\varepsilon/2}-\ln(f(Y))|\leqslant|[\![\ln(z)]\!]_{\varepsilon/2}-\ln(z)|+|\ln(z)-\ln(f(Y))|<\varepsilon/2+|(z-f(Y))/c|<\varepsilon/2+\varepsilon'/(z-\varepsilon')\leqslant\varepsilon$.这时,只要令 $y=[\![\ln(z)]\!]_{\varepsilon/2}$,则有式(30)成立.

否则,若式(31)不成立,则令$\varepsilon'=0.1^n\varepsilon(n\in\mathbb{Z})$.这时,由 $f(Y)>0$ 与假设知:只要 n 足够大,总可以获得一个计算值$(\tilde{Y},\varepsilon')$ 以及对应的 z,使得式(31)成立.这样,新获得的 z 的函数值$[\![\ln(z)]\!]_{\varepsilon/2}$能够满足式(30).证明完毕. □

算法 8. $AccurateEnough\_Ln(\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\},f(x_1,x_2,...,x_n),\varepsilon)$.

Require: $\{(\tilde{y}_i,\varepsilon_i)\mid \tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f(x_1,x_2,\ldots,x_n)$,$f(y_1,y_2,\ldots,y_n)>0$,$\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 $enough$=true,则$|y-\ln(f(y_1,y_2,\ldots,y_n))|<\varepsilon$;否则,当 $enough$=false 时, $\tilde{y}_k$ 的精度ε_k不能满足计算的要求,其精度需要提高至ε_0.

1: $\varepsilon'\Leftarrow 0.1\varepsilon$;

2: $Calculate\_and\_Judge(\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\},f,\varepsilon')$;

3: **while** $(|\tilde{f}|\leqslant 2\varepsilon'\vee 2\varepsilon'>(|\tilde{f}|-\varepsilon')\varepsilon)$ **do**

4: $\varepsilon'\Leftarrow 0.1\varepsilon'$;

5: $Calculate\_and\_Judge(\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\},f,\varepsilon')$;

6: **end while**

7: $y\Leftarrow[\![\ln(\tilde{f})]\!]_{\varepsilon/2}$;

8: **return** $(y,0,0,\text{true})$;

(6) 指数函数

定理 2.7. 若命题 2.2 成立,那么存在一个算法,对于 Y 的一个计算值 $\tilde{Y}$ 与ε,或者能够判定,利用此计算值可以获得一个值 y,使得:

$$|y-e^{f(Y)}|<\varepsilon \tag{32}$$

成立,或者可以获得一个更为精确的计算值,使得利用此新的计算值,可以获得一个满足式(32)的 y.

证明:已知不等式:

$$|z-f(Y)|<\varepsilon' \tag{33}$$

其中, $\varepsilon'=\min\{0.1,\delta(\varepsilon/(2([\![e^{[\![f(\tilde{Y})]\!]_{0.1}+0.1}]\!]_{0.2}+0.2)))\}$.

若式(33)成立,则有$|[\![e^z]\!]_{\varepsilon/2}-e^{f(Y)}|\leqslant|[\![e^z]\!]_{\varepsilon/2}-e^z|+|e^z-e^{f(Y)}|<\varepsilon/2+|e^z-e^{f(Y)}|$.由拉格朗日定理,在$z$与$f(Y)$之间存在一个$c$,使得$|e^z-e^{f(Y)}|=|(z-f(Y))e^c|\leqslant|(y-f(Y))e^{z+0.1}|<\varepsilon'([\![e^{z+0.1}]\!]_{0.2}+0.2)<\varepsilon/2$.这时,只要令 $y=[\![e^z]\!]_{\varepsilon/2}$,则有式(32)成立.

否则,若式(33)不成立,则由假设知:总可以获得一个新的计算值 $\tilde{Y}$,使得由其得到的 z 满足式(33).这样,新获得的 z 的函数值$[\![e^z]\!]_{\varepsilon/2}$能够满足式(32).证明完毕. □

算法 9. *AccurateEnough_Exp*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon$).

Require: $\{(\tilde{y}_i,\varepsilon_i)\,|\,\tilde{y}_i=[\![y_i]\!]_{\varepsilon_i},1\leqslant i\leqslant n\}$,$f(x_1,x_2,\ldots,x_n),\varepsilon>0$;

Ensure:返回一个 4 元组 $\vec{D}=(y,k,\varepsilon_0,enough)$.若 *enough*=true,则 $|y-e^{f(y_1,y_2,...,y_n)}|<\varepsilon$;否则,当 *enough*=false 时,$\tilde{y}_k$ 的精度ε_k不能满足计算的要求,其精度需要提高至ε_0.

