Interval Newton iteration in multiple precision for the univariate case

N. Revol, Lab. ANO, Univ. des Sciences et Technologies de Lille
and INRIA, LIP, École Normale Supérieure de Lyon, France

Dedicated to Claude Brezinski on the occasion of his 60th birthday

Abstract

In this paper, interval arithmetic using an underlying multiple precision arithmetic is briefly presented. Then interval Newton iteration for solving nonlinear equations is introduced. A new Newton's algorithm based on multiple precision interval arithmetic is given, along with its properties: termination, arbitrary accuracy on the computed zeros, automatic and dynamic adaptation of the precision. Finally some experiments illustrate the behaviour of this method.

Keywords: multiple precision interval arithmetic, interval Newton algorithm, arbitrary accuracy, automatic adaptation of the precision.
AMS (MSC) classification: 65G20, 65G30, 65G40, 65H05, 65H20

1 Introduction

The long going increase of the power of computers has led to an increase of the volume of computations. What is now required is not only the quantity of computations, but also some guarantees on the quality of the result. Let us just give an example. It took place at the Vancouver Stock Exchange: in 1982 a new index with a nominal value 1000.00 was introduced and after 2 years its official value had dropped to 524.881, whereas its exact value was 1098.811. This difference can be explained by the biased (downward) rounding mode chosen to perform the index updates (cf. [20]). Even the theoretical studies of the numerical stability of an algorithm rely on strong (and unfulfilled) assumptions. For a factorization of dense matrices, it is classically assumed [11] that the number of elementary operations times the machine epsilon should be less than 1/2: $n^3\varepsilon < 1/2$ with n the dimension of the matrix and $\varepsilon = 1^+ - 1$ where 1^+ is the smallest machine number strictly greater than 1. In IEEE double precision this implies that $n < 2.10^5$: this is not satisfied by very large simulations involving millions of variables, for instance parallel computations, even if in practice the results may be satisfactory. Indeed, these theoretical studies account for dense matrices and worst case behaviour.

Some possible alternatives to the usual floating-point arithmetic are exact arithmetic, multiple precision, interval arithmetic... The obvious advantage of the floating-point arithmetic is that an elementary operation is very cheap since it is performed by the hardware, however inaccuracies can be serious and no hint about the quality of a result can be obtained. The multiple precision arithmetic has been introduced in order to mimic the capabilities of floating-point arithmetic while offering a higher precision: the working precision is chosen by the user and every usual function is available. Nevertheless, this arbitrarily high precision only delays the influence of the finite precision on the results accuracy.

Interval arithmetic is designed to provide guaranteed enclosures of every result; it is usually based on floating-point arithmetic in order to benefit from the low cost of operations. The main issue is to get tight intervals as results; increasing the computing precision enables to have sharp intervals as inputs and consequently sharp intervals as results. We propose to combine multiple precision arithmetic, in order to have accurate results, with interval arithmetic in order to preserve the reliability of computations. Combining multiple precision and intervals into a single arithmetic is not new [1, 5, 7, 8, 14, 15, 16, 19, 21]; what is new here is the use of this arithmetic in the interval Newton algorithm. More precisely, our contribution is twofold. The first aspect is the modification of the algorithm in order to adapt automatically the computing precision; the second one consists in fully exploiting the available arbitrary precision in the stopping criterion.

In this paper we present the implementation of a multiple precision interval Newton for univariate functions. The paper is organized as follows. The basic principles of interval arithmetic are recalled in section 2 and we explain how to implement it with an underlying multiple precision arithmetic. Interval Newton method is introduced in section 3 and its multiple precision counterpart in section 4, along with its properties: this algorithm always terminates and provides enclosures of the zeros with arbitrary precision. In section 5 we illustrate the behaviour of this method on some classical examples, before concluding and giving directions for future work.

2 Multiple precision interval arithmetic

2.1 Interval arithmetic

The main principle of interval arithmetic is to replace every real number by an interval enclosing it and whose bounds are representable by the computer [3, 18]. For instance, π can be represented by the interval $[3.14159, 3.14160]$ if 6 significant radix-10 digits are used. Data known with some degree of uncertainty can also be represented, for instance data measured with bounded measurement errors. Interval vectors and interval matrices are vectors and matrices with interval components.

In this paragraph we assume that an ideal arithmetic is used. The roundings effects will be explained in §2.2. Arithmetic operations are defined for interval operands; the semantic of an interval operation $\diamond$ between two intervals X and Y is the following, if it is defined and with [ ] denoting the smallest enclosing interval: $X \diamond Y = [\{x \diamond y \,/\, x \in X,\, y \in Y\}]$. The definitions of the usual functions can also be extended to include interval arguments: for instance $\exp[-2, 3] = [\exp(-2), \exp(3)]$ since exp is an increasing function, and $\sin[\pi/3, \pi] = [0, 1]$, here care must be taken as sin is not monotonous.

The major advantage of this arithmetic is the fact that every result is guaranteed: if input data are scalars and the computations are performed using intervals, the result of the exact scalar calculation belongs to the resulting interval. Nevertheless, this enclosure can be very large: this is due to two phenomena known as the variable dependency and the wrapping effect. The first one can be observed in $X \times (Y + Z) \subset X \times Y + X \times Z$ with an inclusion instead of the expected equality: $X \times Y + X \times Z$ is equal to $\{x \times y + x' \times z, x \in X, x' \in X, y \in Y, z \in Z\}$ where the identity of x and x' is lost. The second one comes from the *wrapping* of every result in $I\!R^n$ into a box with sides parallel to the axes.

Let f be a function and I an interval vector. Different expressions for f, which are mathematically equivalent, do not yield the same result when variables are replaced by intervals. (From now on, a capital letter will refer to one of the interval enclosures of a scalar function.) Usually, these results are much larger than the exact range of f over I. A way to tighten the resulting intervals consists in using Taylor-Lagrange formulas, the first order one being also known as the mean-value theorem. Another method consists in bisecting the input interval vector X into $X_1 \cup X_2$, since $F(X_1) \cup F(X_2) \subset F(X)$ and frequently $F(X_1) \cup F(X_2)$ is tighter than $F(X)$.

2.2 Multiple precision interval arithmetic

Interval arithmetic has been defined from a theoretical point of view in §2.1. In order to implement this arithmetic, directed roundings are needed. In floating-point arithmetic, these roundings are available to the programmer if both the floating-point unit and the programming language comply with the IEEE-754 norm, as far as arithmetic and algebraic operations are concerned. For instance, the floating point representation of $\sqrt{[2,3]}$ is $[\triangledown\sqrt{2}, \triangle\sqrt{3}]$ the outward rounded representable interval for $\sqrt{[2,3]}$, where $\triangledown$ denotes the downward rounding and $\triangle$ the upward rounding. Floating-point arithmetic can be replaced by another arithmetic as long as it provides directed roundings or at least enclosures of the result of every operation.

In some experiments on interval global minimization [10], we encountered the problem of having input intervals bisected until their relative width was ε, the machine epsilon defined in §1, without having reached a satisfactory accuracy on the result. With "flat valley" functions, it is impossible to distinguish which point is the lowest in the valley, using the IEEE double precision, and a paving of the bottom of the valley was returned. With an "egg-box" function f, there is a large number of local minima and the values of f at these points are very close: only a highly accurate evaluation of f permits to discard local minima. This led us to work using multiple precision interval arithmetic.

Our library for multiple precision interval arithmetic is MPFI for *Multiple Precision Floating-point Interval arithmetic*, it is a joint development with F. Rouillier and it is available at `http://www.ens-lyon.fr/~nrevol/software.html`. We have chosen the multiple precision floating-point library MPFR (cf. `http://www.mpfr.org`): this library is IEEE-754 compliant, even for the elementary functions, *i.e.* it provides the correct rounding of the result of every arithmetic operation or elementary function evaluation. Furthermore it is built upon GMP, a library for exact arithmetic known for its efficiency and its portability. Quite simultaneously, the Arithmos library [6] was developed (which is not open-source).

Even if the combination of multiple precision and interval arithmetic is not new, no library was available and only few algorithms were developed that took benefit of this arithmetic. The first use of this idea is mentioned in the conclusion of Brent [5], as a possible application of his MP library for multiple precision arithmetic; it has been implemented in the Augment library [21], but no algorithm has been developed using Augment. However, these libraries are now "obsolescent", according to Brent (cf. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub043.html`). This combination of arithmetics, in the mid eighties, was called "staggered arithmetic" in the XSC languages [15]; the development of these languages has been stopped for a while, during which the work presented here has been conducted, and has started again recently. In [16], an interval Newton algorithm is given: the extra precision is used solely for the accurate evaluation of the function. Another similar approach is to be found in [1]: the *range* library and the accompanying algorithms [2] are developed to evaluate the accuracy of numerical computations and not as an interval arithmetic library; it is also obsolete, because the employed C++ is too old now. Computer algebra systems such as Maple [7, 8] and Mathematica [14] offer interval arithmetic packages, but, for both systems, these packages are bugged because the underlying multiple precision floating-point arithmetic is not reliable. The Maple package is currently undergoing a complete re-design. Finally, Rump's IntLab [19], a MatLab package, includes multiple precision interval arithmetic, but only few functionalities[1] (π and the exponential function) and no algorithm are available.

It has been proven that for a large class of functions called "arithmetic expressions" in [18], the width of a function enclosure is proportional to the width of the input interval if ideal (exact) arithmetic is used. When replacing this ideal arithmetic by a multiple precision one, for the same class of functions, the width of the computed (outward rounded) function enclosure $\widehat{F(X)}$ exceeds the width of $F(X)$ by a constant times ε, where ε, the machine epsilon, has been defined in section 1 and the constant depends on the function: $w(\widehat{F(X)}) \leq w(F(X)) + c_F\varepsilon$, *i.e.* arbitrary accuracy can be reached for the computed function enclosure, cf. [18].

3 Interval Newton method

3.1 Classical interval Newton method

Since our implementation deals only with univariate functions, we will concentrate on functions $f : \mathbb{R} \to \mathbb{R}$. Assume the zeros of f belonging to a given interval X_0 are sought after. The main principle of interval Newton method [12, 9] consists in enclosing the graph of f in the cone given by

[1] At the date of revision of this paper, the functionalities have improved significantly even if the multiple precision toolbox is still "rudimentary, slow but correct" according to Rump himself.

the extremal tangents, cf. the gray zones of figure 1, and in enclosing the zeros in the intersection of this cone and the real axis. Let us denote by X_k the previous iterate and by x_k the point defining the cone (x_k can be any point of X_k, in the following $x_k = \mathrm{mid}(X_k)$, the middle of X_k); the slopes of the limiting straight lines are the extremal values of $F'(X_k)$ where F' is an enclosure function for f'. Since the zeros are sought in X_k, the new search interval X_{k+1} is defined as

$$X_{k+1} := (x_k - \frac{f(x_k)}{F'(X_k)}) \cap X_k.$$

If there are more than one zero in X_0, the derivative f' will vanish on X_0. If $F'(X_k)$ contains 0, then an extended interval division is performed and after the intersection with X_k, two subintervals contained in X_k are produced, cf. the right part of figure 1. The result of the interval Newton method is a list of intervals whose union contains every zero. Let us insist here on the following property of the interval Newton algorithm: no root belonging to X_0 is lost. In other words, every zero appears in one of the intervals of the resulting list. This list may also contain intervals that do not contain any zero but that could not be discarded if their interval image by f contains 0.

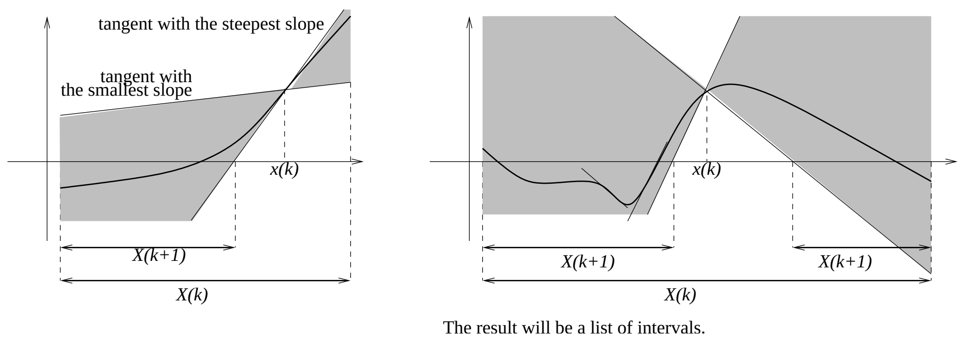

Figure 1: The principle of Newton method.

When the interval arithmetic uses fixed precision floating-point arithmetic to compute the interval bounds, then the stopping criterion is

$$w(X)/|\mathrm{mid}(X)| \le \varepsilon_X \text{ or } w(F(X)) \le \varepsilon_Y.$$

Only one of the two conditions needs to be fulfilled, since it may happen that $w(X)$ is one ulp (*unit in the last place*) of the center of X, *i.e.* $X = [x^-, x^+]$ with x^- and x^+ two consecutive floating-point numbers whereas $F(X)$ cannot be made smaller than ε_Y (and conversely).

3.2 Properties of the classical interval Newton method

These properties have been established for the interval Newton method and they hold for fixed precision floating-point interval arithmetic as well as for multiple precision or even exact (theoretical) interval arithmetic.

Proof of existence of a zero
Brouwer's theorem is effective in interval arithmetic: if $x - f(x)/F'(X) \subset X$ then there exists one zero of f in X. Furthermore, since f is univariate, the intermediate value theorem applies: the existence of a zero in $[a, b]$ is ensured if $f(a) \times f(b) < 0$. With interval arithmetic, this can be translated into "let us denote $F([a])$ by $[\alpha^-, \alpha^+]$ and $F([b])$ by $[\beta^-, \beta^+]$, there exists a zero of f in $[a, b]$ if $\alpha^+ < 0 < \beta^-$ or $\beta^+ < 0 < \alpha^-$".

Proof of existence and uniqueness of a zero
If there is a strict shrink of the search interval during interval Newton iteration, *i.e.* if $x - f(x)/F'(X)$ is strictly contained in the interior of X, then the existence and uniqueness of a zero in X is proven.

4 Multiple precision interval Newton method

When an arbitrary accuracy can be achieved on a function enclosure, then the stopping criterion can be made more stringent. Moreover, in order to ensure the termination of the algorithm, we ensure that the width of the intervals in the queue $\mathcal{L}$ is quickly decreasing. These considerations have led us to propose the following algorithm.

4.1 Multiple precision interval Newton method

Input: f, F', X_0 *the initial search interval*
Initialization: $\mathcal{L} = \{X_0\}$, $\alpha = 0.75$ *(any value in* $]0.5, 1[$ *is suitable)*
Loop: *while* $\mathcal{L} \neq \emptyset$
    Suppress $(X, \mathcal{L})$
    $(X_1, X_2) := (x - \frac{f(x)}{F'(X)}) \cap X$
        (X_2 *can be empty, tests on* X_2 *are performed if* $X_2 \neq \emptyset$*)*
    if $w(X_1) > \alpha w(X)$ *or* $w(X_2) > \alpha w(X)$ *then*
        $(X_1, X_2) :=$ *bisect*(X) *in two halves of equal length*
    Increase the working precision if needed
    if $X_1 \neq \emptyset$ *and* $F(X_1) \ni 0$
        if $w(X_1) \leq \varepsilon_X$ *and* $w(F(X_1)) \leq \varepsilon_Y$ *then*

Insert $(X_1,$ *Res) and test existence and uniqueness*

else

Insert $(X_1, \mathcal{L})$

ditto for X_2

Output: *Res, a list of intervals that may contain the zeros.*

4.2 Properties

The stopping criterion requires that both the width of the zero enclosure and the function evaluation are small enough: the "OR" of §3.1 is replaced by a "AND" in our algorithm. The termination of the algorithm is established by noting that the widths of the examined intervals are upper bounded by a geometric series of ratio α , where $\alpha = 0.75$ in our algorithm but can be any value in $]0.5, 1[$ (for a discussion on stopping criteria, cf. [13]), and also that the widths of the computed function evaluations are upper bounded by a geometric series of ratio α plus the constant term $c_F\varepsilon$ mentioned in §2.2. This can be achieved if the working precision is sufficient.

