Certified and fast computation of supremum norms of approximation errors

Sylvain Chevillard, Mioara Maria Joldes, Christoph Lauter

Certified and fast computation of supremum norms of approximation errors

Sylvain Chevillard ,
Mioara Joldes ,
Christoph Lauter

October 2008

Research Report N^o RR2008-37

Certified and fast computation of supremum norms of approximation errors

Sylvain Chevillard , Mioara Joldes , Christoph Lauter

October 2008

Abstract

In many numerical programs there is a need for a high-quality floating-point approximation of useful functions f, such as $\exp$, $\sin$, erf. In the actual implementation, the function is replaced by a polynomial p, leading to an approximation error (absolute or relative) $\varepsilon = p - f$ or $\varepsilon = p/f - 1$. The tight yet certain bounding of this error is an important step towards safe implementations.
The main difficulty of this problem is due to the fact that this approximation error is very small and the difference $p - f$ is highly cancellating. In consequence, previous approaches for computing the supremum norm in this degenerate case, have proven to be either unsafe, not sufficiently tight or too tedious in manual work.
We present a safe and fast algorithm that computes a tight lower and upper bound for the supremum norms of approximation errors. The algorithm is based on a combination of several techniques, including enhanced interval arithmetic, automatic differentiation and isolation of the roots of a polynomial. We have implemented our algorithm and timings on several examples are given.

Keywords: supremum norm, approximation error, certified computation, elementary function, interval arithmetic, automatic differentiation, roots isolation technique.

Résumé

Beaucoup d'applications numériques nécessitent le calcul précis d'approximations en virgule flottante de fonctions f telles que $\exp$, $\sin$, erf. En pratique, dans l'implantation, la fonction est remplacée par un polynôme p, ce qui conduit à une erreur d'approximation (absolue ou relative) $\varepsilon = p - f$ ou $\varepsilon = p/f - 1$. La majoration fine et néanmoins sûre de cette erreur est une étape importante lorsqu'on vise une implantation certifiée.
La principale difficulté de ce problème vient du fait que l'erreur d'approximation est très petite et est calculée par une différence $p - f$ qui cancelle énormément. Les précédentes approches pour calculer des normes sup sont donc insuffisantes dans ce cas bien précis : ou bien elles n'offrent pas la garantie de sécurité du résultat, ou bien elles donnent des majorations trop grossières, ou enfin elles nécessitent trop de travail manuel au cas par cas.
Nous présentons un algorithme sûr et rapide pour calculer une minoration et une majoration fine de la norme sup d'erreurs d'approximation. L'algorithme s'appuie sur la combinaison de différentes techniques existantes, en particulier une arithmétique d'intervalle perfectionnée, de la différentiation automatique et l'isolation fine des racines d'un polynôme. Nous avons implémenté notre algorithme et nous donnons les temps d'exécution sur plusieurs exemples.

Mots-clés: norme sup, erreur d'approximation, calcul certifié, fonctions élémentaires, arithmétique d'intervalle, différentiation automatique, technique d'isolation de zéros.

1 Introduction

Many numerical programs require computing approximated values of mathematical functions such as $\exp$, $\sin$, $\arccos$, erf, etc. These functions are usually implemented in libraries called `libm`. Such libraries are available on most systems and many numerical programs depend on them. Examples of such `libms` include `CRlibm`, the `glibc`, `libmcr` offered by SUN, and Intel's `MKL`.

In general, these `libms` offer no guarantee on the quality of the computed results. In order to allow for a better portability and a better numerical quality of programs, the revised IEEE 754-2008 standard [15] recommends that the functions provided according to the standard be correctly rounded. Assuming round-to-nearest mode, this means that, if a function f is provided, the function code must always return the floating-point number that is the closest to the exact value $f(x)$.

In order to guarantee such a property, one has to be particularly careful when developing the library. It can be shown [19] that for a large class of functions it is sufficient to compute an approximation y of $f(x)$ accurate enough and then to round y to the target format. Moreover, using properties of the function (such as $\exp(2x) = \exp(x)^2$ for instance) it is possible to perform a so-called *range reduction* so that we need to approximate f on a small closed interval $[a, b]$ only.

