Tools for a gentle slope transition From floating point arithmetic to exact real arithmetic

Valérie Ménissier-Morain

Tools for a gentle slope transition From floating point arithmetic to exact real arithmetic

Valérie Ménissier-Morain*

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
CNRS, UMR 7606, LIP6, Pequan Team, F-75005, Paris, France

Floating point arithmetic (FPA) is one century old [19] and is effectively used commonly since 60 years. Exact real arithmetic (ERA) appears 40 years ago [9] and has been developed essentially since the end of the 80's [22, 2] (see [11] for a survey) resulting both of the dissatisfaction about FPA results and the sharp increase of material computing power that allows computation ambition.

FPA is essentially a fixed precision arithmetic while ERA adapts the precision of each operation to ensure the desired accuracy of the result. However for a long time, we have had to choose between *fast computed, completely wrong, FPA results* on the one hand and *accurate ERA results obtained too late to be useful* on the other hand.

Alternative to FPA such as interval arithmetic (IA) have been designed by mathematicians since the 50's and during the last three decades numerous tools have been designed by computer scientists to reduce the gap between FPA and ERA with two principle directions: evaluate the inaccuracy of the result and try to produce a more accurate result.

Evaluate the inaccuracy of the result

Analysis of FPA behavior

In the case where it is impossible to change radically arithmetic in a program, it is however possible with only a few modifications of the code to analyze its behavior as regards to the accuracy of the result.

It is possible to make static analysis, that is to say without execution of the program, with a tool like Fluctuat developed at CEA List [10, 18] that detects by abstract interpretation which variable and which piece of code is critical for the accuracy of the result and computes a bound on the error, and it can take assertions into account like interval of possible values for data measures or properties of the result. Fluctuat is a non-free multi-platform application to analyze C and Ada95 codes.

As an alternative to this static analysis, there exists a tool for dynamic analysis, Cadna, developed in the Pequan Team at UPMC, based on the notion of stochastic arithmetic of the CESTAC method [21, 4] to estimate round-off errors and detect numerical instabilities. The idea is to execute two or three times the same instruction with different rounding modes (randomly chosen with probability 1/2 for $\underline{x}$ and $\overline{x}$) before next instruction (synchrone evaluation) to see how to propagate round-off errors; for each instruction the significant result will be the common part of the different results. Cadna is an open source library for Fortran and C. There is an old version in Ada, a partial version for BLAS, and now since a few years the SAM version for MPFR multiprecision [12].

Starting from this analysis the programmer is aware of the numerical quality of his result and can try to improve the conditioning of his program.

A survey on rounding errors analysis is given in [5].

Interval arithmetic

The principle is very simple [16]: use intervals rather than numbers for two reasons: often numerical data are soiled with measure errors thus considering intervals of values seems very quickly natural in scientific computations (50's ans 60's) and further it has also been used simply to evaluate the importance of round-off errors in floating point computations.

This leads to two approaches. In the first one, intervals are considered as first class objects and for any function f an algorithm describes how to produce interval $[c, d]$ such that $f(x) \in [c, d]$ for all $x \in [a, b]$ (*inclusion property*) and then

*Valerie.Menissier-Morain@upmc.fr

functional analysis, linear algebra, etc. on this objects are developed. This approach is a branch of pure mathematics and is essentially not concerned with computations. In the second one, intervals $[\underline{x}, \overline{x}]$ represent the result of round-off errors of floating-point computations to guarantee that the final mathematical result is in the computed interval. This approach is the minor one but was clearly boosted by IEEE 754 standard.

There is a free C++ interval library in the Boost repository of free peer-reviewed portable C++ libraries [15].

Try to produce a more accurate result

Error-free transformations and compensated arithmetic

The starting point of this approach is the remark that the rounding error of addition and multiplication of two fp numbers is itself a fp number with rounding to nearest. Furthermore between 1965 and 1975 explicit algorithms have been designed to compute this error [14, 7, 20]. Algorithms that produce both the fp result and the fp error are called Error-Free Transformations (EFT).

This has been used to double the precision of computations based on these operations during this decade and further since 2005 it is used again in a systematic way to obtain by iterative refinement more accurate results for example for summation, product, dot product, polynomial evaluation, solution of triangular linear system [17, 13]. The idea is to re-inject the rounding error of an operation into the following one or to accumulate rounding errors to produce a corrective term to add to the result of the fp computation to produce a fp final result as accurate as if computed in doubled (or more) working precision and rounded to working precision.

Multiprecision

The EFT have also been used since 40 years to implement double-double and quad-double arithmetic in Fortran (Brent MP [3], Bayley MPFUN and QD [1]). These packages can be classified into two formats of numbers: *multiple-digit format* i.e. numbers are represented as a sequence of digits coupled with a single exponent; *multiple-component format*, i.e. numbers represented as an unevaluated sum of two (resp. four) IEEE double precision numbers. The purpose is, as for FPA, to have a fixed precision all along the computation but numbers have larger representations than fp numbers.

