A certified infinite norm for the validation of numerical algorithms

Sylvain Chevillard, Christoph Lauter

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 validation of numerical algorithms

Sylvain Chevillard,
Christoph Lauter

December 2006

Research Report N° RR2006-49

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 validation of numerical algorithms

Sylvain Chevillard, Christoph Lauter

December 2006

Abstract

The development of numerical algorithms requires bounding the image domain of functions, in particular of functions $\varepsilon(x)$ associated to an approximation error. This problem can often be reduced to computing the infinite norm $\|\varepsilon(x)\|_\infty$ of the given function $\varepsilon(x)$. For instance, the development of elementary function operators in hard- and software makes use of such algorithms.

Implementations for computing in practice highly accurate floating-point approximations to infinite norms are known and available. Nevertheless, no highly precise, sufficiently fast and certified or self-validating algorithms are available. Their results could be seen as an element in the correctness proof of safety critical or provenly guaranteed implementations.

We present an algorithm for computing infinite norms in interval arithmetic. The algorithm is optimised for functions representing absolute or relative approximation errors that are ill-conditioned because of high cancellation. It can handle even functions that are numerically unstable on floating-point numbers because they are defined there only by continuous extension.

In addition the given algorithm is capable of generating a correctness proof for an infinite norm instance by retaining its computational tree.

Keywords: infinite norm, optimisation, interval arithmetic, certified algorithm, error analysis, approximation error

Résumé

Le développement d'algorithmes numériques nécessite de borner certaines fonctions, en particulier les fonctions représentant une erreur d'approximation. Ce problème se réduit au calcul de la norme infinie $\|\varepsilon(x)\|_\infty$ de la fonction d'erreur $\varepsilon(x)$. Par exemple, le développement de fonctions élémentaires, tant au niveau logiciel que matériel, utilise ce genre de calcul.

Il existe déjà des implémentations de la norme infinie fournissant une très bonne approximation de la valeur réelle de la norme. Cependant, il n'existe pas d'algorithme capable de fournir un résultat à la fois précis et sûr. On entend par *sûr*, un algorithme qui renvoie une valeur majorant la norme réelle et qui fournit par ailleurs un certificat prouvant la validité de cette majoration.

Nous proposons un algorithme de calcul de la norme infinie utilisant l'arithmétique d'intervalle. Cet algorithme est optimisé pour les fonctions correspondant à une erreur relative ou absolue, c'est-à-dire des fonctions numériquement très mal conditionnées du fait d'importantes cancellations. Notre algorithme peut aussi, dans une certaine mesure, travailler avec des fonctions numériquement instables à proximité de certains points flottants où elles ne sont définies que par continuité.

Enfin, notre algorithme peut retenir l'arbre des calculs qu'il a effectués afin de produire une preuve de correction du résultat de son calcul.

Mots-clés: norme infinie, optimisation, arithmétique d'intervalles, algorithme certifié, analyse d'erreur, erreur d'approximation

1 Introduction

The development of a numerical algorithm, such as scientific code[8, 6], elementary function implementation[1] or control applications, consists generally of three steps. Firstly, the given problem is expressed as a mathematical model. This mathematical model may still make usage of high level concepts or functions that are not directly supported by current combinations of processors, programming languages etc. such as composed, non-elementary functions. Secondly, the mathematical model is simplified to match already more closely the available hardware and software system. In this step, approximations take place. For example, transcendental functions may be approximated by rational functions. Further composed functions may be replaced by a combination of approximations of elementary functions. In a third step, the simplified and approximated rational mathematical model is implemented in a floating-point environment, provided, for example, by the IEEE 754 standard[2]. Floating-point numbers generally are finite dyadic approximations to real numbers in a finite range around zero.

All three steps of modelling a given problem imply errors. The mathematical model does not exactly match reality. Simplified to a rational model, it is subject to approximation errors. Finally, floating-point computations induce round-off error. In order to be certain of the significance of a numerical result, quantities appearing in the given model must be bounded. Computed values must be proven to be contained in the finite range of the floating-point environment. In addition, approximation and round-off errors must be shown to be less than a priori specified bound.

Round-off errors induced by a floating-point arithmetic are discrete, non-analytical, discontinuous functions of the inputs of the different basic arithmetical operators. Their bounding has been studied for example in [6]. Automatic tight boundings can be computed and proven using for example the Gappa* tool[4]. In this article we will not further consider them.

Values and approximation errors in a given model can be considered as almost everywhere continuous functions $\varepsilon : \mathbb{R}^n \to \mathbb{R}$ of the inputs. Bounding them means computing their extrema on a given domain $I \subseteq \mathbb{R}^n$. If quantities, especially errors, are mostly symmetric or strictly positive or negative, sufficient bounding may be achieved by computing the infinite (or infinity) norm (infnorm) of the function ε defined as

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

Computing the infinite norm of a function ε, given as a expression or a numerical operator, is for itself a numerical problem. High-quality, approximate, floating-point solutions to the infinite norm computational problem exist. General techniques and considerations are described in [10]. In the case where ε is a multivariate function, computing an infinite norm is a particular case of global optimisation. In this article we will consider only univariate functions $\varepsilon : \mathbb{R} \to \mathbb{R}$. We attract the reader's attention to [5] concerning the multivariate case.

