

A certified infinite norm for the implementation of elementary functions

Sylvain Chevillard, Christoph Quirin Lauter

▶ **To cite this version:**

Sylvain Chevillard, Christoph Quirin Lauter. A certified infinite norm for the implementation of elementary functions. [Research Report] Laboratoire de l'informatique du parallélisme. 2007, 2+11p. hal-02102507

HAL Id: hal-02102507

https://hal-lara.archives-ouvertes.fr/hal-02102507

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

A certified infinite norm for the implementation of elementary functions

Sylvain Chevillard,
Christoph Quirin Lauter

June 2007

Research Report N° RR2007-26

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`







A certified infinite norm for the implementation of elementary functions

Sylvain Chevillard, Christoph Quirin Lauter

June 2007

Abstract

The high-quality floating-point implementation of useful functions $f : \mathbb{R} \to \mathbb{R}$, such as exp, sin, erf requires bounding the error $\varepsilon = \frac{p-f}{f}$ of an approximation p with regard to the function f. This involves bounding the infinite norm $\|\varepsilon\|_\infty$ of the error function. Its value must not be underestimated when implementations must be safe.

Previous approaches for computing infinite norm are shown to be either unsafe, not sufficiently tight or too tedious in manual work.

We present a safe and self-validating algorithm for automatically upper- and lower-bounding infinite norms of error functions. The algorithm is based on enhanced interval arithmetic. It can overcome high cancellation and high condition number around points where the error function is defined only by continuous extension.

The given algorithm is implemented in a software tool. It can generate a proof of correctness for each instance on which it is run.

Keywords: Certified infinite norm, interval arithmetic, elementary functions

Résumé

Pour garantir la qualité de l'implémentation en arithmétique flottante de fonctions usuelles $f : \mathbb{R} \to \mathbb{R}$ telles que exp, sin, erf, il faut borner l'erreur $\varepsilon = \frac{p-f}{f}$ commise entre f et une approximation p. Cela implique de borner la norme infinie $\|\varepsilon\|_\infty$ de la fonction d'erreur. Si on veut que l'implémentation soit sure, on ne doit en aucun cas renvoyer une borne inférieure à la valeur exacte.

Nous montrons que les approches précédentes visant à calculer la norme infinie ne sont pas satisfaisantes : soit elles ne sont pas sures, soit pas assez précises, soit elles nécessitent un travail manuel trop fastidieux.

Nous présentons un algorithme sûr, qui fournit une preuve de sa propre correction, et qui minore et majore automatiquement la norme infinie de fonctions d'erreur. Cet algorithme est fondé sur une version améliorée d'arithmétique d'intervalle. Il peut contourner les difficultés dues à une grande cancellation et un mauvais conditionnement autour de points où la fonction d'erreur n'est définie que par continuité.

L'algorithme proposé a été implémenté dans un outil logiciel. Il peut générer une preuve de correction pour toute instance sur laquelle il est exécuté.

Mots-clés: Norme infinie certifiée, arithmétique d'intervalles, fonctions élémentaires

1 Introduction

Floating-point environments are the base of many software systems. Examples include scientific computing, financial applications and embedded systems. These software systems use floating-point implementations of useful functions $f : \mathbb{R} \to \mathbb{R}$: exp, sin, cos, erf or some composites like $\exp -1$, for instance.

When it comes to implement such a function f, an approximation to f must be used [14, 13]. This approximation leads to some error $\varepsilon(x)$ for each argument x in the definition domain I.

The quality of an implementation is determined by the maximal error in the definition domain. High quality implementations must provide guarantees of their validity. Considering the error ε as a function of x, determining and certifying the maximal error means computing the infinite norm of ε,

$$\|\varepsilon\|_\infty^I = \max_{x \in I} |\varepsilon(x)| ,$$

without underestimating it.

This paper analyzes the requirements to an algorithm for such infinite norms. Previous approaches are shown to be unsafe or unsatisfactory. We propose a self-validating algorithm for this task. We have implemented the algorithm in a software tool*.

1.1 Framework: implementation of functions

The most useful mathematical functions $f : \mathbb{R} \to \mathbb{R}$, as for example exp, sin, log or erf, some of which are called elementary [14], are implemented in so-called mathematical libraries (`libm`). Techniques [14, 6] used for implementing elementary functions transpose to other useful smooth functions $f \in \mathcal{C}^\infty$.

Implementations are mostly based on the IEEE 754 floating-point standard [3]. This standard defines binary floating-point formats $\mathcal{F}_t = \left\{2^E \cdot m | E \in \mathbb{Z}, m \in \mathbb{Z}, 2^{t-1} \leq |m| \leq 2^t - 1\right\} \cup \{0\}$ of precision t. For example, with the required bounds on E, $\mathcal{F}_{24}$ is the single precision format. The standard specifies also rounding modes, for example round-to-nearest, $\circ(x)$. The basic operations defined by the standard, $\oplus, \ominus, \otimes, \oslash$ and `sqrt`, are the rounding of their infinitely precise equivalents, $+, -, \times, /$ and $\sqrt{}$. For example, $a \oplus b = \circ(a + b)$.

