Exact Real Computation in Computer Algebra*

Gábor Bodnár, Petru Pau, Barbara Kaltenbacher, Josef Schicho
SFB Numerical and Symbolic Scientific Computation
Johannes Kepler University Linz
A-4040 Linz, Austria

Abstract

This talk describes our approach to represent computable real numbers in Maple. We aim at a model that can be used in symbolic computation.

The main topics we discuss are: representation and arithmetic of exact real numbers, vectors and matrices; polynomial arithmetic and root computation. Some operations raise ill-posed problems; we used regularization methods to solve them.

1 Extended Abstract

Many algorithms in symbolic and algebraic computation rely on exactness. The usual way is to work with the subfield of rationals or real algebraic numbers; the later are traditionally represented by the minimal polynomial and either an isolated interval or a sequence of signs [7]. It is sometimes a problem that these subfields are not closed under transcendental functions such as exponentiation or trigonometric functions. Even if we stay within algebraic operations only, it turns out that the manipulation of real algebraic numbers is very expensive [10].

The method of *exact real computation* seems to be a reasonable alternative: it provides a *a mathematically consistent representation* of the represented reals, as the models behave exactly like the mathematical objects which they represent. Recall that this is not the case for floating point numbers, and this is one of the reasons why working with floating point numbers is so often difficult in computer algebra.

Several ways to represent the reals have already been invented; we mention here infinite (lazy) strings of digits ([6], with golden ratio notation, [9], [5]) and a functional approach, where the reals are represented as functions that produce approximations on demand [3].

Our approach is more pragmatically oriented towards the usability within computer algebra. Of course, efficiency of the arithmetic has been an issue, but our intention was not to invent a new arithmetic that could compete with the fastest existing ones. Instead, we took an objectual approach, which makes it easy to change the details in the representation at the lowest level independently of the higher level algorithms. It may be remarked that this has already paid off, as we were already forced to change details on the low level at a state where a lot of higher algorithms already existed.

A general performance problem in computer algebra is the one of *intermediate expression swell*. We still have it when using exact real computation, but much more moderate: in symbolic-exact arithmetic, the expressions may grow exponentially with the number of arithmetic operations (e.g. minimal polynomials under addition and multiplication of algebraic numbers, or mantissa lengths under powering of rational numbers). But in exact real computation we can always achieve linear growth. The size of a result is always equal to the size of the arguments plus a constant overhead for the operation itself. On the other hand, it has to be remarked that intermediate simplification (e.g. canceling) does not occur in exact real computation.

*Supported by the Austrian *Fonds zur Förderung der wissenschaftlichen Forschung* in the frame of the SFB 013.

We utilize the fact that most existing computer algebra systems already have a built-in arbitrary precision arithmetic and conversion algorithms between numerical and symbolic data. For many problems, it suffices to supply small wrappers which encode the necessary information about the error propagation. Sometimes this information is already available in the literature of numerical analysis (see e.g. [8]). An example is our algorithms for computing roots of squarefree polynomials (section 4.3).

However, there are other problems which lead to fundamental difficulties because their solution does not depend continuously on the input data: *pseudo-inverse of rank-deficient matrices, polynomial greatest common divisor (gcd) computation, multivariate polynomial factorization, curve and surface parameterization.* Strictly speaking, these problems cannot be solved within exact real computation (see [1]). As we definitely need a substitute for these ill-posed problems, we apply the common technique of *regularization*: we replace the problem by a nearby continuous problem (the distance to the exact problem is an additional input parameter). We apply Tikhonov regularization [2] (see also [4]) to the computation of pseudo-inverses and gcd's.

References

[1] Aberth, O. *Precise numerical methods using C++.* Academic Press Inc., San Diego, CA, 1998.

[2] Arsenin, V., and Tikhonov, A. N. *Solutions of ill-posed problems.* Wiley, 1977.

[3] Boehm, H., and Cartwright, R. Exact real arithmetic, formulating real numbers as functions. In *Research Topics in Functional Programming*, D. Turner, Ed. Addison-Wesley, 1990, pp. 43–64.

[4] Engl, H. Regularization methods for solving inverse problems. In *Proc. ICIAM 99* (1999).

[5] Escardó, M. H. Exact numerical computation. In *ISSAC'00* (2000), ACM Press. tutorial.

[6] Gianantonio, P. Real number computability and domain theory. In *Proc. Math. Found. Comp. Sci. '93* (1993), Springer, pp. 413–422.

[7] Gonzalez-Vega, L., Rouillier, F., Roy, M.-F., and Trujillo, G. Symbolic recipes for real solutions. *Algorith. Comp. Math. 4* (2000), 121–167.

[8] Higham, N. J. *Accuracy and stability of numerical algorithms.* SIAM, 1993.

[9] Ménissier-Morain, V. Arbitrary precision real arithmetic: design and algorithms. *J. Symb. Comp.* (2001). to appear.

[10] Roy, M.-F., and Szpirglas, A. Complexity of computation of real algebraic numbers. *J. Symb. Comp. 10* (1990), 39–51.