Reliable computing with GNU MPFR

Paul Zimmermann

LORIA/INRIA Nancy-Grand Est, Équipe CARAMEL - bâtiment A,
615 rue du jardin botanique, F-54603 Villers-lès-Nancy Cedex

Abstract. This article presents a few applications where reliable computations are obtained using the GNU MPFR library.

Keywords: reliable computing, correct rounding, IEEE 754, GNU MPFR

The overview of the 3rd International Workshop on Symbolic-Numeric Computation (SNC 2009), held in Kyoto in August 2009, says: *Algorithms that combine ideas from symbolic and numeric computation have been of increasing interest over the past decade. The growing demand for speed, accuracy and reliability in mathematical computing has accelerated the process of blurring the distinction between two areas of research that were previously quite separate. [...]* Using numeric computations certainly speeds up several symbolic algorithms, however to ensure the correctness of the final result, one should be able to deduce some *reliable facts* from those numeric computations. Unfortunately, most numerical software tools do not provide any accuracy guarantee to the user. Let us demonstrate this fact on a few examples. Can we deduce the sign of $\sin(2^{100})$ from the following computation with Maple version 13?

```
> evalf(sin(2^100));
                                0.4491999480
```

This problem concerns other computer algebra systems, for example Sage 4.4.2, where the constant $C = e^{\pi\sqrt{163}} - 262537412640768744$ evaluated to different precisions yields different signs:

```
sage: f=exp(pi*sqrt(163))-262537412640768744
sage: numerical_approx(f, digits=15)
448.000000000000
sage: numerical_approx(f, digits=30)
-5.96855898038484156131744384766e-13
```

or Mathematica 6.0, where the same constant $C \approx -0.75 \cdot 10^{-12}$, integrated from 0 to 1, yields a result too large by several orders of magnitude:

```
In[1]:= NIntegrate[Exp[Pi*Sqrt[163]]-262537412640768744,{x,0,1}]

Out[1]= -480.
```

Another example is FFTW, which is a very efficient Fast Fourier Transform (FFT) library. On `http://www.fftw.org/accuracy/comments.html` one can

read: *Our benchmark shows that certain FFT routines are more accurate than others. In other cases, a routine is accurate on one machine but not on another. [...]* This non-reproducibility from one machine to another one is quite annoying, since this means that a given program using FFTW might give different results on different computers. One reason of this non-reproducibility is the fact that FFTW uses *trigonometric recurrences* to compute the *twiddle factors* $e^{2ik\pi/2^n}$ needed in the FFT. On some architectures, those recurrences are evaluated in double-extended precision — significand of 64 bits — instead of double-precision — significand of 53 bits.

To avoid the above problems, developers of numerical software tools should provide primitives with *well-defined semantics*. This is not an easy goal. The more complex the numerical primitive is, the more difficult it is to provide rigorous bounds on the error. For example consider the implicit two-dimensional plot of $|\cos((x+iy)^4)| = 1$ for $-3 \le x, y \le 3$; the computation to 20 decimal places of $\rho(10)$ where ρ is Dickman's function, defined by the difference-differential equation $x\rho'(x) + \rho(x-1) = 0$, with initial conditions $\rho(x) = 1$ for $0 \le x \le 1$; or the computation to 20 decimal places of the singular values of the Hilbert matrix of order 50, defined by $M_{i,j} = 1/(i+j)$.

There are several ways for a numerical primitive to return valuable information. One way is to give, in addition to the numerical approximation, a bound on the absolute or relative error; or alternatively an interval enclosing the true result. The ultimate — and more difficult solution, from the implementer point of view — is to guarantee *correct rounding* as in IEEE 754 [6], i.e., that the given approximation is the best possible according to the target precision. To rigorously define what we mean by "best possible", we have to introduce *rounding directions*, which determine in which direction to round the approximation with respect to the exact result, and how to break ties.

GNU MPFR (MPFR for short) is a C library implementing correct rounding for basic arithmetic operations and mathematical functions in binary multiple-precision. We refer the reader to [4] for a technical description of MPFR. We focus here on a few applications of MPFR.

Companion Libraries MPFI and MPC. The MPFI library implements arbitrary precision interval arithmetic on top of MPFR. It was originally designed by N. Revol and F. Rouillier. For a monotonic function, implementing an interval routine is trivial: just call the corresponding MPFR function with rounding modes towards $-\infty$ and $+\infty$. However for a non-monotonic function, it requires more work; for example does $\cos([103992, 103993])$ contain 1?

The MPC library, developed by A. Enge, Ph. Théveny and the author, is another companion library to MPFR, which provides arbitrary precision complex floating-point numbers with correct rounding. MPC uses the cartesian representation $z = x + iy$, where both x and y are correctly rounded. If MPFR provides r rounding modes — $r = 5$ in MPFR 3.0.0 — then MPC can provide up to r^2 complex rounding modes. The MPC library implements all functions from the C99 standard.