Current hardware shows particularly high performance on addition and multiplication [13]. Elementary functions $\tilde{f} : \mathbb{R} \to \mathbb{R}$ are thus implemented in software or microcode following an approach where tabulation is combined with polynomial approximation:

1. Argument reduction uses algebraic properties of the function $\tilde{f}$ to relate it to a function f. The argument x of f lies in a small domain I, most usually around 0. Argument reduction may use tables with pre-computed values. This step may induce some error, called reduction error.

2. In the small domain I, the function is then approximated by a polynomial $p = c_0 + x \cdot (c_1 + x \ldots)$ of some degree with floating-point coefficients $c_i \in \mathcal{F}_t$. This step causes the so-called approximation error $\varepsilon = \frac{p-f}{f}$.

3. Implemented in floating-point arithmetic, for example as $P = c_0 \oplus x \otimes (c_1 \oplus x \ldots)$, the polynomial p is evaluated in an arithmetic subject to rounding. The induced error is called round-off error $E = \frac{p-P}{p}$ [7].

4. A reconstruction step finally combines table values and polynomial approximations in order to retrieve the original function $\tilde{f}$ from the approximation p of f. Its error is called reconstruction error [14, 13].

The errors of the different steps combine to one overall-error, which is the error of the floating-point number returned by the code of the function on argument x with regard to the real value $\tilde{f}(x)$ [1, 7].

*available at http://lipforge.ens-lyon.fr/projects/arenaireplot/

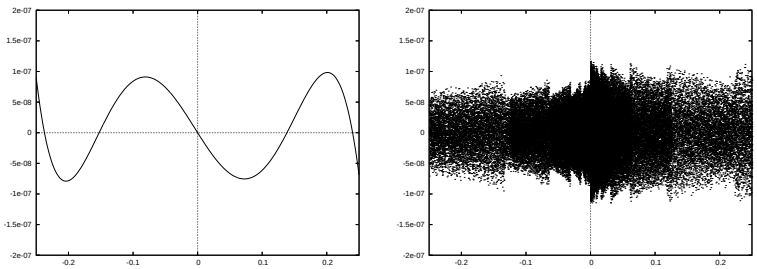
As an example consider the function $f = \exp -1$. Suppose that error-free argument reduction has already brought its argument on the domain $I = \left[-\frac{1}{4}; \frac{1}{4}\right]$. The function can be approximated on this domain using the polynomial

$$p(x) = x \cdot (1 + x \cdot (2097145 \cdot 2^{-22} + x \cdot (349527 \cdot 2^{-21} + x \cdot (87609 \cdot 2^{-21} + x \cdot 4369 \cdot 2^{-19}))))$$

Let this polynomial be implemented in Horner's scheme in IEEE 754 single precision:

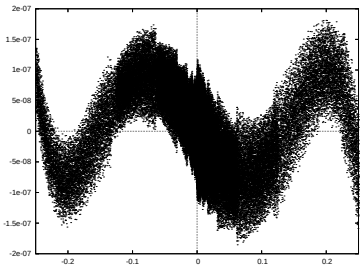
$$P(x) = x \otimes (1 \oplus x \otimes (2097145 \cdot 2^{-22} \oplus x \otimes (349527 \cdot 2^{-21} \oplus x \otimes (87609 \cdot 2^{-21} \oplus x \otimes 4369 \cdot 2^{-19}))))$$

Figure 1 plots the approximation and round-off-error and their combination in the overall error.



(a) Approximation Error ε

(b) Roundoff Error E



(c) Overall Error

Figure 1: Sources of error as functions of x

1.2 Challenge and contributions

The quality of an implementation of a function is mainly determined by its overall-error [7, 1, 13]. Moreover, specifications for the implementation of functions often fix a maximum bound for this error [7, 6].

Showing that the overall-error of an implementation is less than a specified bound is the key to the proof of its correctness. Since the safety of a software system may rely on an implementation of a function, the bound on the overall-error must be computed in a safe way. When it comes to guarantee the bound, it must not be underestimated in any case.

The overall-error is generally upper bounded considering the different errors separately. Amongst them, the reduction and reconstruction error can be handled by ad-hoc means [1, 14, 7, 10].

The round-off and approximation error are the main issue. The situation in their analysis is not balanced:

- The round-off error E is a discrete function $\mathcal{F}_t \to \mathbb{R}$ with chaotic behavior (see Figure 1(b)). Although classical analysis does not allow to bound it, different means are known and useable.

  Firstly, a manual study of the error terms induced by each operation [10] allows for obtaining relatively tight bounds and quite satisfactory safety. Secondly, approaches using formal proof checkers like HOL, COQ or PVS [9, 5] increase the safety at the cost of more tedious and

complex proofs [7]. Finally, tools like Gappa allow for automatic analysis of round-off error and verification in formal proof checkers [7].