For computing the approximation y, a polynomial p may be used. One must then prove that for each point $x \in [a, b]$, the error between $p(x)$ and $f(x)$ is small enough: $\forall x \in [a, b]$, $|\varepsilon(x)| \leq \mu$ where μ is the target approximation quality and ε is the approximation error defined by

$$\varepsilon(x) = \frac{p(x)}{f(x)} - 1 \quad \text{or} \quad \varepsilon(x) = p(x) - f(x)$$

depending on whether the relative or absolute error is considered.

In other words, we want to prove that $\|\varepsilon\|_\infty \leq \mu$. Here, $\|\varepsilon\|_\infty$ denotes the infinity or supremum norm, defined by $\|\varepsilon\|_\infty = \sup_{x\in[a,\,b]}\{|\varepsilon(x)|\}$.

A simple numerical computation of the norm is not satisfying: guaranteeing correct rounding means guaranteeing a precise mathematical property and this requires proving each assertion. We will address here the problem of computing a certified bound for the supremum norm of an approximation error: given p and f, find an interval $\mathbf{r}$ (as thin as desired) such that $\|\varepsilon\|_\infty \in \mathbf{r}$.

By *certified*, we want to express the fact that the correctness of the algorithm must be proven. Thus, if there are no bugs in the implementation, the result given by the algorithm can be trusted.

In order to protect oneself from programming errors, one might want to compute, in the same time as $\mathbf{r}$, a certificate that can formally be checked using a proof assistant such as COQ [2] or PVS [26]. We do currently not compute such a certificate. In this article we will not address that point, that we consider to be future work.

This article presents a certified algorithm for computing the supremum norm of an approximation error. The function f involved in the approximation error is supposed to be differentiable up to a sufficient order n on the interval $[a, b]$. In Section 2, we briefly provide a general view of previous approaches for solving this question and we show why they seem inappropriate for the particular problem we address herein. In Sections 3.1 and 3.2, we first recall some classical techniques necessary for understanding our algorithm. Subsequently, in Section 3.3 we explain our algorithm. Finally, in Section 4, we present our implementation and we show experimental results.

2 Previous work

2.1 Interval Arithmetic

The idea of using interval arithmetic for obtaining guarantees on a numerical result, for encompassing finiteness, round-off errors and uncertainty is well established in the literature, e.g. [22], [16] [24]. It is a classical way of performing validated computations with floating-point arithmetic. The fundamental principle consists in replacing each number by an enclosing interval. In the framework of interval arithmetic, we define an interval $\mathbf{x}$ as a pair $\mathbf{x} = [\underline{x}, \overline{x}]$ consisting of two numbers $\underline{x}$ and $\overline{x}$ with $\underline{x} \leq \overline{x}$. A real number $x \in \mathbb{R}$ is *contained* in an interval $\mathbf{x}$, i.e., $x \in \mathbf{x}$, iff $\underline{x} \leq x \leq \overline{x}$. An interval may contain only one real number (i.e. it has zero width) and is denoted by $\mathbf{x} = [x, x]$, where x is the unique point it contains. In this case we say that $\mathbf{x}$ is a point or degenerated interval. We define the midpoint of the interval $\mathbf{x}$, defined as: $\mathrm{mid}\,(x) = {}^{(\underline{x}+\overline{x})}/_{2}$. The width of an interval is $\overline{x} - \underline{x}$. Moreover, we denote by $\varphi(\mathbf{x}) = \{\varphi(x) | x \in \mathbf{x}\}$ the exact image of the function φ over the interval $\mathbf{x}$.

The elementary arithmetic operations, as well as the elementary functions can be straightforwardly extended to handle intervals [22], [16], [24].

Although defining the interval arithmetic operations on real numbers proves to be straightforward, one has to take into account that the endpoints $\underline{x}$ and $\overline{x}$ of an interval $\mathbf{x}$ might not be representable on a given computer. For instance, the interval $[{}^1/_3,\ {}^2/_3]$ cannot be represented exactly using floating-point numbers. In such a case outward rounding, possibly in multiprecision arithmetic, is performed [10]. The `MPFI` library[*] provides such a multiprecision interval arithmetic: when performing an operation, the user chooses the precision used for representing the bounds of the result. The precision may be arbitrarily high. Consequently, $[{}^1/_3,\ {}^2/_3]$ for instance is replaced by an enclosing interval $[u,\ v]$ where $u \leq {}^1/_3$, ${}^2/_3 \leq v$ and u and v can be arbitrarily close to the exact values ${}^1/_3$ and ${}^2/_3$.