A second peculiarity of our algorithm is its automatic and dynamic adaptation of the working precision. The precision is increased when one of the following conditions holds:

- $w(X_1)$ or $w(X_2) \geq w(X)$: the width of X is one ulp of $\mathrm{mid}(X)$ and no further refinement is possible using the current precision;
- $w(\widehat{F(X_1)})$ or $w(\widehat{F(X_2)}) \geq w(\widehat{F(X)})$: it implies that $w(F(X)) \leq w(\widehat{F(X_1)})$ $\leq w(F(X_1)) + c_F\varepsilon \leq w(F(X)) + c_F\varepsilon$, *i.e.* the rounding error is greater than the gain obtained by replacing X by X_1 or X_2 in the evaluation by F.

Since Newton's algorithm doubles the accuracy on the zero at each iteration (at least for a single root), we chose to double the current precision when it needs to be increased. In order to minimize the computational cost, the working precision is stored with the interval in the queue $\mathcal{L}$ and is restored when the corresponding interval is examined.

A particularly pleasant feature of Newton's algorithm is that it is autocorrective: when the precision is increased, only further computations use this higher precision, but the whole computation does not need to be restarted.

5 Experimental results

Our implementation of Newton method presented in §4 deals only with univariate functions for the time being. The multidimensional case requires the solution of a linear system involving the Jacobian and this is not yet implemented for the multiple precision interval arithmetic.

The experiments presented here illustrate the behaviour of this algorithm for some quite difficult problems, even if they imply only polynomials.

The first series concerns the Chebychev polynomials which can be defined for instance by: $C_n(\cos\theta) = \cos n\theta$. The degree of C_n is n. They are known to be difficult to evaluate accurately even if they take their values in $[-1, 1]$, because their coefficients are large. A consequence is thus that it is quite difficult to get a small "residual" $F(X)$, smaller than the stopping threshold ε_Y. The benefit of using a multiple precision arithmetic appears clearly: the polynomial evaluations are tighter and thus intervals are rejected earlier when a high precision is used than with a low precision. Furthermore, even if all roots of Chebychev polynomials are simple roots, most of them form two clusters, one close to -1 and symmetrically the other one close to 1. This means that they are difficult to determine numerically. For instance, MatLab determines only 6 roots of C_{30}, the Chebychev polynomial of degree 30 (it finds 24 complex roots for the 24 remaining ones), with 5 correct decimal digits. It finds only 8 roots of C_{26}, with 3 correct decimal digits. Yet the coefficients of C_{26} or of C_{30} are exactly representable by machine numbers and these results are not due to the approximation of the coefficients by double precision floating-point numbers. Figure 2 illustrates the power of interval Newton iteration: the starting search interval is $[-2, 2]$, $\varepsilon_X = \varepsilon_Y = 10^{-6}$ and the starting precision is 24 bits, which corresponds to the IEEE single precision. At most two doublings of the precision occurred. The results are very satisfactory since no superfluous intervals are given in the list of potential results and the existence (and uniqueness) of the roots in each isolating interval is proven for most of the intervals.

A second series presents quite the same conclusions obtained with the Wilkinson polynomials $W_n(x) = \prod_{i=1}^{n}(x - i)$ written in the expanded form. The initial precision is chosen large enough to enable the exact representation of the coefficients. These polynomials are difficult to evaluate precisely even in multiple precision "scalar" arithmetic; this can easily be checked

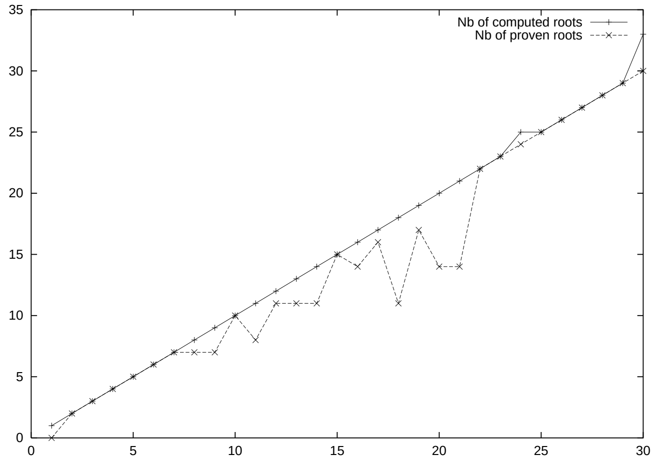

Figure 2: Number of computed roots and roots with proven existence.