Constant Folding in GCC. The GCC compilers for the C and Fortran languages use MPFR for constant folding (see details in [5]). In short, when one writes `double y = sin(17.42)` in a C program, GCC replaces this at *compile time* by `double y = -0.99004214446851813`, using MPFR to perform the corresponding computation. The advantage is twofold: on the one hand MPFR will yield the same numerical result on any configuration, whatever the operating system, the processor word size; and on the other hand MPFR guarantees correct rounding for the sine mathematical function (which de facto implies reproducibility). Up from version 4.5, GCC also uses the MPC library to provide constant folding of complex floating-point expressions.

Maple. Since version 11, the Maple computer algebra system uses MPFR for approximating real solutions of polynomial systems in the `RootFinding` package:

```
> sys:=[x^2+y^2-1, y-x^2]:
> RootFinding[Isolate](sys, [x,y], digits=1, output=interval);
       -7251005348244714239  -906369071450328423
[[x = [--------------------, --------------------],
       9223372036854775808   1152921504606846976

         1414187776304389027  5711861873363103083
    y = [-------------------, -------------------]], [...]]
         2305843009213693952  9223372036854775808
```

This computation uses the Rational Univariate Representation designed by F. Rouillier, and then the MPFI interval arithmetic library — which in turn uses MPFR — is used to isolate the roots or perform arithmetic operations on the roots.

MPFR in Sage. The open-source Sage computer algebra system (`sagemath.org`) uses MPFR for its arbitrary precision floating-point arithmetic, and MPFI for the corresponding interval arithmetic. It should be noted that the user has access to the MPFR rounding modes and to the exact representation $m \cdot 2^e$ of binary floating-point numbers, as the following example (with Sage 4.4.2) shows:

```
sage: D = RealField(42, rnd='RNDD'); U = RealField(42, rnd='RNDU')
sage: D(pi), U(pi)
(3.14159265358, 3.14159265360)
sage: D(pi).exact_rational()
3454217652357/1099511627776
sage: x = RealIntervalField(42)(pi); x.lower(), x.upper()
(3.14159265358, 3.14159265360)
```

MPFR is also used by the Magma computational number theory system, and by the Mathemagix free computer algebra system (in the `numerix` package).

Apart from the above indirect applications, where the final user is not always aware that she/he is using MPFR, we mention here a few selected "direct" applications of MPFR (more details can be found on `http://www.mpfr.org/pub.html`). MPFR is often used as a reference correctly-rounded implementation

for mathematical functions in double precision [3, 8]. P. Kornerup, V. Lefèvre, N. Louvet and J.-M. Muller used MPFR to prove — among other results — that there exists no algorithm in less than 6 arithmetic operations to compute the "TwoSum" of two floating-point numbers a and b, i.e., x and y such that x is the rounding to nearest of $a + b$, and $y = a + b - x$; they used an exhaustive search approach [7]. F. Chiba and T. Ushijima used MPFR to study waves produced by a disc [1, 2].

Some funny applications of MPFR are the following. K. Briggs used MPFR to find a new worst approximable pair; this result required computing 10^7 terms of the continued fraction of $2\cos(2\pi/7)$. D. de Rauglaudre used MPFR to zoom in the Mandelbrot/Julia sets, since from a given depth on, double precision is not sufficient; the corresponding videos are available on Youtube[1].

Conclusion. Numeric tools with well-defined semantics help improving the reliability and portability of mathematical software. MPFR does not solve all problems: it only guarantees correct rounding for an atomic operation, thus for a sequence of operations like the constant C in the introduction, one has to use other means like interval arithmetic or a Real RAM implementation (like the iRRAM package from N. Müller). We advise the developers of numeric tools to provide such well-defined semantics, and their users to make good use of them!

Acknowledgement. The author thanks Nathalie Revol who noticed some typos in a earlier version of that article.

References

1. Chiba, F., and Ushijima, T. Computation of the scattering amplitude for a scattering wave produced by a disc – approach by a fundamental solution method. *Journal of Computational and Applied Mathematics 233*, 4 (2009), 1155–1174.
2. Chiba, F., and Ushijima, T. Exponential decay of errors of a fundamental solution method applied to a reduced wave problem in the exterior region of a disc. *Journal of Computational and Applied Mathematics 231*, 2 (2009), 869–885.
3. de Dinechin, F., Ershov, A. V., and Gast, N. Towards the post-ultimate libm. In *Proceedings of 17th IEEE Symposium on Computer Arithmetic* (Cape Cod, USA, 2005), pp. 288–295.
4. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.
5. Ghazi, K. R., Lefèvre, V., Théveny, P., and Zimmermann, P. Why and how to use arbitrary precision. *Computing in Science and Engineering 12*, 3 (2010), 62–65.
6. IEEE standard for floating-point arithmetic, 2008. Revision of ANSI-IEEE Standard 754-1985, approved June 12, 2008: IEEE Standards Board.
7. Kornerup, P., Lefèvre, V., Louvet, N., and Muller, J.-M. On the computation of correctly-rounded sums. In *Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH'19)* (2009), J. D. Bruguera, M. Cornea, D. DasSarma, and J. Harrison, Eds., IEEE Computer Society, pp. 155–160.

[1] `http://www.youtube.com/view_play_list?p=56029CB07C72B4A4`

8. Lauter, C. Q., and Lefèvre, V. An efficient rounding boundary test for pow(x, y) in double precision. *IEEE Trans. Comput. 58*, 2 (2009), 197–207.