Using Gappa, the analysis of the discrete round-off error function becomes relatively easy and completely safe. We will not further address this problem.

- In contrast, the approximation error $\varepsilon = \frac{p-f}{f}$ of a polynomial p with regard to a smooth function f gives also a smooth function (see Figure 1(a)) in practical cases. Classical analysis is thus appropriate for calculating and proving a bound on the infinite norm $\|\varepsilon\|_\infty^I$. The problem may seem easy.

  Nevertheless, previous approaches and algorithms do not satisfy the needs of a safe implementation: the infinite norm is either underestimated or the approach is too tedious and case-specific.

This paper studies the related approaches for bounding such infinite norms in Section 2.1. This study makes it possible to specify a new algorithm in Section 2.2. It proposes an implementation of this algorithm responding to these specifications in Section 3. It is useful on real-life examples encountered in the implementation of functions. Section 4 illustrates this point. Before concluding in Section 6, some of its limitations will be shown in Section 5.

2 Analysis of the problem and specifications of the algorithm

2.1 Related work

Previous approaches for bounding the infinite norm $\|\varepsilon\|_\infty$ of an error function $\varepsilon = \frac{p-f}{f}$ fall in two categories with regard to their incompatibility with safe and fully automated implementation of functions:

- Floating-point techniques as proposed by Brent [4] may return underestimations to the infinite norm. They are therefore qualified as *unsafe* in this paper. Common software tools like Maple and Matlab implement similar algorithm.

  For example, reconsider the function $f = \exp -1$ and the polynomial p given in Section 1.1. Independently of its internal precision, Maple returns the value $0.983491319532\ldots \cdot 10^{-7}$ for $\left\|\frac{p-f}{f}\right\|_\infty^{\left[-\frac{1}{4},\frac{1}{4}\right]}$ (see Section 4 for more details). This is an underestimate by at least $1.8 \cdot 10^{-17}$ because $\varepsilon(843485 \cdot 2^{-22}) = 0.983491319722\ldots \cdot 10^{-7}$.

  Known software glitches show that even such small differences must not be neglected. Nevertheless, Maple has been used for bounding error functions in libraries like CRLibm that claim to have proofs for providing correctly rounded, bit-exact results [1].

- Other approaches increase safety but require much more tedious, manual work or very high computation time for obtaining tight bounds. They are all based on a high order Taylor development p^* of the function f. The infinite norm $\|p-f\|_\infty$ is upper bounded using the triangular inequality $|p(x)-f(x)| \leq |p(x)-p^*(x)| + |p^*(x)-f(x)|$. The bound on $\|p^*-f\|_\infty$, i.e. on the remainder term the development p^*, is generally shown using paper and pencil. This may become very difficult for composites of basic functions, even more if the process is to be automated.

  Krämer gives a technique used in the development of the FL_LIB library [12]. The approach uses interval arithmetic for bounding $\|p-p^*\|_\infty$. The remainder bound is shown on paper. The result may suffer from a bug in the implementation. No formal proof is produced. The results are not very tight if they come near the machine precision [11].

Harrison shows the correctness of an implementation of the function exp in the formal proof checker HOL [9]. His proof is very tedious and not resistant to changes in the implementation. The bound on the remainder term is shown manually and just checked automatically.

Taylor model based approaches allow for computing a bound on the remainder term $p^* - f$. Techniques have been proposed for PVS and other Taylor model based tools [5, 15]. They often require expensive computations [5].

2.2 Specifications

In order to implement a function f using a polynomial approximation p in a safe way, one wants to know a bound u such that, for each point $x \in I$, the approximation error $|\varepsilon(x)|$ is not greater than u. The infinite norm $\|\varepsilon\|_\infty$ is the best possible answer but this result is rarely directly reachable and one just knows an approximated value. This leads us to our first requirement for an algorithm bounding infinite norms:

Requirement 1 (Safety)**.** *When the algorithm cannot return the exact value of $\|\varepsilon\|_\infty$, it shall return an upper-approximated value u.*

This requirement is essential to the safety of the implementation. However it does not imply anything about the quality of the approximated infinite norm u with respect to the value $\|\varepsilon\|_\infty$ it is supposed to represent. This leads to the following requirement:

Requirement 2 (Quality)**.** *The algorithm shall return a lower-approximated value ℓ of $\|\varepsilon\|_\infty$.*

Thus, one knows a range $[\ell, u]$ where the exact value lies. An algorithm implementing the specifications inhere may depend on some parameters. If the range $[\ell, u]$ is too large, u may overestimate $\|\varepsilon\|_\infty$ too much. In this case, one may restart the algorithm with better chosen parameters in order to get a better estimation of the actual value $\|\varepsilon\|_\infty$.

Let us remark that classical numerical analysis techniques generally allow one to get a point x_0 where the infinite norm is almost reached. Most of the time, it is hence not very difficult to obtain such a lower bound ℓ: $|\varepsilon(x_0)|$ is a good one. The difficulty of the problem we address comes from the fact that we want to get $u \approx \|\varepsilon\|_\infty$ with the guarantee that it is an upper bound: $u \geq \|\varepsilon\|_\infty$.

