Should We be Concerned about Roundoff Error?

Dr. Anthony P. Leclerc
Department of Computer Science
The College of Charleston
66 George Street
Charleston, SC 29424
leclerc@cs.cofc.edu

1 An Example of Roundoff Error

Some might contend that, today, roundoff error is not a critical issue because of the extended precision floating point numbers available on many computers. However, consider an example of S. M. Rump [3] where the following innocuous-looking function

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y) \qquad (1)$$

was evaluated at $x = 77617$ and $y = 33096$.

The FORTRAN program to evaluate this function at various precisions was compiled and executed on a SPARCstation SLC. The accuracy of the three precisions tested, single, double, and extended, were 6, 14, and 35 decimal digits, respectively. The powers on x and y were evaluated with multiplications rather than with the built-in library *power* function in order to test only the basic arithmetic operations. The following results were obtained:

$$\begin{aligned} \text{(single precision)} \quad & f = 6.33825 \times 10^{29} \\ \text{(double precision)} \quad & f = 1.1726039400532 \\ \text{(extended precision)} \quad & f = 1.17260394005317863185883490452018 38 \end{aligned}$$

By observing these results, one might conclude that single precision result is wrong. One might further (and erroneously) conclude that the double precision result is accurate since it agrees to 13 digits with the extended precision result. Surprising to many, all three results are wrong *even in the first digit*! For that matter, *the sign itself* is incorrect!

The exact result, obtained using the variable precision interval arithmetic of VPI [1] with about 40 decimal digits of accuracy, is "trapped" tightly in the following interval:

$$\begin{bmatrix} -0.827396059946821368141165095479816292005, \\ -0.827396059946821368141165095479816291986 \end{bmatrix}$$

The point to be made here is two-fold:

1. Roundoff error can seriously compromise the reliability of results for *any* fixed precision floating point computation.

2. By simply observing floating *point* results at increasing precisions (single, double, and extended), *no* indication of the seriousness of roundoff error may be given.

2 How Can Roundoff Error be Controlled?

It seems that roundoff error plagues *all* computations performed with fixed precision floating point arithmetic. How can one hope to alleviate this error?

One tedious effort is to perform floating point error analysis. However, such analyses are extremely cumbersome and usually only applied to simple functions. In addition, floating point error analysis becomes more difficult with iterative algorithms in which roundoff error can be propagated from one iteration to the next.

Another attempt is to "estimate" the accuracy of results from a floating point computation by using stochastic approaches such as the CESTAC method (or Permutation-Perturbation method) [4]. Such methods, however, are probabilistic in nature and cannot reliably guarantee the accuracy of the results.

To date, the author knows of only one technique which easily and reliably controls roundoff error. This technique is *interval arithmetic*.

3 Interval Arithmetic: A Tool to "Trap" Roundoff

Just *one* evaluation of a function using interval arithmetic provides upper and lower bounds on the range of values of the function over a *set* of values (a continuum) of the arguments.

3.1 What is Interval Arithmetic?

An interval is a closed bounded set of real numbers

$$[a, b] = \{x : a \leq x \leq b\}.$$

Arithmetic with intervals is simply *arithmetic with inequalities*. For instance, if $a \leq x \leq b$ and $c \leq y \leq d$, then $a + c \leq x + y \leq b + d$. Thus, the addition of two intervals is defined by:

$$[a, b] + [c, d] \quad = \quad [a + c, b + d]$$

Likewise, using the properties of inequalities, the four basic interval operations are defined as follows:

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d] \\
[a,b] - [c,d] &= [a-d, b-c] \\
[a,b] \times [c,d] &= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\
[a,b] \div [c,d] &= [a,b] \times [1/d, 1/c] \text{ if } 0 \notin [c,d]
\end{aligned}$$

The implementation of interval arithmetic on a computer is easy. Since the endpoints a and b of a given interval $[a,b]$ may not be machine representable numbers, a is rounded to the largest machine number, say a_m, which is less than or equal to a, and b is rounded to the smallest machine number, say b_m, which is greater than or equal to b.

This outward-rounded *machine interval*, $[a_m, b_m]$, contains $[a,b]$. Simply put, $[a,b] \subseteq [a_m, b_m]$. The basic principle of interval arithmetic is preserved in that the exact result is contained in the corresponding known machine interval, **with roundoff error controlled**.

3.2 An Example of Interval Arithmetic

Consider the following interval computation of:

$$x(u,t) = \frac{u^2 t}{u^2 + t^2 + 1}$$

where $u = [.1, .3]$ and $t = [.2, .6]$. Evaluating each sub-expression, one obtains:

$$\begin{aligned}
u^2 &= [.01, .09] \\
t^2 &= [.04, .36] \\
u^2 t &= [.002, .054] \\
u^2 + t^2 + 1 &= [1.05, 1.45] \\
\frac{u^2 t}{u^2 + t^2 + 1} &= \frac{[.002, .054]}{[1.05, 1.45]} = \left[\frac{.002}{1.45}, \frac{.054}{1.05}\right] \subseteq [.00137, .05143]
\end{aligned}$$

3.3 Where is Interval Arithmetic Available?

There are several extensions or libraries in the FORTRAN, Pascal, and the C++ programming languages with facilities for fixed or arbitrary precision interval arithmetic [1, 2]. In addition, Maple V version 2 and Mathematica version 2.2 provide for "range" and "Interval" arithmetic, respectively. However, the "range" arithmetic in Maple V version 2 (using *evalr()*) does not seem to do outward rounding (or outward-rounded I/O) properly. For this reason, Maple was not able to provide guaranteed "traps" for several interval computations tested.

4 Should We Teach Students about Roundoff?

In my opinion, yes. The concepts of a fixed precision floating point number, the operations defined on such numbers, and the strengths (efficiency) and weaknesses (limited precision, roundoff) of computing with these types of numbers are not difficult. A student, scientist, mathematician, or engineer, understanding roundoff error and equipped with the tools to "trap" roundoff, will be able to detect when roundoff has corrupted a result and hopefully find better ways to formulate or evaluate the computation.

References

[1] J. S. Ely. *Prospects for Using Variable Precision Interval Software in C++ for Solving Some Contemporary Scientific Problems*. PhD thesis, The Ohio State University, 1990.

[2] A. P. Leclerc. *Efficient and Reliable Global Optimization*. PhD thesis, The Ohio State University, 1992.

[3] S. M. Rump. *Reliability in Computing. The Role of Interval Methods in Scientific Computing*. Academic Press, 1988. roundoff error example.

[4] Jean VIGNES. *New Methods in Optimization and their Industrial Uses*, pages 219–227. Birkhäuser Verlag, Basel, 1989. Estimation of the accuracy of results.