Tools like Maple† or Matlab‡ implement general purpose numerical approximation algorithms for computing an infinite norm of a univariate function. Both algorithms are not clearly specified in terms of the quality of the returned approximation. Matlab uses hardware, i.e.

*available at http://lipforge.ens-lyon.fr/www/gappa/

†cf. www.maplesoft.com

‡cf. www.mathworks.com

IEEE 754 double precision, and is hence limited to well-conditioned infnorm problems. The infnorm algorithm in Maple's `numapprox` package tends to provide overestimations of the infinite norm's true value but can be shown also to return underestimations on some particular functions.

Such approximate solutions may be sufficient in the development phase of a numerical implementation. But whenever it comes to prove its correctness, in particular if the implementation is safety critical, numerical approximations do no longer suffice. An algorithm that provides a certified or self-validating result such as an interval with guaranteed lower and upper bounds of the computed approximation of the infinite norm is needed here.

The authors' work has been motivated by infinite norm problems in the development of correct rounding transcendental elementary functions such as e^x, $\log x$, $\sin x$. The correct rounding, i.e. bit-exact result, correctness proof mainly relies on showing a maximal approximation and round-off error bound[4]. Similar problems in the context of safety critical implementations of combined transcendental functions have been considered for example in[3]. The computation of well-specified approximations to infinite norms are also at the base of works like[11].

In this article we present an algorithm for computing a upper and a lower bound for infinite norms on univariate functions $\varepsilon(x) \in \mathcal{C}^2$ in self-validating and hence certified way. The algorithm is especially optimised for functions $\varepsilon(x)$ that are ill-conditioned because of cancellation and numerically unstable at some floating-point numbers because they are defined only by continuous extension at these points. The implementation of the algorithm is still under development and is integrated into a software tool[§].

This article is organised as follows: in the next section 2, we give the specifications of our algorithm and explain these design choices. In section 3.1 we give the algorithm as well as a correctness proof sketch. This general algorithm makes use of some particular interval arithmetic evaluation techniques that we present in section 3.2. These techniques are used in particular for bracketing the zeros of a function. Section 3.3 clarifies this point. Our algorithm is capable of retaining its computational tree for generating a proof of the generated result. The main considerations on this point are given in section 4. Some examples in section 5 lead the reader to our conclusions in section 6.

2 Specifications of our infnorm algorithm

Let $f : \mathbb{R} \mapsto \mathbb{R}$ be a function to be shown to be correctly implemented, i.e. approximated within a specified error bound. Let $p : \mathbb{R} \mapsto \mathbb{R}$ be the approximation to f used in the implementation. So the absolute (respectively relative) approximation error of p with regard to f is a function $\varepsilon : \mathbb{R} \mapsto \mathbb{R}$ defined as $\varepsilon(x) = p(x) - f(x)$ (respectively $\varepsilon(x) = \frac{p(x)-f(x)}{f(x)}$). In the framework of elementary function development, f is a transcendental function and p a polynomial with floating-point coefficients[1, 3]. If p and f are continuous and continuously differentiable functions that are not identically zero on no sub-interval and if ε is finite everywhere (which is the case in practical implementations), ε is almost everywhere continuous and continuously differentiable.

The first requirement our algorithm shall fulfil is implied by the fact that we want a certified result:

[§] available under the GPL at `http://lipforge.ens-lyon.fr/projects/arenaireplot`

Requirement 1. *The algorithm implementing the infinite norm of a function must always give an* **upper***-bound of the real value of the infnorm of the function.*

It would be possible to always answer $+\infty$ but this would be perfectly useless in practice. In order to estimate the order of magnitude of the error made by a certified infinite norm algorithm, a lower bound for the actual value is needed. This can be obtained with this second requirement:

Requirement 2. *The algorithm shall give a* **lower***-bound of the real value of the infnorm of the function. Thus if the orders of magnitude of the upper and the lower bounds are the same, it can be concluded that the result of the algorithm is accurate enough for the problem in consideration.*

Since p approximates f, the order of magnitude of $\varepsilon(x)$ is much lower than the order of magnitude of p or f. In other words, the functions $\varepsilon(x) = p(x) - f(x)$ (respectively $\varepsilon(x) = \frac{p(x)-f(x)}{f(x)}$) are ill-conditioned due to the cancellation in the subtraction $p(x) - f(x)$. This observation leads to a third requirement to our algorithm:

Requirement 3. *The algorithm shall take in input functions defined by an explicit expression tree. Ill-conditioned functions defined in this way shall be overcome by the usage of high intermediate precision and re-correlation[5, 3] techniques for interval arithmetic.*

Let us still make one observation on the functions $\varepsilon(x) = \frac{p(x)-f(x)}{f(x)}$ we are especially interested in. Suppose that in the given domain $I = [a; b]$ the infinite norm $\|\varepsilon(x)\|_\infty^I$ is to be computed on, $f(x)$ has a zero z, i.e. $f(z) = 0$. If $p(z) = 0$ at the same point z and $\lim_{x\to z} p'(x) - f'(x) = c_1$ and $\lim_{x\to z} f'(x) = c_2$ exist, $\varepsilon(z) = c = \frac{c_1}{c_2} \in \mathbb{R}$ is nevertheless well-defined at z by continuous extension. In consequence $\|\varepsilon(x)\|_\infty^I \neq \infty$ even if the pole at z of $\varepsilon(z)$ and pure interval arithmetic might suggest the opposite.