The algorithm to compute u will probably be complex and its implementation could contain some bugs. This is why we introduce a third requirement to guarantee the safety of the result:

Requirement 3 (Automatic proof)**.** *Together with the numerical result $[\ell, u]$, the algorithm shall return a proof of the claim $\|\varepsilon\|_\infty \in [\ell, u]$ that can be checked independently (ideally with an automatic proof checker like COQ or PVS).*

Now, let us make two remarks regarding the specificity of the context in which we compute infinite norms. First notice that the algorithm will have to subtract $f(x)$ from $p(x)$, two quantities very close to each other:

Requirement 4 (High-cancellation)**.** *The algorithm should return accurate results, even when p is an excellent approximation to f, e.g. $\varepsilon(x)$ is obtained from a highly-cancellating subtraction.*

There is often a point $z \in I$ where the expression $\varepsilon = (p - f)/f$ is not defined because the function f has a zero at z. Nevertheless, the developer who implements such a function always tries to keep ε bounded in the neighborhood of z. For this reason, most of the time, p has a zero at z at least of the same order as f. Thus, the function ε extends by continuity at z, even if the expression in undefined in z. This leads to our last requirement:

Requirement 5 (Continuous extension)**.** *The algorithm should be able to return accurate results even when function ε is only defined by continuous extension at some point z. However, the safety shall not be compromised by this requirement: the value $+\infty$ is obviously better than a value less than $\|\varepsilon\|_\infty$.*

3 The algorithm

3.1 Assumptions

In order to satisfy these requirements, in particular the safety requirement, we have chosen to use multi-precision interval arithmetic [16, 2]. The general property of interval arithmetic is the so-called inclusion property: given a function φ and an interval I, the computer evaluation of φ on I in interval arithmetic returns an interval J such that $\varphi(I) \subseteq J$. Interval arithmetic naturally takes round-off into account internally. It provides mathematically valid results. We will define below a procedure `eval` satisfying this property.

In the following, the considered intervals are always supposed compact. If I denotes an interval, $\overline{I}$ will denote its upper bound, $\underline{I}$ will denote its lower bound, $\mathrm{mid}(I) = (\overline{I}+\underline{I})/2$ and $\mathrm{diam}(I) = \overline{I}-\underline{I}$.

We will further assume that f and $\varepsilon = (p-f)/f$ are smooth functions on I given as expression trees. Our algorithm manipulates some derivatives of functions. These derivatives are obtained by symbolic differentiation. This is a design choice; other techniques may be appropriated (see Section 5).

3.2 General scheme of the algorithm

The proposed algorithm uses the following elementary theorem:

Theorem 3.1. *Let φ be a differentiable function on a closed interval $[a, b]$. The function has a maximum on $[a, b]$ and this maximum is reached:*

- *either at a or b;*
- *or at a point c such that $\varphi'(c) = 0$.*

The same holds for the minimum.

The principle of our algorithm consists in applying the previous theorem to ε, and boxing rigorously the zeros of function ε', using a sub-procedure `boxZeros` (described in Section 3.4). The general scheme of the algorithm is shown in Algorithm 1.

Algorithm: `CertifiedInfnorm`
Input: An error function $\varepsilon = (p - f)/f$ and a closed interval $[a, b]$
A parameter t controlling the internal precision to be used in the computations
A parameter Δ controlling the maximal diameter of the zero boxes used in the algorithm
A parameter N controlling the maximal degree of recursion in `eval`
Result: An interval $[\ell, u]$ such that $\|\varepsilon\|_\infty \in [\ell, u]$
begin
  Box the zeros of ε': $\mathcal{B} :=$ `boxZeros`$(\varepsilon', [a, b], \Delta)$;
  Add the two endpoints: $Z_{\text{left}} := [a, a]$; $Z_{\text{right}} := [b, b]$; $\mathcal{Z} := \{Z_{\text{left}}\} \cup \mathcal{B} \cup \{Z_{\text{right}}\}$;
  forall $Z_i \in \mathcal{Z}$ **do**
    Evaluate ε on the box Z_i: $Y_i :=$ `eval`(ε, Z_i, t, N);
  end
  Deduce an interval $[\ell, u]$ around the infinite norm;
  (If asked for, generate a proof of the result);
  return $[\ell, u]$;
end

Algorithm 1: General scheme of our algorithm

By boxing the zeros, we mean finding a finite list $\mathcal{B}$ of disjoint intervals $\mathcal{B} = \{Z_1, Z_2, \ldots\}$ such that every zero of ε' lies in one Z_i. A parameter Δ controls the maximal diameter allowed for a Z_i. Note that there can be some $Z_i \in \mathcal{Z}$ that does not contain any zero of ε' and that a Z_i may contain two distinct zeros.

Theorem 3.1 indicates that the extrema of ε are reached either at a zero of ε' or at an endpoint of the input interval $[a, b]$. This is why the two thin intervals $[a, a]$ and $[b, b]$ are then added to $\mathcal{B}$ for obtaining a list $\mathcal{Z}$.

Then the algorithm evaluates ε on the boxes: it means that for each $Z_i \in \mathcal{Z}$, we call our function `eval` that returns an interval Y_i that $\varepsilon(Z_i) \subseteq Y_i$. Two parameters t and N control the accuracy of the result Y_i with regard to $\varepsilon(Z_i)$. The details of the algorithm and how the parameters influence the result will be explained in Section 3.3.

We explain now how bounds ℓ and u for the maximum of ε can be deduced from the list of the Y_i. The same method applies for the minimum. From bounds for the minimum and the maximum, it is easy to get bounds for the infinite norm. Let $x^\star$ denote a point where the maximum of ε is reached. By construction of $\mathcal{Z}$, $x^\star$ lies in a $Z \in \mathcal{Z}$.

As shown in Figure 2, and since $\varepsilon(Z) \subseteq Y$, the upper-bound $\overline{Y}$ is greater than the maximum of the function. Hence, the maximum of all the $\overline{Y_i}$ is greater than the maximum of ε on $[a, b]$.

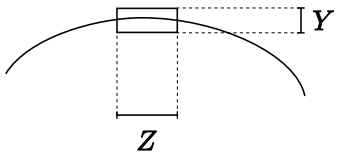


Figure 2: Zoom on the maximum

Let us remark that if $\mathrm{diam}(Z)$ is small enough, $\varepsilon(Z)$ will be small too. So, if the result Y of $\texttt{eval}(\varepsilon, Z, t, N)$ is tight enough, $\mathrm{diam}(Y)$ is also small. Thus, the lower-bound $\underline{Y}$ of Y is a pretty good under-approximation of the maximum (see Figure 2). It follows that the maximum of all the $\underline{Y_i}$ is a good under-approximation of the maximum.

These techniques are known as the computation of an inner enclosure and an outer enclosure of $\varepsilon(Z)$. For technical details about it, see [16] for example.

Assume that `eval` and `boxZeros` can produce a formal proof of their result (see Sections 3.3 and 3.4). To get a proof for the whole algorithm, one just needs to implement a proof for Theorem 3.1 and for the algorithm computing the inner- and outer- enclosure of the image of an interval by ε. The current version of our software produces proofs written in English. However, the method has been designed with the goal of generating formal proofs, which should not be much more difficult.

As will be seen, procedure `boxZeros` needs to evaluate functions on intervals; this is why we first explain procedure `eval`.

3.3 Evaluating a function on an interval

In this section, we are going to present an algorithm $\texttt{eval}(\varphi, I, t, N)$ that computes J such that $\varphi(I) \subseteq J$. It is given φ as an expression, the interval I, an internal precision t (in bits) and a level of recursion N. It is based on another algorithm for direct interval evaluation, `direval`, detailed first.

The expression φ is built up of n-ary basic functions ψ such as: $+$, $-$, $\times$, $/$, exp, sin, erf. For all these functions ψ, we have a procedure $\texttt{baseeval}(\psi, I_1, \ldots, I_n, t)$ that computes an interval J such that $\psi(I_1, \ldots, I_n) \subseteq J$. The interval J has floating-point endpoints. Their precision is controlled by the parameter t. A procedure $\texttt{direval}(\varphi, I, t)$ returning $J \supseteq \varphi(I)$ can be built as follows: expression trees are recursively evaluated bottom-up using `baseeval` for the basic functions. The correctness of this algorithm follows from the inclusion property of interval arithmetic [2] by induction on the expression tree φ. The MPFI library[†] implements an evaluation procedure `baseeval` for common functions. We use that library.

This naïve algorithm does not respect Requirement 5 given in Section 2.2. In the case of a division, $\texttt{baseeval}(/, J_1, J_2, t)$ returns $[-\infty, +\infty]$ if the denominator interval J_2 contains 0. Nevertheless in such a situation it may be the case that function $\varphi = \theta_1/\theta_2$ can be extended by

[†] available at http://gforge.inria.fr/projects/mpfi/

continuity in a zero $z \in I$ of θ_2 and that it has finite bounds. An interval variant of L'Hôpital's rule solves the issue:

Proposition 3.1. *If θ_1 and θ_2 are $\mathcal{C}^\infty$ on I, if there is $z \in I$ such that $\theta_1(z) = \theta_2(z) = 0$ and if θ_1/θ_2 is nevertheless $\mathcal{C}^\infty$ on I then*

$$\frac{\theta_1}{\theta_2}(I) \subseteq \left\{ \frac{\theta_1'(x)}{\theta_2'(y)}, \text{ with } (x,y) \in I^2 \right\} = \frac{\theta_1'(I)}{\theta_2'(I)}.$$