1: $\varepsilon'\Leftarrow 0.1$;
2: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f,\varepsilon'$) ;
3: $\varepsilon''\Leftarrow 0.2$;
4: $y\Leftarrow[\![e^{\tilde{f}+\varepsilon'}]\!]_{\varepsilon''}$;
5: $\varepsilon'''\Leftarrow\min\{0.1,\delta(\varepsilon/(2(y+\varepsilon'')))\}$;
6: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f,\varepsilon'''$) ;
7: $y\Leftarrow[\![e^{\tilde{f}}]\!]_{\varepsilon/2}$;
8: **return** (y,0,0,true);

2.1.3 递归计算子算法

首先,对于下面式(34)中的函数,我们有式(35)中对应的等式:

$$\cos(f_1),\tan(f_1),\cot(f_1),\sec(f_1),\csc(f_1),\arcsin(f_1),\arccos(f_1),\operatorname{arccot}(f_1),\log_{f_1} f_2,\sinh(f_1),\cosh(f_1) \tag{34}$$

$$\begin{cases}\sin(\pi/2-f_1),\sin(f_1)/\cos(f_1),\cos(f_1)/\sin(f_1),1/\cos(f_1),1/\sin(f_1),2\arctan(f_1/(1+\sqrt{1-f_1^2}))\\ \pi/2-\arcsin(f_1),\pi/2-\arctan(f_1),\ln(f_1)/\ln(f_2),(e^{f_1}-e^{-f_1})/2,(e^{f_1}+e^{-f_1})/2\end{cases} \tag{35}$$

然后,根据前面的两个子算法,我们可以得出递归计算的子算法.其中,f 为$-f_1$, $\sum_{i=1}^{n_1} f_i$, $f_1^{f_2}$ 或式(34)中之一.

算法 10. *Sub*3_*AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f(x_1,x_2,...,x_n),\varepsilon$).

1: **if** $f=-f_1$ **then**
2: $\vec{D}_1\Leftarrow$ *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_1,\varepsilon$) ;
3: $\vec{D}\Leftarrow(-Fst(\vec{D}_1),Snd(\vec{D}_1),Thd(\vec{D}_1),Fth(\vec{D}_1))$;
4: **else if** $f=\sum_{i=1}^{n_1} f_i$ **then**
5: **for** i=1 to n_1 **do**
6: *Calculate_and_Judge*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, f_i,\delta(\varepsilon/n_1)$) ; //参见算法 4
7: **end for**
8: $\vec{D}\Leftarrow(\sum_{i=1}^{n_1}\tilde{f}_i,0,0,\text{true})$;
9: **else if** $f=f_1^{f_2}$ **then**
10: **if** f_1=0 and f_2>0 **then**
11: $\vec{D}\Leftarrow(0,0,0,\text{true})$;
12: **end if** f_1>0 **then**
13: $\vec{D}\Leftarrow$ *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, e^{f_2\times\ln(f_1)},\varepsilon$)
14: **end if** f_1<0,$f_2=n'/n''$(不可约),n'是偶数,and n''是奇数 **then**
15: $\vec{D}\Leftarrow$ *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, e^{f_2\times\ln(-f_1)},\varepsilon$) ;
16: **end if** f_1<0,$f_2=n'/n''$(不可约),and n'与 n''均是奇数 **then**
17: $\vec{D}\Leftarrow$ *AccurateEnough*($\{(\tilde{y}_i,\varepsilon_i)\,|\,1\leqslant i\leqslant n\}, -e^{f_2\times\ln(-f_1)},\varepsilon$) ;
18: **end if**
19: **end if** f=式(34)中的表达式 **then**

20: $\vec{D} \Leftarrow AccurateEnough(\{(\tilde{y}_i,\varepsilon_i)\mid 1\leqslant i\leqslant n\},z,\varepsilon)$; //$z$ 为式(35)中对应等价表达式.

21: **end if**

2.2 举例说明

已知函数 $f(x_1,x_2)=\sin(1000x_1)+x_1x_2$, $Y=(y_1,y_2)=(\pi,\sqrt{2})$,误差限$\varepsilon=0.1^3$.