We formulate thus the following additional requirement:

Requirement 4. *If the expression tree for $\varepsilon(x)$ has some pole on a floating-point number in the given input domain that may be extended by continuity, the algorithm for computing the infinite norm of ε on I shall return an upper bound different from $+\infty$.*

In order to ensure that the algorithm respects its specifications, it should be carefully proven. However, the implementation could contain bugs. Moreover, some users of the algorithm want to provide proofs for the results of intermediate computations in the development of an algorithm they had been using an infinite norm algorithm for. This yields to a last requirement:

Requirement 5. *The algorithm shall give, in addition to the result, a formal proof which can be checked externally and which ensures that the interval result is really bounding the value of the infinite norm $\|\varepsilon(x)\|_\infty$.*

Naturally, we want our algorithm to have the best possible performance using the least memory possible.

3 The algorithm

We are going to present now our infinite norm algorithm. Let us remember that the algorithm requires the function $\varepsilon(x)$, given as an expression tree, to be at least $\mathcal{C}^2$ and formally differentiable.

In the following, if ε is a function and I an interval, $\varepsilon[I]$ will denote the set

$$\varepsilon[I] = \{y \in \mathbb{R},\, \exists x \in I,\, y = \varepsilon(x)\} \quad .$$

It is well-known that this set is an interval when ε is continuous.

3.1 General scheme of the algorithm

The algorithm for computing the infinite norm basically decomposes into six steps which will are given in the following and detailed below:

`infnorm`: input a function $\varepsilon \in \mathcal{C}^2(I)$ and a compact interval $I = [a, b]$:

1. formally differentiate the function ε;
2. search a list of intervals $I_1, \cdots, I_p$ such that every zero of ε' lies in at least one of the I_k. Note that some I_k may not contain any zero of ε' and reciprocally that two zeros may belong to the same I_k;
3. add $I_0 = [a, a]$ and $I_{p+1} = [b, b]$ to the list;
4. compute $J_0, \cdots, J_{p+1}$ such that for each k, $\varepsilon[I_k] \subseteq J_k$.
5. compute the inner- and outer- enclosure of $g[I]$ from the intervals $J_0, \cdots, J_{p+1}$, i.e. compute intervals IE and OE such that

   $$\forall y \in IE,\, \exists x \in I,\, y = \varepsilon(x)$$

   and such that

   $$\forall k,\, \forall y \in J_k,\, y \in OE$$

   The computation of this enclosures will be explained below.
6. Return

   $$[\max\{|IE_\ell|, |IE_r|\}, \max\{|OE_\ell|, |OE_r|\}]$$

   as an interval containing $\|\varepsilon\|_\infty^I$

Since the interval $I = [a, b]$ is compact and ε is continuous, we know that ε reaches its minimum at a point x_m and its maximum at a point x_M. Since ε is differentiable, x_m is either a bound of the domain (a or b) or $\varepsilon'(x_m) = 0$. The same holds for x_M. It follows that both x_m and x_M are in some interval of the list $I_0, \cdots, I_{p+1}$.

Lemma 3.1. *$\varepsilon(x_m)$ does not belong to the interior of IE. The same thing holds for x_M.*

Proof. Suppose that $\varepsilon(x_m)$ belongs to the interior of IE. Thus, there exists some $z \in IE$ such that $z < \varepsilon(x_m)$. By applying the definition of IE, we would have some x such that $\varepsilon(x) = z$ yielding contradiction since $\varepsilon(x_m)$ is minimal and $\varepsilon(x) = z < \varepsilon(x_m)$. The proof is the same for x_M. □

Lemma 3.2. *$\varepsilon(x_m) \in OE$. The same holds for x_M.*

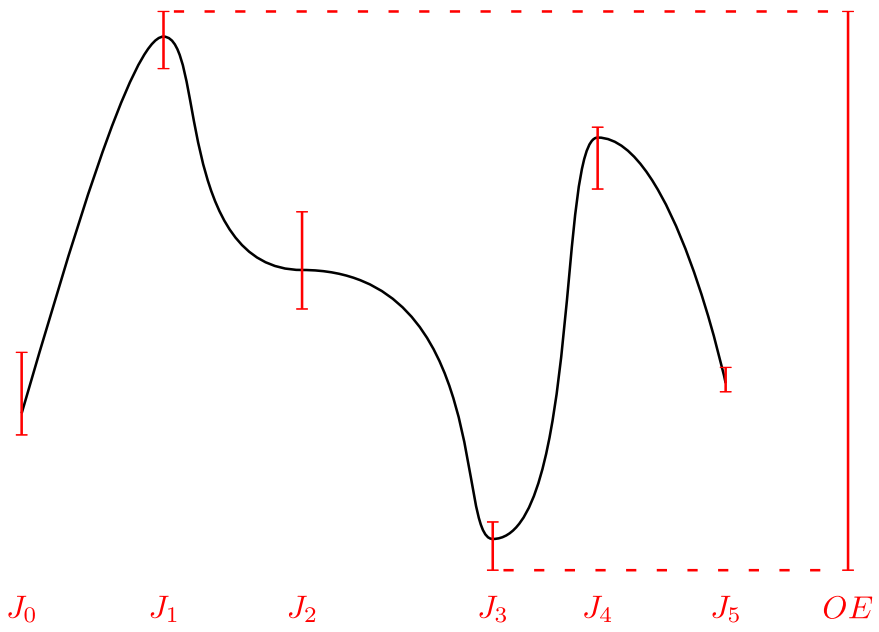
Proof. Since x_m belongs to one of the I_k, $\varepsilon(x_m)$ belongs to $\varepsilon[I_k]$ and then $\varepsilon(x_m) \in J_k$. Applying the definition of OE, $\varepsilon(x_m) \in OE$. The proof is the same for x_M. □

Let us see now how to compute IE and OE. For OE we take the convex enclosure of the union of the J_k which is defined by

$$OE_\ell = \min\{J_{0\ell}, \cdots, J_{(p+1)\ell}\} \quad \text{and} \quad OE_r = \max\{J_{0r}, \cdots, J_{(p+1)r}\}$$

where $OE = [OE_\ell, OE_r]$. It trivially satisfies the required property for OE.

Figure 1: The outer enclosure of the J_k.



For IE we take $]IE_\ell, IE_r[$ where

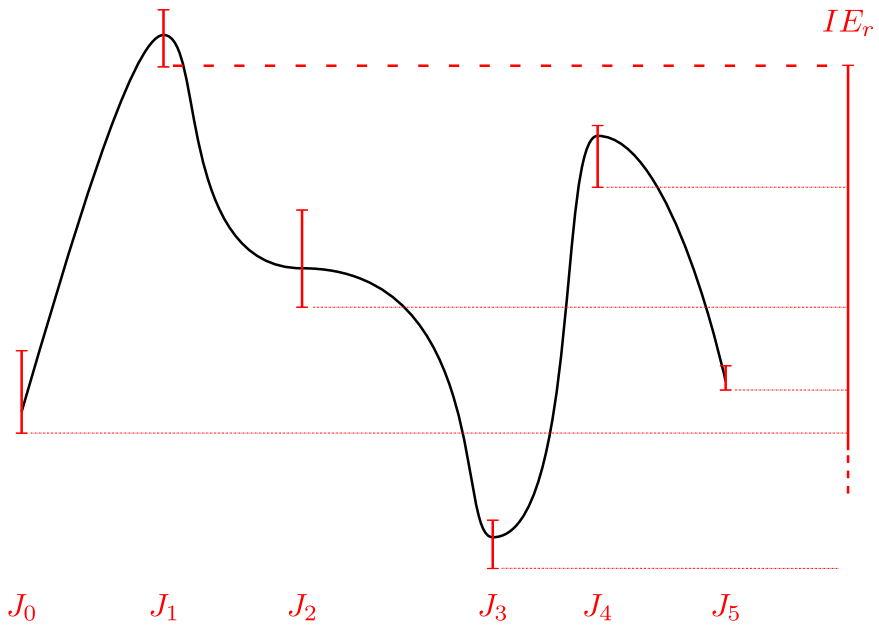
$$IE_\ell = \min\{J_{0r}, \cdots, J_{(p+1)r}\} \quad \text{and} \quad IE_r = \max\{J_{0\ell}, \cdots, J_{(p+1)\ell}\}$$

where the subscript ℓ denotes the lower bound of an interval and the subscript r denotes the upper bound.

Lemma 3.3. *The previous way of computing IE actually provides an inner enclosure as defined above.*

Proof. Let $IE_\ell \leq y \leq IE_r$. Thus $y \geq \min\{J_{0r}, \cdots, J_{(p+1)r}\}$ holds. Let k be the index for which the minimum is reached: $y \geq J_{kr}$. Since $\varepsilon[I_k] \subseteq J_k$ there exists u such that $\varepsilon(u) \leq J_{kr} \leq y$. With the same argument, there exists v such that $y \leq \varepsilon(v)$. Since $\varepsilon[I]$ is a interval, $y \in \varepsilon[I]$ and then $\exists x \in I$, $y = \varepsilon(x)$ which is the required property. □

Figure 2: The (right bound of the) inner enclosure of the J_k.



The computation of IE and OE from the J_k can be performed incrementally as the J_k are calculated.