Proof. Let x be in I. If $\theta_2(x) \neq 0$, it holds by the mean value theorem, as claimed:

$$\exists \xi_1 \in I, \exists \xi_2 \in I, \frac{\theta_1(x)}{\theta_2(x)} = \frac{\theta_1(z) + (x-z)\cdot\theta_1'(\xi_1)}{\theta_2(z) + (x-z)\cdot\theta_2'(\xi_2)} = \frac{\theta_1'(\xi_1)}{\theta_2'(\xi_2)}$$

If $\theta_2(x) = 0$, we may assume $x = z$. If z is an accumulation point of the zeros of θ_2, it is easy to see that it is also an accumulation point of zeros of θ_2'. In particular, by continuity of θ_2', $\theta_2'(z) = 0$. Thus, $\theta_1'(I)/\theta_2'(I) = [-\infty, +\infty]$ and the proposition holds.

If z is not an accumulation point of the zeros of θ_2, there exists an interval $\tilde{I} \subseteq I$ where θ_2 has only one zero z. Since $\forall x \neq z \in \tilde{I}$,

$$\frac{\theta_1(x)}{\theta_2(x)} \in \frac{\theta_1'(I)}{\theta_2'(I)}$$

by the mean value theorem, and since $\theta_1'(I)/\theta_2'(I)$ is a closed interval in $\mathbb{R} \cup \{-\infty, +\infty\}$,

$$\left(\lim_{x \to z} \frac{\theta_1(x)}{\theta_2(x)} \right) \in \frac{\theta_1'(I)}{\theta_2'(I)}.$$

□

The application of L'Hôpital's rule must not endanger the safety of the algorithm. Finding a zero $z \in \mathcal{F}_t$ is easy in common cases using a floating-point Newton-Raphson iteration. However, since it is obtained by an uncertified floating-point process, the found z must not be used immediately. It must be proven that $\theta_1(z) = \theta_2(z) = 0$. The inclusion property of interval arithmetic provides the base: since $\theta_i([z,z]) \subseteq \texttt{direval}(\theta_i, [z,z], t)$, if $\texttt{direval}(\theta_i, [z,z], t) = [0,0]$, it holds that $\theta_i([z,z]) = [0,0]$ and $\theta_i(z) = 0$. If the interval evaluation of $\texttt{direval}(\theta_i, [z,z], t)$ does not permit concluding, the rule is not applied, in which case the bound is infinite.

It might be argued that the application of L'Hôpital's rule in our algorithm is subject to too many conditions that are all influenced by overestimates in the underlying interval arithmetic. Although it may not work for general functions φ, it is appropriated for functions $\varepsilon = \frac{p-f}{f}$ or ε' that are encountered in the implementation of functions f. See Section 5 for more detailed information.

The complete algorithm `direval` using L'Hôpital's rule for evaluating a function φ on I is given in Algorithm 2.

The procedure `direval` does not always respect Requirement 4: cancellation and decorrelation effects [2] may lead to high overestimates of $\varphi(I)$ by the result of $\texttt{direval}(\varphi, I, t)$. By the mean value theorem we can use a centered form [2] in an interval Taylor evaluation approach: choosing a center $m \in I$, we have

$$\varphi(I) \subseteq \varphi([m,m]) + (I - [m,m]) \cdot \varphi'(I)$$

The procedure $\texttt{eval}(\varphi, I, t, N)$ thus evaluates φ on the thin interval $[m,m]$ using `direval` and recursively calls itself with $\texttt{eval}(\varphi', I, t, N-1)$ until $N = 0$, in which case φ' is evaluated using `direval`. As shown in [2], if the diameter $\text{diam}(I)$ of the interval I is less than 1, the overestimate in the returned J with regard to $\varphi(I)$ decreases exponentially when N increases.

For proof generation, a trace of the computations in `eval` is kept. This trace includes information for the use of interval Taylor and interval L'Hôpital's rule.

Algorithm: `direval`(φ, I, t)

Input: A function φ given as an expression tree, an interval I and a precision t
Result: An interval J such that $\varphi(I) \subseteq J$
begin
  if φ *is a leaf in the expression tree* **then return** `baseeval`(φ, I, t)
  else
    Let $\theta_1, \ldots, \theta_n$ be such that $\varphi = \psi(\theta_1, \ldots, \theta_n)$ and let $J_i =$ `direval`(θ_i, I, t)
    if ψ *is a division and* $0 \in J_2$ **then**
      Compute an approximate zero $z \in \mathcal{F}_t$ of $\theta_2(z)$ using Newton-Raphson iteration
      Let $T_1 =$ `direval`$(\theta_1, [z, z], t)$ and $T_2 =$ `direval`$(\theta_2, [z, z], t)$
      if $T_1 = [0, 0]$ *and* $T_2 = [0, 0]$ **then return** `direval` $(\psi(\theta_1', \theta_2'), I, t)$
      else return $[-\infty, +\infty]$
    end
    else return `baseeval`$(\psi, J_1, \ldots, J_n, t)$
  end
end

Algorithm 2: `direval` - Direct interval evaluation

3.4 Boxing the zeros of a function

In order to box the zeros of a function φ on an interval I we use a bisection algorithm: we first evaluate ϕ on I with `eval` thus getting an interval J; since $\varphi(I) \subseteq J$, if 0 does not belong to J then φ does not have any zero in I.

If, on the contrary, $0 \in J$, this does not necessarily mean that $0 \in \varphi(I)$ but there is a suspicion. That is where we bisect: we cut I into two halves $I_1 \cup I_2$ and call `boxZeros` recursively on I_1 and I_2. We stop this process when the diameter of the input interval is smaller than a parameter Δ.

Algorithm: `boxZeros`

Input: A function φ; an interval I; a parameter Δ
Result: A list $\mathcal{B}$ of intervals boxing the zeros of φ
begin
  $J :=$ `eval`(φ, I, t, N) ;
  if $0 \in J$ **then**
    if $\mathrm{diam}(I) < \Delta$ **then return** $\{I\}$;
    else $I_1 := [\underline{I}, \mathrm{mid}(I)]$; $I_2 := [\mathrm{mid}(I), \overline{I}]$;
    return `boxZeros`$(\varphi, I_1, \Delta) \cup$ `boxZeros`(φ, I_2, Δ) ;
  end
  else return $\{\}$;
end

Algorithm 3: How to box the zeros

In order to generate a proof of the result, the algorithm just retains the decisions made during the algorithm and writes theorems of the form $(\varphi(I) \subseteq J) \wedge (0 \notin J) \Rightarrow 0 \notin \varphi(I)$. The proof of $\varphi(I) \subseteq J$ is given by `eval`.

Note that the bisection algorithm is a bit naïve. A more sophisticated algorithm like an interval version of Newton's iteration process (see [2]) could also be used but we have not implemented it yet.

4 Examples

We present now two examples that show the practical results of our algorithm. We have implemented the algorithm in a software tool[‡]. In all the examples, we use the release 0.0.2.2-alpha of the tool and compare the results with Maple 10 (Build ID 190196). Our experiments have been done on a computer with a 2.5 GHz processor Intel Pentium 4 running GNU/Linux (kernel 2.6.19.2-ws #1 SMP i686).

We used the procedure `infnorm` in the package `numapprox` of Maple. We set `Digits` to 100.

We will use a short notation for the result $[\ell, u]$ of our algorithm: we write the common digits of ℓ and u followed by the range of possible next digits. For instance, if the result of our algorithm is $[0.123456, 0.1234789]$, we will write $0.1234[5-8]$.

Worked example: We consider the example described in Section 1.1. Let f be the function $\exp -1$, $I = [-0.25, 0.25]$ and the polynomial p given above. We obtain for $\|\varepsilon\|_\infty^I$:

Maple	$0.9834913195329190\ldots\mathrm{e}-7$
Our algorithm ($N=0$, $t=165$, $\Delta=2^{-27}$)	$0.98349131972[1-3]\mathrm{e}-7$
Exact value	$0.9834913197221\ldots\mathrm{e}-7$

As can be seen, the result of Maple is underestimated. This estimation does not become better as `Digits` increases as can be verified by increasing `Digits` to greater values than 100. Contrary to what is often believed, the results of Maple seem not to converge towards the exact value when `Digits` goes to infinity.

This underestimation may affect the correctness of the implementation of function f. The result of our algorithm gives approximately the same number of correct digits as Maple, but it bounds rigorously the exact value giving a trustful result. Moreover, increasing t and decreasing Δ, we get tighter bounds of the exact value.

Log for CRLibm : The second example implements the function $f : x \mapsto \log_2(1+x)$ and is used in the library CRLibm [1]. The infinite norm of $\varepsilon = (p-f)/f$ must be computed on $[-1/512, 1/512]$ where

$$\begin{aligned} p(x) \quad = \quad & x \cdot \left(\frac{11704532700986780303630157 4157545}{2^{106}} + x \cdot \left(\frac{-58522663504933901606981166592605}{2^{106}} + \right.\right. \\ & x \cdot \left(\frac{8663094464742397}{2^{54}} + x \cdot \left(\frac{-6497320848515433}{2^{54}} + x \cdot \left(\frac{2598928339549937}{2^{53}} + \right.\right.\right. \\ & \left.\left.\left. x \cdot \left(\frac{-541446114948727}{2^{51}} + x \cdot \frac{3712726891772213}{2^{54}} \right) \right) \right) \right) \right) . \end{aligned}$$

Maple	$0.21506063319877\ldots\mathrm{e}-21$
Our algorithm ($N=2$, $t=165$, $\Delta=2^{-88}$)	$0.215060633232252001406277045[72-80]\mathrm{e}-21$
Exact value	$0.21506063323225200140627704573738 2\ldots\mathrm{e}-21$

Maple returns its result quite instantaneously but underestimates the real value. Our algorithm needs about 320 seconds to produce a safe result.

5 Limitations of the algorithm

Our infinite norm algorithm given in the previous Section 3 suffers from some limitations. These limitations are of different kinds:

[‡]available at `http://lipforge.ens-lyon.fr/projects/arenaireplot/`

- On some instances for $\varepsilon = \frac{p-f}{f}$, the algorithm fails to deliver a finite bounding for the infinite norm $\|\varepsilon\|_\infty$ because of the lack of symbolic simplification. Symbolic derivatives used in the algorithm for interval Taylor evaluation and L'Hôpital's rule may contain subexpressions that cancel out symbolically but are the source of numerical instabilities, such as decorrelations. A typical example is the subexpression $\frac{\sin\sqrt{x}}{\sqrt{x}}$ when evaluated on an interval containing 0. Use of L'Hôpital's rule without symbolical simplification of the subexpression $\sqrt{x}$ does not permit computing finite bounds.

- On the contrary, the use of symbolic differentiation for evaluating the derivative of a function may not be appropriate. The size of the expressions representing the successive derivatives of a function may grow exponentially. In particular fractional terms hinder the use of high order derivatives. Functions $\varepsilon = \frac{p-f}{f}$ contain such fractions. Automatic differentiation [8] is investigated for overcoming this limitation.

- The presented algorithm has several parameters: the precision of the interval arithmetic t, the maximal zero box diameter Δ, and the interval Taylor recursion level N. These parameters all have some influence on the tightness of the result $R = [l, u]$. Although experienced users of the algorithm may find well-behaving values by intuition, their influence may be too unpredictable in general. In particular, one observes abrupt changes of tightness in R with regard to Δ.

- Proof generation has quite a few drawbacks. Besides the fact that it does not yet directly interface with a formal proof checker, the size of the proofs may be too large. Proofs explicitly list all interval evaluations of basic functions, all symbolical derivatives, each simplification step etc. For instance, for the worked example (Section 4), about 45 000 theorems and lemmas are listed. We lack a means of simplifying the proof afterwards.

6 Conclusion

The implementation of functions f requires bounding the approximation error $\varepsilon = \frac{p-f}{f}$ of a polynomial p with respect to f. Previous approaches are unsatisfactory. As a solution, we have given a safe algorithm for bounding the infinite norm $\|\varepsilon\|_\infty$ of a smooth function ε.

In the given framework, functions $\varepsilon = \frac{p-f}{f}$ often have a high condition number and present difficulties at the zeros of f. Our algorithm can overcome both issues in typical cases. It uses a particular multi-precision interval evaluation algorithm. This algorithm combines interval Taylor evaluation with heuristics for the use of L'Hôpital's rule. The heuristics do not endanger safety: the algorithm automatically proves the necessary conditions.

Our algorithm can generate a proof written in English for each instance. This way it becomes self-validating. The proof certifies that no contingent bug has affected correctness. This is a first step: the real goal is to interface directly with formal proof checkers.

Our algorithm has some limitations. For instance, some functions ε require symbolic simplification. We are planning to work on that.

The implementation of our algorithm has successfully been used on real-life problems. All instances, taken out of the CRLibm library [1], could be handled with quite satisfactory ease.

References

[1] CRLibm, a library of correctly rounded elementary functions in double-precision. http://lipforge.ens-lyon.fr/www/crlibm/.

[2] G. Alefeld and D. Claudio. The basic properties of interval arithmetic, its software realizations and some applications. *Computers and Structures*, 67:3–8, 1998.

[3] ANSI/IEEE. Standard 754-1985 for binary floating-point arithmetic, 1985.

[4] R. P. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.

[5] M. Daumas, G. Melquiond, and C. Muñoz. Guaranteed proofs using interval arithmetic. In P. Montuschi and E. Schwarz, editors, *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 188–195, Cape Cod, Massachusetts, USA, 2005.

[6] F. de Dinechin, A. Ershov, and N. Gast. Towards the post-ultimate libm. In *17th IEEE Symposium on Computer Arithmetic*, Cape Cod, Massachussets, June 2005.

[7] F. de Dinechin, Ch. Q. Lauter, and G. Melquiond. Assisted verification of elementary functions using Gappa. In P. Langlois and S. Rump, editors, *Proceedings of the 21st Annual ACM Symposium on Applied Computing - MCMS Track*, volume 2, pages 1318–1322, Dijon, France, April 2006. Association for Computing Machinery, Inc. (ACM).

[8] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2000.

[9] J. Harrison. Floating point verification in HOL light: the exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[10] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[11] W. Hofschuster and W. Krämer. FI_LIB, eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-double-Format. Technical Report 98/7, Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, Universität Karlsruhe, 1998.

[12] W. Krämer. Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen. Technical report, Institut für angewandte Mathematik, Universität Karlsruhe, 1996.

[13] P. Markstein. *IA-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.

[14] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, Boston, 1997.

[15] M. Neher. ACETAF: A software package for computing validated bounds for Taylor coefficients of analytic functions. *ACM Transactions on Mathematical Software*, 2003.

[16] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.