例 2.1:假设$\varepsilon_1=0.1^2$,$\varepsilon_2=0.1^3$.这时,我们取 $\tilde{Y}=(\tilde{y}_1,\tilde{y}_2)=(3.14,\ 1.414)$.判断由 $f(\tilde{Y})$ 能否获得一个 y,满足:

$$|y-f(Y)|<\varepsilon \tag{36}$$

首先,平均分配误差:令 $\sin(1000x_1)$与 x_1x_2 的误差限均为:

$$\varepsilon'=\varepsilon/2 \tag{37}$$

然后,对于正弦函数来说,由算法 6 知,其自变量 $1000x_1$ 的误差限应为$\varepsilon''=\varepsilon'/2=\varepsilon/4$;继而,由乘法的相关算法可推出 x_1 的精度(或误差限)应为:

$$\varepsilon'''=\varepsilon''/1000=0.1^6/4 \tag{38}$$

由于$\varepsilon'''<\varepsilon_1$,因此,由 $\tilde{Y}$ 不能确保可以得出一个满足式(36)的 y.这样,建议 $\tilde{y}_1$ 的精度ε_1 提高至式(38)中的值.

例 2.2:设$\varepsilon_1=0.1^7$,$\varepsilon_2=0.1^3$.这时,取 $\tilde{Y}=(\tilde{y}_1,\tilde{y}_2)=(3.1415927,1.414)$.判断由 $f(\tilde{Y})$ 能否获得一个 y,满足式(36).

由例 2.1 知, $\tilde{y}_1$ 的精度已能满足 $\sin(1000x_1)$的精度要求.

下面,我们分析$\varepsilon_1,\varepsilon_2$ 是否能够满足式(37)中对 x_1x_2 的精度要求.

由于 $\varepsilon_1(|\tilde{y}_2|+\varepsilon_2)+\varepsilon_2(|\tilde{y}_1|+\varepsilon_1)=0.1^7\times(|1.414|+0.1^3)+0.1^3\times(|3.1415927|+0.1^7)=0.0031417343>\varepsilon'$,即计算值不满足不等式(12),因此,由 $\tilde{Y}$ 不能保证可以得出一个满足式(36)的 y.

另外,由于 $\varepsilon_1(|\tilde{y}_2|+\varepsilon_2)<\varepsilon_2(|\tilde{y}_1|+\varepsilon_1)$,因此,令 $\varepsilon_2(|\tilde{y}_1|+\varepsilon_1)<\varepsilon'/2$,即 $\varepsilon_2<\varepsilon'/(2\times(|\tilde{y}_1|+\varepsilon_1))=0.000079577...$,因此,我们建议$\varepsilon_2=\delta(0.000079577\ldots)=0.1^5$.

例 2.3:设$\varepsilon_1=0.1^7$,$\varepsilon_2=0.1^5$.这时,取 $\tilde{Y}=(\tilde{y}_1,\tilde{y}_2)=(3.1415927,1.41421)$.由上例(即例 2.2)知:通过 $f(\tilde{Y})$,我们可以获得一个值 y,满足式(36)中的条件.

可以验证, $f(\tilde{Y})=4.4429182...$, $f(Y)=4.4428829...$.从而, $y=[\![f(\tilde{Y})]\!]_{\varepsilon}=4.443$.它在满足 $|y-f(\tilde{Y})|<\varepsilon$ 的同时,也满足不等式(36):$|y-f(Y)|=|4.443-4.4428829\ldots|=0.000117\ldots<\varepsilon$.

3 循环迭代程序的可信计算算法

对于式(3)来说,由 *AccurateEnough* 算法(即算法 0)可得:

定理 3.1. 存在一个算法,对于任意的$\varepsilon>0$,可以获得 y_n 的一个计算值 $\tilde{y}_n$,使得 $|\tilde{y}_n-y_n|<\varepsilon$.

证明:下面使用归纳法进行证明.

当 $n=1$ 时,由于只是计算一个表达式的值,因此我们可以利用一些算法获得其误差可控的值.假设 $n\leqslant k$ 时结论也成立,即,由算法可以获得 $y_1,y_2,\ldots,y_k$ 任意精度的值.再设 $(\tilde{y}_1,\tilde{y}_2,...,\tilde{y}_k)$ 是$(y_1,y_2,\ldots,y_k)$的一个计算值.这时,由 *AccurateEnough* 算法知,我们可以判断其是否足够精确,以至可以利用它获得 $f_{k+1}(y_1,y_2,\ldots,y_k)$误差可控的值.若该值足够精确,那么意味着我们也能够获得 $y_{k+1}=f_{k+1}(y_1,y_2,\ldots,y_k)$给定精度的值.从而定理成立;否则,若该值不够精确,那么利用 *AccurateEnough* 算法,通过提高有关值的精度,总可以找到一个计算值,使得该值足够精确,以至可以获得 $y_{k+1}=f_{k+1}(y_1,y_2,\ldots,y_k)$给定精度的值.从而定理获证.证明完毕. □

根据定理 3.1,我们可得下列主算法.它计算 $y_n=f_n(y_1,y_2,\ldots,y_{n-1})$的值.