It is clear that $IE \subseteq OE$ and, hence, $OE \backslash IE$ consists in the union of two intervals: $[OE_\ell, IE_\ell]$ and $[IE_r, OE_r]$. By lemmata 3.1 and 3.2 the minimum and maximum of ε lie in these intervals. It follows that

$$\|\varepsilon\|_\infty^I \in [\max\{|IE_\ell|, |IE_r|\}, \max\{|OE_\ell|, |OE_r|\}] \quad .$$

This is the value returned by the algorithm given above.

Let us now show how the I_k are found and in which way the J_k are computed out of them. Clearly if every J_k were equal to $\varepsilon[I_k]$, OE would precisely be equal to $[\varepsilon(x_m), \varepsilon(x_M)]$. Let I_k denote the interval containing x_M. If J_k is affected by arithmetical errors, we have $\varepsilon[I_k] \subset J_k$ and OE_r is hence greater or equal to J_{kr}. It follows that the greater the overestimate of $\varepsilon[I_k]$ by the J_k, the greater the overestimate of the function's image domain by OE will be.

Let I_k be an interval such that $x_M \in I_k$. If I_k is exactly $[x_M, x_M]$), we have $\varepsilon[I_k] = [\varepsilon(x_M), \varepsilon(x_M)]$. But if I_k is wider, $\varepsilon[I_k]$ will be of the form $[u, \varepsilon(x_M)]$ with u the smaller as I_k becomes the wider. Thus, the contribution of $J_{k\ell}$ to IE_r will be less or equal to u. In consequence, the greater the overestimate of I_k, the greater the underestimate of the function's image domain by IE will be.

This shows that it is important to take care of the way the I_k and the J_k are computed. We will focus on this point in the following two paragraphs.

3.2 Interval evaluation of functions - computation of J_k

Our goal is to compute out of I_k an interval J_k as precisely as possible with regard to $\varepsilon[I_k]$. We can suppose that I_k is a small interval, i.e. in practice its diameter is a lot smaller than 1. We use the library `MPFI`[¶] which implements the interval arithmetic with arbitrary precision. The precision used in the computations is a parameter of our algorithm. For each function f known by `MPFI`, and for each interval I, the evaluation of f on I by `MPFI` produces an interval J such that $f[I] \subseteq J$. We will denote by $f(I)$ the interval computed by `MPFI`.

`MPFI` implements the standard functions $+$, $-$, $/$, $\times$, $\sqrt{\cdot}$, exp, sin, etc. For more complex functions such as $h(x) = \exp(\sin(x+\ln(x)))$ we have to decompose the expression and evaluate each subterm separately. For example, to compute h on I, we will first compute $J_1 = \ln(I)$, then $J_2 = I + J_1$, then $J_3 = \sin(J_2)$ and finally $J_4 = \exp(J_3)$. For each operation, `MPFI` is very precise. In contrast, the interval evaluation of composed functions is subject to error accumulation and cancellation effects caused by decorrelation. The final result of such an evaluation may thus be inaccurate.

We can use the mean value theorem for obtaining an evaluation satisfying $\mathrm{diam}(J) = \mathcal{O}(\mathrm{diam}(I)^2)$ (instead of $\mathrm{diam}(J) = \mathcal{O}(\mathrm{diam}(I))$ in the previous case), which performs well for the used small I_k. Let z be a point in I_k (we choose the middle of I_k), for each $x \in I_k$, $\exists c \in I_k$, $\varepsilon(x) = \varepsilon(z) + (x-z)\varepsilon'(c)$. It follows that $\varepsilon[I_k] \subseteq \varepsilon(z) + (I_k - z)\varepsilon'[I_k]$. So we can compute $\varepsilon'(I_k)$ using `MPFI` and then take for J_k the interval $\varepsilon(z) + (I_k - z)\varepsilon'(I_k)$. The interest of this method comes from the fact that the errors in the evaluation of $\varepsilon'(I_k)$ are multiplied by $(I_k - z)$ which is a very small interval centred in 0.

Obviously, we can use this technique recursively and compute $\varepsilon'(I_k)$ using $\varepsilon''(I_k)$ and so on. This allows theoretically to obtain $\mathrm{diam}(J) = \mathcal{O}(\mathrm{diam}(I)^n)$. A possible problem is that the successive derivatives of ε are more and more complex expressions and their evaluation may lead to so imprecise results whilst using great amounts of memory; thus, the technique may become useless. At the moment, the number of step of recursion is just a parameter of the algorithm that the user can fix following its intuition about the complexity of the successive derivatives.

In order to limit the explosion of the expression of the successive derivatives of a function, we implement an additional special optimisation for fractions $\varepsilon(x) = \frac{f(x)}{g(x)}$. As long as $g(x)$ has no zero in the given interval, instead of evaluating $\frac{f(z)}{g(z)} + (x-z) \cdot \frac{g(x)\cdot f'(x) - f(x)\cdot g'(x)}{g^2(x)}$, we evaluate $\frac{f(z)+(x-z)\cdot f'(x)}{g(z)+(x-z)\cdot g'(x)}$. This is more efficient since the induced expression trees are smaller than the tree for $\frac{g(x)\cdot f'(x) - f(x)\cdot g'(x)}{g^2(x)}$.