Later multiprecision appeared to allow more powerful and flexible computations. We can cite GMP/MPF since 1996, MPFR [8] since 2000 and its interval version MPFI since 2002 for C language, with interfaces for lots of other languages. Each number has its own precision and the result is correctly rounded to the precision of the target variable, in any of the four IEEE-754 rounding modes. If the programmer chooses convenient precision for each variable he can obtain an accurate result as with an ERA computation. He simply needs to analyze the computation to deduce the required precision everywhere while ERA computes the result without any precision indication.

Almost ERA with Sollya

Sollya, developed by PEQUAN UPMC team and APICS INRIA team [6], is both an interactive application and a library for safe C floating-point code development. It is particularly targeted to the automatized implementation of mathematical floating-point libraries. Sollya uses multi-precision arithmetic (MPFR) and interval arithmetic to produce tight and safe results with the precision required by the user.

The printed value is either exact or generally a faithful rounding of the exact value at the working precision. If a faithful rounding cannot sufficiently quickly be computed, a value is printed that was obtained using floating-point approximations without control on the final accuracy. Whenever something is not exact, a warning is systematically printed.

Conclusion

These few tools show that there's nothing to despair: even if it is not possible to use ERA, there exists a variety of tools to watch over or insure the accuracy of the results with non-ERA packages.

References

[1] D. H. BAILEY. High-precision software directory. `http://crd-legacy.lbl.gov/ dhbailey/mpdist/`.

[2] H. J. BOEHM. Constructive Real Interpretation of numerical Programs. In *Proceedings of the 1987 ACM conference on Interpreters and Interpretives Techniques* (1987).

[3] R. BRENT. A fortran multiple precision arithmetic package. *ACM Trans. Math. Soft 4* (1978), 57–70.

[4] J.-M. CHESNEAUX. *L'arithmétique stochastique et le logiciel CADNA*. Habilitation à diriger des recherches, UPMC, 1995.

[5] J.-M. CHESNEAUX, S. GRAILLAT, AND F. JÉZÉQUEL. *Encyclopedia of Computer Science and Engineering*, vol. 4. Wiley, 2009, ch. Rounding Errors, p. 2480–2494.

[6] S. CHEVILLARD, M. JOLDEŞ, AND C. LAUTER. Sollya: An environment for the development of numerical codes. In *Mathematical Software - ICMS 2010* (2010), vol. 6327 of *Lecture Notes in Computer Science*, Springer, pp. 28–31.

[7] T. J. DEKKER. A floating point technique for extending the available precision. *Numerische Mathematik 18*, 3 (1971), 224–242.

[8] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, AND P. ZIMMERMANN. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software 33*, 2 (June 2007), 13:1–13:15.

[9] B. GOSPER. Continued Fraction Arithmetic. HAKMEM Item 101B, MIT AI MEMO 239, Feb. 1972.

[10] E. GOUBAULT, S. PUTOT, P. BAUFRETON, AND J. GASSINO. Static analysis of the accuracy in control systems: Principles and experiments. In *Proceedings of Formal Methods in Industrial Critical Systems* (2007), vol. 4916 of *LNCS*.

[11] P. GOWLAND AND D. LESTER. A survey of exact arithmetic implementations. In *Computability and Complexity in Analysis* (2001), vol. 2064 of *Lecture Notes in Computer Science*, pp. 30–47. 4th International Workshop, CCA 2000, September 2000.

[12] S. GRAILLAT, F. JÉZÉQUEL, S. WANG, AND Y. ZHU. Stochastic arithmetic in multiprecision. *Mathematics in Computer Science 5* (2011), 359–375.

[13] S. GRAILLAT, P. LANGLOIS, AND N. LOUVET. Accurate and validated polynomial evaluation in floating point arithmetic. In *Algebraic and Numeric Algorithms and Computer-assisted Proofs* (2005), no. 5391 in Dagstuhl Seminar.

[14] D. E. KNUTH. *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2. Addison-Wesley, 1969.

[15] G. MELQUIOND, S. PION, AND H. BRÖNNIMANN. Boost interval arithmetic library. `http://www.boost.org/doc/libs/1_55_0/libs/numeric/interval/doc/interval.htm`.

[16] R. E. MOORE, R. B. KEARFOTT, AND M. J. CLOUD. *Introduction to Interval Analysis*. SIAM Press, 2009.

[17] T. OGITA, S. M. RUMP, AND S. OISHI. Accurate sum and dot product. *SIAM Journal on Scientific Computing 26*, 6 (2005), 1955–1988.

[18] S. PUTOT. Fluctuat. `http://www.lix.polytechnique.fr/Labo/Sylvie.Putot/fluctuat.html`.

[19] L. TORRES Y QUEVEDO. Ensayos sobre automatica–su defmicion. extension teorica de sus aplicaciones. *Revista de la Real Academia de Ciencias Exactas, Fisicas y Naturales 12* (1913), 391–418.

[20] G. W. VELTKAMP. Algol procedures voor het berekenen van een inwendig product in dubbele precisie. RC-Informatie 22, Technische Hogeschool Eindhoven, 1968.

[21] J. VIGNES. A stochastic arithmetic for reliable scientific computation. *Mathematics ans Computers in Simulation 35* (1993), 233–261.

[22] J. VUILLEMIN. Exact real computer arithmetic with continued fractions. Research report 760, INRIA, 1987.