算法 11. *Main*($\tilde{x}_1,\tilde{x}_2,...,\tilde{x}_m,f_1,f_2,...,f_n,\varepsilon$).

Require:$m,\{\tilde{x}_i\mid 1\leqslant i\leqslant m\}$,$n,\{y_i|y_1=f_1(x_1,x_2,\ldots,x_m),y_i=f_i(y_1,y_2,\ldots,y_{i-1}),2\leqslant i\leqslant n\},\varepsilon>0$;

Ensure: $\tilde{y}_n\approx y_n,|\tilde{y}_n-y_n|<\varepsilon$;

1: $l\Leftarrow 1$; //初始化

2: $extra\_precision\Leftarrow 0.1^l$; //初始化精度步长

3: $j\Leftarrow 1$; //初始化

4: **while** $j \leqslant n$ **do**
5: $\beta[j] \Leftarrow extra\_precision^{n-j}$;
6: **if** j=1 **then**
7: $\tilde{y}_1 \Leftarrow [\![f_1(\tilde{x}_1,\tilde{x}_2,...,\tilde{x}_m)]\!]_{\varepsilon\times\beta[1]}$; //计算 f_1 精度为 $\varepsilon\times extra\_precision^{n-1}$ 的值.
8: $j \Leftarrow j+1$;
9: **else if** j>1 **then**
10: $(\tilde{y}_j, k, \varepsilon_0, enough) \Leftarrow AccurateEnough(\{(\tilde{y}_i, \beta[i]) \mid 1 \leqslant i \leqslant j-1\}, f_j, \varepsilon \times \beta[j])$;
11: **if** *enough*=true **then**
12: $j \Leftarrow j+1$;
13: **else**
14: $l \Leftarrow l+1$; //修改 l 的值.
15: $extra\_precision \Leftarrow 0.1^l$; //修改精度步长
16: $j \Leftarrow 1$; //重新开始
17: **end if**
18: **end if**
19: **end while**
20: **return** $\tilde{y}_n$;

在上述算法中,为了提高计算效率,我们令 y_i 的精度超过 $y_{i+1}(1 \leqslant i < n)$的精度.并且,所有精度构成等比数列.需要注意的是:若某一次的精度不满足要求,那么一定要像第 16 行那样(提高精度后)从头开始计算.

4 案例分析与验证

对于前述算法,我们已用 C++语言编程实现于 ISReal[7,8].程序及有关文档的下载网址是:http://www.isrealsoft.cn/download.htm.ISReal 的现有版本采用 GMP 的整数运算作为底层运算,并且输入与输出均是字符串,所以采用前述算法的 ISReal 得出的结果是正确结果.除此之外,ISReal 用“*y*”与“:=”分别代表迭代的函数名与赋值符号.比如,式(1)中第 1 个赋值语句可以写成这种形式:*y*1:=2.另外,为了方便操作,ISReal 利用小数位数取代误差限(或精度).比如:若结果的误差限为 0.1^n,则 ISReal 会令结果保留 n 位小数,对应设置函数(或命令)为 *DecimalPlaces*.

例 4.1:计算式(1)中的 u_{30},结果保留 16 位有效数字.

由于迭代的极限值为 6,因此不妨令结果保留 15 位小数.这时,由程序的运行过程可知:只有 u_3 重复计算了 5 次,即 l=5,$extra\_precision$=0.1^5.这样,计算时,$u_i(1 \leqslant i \leqslant 30)$的误差限为 $0.1^{15+(30-i)\times 5}$.最终的输出结果为:u_{30}=6.006786093031206.

下面的图 2 展示了 ISReal 在计算 u_{30} 时的环境界面,其中包括 5 行输入与 1 行输出.

例 4.2:计算式(2)中的 u_{32},结果保留 100 位有效数字.

类似于例 4.1,由于迭代的极限值为 5,因此不妨令结果保留 99 位小数.这时,由程序的运行过程可知:只有 y_3 重复计算了 5 次,即 l=5,$extra\_precision$=0.1^5.这样,计算时,$y_i(1 \leqslant i \leqslant 32)$的误差限为 $0.1^{99+(32-i)\times 5}$.最终输出结果为:y_{32}=4.999999734711331524163448988670387320907181558470424064116020671501994740701184553230083295123968309.

例 4.3:计算式(6)中的 y_9,结果保留 16 位有效数字.

由于迭代的每个值均为 0.5,因此令结果保留 16 位小数.这时,由程序的运行过程可知:只有 y_2 重复计算了 2 次,即 l=2,$extra\_precision$=0.1^2.这样,计算时,$y_i(1 \leqslant i \leqslant 9)$的误差限为 $0.1^{16+(9-i)\times 2}$.最终输出结果为 y_9=0.5.

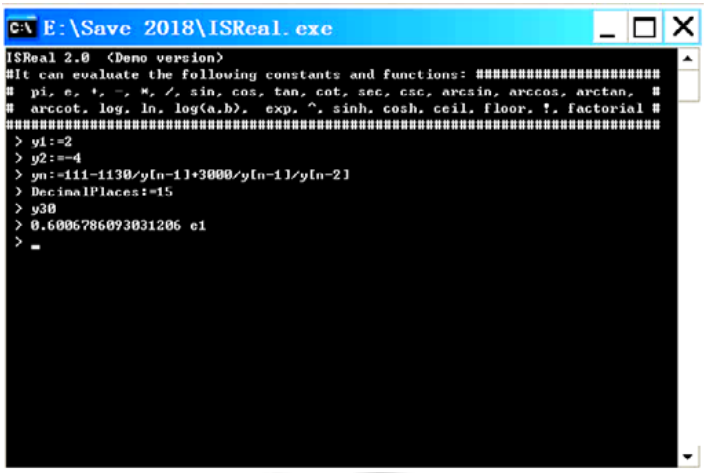

Fig.2 ISReal's interface for computing u_{30} of Eq.(1)

图 2 ISReal 计算迭代式(1)的 u_{30} 时的运行环境

对于上述几个案例来说,均预先知道极限值,因此通过 *DecimalPlaces* 可以对需要计算的小数位数进行一次性设置.若预先不清楚极限值,那么可以通过增大 *DecimalPlaces* 的值来获得极限值.

5 结束语

本文给出了一个通用的算法,利用它可以解决一些用其他方法解决不了的循环迭代难题.对于 Kahan 提出的问题:"How can distress caused by roundoff be diagnosed reliably? How can it be cured?"[3],显然,本文的算法是解决方案之一.通过与本算法(或软件)的计算结果的比较,程序员可以了解或验证其循环迭代程序运行结果的正确与否.

致谢 感谢与日本新潟大学刘学峰博士的讨论.

References:

[1] Muller JM. Arithmétique des Ordinateurs. Paris: Masson, 1989 (in French).

[2] Muller JM, Brisebarre N, Dinechin FD, *et al*. Handbook of Floating-point Arithmetic. Boston: Birkhauser Boston, 2010. 8–10.

[3] Kahan W. How futile are mindless assessments of roundoff in floating-point computation? 2006. http://www.cs.berkeley.edu/~wkahan/Mindless.pdf

[4] Benz F, Hildebrandt A, Hack S. A Dynamic program analysis to find floating-point accuracy problems. In: Proc. of the 2012 ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2012. 453–462.

[5] Zhao SZ. Reasons of miscalculation in floating point arithmetic. Sciencepaper Online. 2017 (in Chinese with English abstract). http://www.paper.edu.cn/releasepaper/content/201707-86

[6] Zhao SZ. CuoShu of six binary basic elementary operations and functions. Sciencepaper Online. 2019. (in Chinese with English abstract) http://www.paper.edu.cn/releasepaper/content/201910-3

[7] Zhao SZ. A reliable computing algorithm and its software ISReal for arithmetic expressions. Scientia Sinica Informationis, 2016, 46(6):698–713 (in Chinese with English abstract).

[8] Zhao SZ, Liu XF, Song F. Error-Controlled computation of expressions. In: Proc. of the 18th Int'l Symp. on Scientific Computing, Computer Arithmetic, and Verified Numerical Computations. 2018. 172–173.

附中文参考文献:

[5] 赵世忠.浮点运算错误计算原因.中国科技论文在线.2017. http://www.paper.edu.cn/releasepaper/content/201707-86

[6] 赵世忠.二元运算的错数.中国科技论文在线.2019. http://www.paper.edu.cn/releasepaper/content/201910-3

[7] 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(6):698−713.

赵世忠(1968－),男,博士,讲师,CCF 专业会员,主要研究领域为可信计算,符号计算.

陈冬火(1974－),男,博士,讲师,CCF 专业会员,主要研究领域为程序验证,模型检验,机器学习.

刘静(1964－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件模型,形式化方法,可信软件.