Another problem can arise: some functions have a so-called removable singularity: at some point z, the function $\varepsilon(x)$ is of the form $\frac{f(z)}{g(z)}$ with $f(z) = g(z) = 0$. However, the function may be prolongated by continuity. It is the case, for instance, for the function $\frac{\sin(x)}{x}$ at 0. Mathematically, the function remains well defined, but numerically, will perform very badly. The round-off errors become very big; if using interval arithmetic, a division by an interval containing 0 occurs and produces a NaN or an infinity. In order to solve the problem, we have to detect this case and find a solution. If we can detect it (that is if we find a point z and we can prove that $f(z) = g(z) = 0$), we can use a variant of the so-called *L'Hôpital's rule*:

$$\forall x \in I,\ \exists (c,\ d) \in I^2,$$

[¶] distributed under the LGPL at http://gforge.inria.fr/projects/mpfi/

$$\frac{f(x)}{g(x)} = \frac{f(z) + (x-z)f'(c)}{g(z) + (x-z)g'(d)} = \frac{f'(c)}{g'(d)} \quad .$$

Thus $(f/g)[I] \subseteq (f'/g')[I]$. Once again, we can use the rule recursively if f'/g' has a removable singularity in the interval.

For detecting a removable singularity, we firstly test if the function to evaluate is a quotient. If so, we evaluate an interval J containing the denominator $g[I]$ (using Taylor and `MPFI`). If J does not contain 0, we are sure that there is no singularity. If it contains 0, there is a doubt: we search a floating-point (not interval) zero using the Newton-Raphson method. If we do not find any, we cannot do anything. But if we find one z, it is a potential removable singularity. We firstly evaluate g on the interval $[z, z]$ using the described interval evaluator. If the result is [0, 0] we **know** that z is a zero of g (if it is not, we cannot conclude). Since we are now sure that z is a zero of g, we evaluate f on $[z, z]$ the same way and if the result is [0, 0], we know that L'Hôpital's rule can be applied. In every other case, we just let the normal interval evaluation continue, leading to a final result that is NaN or infinity.

It can be argued that this technique is useless because its detection is subject to too much floating-point noise. If it works, it would just be luck. We must have simultaneously discovered by a floating-point Newton-Raphson the precise real point z, obtain $f([z, z]) = [0, 0]$ and $g([z, z]) = [0, 0]$. This is right but it is the only way to be sure that we actually have a removable singularity. Besides, let us recall that we need to be sure in order to prove the correctness of our final result. Moreover, it works more often than it seems. Actually, if the singularity is not at a floating-point number, the only way to show the infinite norm is finite is to show formally that $g(x) = 0$. Further, the function would be so ill-conditioned near the singularity, that it cannot be evaluated in practice. So, Newton-Raphson will almost surely detect the point for functions that are practically evaluated e.g. in elementary function libraries. In addition, z will probably be some simple point such as 0 or an integer and if the functions are not to complicated, it is probable that for this special point, no round-off errors at all occurs during all the computation. For example,

$$\frac{\exp(\arcsin(x)) - 1}{\arcsin(x)}$$

works well because 0 is detected as a potential singularity, and `MPFI` knows that $\arcsin([0, 0]) = [0, 0]$, $\exp([0, 0]) = [1, 1]$, etc.

Evaluation algorithm:
After this introductory discussion let us give our algorithm for evaluating a composed function ε on a domain I for a result J satisfying $\mathrm{diam}(J) = \mathcal{O}(\mathrm{diam}(I)^{r+1}$.

`evaluate`: input an expression representing a function ε, an interval I, and a parameter `rec_level` :

1. if `rec_level` > 0: differentiate ε ; compute the mid-point z of the interval I. Return `evaluate`$(\varepsilon, [z, z], 0)$ $+(I - z)\cdot$ `evaluate`$(\varepsilon', I,$`rec_level`$-1)$ using `MPFI` for the addition and the multiplication.

2. else:

   (a) if ε is not a quotient: ε is of the form `op`(h) (or h_1 `op` h_2). Return `op`(`evaluate`$(h, I, 0)$) performing `op` with `MPFI` (idem if there are two operands).

(b) else $\varepsilon = h_1/h_2$. Let $J_2 =$ `evaluate`$(h_2, I, 0)$.

i. If J_2 does not contain 0, let $J_1 =$ `evaluate`$(h_1, I, 0)$ and return J_1/J_2 performed by `MPFI`.

ii. else test if L'Hôpital's rule can be applied as explained above. If the test succeeds, formally differentiate h_1 and h_2 and return `evaluate`$(h_1'/h_2', I, 0)$.