thanks to the forward error bound given in [11]:

$$|p(x) - \widehat{q_0}| \leq 2n\varepsilon \sum_{i=0}^{n} |a_i|.|x|^i$$

where $p(x) = \sum_{i=0}^{n} a_i x^i$ is the polynomial to evaluate and $\widehat{q_0}$ is the value computed by Horner's rule. Since the coefficients of W_n are of the order of magnitude of $n!$, the assertion concerning the bad quality of the evaluation of Wilkinson polynomials is obtained. The results confirm the difficulty of accurately evaluating a Wilkinson polynomial: the required precision on the range (called ε_Y in our algorithm) cannot be made much smaller than $\mathcal{O}(n!)$. Consequently it is very difficult for our algorithm (essentially very time-consuming) to discard intervals not containing zero. The results are thus small enclosures for the roots along with a proof of their existence and uniqueness and a long list of other intervals, covering almost the whole interval $[1, n]$.

For $n = 20$ and with the coefficient of X^{19} perturbed by the interval $[-2^{-19}, 2^{-19}]$, then every point between 8 and 20 is a root of a perturbed polynomial belonging to this interval polynomial; indeed, our algorithm returns small enclosures for the roots 1 to 7 and a covering of $[7.91, 22.11]$.

Lastly, a polynomial having a double root is tested [17]. It is known that for a root of multiplicity μ the backward error is of order $\varepsilon^{1/\mu}$, or roughly speaking that in order to get an error of order ε_X on the root, an error ε_X^{μ} must be obtained for the polynomial evaluation. This intuitive fact is experimentally confirmed. Moreover, interval Newton iteration even enables to prove that the decimal to binary coefficient conversion with rounding to nearest has transformed the multiple root into two close single roots.

6 Conclusion and future work

In this paper we have presented an algorithm for the determination of real zeros of univariate equations. This algorithm is based on interval arithmetic in order to provide guaranteed enclosures of every zero in the initial search domain and it also benefits from multiple precision arithmetic which enables it to compute arbitrarily accurate enclosures. Its convergence towards arbitrarily tight enclosures and its termination property have been proven. Experimental results illustrate the behaviour of this algorithm on some difficult problems and the results are conform to what was expected.

A promising direction consists in exploring multiple precision interval arithmetic in conjunction with higher order methods, such as those described in [4]. For these methods indeed, the error due to the limited computing precision becomes large in comparison with the error of approximation, as the algorithm proceeds, quicker than with lower order methods. Being able to increase the computing precision reveals even more useful.

The next step of this work consists in the study and implementation of an algorithm solving linear systems. It will enable us to implement a multivariate Newton method.

References

[1] O. Aberth, M.J. Schaefer. Precise computation using range arithmetic, via C++. *ACM TOMS*, 18(4):481–491, 1992.

[2] O. Aberth. Precise numerical methods using C++. Academic Press, 1998.

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

[4] G. Alefeld, F. Potra. A new class of interval methods with higher order of convergence. *Computing*, 42(1):69–80, 1989.

[5] R. P. Brent. A Fortran multiple-precision arithmetic package. *ACM TOMS*, 4:57–70, 1978.

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

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

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

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

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

[11] N. Higham. *Accuracy and stability of numerical algorithms.* 2nd ed., SIAM, 2002.

[12] R.B. Kearfott. Empirical evaluation of innovations in interval branch and bound algorithms for nonlinear systems. *SIAM J. Sci. Comput.*, 18(2):574–594, 1997.

[13] R.B. Kearfott, G.W. Walster. On stopping criteria in verified nonlinear systems or optimization algorithms. *ACM TOMS*, 26(3):373–389, 2000.

[14] J. Keiper. Interval arithmetic in Mathematica. *Interval Computations*, (3), 1993.

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

[16] W. Krämer, U. Kulisch, R. Lohner. *Numerical toolbox for verified computing II – Advanced Numerical Problems.* To appear (1998).

[17] P. Langlois, N. Revol. Validating polynomial numerical computations with complementary automatic methods. RR 2001-18, LIP, ENS Lyon, `http://www.ens-lyon.fr/LIP/Pub/rr2001.html`, 2001.

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

[19] S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.

[20] G.W. Stewart. What is rounding error? `http://www.cs.umd.edu/~stewart`, in `/pub/misc`, 2000.

[21] J. M. Yohe. Portable software for interval arithmetic. *Computing*, Suppl. 2, 211–229, 1980.