One fundamental use of interval arithmetic is bounding the image of a function over an interval. However, it is very well known that interval calculations generally overestimate the image of a function, phenomenon known under the name of “interval blow-up” [25]. This phenomenon is in general proportional to the width of the interval, and thus we are interested in using thin intervals for obtaining a reasonably tight bounding of the function image.

In particular, when evaluating a function φ over a point interval $\mathbf{x} = [x, x]$, the interval enclosure of $\varphi(x)$ can be made arbitrarily tight by increasing the precision used for evaluation until a satisfactory tightness is obtained. This implies that unless the value of $\varphi(x)$ is zero, the sign of $\varphi(x)$ can always be safely determined.

When evaluating functions over non-degenerated intervals, the algorithm presented in [6] can be used. This algorithm is based on a Taylor expansion of order 1 and allows for obtaining a smaller overestimation of the range of a function. This leads to results of superior quality compared to the usage of straightforward interval arithmetic. Another useful feature of this algorithm is that it is able to overcome removable singularities of functions using a heuristic based on L’Hôpital’s rule. For example, the algorithm returns a bounded interval for enclosing the image of the function $x \mapsto {}^{\sin(x)}/_{x}$ over an interval that contains 0; straightforward interval arithmetic would divide by zero and return $[-\infty,\ +\infty]$ as a bound.

2.2 Previous solutions to a specific problem

Bounding the supremum norm $\|\varepsilon\|_\infty$ of an approximation error $\varepsilon = {}^p/_f - 1$ (or $\varepsilon = p - f$) can be viewed as a global optimization problem [11].

There are efficient floating-point techniques for finding the global extremum of a function (see [3] for instance). These techniques consist in numerically finding a point x that is very close to a point $x^\star$ where the extremum of ε is reached. An approximate value of the extremum $\varepsilon(x^\star)$ is given by $\varepsilon(x)$. Consequently these techniques generally underestimate the supremum norm. Algorithms implemented in software tools like Maple and Matlab typically suffer from this issue [6]. The particular difficulty of the problem we address here is to find a value that is surely an upper-bound of the actual value $\|\varepsilon\|_\infty$.

The problem of finding a safe enclosure of the global optimum of a function has been studied [16, Chap.4], [21, Chap.1,2], [11]. However, the proposed algorithms are not suitable for the case of an approximation error. Indeed, the function ε is obtained by a subtraction between p and f. Since p approximates very well the function f, the leading bits of both components of the

[*]distributed under the LGPL and available at `http://gforge.inria.fr/projects/mpfi/`

subtraction cancel out. An arithmetic with arbitrary precision may reduce this phenomenon but it does not suppress it. This leads to very loose resulting interval bounds. For instance, GlobSol [17] does not go beyond double precision in its interval arithmetic library and is not tailored for such a specific problem.

There are approaches for this specific problem. In order to avoid the phenomenon of cancellation, they use a high order Taylor expansion T of the function ε. This way, the cancellations appear when computing the coefficients of the expansion and not when evaluating the approximation error function itself. Afterwards, a triangular inequality is used for upper-bounding the supremum norm of the approximation error: $\|\varepsilon\|_\infty \leq \|T\|_\infty + \|\varepsilon - T\|_\infty$. Two key problems occur: there is a need for a tight bounding of $\|T\|_\infty$ as well as for the Taylor remainder $\|\varepsilon - T\|_\infty$. Krämer used this approach and interval arithmetic in the development of the FI_LIB library [18]. However, his method has the disadvantage that no formal proof is produced, the results are not very tight if they come near the machine precision [14] and the remainder bound is computed manually.

Harrison also used the idea of working with a Taylor expansion of ε. He tightly bounds the supremum norm of the polynomial $\|T\|_\infty$ using a Sum of Squares (SOS) decomposition algorithm [13]. Usually the SOS algorithms are either very slow or they present numerical problems [13, Section 7]. The particular SOS algorithm presented in [13] solves some of these issues. The approach allows for verifying the results in HOL [12]. Nevertheless, the second problem of automatically bounding the Taylor remainder is not addressed.

In order to be able to bound the Taylor remainder automatically, solutions based on Taylor models have been proposed [7, 23]. Current implementations like COSY-INFINITY[†] are either limited to double precision or do not seem to offer the required safety. In particular, the Taylor remainder bounds are analyzed by hand [29]. Taylor models checked by formal tools like PVS may require expensive computations [7].

Chevillard and Lauter proposed an algorithm for computing the supremum norm of an approximation error that has the advantage of offering a safe and automatically validated result [6]. They assume that the function ε is at least twice continuously differentiable on the considered interval. They formally differentiate ε and look for the zeros of ε'. All zeros x_i of this derivative are enclosed by intervals $\mathbf{x_i}$, such that the diameters of $\mathbf{x_i}$ are less than a predefined small bound d, which is a parameter of their algorithm. These tight intervals can be computed by dichotomy or with interval Newton iteration. Then the function ε is evaluated on each interval $\mathbf{x_i}$ using the algorithm mentioned in Section 2.1. The resulting inner and outer enclosures [6] are combined to obtain an interval enclosure of $\|\varepsilon\|_\infty$ over $[a, b]$. However, when the degree of the approximation polynomial p is greater than 10, the problem of bounding the supremum norm $\|\varepsilon\|_\infty$ generally becomes unfeasible because of the computation time.

3 A polynomial-based approach

3.1 Some techniques derived from automatic differentiation

Given a function τ defined on an interval $[a, b]$ and a real number θ, we want to automatically compute a polynomial T such that $\|T - \tau\|_\infty \leq \theta$. In practice, τ is given as an expression tree that represents the function. For T, we might take a Taylor expansion of τ with a well-chosen order $n - 1$, or we may try to be more clever, e.g. by using an interpolation polynomial of degree $n - 1$ (see Section 3.3.3).

In both cases, denoting by $\tau^{(i)}$ the i-th derivative of τ, we need to bound the ratio $\tau^{(n)}/n!$ on $[a, b]$ automatically in order to ensure that $\|T - \tau\|_\infty \leq \theta$ (see Section 3.3.3). Moreover, if we choose to use a Taylor expansion, we need to compute it, which implies to compute $\tau^{(i)}/i!$ for each integer $i \in \{0, \dots, n-1\}$.

The naive approach consists in formally differentiating function τ n times and evaluating the expression obtained this way. This is utterly inefficient, in particular because the size of the

[†]http://bt.pa.msu.edu/index.htm

expressions grows too fast with the order of derivation.

Automatic differentiation is a technique that makes it possible to evaluate the derivative of a numerical function at some point, without having to formally differentiate it. Most of the time, this technique is described as a code transformation process: given the source code of a program that computes a numerical function, it deduces a program that computes the derivative. In general the considered functions have several variables.

Some authors have focused more precisely on the problem of efficiently computing the coefficients of the Taylor expansion of a univariate function [1], [23]. This is done again by a code transformation process (given a program computing a function, deduce the expansion of this function). There is no difficulty in adapting these techniques for working with expressions instead of code.

The main idea is the following. We want to compute the first n derivatives of τ in a given point x_0. So, we are not interested in knowing the expressions of the successive derivatives. All we need is given by an array of $n+1$ numbers: the array $[\tau_0, \ldots, \tau_n]$ where $\tau_i = {}^{\tau^{(i)}(x_0)}/_{i!}$. So, instead of working with expressions, we will work with arrays of $n+1$ numbers. This is much more convenient in terms of memory usage.

Basically, automatic differentiation consists in doing the same operations that would be done when evaluating the expression of the derivative, but without effectively writing the expression of the derivative. Consider for instance the addition of two functions u and v (supposed to be n times differentiable). If $[u_0, \ldots, u_n]$ and $[v_0, \ldots, v_n]$ are given, the array for the function $u+v$ is simply given by $(u+v)_i = u_i + v_i$.

The same way, using Leibniz' formula, it is easy to see that

$$(u \cdot v)_i = \sum_{k=0}^{i} u_k \cdot v_{i-k}.$$

Formulas for $\exp(u)$, $\cos(u)$, etc. may also be written. The interested reader will find in [9, Chap. 10] or [1] a complete introduction to this domain.

More generally, it is possible to write a recursive procedure that computes $(u \circ v)_i$ from the (v_k) computed in x_0 and the (u_k) computed in $v(x_0)$. This allows one to efficiently compute the first n derivatives of any function τ given by an expression.

Besides, these procedures may be used with interval arithmetic: if x_0 is replaced by an interval $\mathtt{x}$, and if the procedures are applied with interval arithmetic, the result is an interval $\boldsymbol{\tau}_\mathtt{i}$ that encloses the scaled image ${}^{\tau^{(i)}(\mathtt{x})}/_{i!}$ of the i-th derivative of τ on the interval $\mathtt{x}$. In the following, we denote by $\mathtt{AutoDiff}(\tau, n, \mathtt{x})$ a procedure that returns an array $[\boldsymbol{\tau}_\mathtt{0}^\mathtt{x}, \boldsymbol{\tau}_\mathtt{1}^\mathtt{x}, \ldots, ..., \boldsymbol{\tau}_\mathtt{n}^\mathtt{x}]$, where

$$\frac{\tau^{(i)}(\mathtt{x})}{i!} \subseteq \boldsymbol{\tau}_\mathtt{i}^\mathtt{x}.$$

Notice that by applying this to the interval $\mathtt{x} = [x_0, x_0]$ and using a multiprecision interval arithmetic, we obtain a safe enclosure of ${}^{\tau^{(i)}(x_0)}/_{i!}$ that can be made as accurate as desired. This will be useful for safely computing the Taylor coefficients of τ.

3.2 Isolation of roots of polynomials

Given a univariate real polynomial, we now want to compute thin isolating intervals for all of its real roots. These intervals should be disjoint and each interval should contain exactly one root. There are two main classes of methods for isolating the real roots of univariate polynomials.

On one hand we have the methods based on "Descartes' rule of signs", which upper-bounds the number of positive real roots of a polynomial by the number of sign changes in its coefficients. Descartes based strategies are well developed in the literature (see [28], [8, Chap. 5]). Roughly speaking, the algorithm is based on bounding the number of positive real roots by the sign changes in the coefficients of the polynomial. This bounding criterion can efficiently be applied for isolating the roots of a polynomial on the interval $]0, 1]$. For that purpose elementary transformations of the initial polynomial (like translation, homothety, reversal) are used. Furthermore, a strategy of successively bisecting the search interval is employed.

Although the computation of the number of roots in an interval is replaced by an upper-bound, we have the guarantee that this algorithm ends if the width of the search interval is sufficiently small (cf. Vincent's Theorem [28]).

On the other hand, another class of algorithms for isolating the real roots of a polynomial are the algorithms based on Sturm method (see [4] for an overview of these methods). Specifically, the exact number of distinct real roots is computed using an algorithm based on counting the sign changes in the Sturm sequence. Sturm method offers the advantage of an exact number of roots compared to Descartes' approach which gives an upper-bound for the number of roots.

For isolating and then refining the bounding of the roots, both classes of algorithms use a dichotomic approach [28], and/or a variant of Newton's iteration process [8, Chap. 5].

However, the computation of the Sturm polynomials poses more numerical problems [4] and has a higher complexity compared to Descartes based process [27]. Descartes' method requests the precondition that the polynomial should be square-free. In general, this condition can be insured by computing the greatest common divisor between the polynomial and its first derivative, which might prove as hard as computing the Sturm sequence, unless techniques based on modular arithmetic (computations of gcd mod p where p belongs to a well chosen family of primes) are used.

According to [28], isolating the roots of polynomials can be done faster and safely using a hybrid algorithm based on the Descartes' method and interval arithmetic. In fact, in that algorithm, each coefficient of the polynomial is replaced by a tight interval that encloses it. