Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library

N. Revol[1] and and F. Rouillier[2]

[1]Lab. ANO, University of Lille and
and CNRS/ENSL/INRIA Project Arenaire
LIP, École Normale Supérieure de Lyon, France
[2]Project Spaces, LORIA/INRIA/LIP 6, France

1 Motivations for Changing Arithmetic

Nowadays, computations involve more and more operations and consequently errors. The limits of applicability of some numerical algorithms are now reached: for instance the theoretical stability of a dense matrix factorization (LU or QR) is ensured under the assumption that $n^3u < 1$, where n is the dimension of the matrix and $u = 1^+ - 1$, with 1^+ the smallest floating-point larger than 1; this means that n must be less than 200,000, which is almost reached by modern simulations. The numerical quality of solvers is now an issue, and not only their mathematical quality. Let us cite studies performed by the CEA (French Nuclear Agency) on the simulation of nuclear plant accidents and also softwares controlling and possibly correcting numerical programs, such as Cadna [10] or Cena [20].

Another approach consists in computing with certified enclosures, namely interval arithmetic [21, 2, 18]. The fundamental principle of this arithmetic consists in replacing every number by an interval enclosing it. For instance, π cannot be exactly represented using a binary or decimal arithmetic, but it is certified that π belongs to $[3.14159, 3.14160]$. The advantages of interval arithmetic are numerous. On the one hand, it exhibits the property of *validated* or *certified computing*. On the other hand, computer implementations are based on outward roundings and thus computed results take into account rounding errors and constitute a way to estimate these errors. A last and very important advantage, even if it is often less known, is that this arithmetic provides global information: for instance, it provides the range of a function over a whole set S, which is crucial for global optimization; furthermore, if this range is a (strict) subset of S, then Brouwer's theorem states that this function has a (unique) fixed-point and this can be used by Newton's algorithm for instance. Such properties cannot be reached without set computing, and interval arithmetic computes with sets and is easily available.

However, in spite of the improvements in interval analysis, the problem of overestimation, *i.e.* of enclosures which are far too large and thus inaccurate, seems to be the destiny of interval computation when it is implemented using fixed-precision floating-point arithmetic. Using a multiple precision postpones the occurrence of numerical problems, however the number of correct figures remains unknown. Computing with intervals provides guaranteed results, but the bounds can be far apart even when the input data are provided with the machine precision; a remedy for this phenomenon consists in computing with a higher precision. This proposal is the core of the MPFI library (*Multiple Precision Floating-point Interval arithmetic library*), a library implementing arbitrary precision interval arithmetic which is described in this paper.

This quest for extra accuracy can be found in other works such as those by [9] where polynomial expressions are symbolically rewritten before being evaluated, so as to reduce the overestimation due to dependency, or by [5] where high-order Taylor expansions are used. In this latter work, the time overhead is about 1500 for a single evaluation, however it is compensated by the reduction in the number of steps performed by the algorithm. Real-world applications where extra accuracy is required are to be found in automatics (we have been asked to integrate linear systems with high accuracy) or chemistry: determining a molecular conformation entails the minimization of an energy function and requires accurate evaluations of this energy function.

Several multiple precision interval packages are available. Let us quote for instance intpak [11] and intpakX [13] for Maple or a similar package for Mathematica [17]. Due to unverified assumptions on the roundings of elementary functions (0.6 ulp for intpak in Maple, 1 ulp for Mathematica), to bugged roundings (for instance, with 3 decimal digits, the rounding towards $-\infty$ of $1-9.10^{-5}$ gives 1 instead of 0.999 in Maple v6 and v7), and to several undue assumptions, these packages cannot be considered as reliable. Earlier works include the "range arithmetic" [1], a multiple precision library which aims at indicating the number of correct digits rather than at performing interval arithmetic, and IntLab [27] which primarily implements efficiently interval algorithms using MatLab, and, besides, mainly provides a type for arbitrary precision computations but implements few related functionalities. Such an arithmetic was also mentioned as an easy-to-implement extension to Brent's multiple precision package MP as early as 1981 [7]. Anyway, none of the aforementioned packages implements a complete and really reliable arbitrary precision interval arithmetic and this led us to implement our own library.

2 Theoretical Background

The theoretical result underlying this idea can be found in [23]: let us denote by X an interval, by f a function and by F an interval extension of f, where F is given by a Lipschitz expression, let ε correspond to the current computing precision p: $\varepsilon = 2^{-p}$, then $F(X)$ overestimates $f(X)$ and the overestimation is bounded by

$$q(f(X), F(X)) \leq c_1 w(X) + c_2 \varepsilon \tag{1}$$

where q is the Hausdorff distance, $w(X)$ is the width of X and the constants c_1 and c_2 depend on F. This means that the computing precision can become a limiting factor and that being able to increase it can be an issue.

Furthermore, a classical procedure in interval analysis is the bisection one: if the output width is too large, then the inputs are split in two (or more) parts and the computation is repeated on each part. Bisection is a way to escape the wrapping effect by providing a paving of the sought set, and also the dependency problem, even if, in that respect, nothing supersedes the use of a good formulation. Bisection is often the last resort to get more accuracy, by reducing in formula (1) the quantity $w(X)$. In cases where $w(X) = u$ (which happened in our experiments on global optimization), only an increase in the computing accuracy, by "adding new floating-point numbers" between the endpoints of X, would have yielded a solution.

It can be noticed that the rule of thumb "*to get more digits, one has to increase the computing precision by roughly the same number of digits*" can fail, for instance when computing a square root or more generally a $1/n$-th power close to 0. However, the rule of thumb becomes in such cases "*to get α more digits, one has to increase the computing precision by roughly $n\alpha$ digits*". In other words, in most cases an increase in the computing precision yields an improved accuracy on the results.

This is also the starting point of Müller's work on an effective simulation of a Real RAM [22], following the theoretical results by Brattka and Hertling [6] on the feasibility of a Real RAM. In Müller's work, a computation is performed and, if the final accuracy is not sufficient, then the whole computation is restarted with an increased precision; this is reiterated until the outputs are accurate enough.

3 The MPFI Library

In order to implement an arbitrary precision interval arithmetic, a multiple precision library was needed. By multiple precision, it is meant that the computing precision is not limited to the single or double precision of machine floating-point numbers; on the contrary, arbitrary precisions should be available. Furthermore, this computing precision must be dynamically adjustable to fulfill the accuracy needs. A more precise requirement for interval arithmetic is the outward rounding facility: this ensures that for each operation, the interval computed using floating-point arithmetic contains the interval obtained if exact real arithmetic were used. Even more desirable is *exact* directed rounding to avoid losing accuracy, *i.e.* the interval computed using floating-point arithmetic is the smallest one (for inclusion); however, it is rarely fulfilled for elementary functions. To sum up, compliance with the IEEE 754 standard for floating-point arithmetic, extended to elementary functions, is welcome.

The Arithmos project of the CANT team, U. Antwerpen, Belgium [8], or the MPFR library (*Multiple Precision Floating-point Reliable library*), developed by the Spaces team, INRIA Lorraine, France [12], are such libraries. For portability and efficiency reasons (MPFR is based on GMP and efficiency is a motto for its developers) and also because of the availability of the source code, we chose MPFR. The corresponding library, named MPFI [25], is a portable library written in C for arbitrary precision interval arithmetic. It is based on the GNU MP library and on the MPFR library and is part of the latter. The largest achievable computing precision is provided by MPFR and depends in practice on the computer memory on which it runs. The only theoretical limitation, which may be removed soon, is that the exponent must fit in an integer. Let us just say that it is possible to compute with numbers of several millions of binary digits if needed.

Intervals are implemented using their endpoints, which are MPFR reliable floating-point numbers: this is not visible for the user but ensures that the swelling of intervals' widths is less important than with the midpoint-radius representation such as implemented by Rump in IntLab [27, 28]. Indeed, switching the rounding modes incurs no penalty with multiple precision arithmetic and the motivation for this choice in IntLab does not hold for MPFI: every multiple precision operation is a software one. The arithmetic operations are implemented and the elementary functions available up to now are exp, log, sine and cosine; all functions provided by MPFR will shortly be included as well (trigonometric and hyperbolic trigonometric functions and their reciprocals).

The planned functionalities, that will be added in a near future, include a C++ interface à la Profil/BIAS [19] for ease of use, basic tools for linear algebra (vector and matrix data types, additions and multiplications) and automatic differentiation (forward differentiation by overloading operators and functions).

4 Applications

The MPFI library is already in use. Rouillier and Zimmermann [26] have developed a hybrid algorithm (symbolic/interval) for isolating real roots of polynomials, Revol [24] has implemented interval Newton algorithm [15] adapted to multiple precision computations. A main advantage of using MPFI is that one is no more limited by the computing precision: for instance one can impose arbitrary accuracy on both the root and the residual in Newton's algorithm [4]. Furthermore, the aforementioned implementations manage to adapt dynamically the precision to the computing needs without restarting the whole program. This desirable feature will be sought after for future implementations of other algorithms.

5 Conclusion

MPFI is a library for multiple precision interval arithmetic. It is written in C and built upon MPFR and GMP and can be freely downloaded. It is still under development: new facilities such as automatic differentiation and linear algebra will be added in the near future. It still has enabled us to implement and test some algorithms and this will be pursued with a careful study of the solution of linear systems and of global optimization of continuous functions [14, 3]. Applications such as parameter estimation in automatics [16] will offer the opportunity to gain further insight in the development of new algorithms.

References

[1] O. Aberth and M. J. Schaefer, "Precise computation using range arithmetic, via C++", *ACM TOMS*, 1992, Vol. 18, No. 4, pp. 481–491.

[2] G. Alefeld and J. Herzberger, *Introduction to interval analysis*, Academic Press, 1983.

[3] R. Baker Kearfott, *Rigorous global search: continuous problems*, Kluwer, 1996.

[4] R. B. Kearfott and G. W. Walster, "On stopping criteria in verified nonlinear systems or optimization algorithms", *ACM TOMS*, 2000, Vol. 26, No. 3, pp. 373–389.

[5] M. Berz and J. Hoefkens, "Verified high-order inversion of functional dependencies and interval Newton methods", *Reliable Computing*, 2001, Vol. 7, pp. 1–20.

[6] V. Brattka and P. Hertling, "Feasible real random access machines", *J. of Complexity*, 1998, Vol. 14, No. 4, pp. 490–526.

[7] R. P. Brent, "A Fortran multiple-precision arithmetic package", *ACM TOMS*, March 1978, Vol. 4, pp. 57–70.

[8] CANT Research Group, *Arithmos: a reliable integrated computational environment*, University of Antwerpen, Belgium, `http://win-www.uia.ac.be/u/cant/arithmos`, 2001.

[9] M. Ceberio and L. Granvilliers, "Solving Nonlinear Systems by Constraint Inversion and Interval Arithmetic", In *Int. Conf. on Artificial Intelligence and Symbolic Computation (AISC'2000)*, LNAI 1930.

[10] J.-M. Chesneaux, S. Guilain, and J. Vignes, *La bibliothèque CADNA: présentation et utilisation*. `http://www-anp.lip6.fr/cadna`, 1996.

[11] A. E. Connell and R. M. Corless, "An experimental interval arithmetic package in Maple", In *Num. Analysis with Automatic Result Verification*, 1993.

[12] D. Daney, G. Hanrot, V. Lefèvre, F. Rouillier, and P. Zimmermann, *The MPFR library*, `http://www.mpfr.org`, 2001.

[13] I. Geulig and W. Krämer, *Intervallrechnung in Maple - Die Erweiterung intpakX zum Paket intpak der Share-Library*, Technical Report 99/2, Universität Karlsruhe, 1999.

[14] E. Hansen, *Global optimization using interval analysis*, Marcel Dekker, 1992.

[15] E. Hansen and R.I. Greenberg, "An interval Newton method", *J. of Applied Math. and Computing*, 1983, Vol. 12, pp. 89–98.

[16] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied interval analysis*, Springer Verlag, 2001.

[17] J. Keiper, "Interval arithmetic in Mathematica", *Interval Computations*, 1993, No. 3.

[18] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, and A. Wiethoff, *C-XSC: A C++ class library for extended scientific computing*, Springer Verlag, 1993.

[19] O. Knueppel, "PROFIL/BIAS - a fast interval library", *Computing*, 1994, Vol. 53, No. 3-4, pp. 277–287.

[20] P. Langlois, "Automatic linear correction of rounding errors", *BIT Numerical Algorithms*, 2001, Vol. 41, No. 3, pp. 515–539.

[21] R.E. Moore, *Interval analysis*, Prentice Hall, 1966.

[22] N. Müller, "The iRRAM: Exact Arithmetic in C++", In *Proc. Workshop on Constructivity and Complexity in Analysis*, Swansea, 2000.

[23] A. Neumaier, *Interval methods for systems of equations*, Cambridge University Press, 1990.

[24] N. Revol, *Newton's algorithm using multiple precision interval arithmetic*, Research Report 4334, INRIA, 2001; submitted to *Numerical Algorithms.*

[25] N. Revol and F. Rouillier, *The MPFI library* `http://www.ens-lyon.fr/~nrevol`, 2001.

[26] F. Rouillier and P. Zimmermann, *Efficient isolation of polynomial real roots*, 2001.

[27] S. Rump, "INTLAB - Interval Laboratory", In: T. Csendes (ed.), *Developments in Reliable Computing*, Kluwer, 1999, pp. 77–104.

[28] S. Rump, "Fast and parallel interval arithmetic", *BIT*, 1999, Vol. 39, No. 3, pp. 534–554.