The actual implementation of this algorithm integrates some additional improvements with regard to the performance and the accuracy of the produced results:

- Formally differentiated expressions are simplified exactly as well as possible. The simplification comprises the evaluation of constant sub-expressions as long as no rounding occurs, conversion of polynomial sub-expressions into Horner's scheme, elimination of additions and subtractions with 0, multiplications and divisions with 0 and 1 and some other simple formal simplifications.

- If their necessity is known in advance, functions are differentiated only once for several evaluations.

- Cancellation in additions and subtractions during interval evaluation of sub-expressions are detected if possible by simple tests. Intermediate Taylor evaluations allow here for improving the accuracy of the result.

3.3 Intervals bounding the zeros of a function - determination of I_k

We have seen that the intervals I_k should be small in order to contribute efficiently to the inner enclosure. For the economy of useless computations, we should try to select only those intervals which actually contain some zeros of the derivative. In contrast, for ensuring the correctness of the algorithm, we have to be sure that every zero lies in a I_k.

In our approach we therefore fix an appropriate diameter δ as a parameter of the algorithm. We use a bisection algorithm. At first, we evaluate ε' on the whole interval I (using all the optimisations of the `evaluate` function). If the returned interval J contains 0, we cut I in two parts I_1 and I_2 and we recurse on each sub-interval.

If we find an interval I' such that J' does not contain 0 at one moment of this procedure, then we are sure that ε' has no root in I' and we can just eliminate the interval I'. We stop branching when we have an interval I' which diameter is less than δ.

At the end of the procedure, we get a list $\mathcal{I}_1, \cdots, \mathcal{I}_t$ such that every zero of ε' lies in one $\mathcal{I}_k$ and which diameters are all less than δ.

Let z be a zero of ε' and $\mathcal{I}_k = [a, b]$ the selected interval in which it lies. In practice, the algorithm computes very often additional intervals $\mathcal{I}_{k-1}$ and $\mathcal{I}_{k+1}$ that actually do not contain any zero of ε' and are of the form $[a', a]$ and $[b, b']$ because $\varepsilon[\mathcal{I}_{k-1}]$ and $\varepsilon[\mathcal{I}_{k+1}]$ are too close to zero to be discarded by interval evaluation. In contrast, if ε has a removable singularity in z, the evaluation of $\varepsilon(\mathcal{I}_{k-1})$ and $\varepsilon(\mathcal{I}_{k+1})$ will be very unstable since ε is ill-conditioned near z and may yield to imprecise results. However, if we join the intervals, obtaining one interval $I' = \mathcal{I}_{k-1} \cup \mathcal{I}_k \cup \mathcal{I}_{k+1}$, we can apply L'Hôpital's rule on the whole interval I' which yields to better results, even if the interval is three times wider than the previous one.

Thus, we replace every series of consecutive intervals in the list $\mathcal{I}_1, \cdots, \mathcal{I}_t$ by their union unless the union becomes more than 4 times greater in diameter than the parameter δ fixed previously. This yields to the final list $I_1, \cdots, I_p$ used in the following of the algorithm.

Remark that instead of using a bisection to bracket the zeros of the derivative $\varepsilon'(x)$ of the given function, the interval Newton method as described in [5] could also be used. We would have to require the input functions ε to be at least C^3 in this case.

4 Generating a proof for infinite norm results

Interval arithmetic, satisfying the so-called inclusion property, has strong links to numerical proving of mathematical properties. As shown in [3], libraries for certifying interval computations in proof checkers, such as for example the Prototype Verification System (PVS) [9] exist. The general idea consists in retaining the complete computational and decisional tree of an instance of an interval algorithm, for instance, our infinite norm algorithm, and to generate a lemma for each invocation of an interval function or logical element such as bisection, Taylor series expansion, L'Hôpital's rule etc.

Our algorithm is currently not yet capable of producing PVS or COQ [7] readable proofs. Nevertheless, it is already possible to store the complete computational tree and to generate an English written proof for each instance of the infinite norm algorithm. The last element that is still lacking to us to provide this additional safety to the correctness of the results computed and inherently proven by the interval algorithm is the difficulty to handle transcendental functions in formal proof checkers. Such a library is partially available for PVS but as shown in [3], computation times for proof checking are still very high.

Most current proofs are still intractable in PVS because of the complexity of the implied numbers. We are working to provide a means of simplifying a proof in terms of the bit-length of used numbers.

In some examples, the proof generated by our algorithm could already be checked by a tool like Gappa. This is due to the fact that the derivatives of some transcendental functions such as $\log(x)$ are rational.

5 Examples

Let us give some examples of the behaviour of our algorithm. The examples are mainly taken out of problems in the development of the `crlibm` library for correctly rounding elementary functions[1].

1. The first example is a toy problem: let be $\varepsilon(x) = \frac{\log(1+x)}{x}$. The function $\varepsilon(x)$ is defined in 0 only by continuous extension. Our algorithm answers for $\|\varepsilon(x)\|_\infty^{[-2^{-6};2^{-6}]}$ the following

$$\|\varepsilon(x)\|_\infty^{[-2^{-6};2^{-6}]} \in [541109425 \cdot 2^{-29}; 270554713 \cdot 2^{-28}]$$

   This is equivalent to an accuracy of 27 correct bits; the computing precision has been 30 bits. Computation is instantaneous on current desktop machines.

2. The second example is the computation of a bound for the approximation error of the

polynomial $p(x) = x - \frac{1}{2} \cdot x^2 + a_3 \cdot x^3 - a_4 \cdot x^4 + a_5 \cdot x^5 - a_6 \cdot x^6 + a_7 \cdot x^7$ where

$$\begin{aligned} a_3 &= 6004799503160663 \cdot 2^{-54} \\ a_4 &= 9007199254173073 \cdot 2^{-55} \\ a_5 &= 3602879701310655 \cdot 2^{-54} \\ a_6 &= 6004904200786859 \cdot 2^{-55} \\ a_7 &= 40211673751819 \cdot 2^{-48} \end{aligned}$$

with respect to the function $f(x) = \log(1+x)$ in the domain $I = [-129 \cdot 2^{-15}; 129 \cdot 2^{-15}]$ (i.e. $\varepsilon(x) = p(x) - f(x)$). Our algorithm returns lower and upper bounds which are close enough that they can both be considered as a result accurate up to 29 bits. The computing precision has been 100 bits and rec_level has been set to 2. Computational time is around 15 seconds on current desktop machines.

3. For the third example, let $f(x) = \log(1+x)$, $p(x) = x - a_2 \cdot x^2 + a_3 \cdot x^3 - a_4 \cdot x^4 + a_5 \cdot x^5 - a_6 \cdot x^6 + a_7 \cdot x^7$ where

$$\begin{aligned} a_2 &= 9223372036854776725 \cdot 2^{-64} \\ a_3 &= 6148914691236520117 \cdot 2^{-64} \\ a_4 &= 18446744071800930591 \cdot 2^{-66} \\ a_5 &= 7378697627908458209 \cdot 2^{-65} \\ a_6 &= 3074519401226530361 \cdot 2^{-64} \\ a_7 &= 5270640148006219133 \cdot 2^{-65} \end{aligned}$$

and $\varepsilon(x) = p(x) - f(x)$. Our algorithm (with rec_level set to 2) returns as an upper bound for $\|\varepsilon(x)\|_\infty^{[-129\cdot 2^{-15};129\cdot 2^{-15}]}$ a value which is an approximation up to 94 bits according to the returned lower bound.

6 Conclusion

We have given an algorithm for computing a self-validating interval result for the infinite norm $\|\varepsilon(x)\|_\infty^I$ of an univariate function $\varepsilon(x) \in C^2$ on some domain I. The algorithm can handle ill-conditioned functions by overcoming ill-conditioning and resulting high decorrelation by the usage of high intermediate (multi-) precision and (recursive) interval Taylor evaluation. Functions with discontinuities at floating-point values that can be extended by continuity can be handled by the use of L'Hôpital's rule, too. The algorithm proves in this case automatically that the L'Hôpital's rule can be applied. Such functions are common as approximation errors in the development of numerical algorithms, in particular, elementary functions.

The performance of the implementation of our algorithm is sufficient for common problems on current machines. Examples taken out of the development of `crlibm`[1], an implementation of correct rounding elementary functions in double precision, can all be handled in some minutes of computation.

The algorithm can retain its computational and decision tree for the generation of an English written proof of an instance. Such a proof may be used in the certification process of a numerical algorithm analysed with our infinite norm. Currently, generation of a PVS or COQ readable proof is impossible because of the difficulty to handle transcendental functions.

We know some examples of functions, as for instance $\frac{\log\sqrt{x}}{\sqrt{x}}$ whose derivatives, as they are provided by our automatic differentiation algorithm, are numerical unstable and cannot be handled by our algorithm. Nevertheless, these derivatives could be formally simplified and brought to an evaluable form. In future our work, we will try to integrate a little more formal simplification into the presented algorithm.

We are currently lacking knowledge of other algorithms and approaches with similar specifications to compare our algorithm and implementation with. Naturally, this might also be due to the fact that our infinite norm problems arise only in particular situations and that no other approaches exist.

References

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

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

[3] Marc Daumas, Guillaume Melquiond, and César Muñoz. Guaranteed proofs using interval arithmetic. In Paolo Montuschi and Eric Schwarz, editors, *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 188–195, Cape Cod, Massachusetts, USA, 2005.

[4] 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*, April 2006.

[5] Eldon Hansen and G. William Walster. *Global Optimization Using Interval Analysis, Second Edition, Revised and Expanded*. Marcel Dekker, Inc., New York, Basel, 2004.

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

[7] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. *The Coq proof assistant: a tutorial: version 8.0*, 2004.

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

[9] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752, Saratoga, NY, June 1992.

[10] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C, The Art of Scientific Computing, 2nd edition*. Cambridge University Press, 1992.

[11] Arnaud Tisserand. Hardware operator for simultaneous sine and cosine evaluation. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 992–995, Toulouse, France, May 2006.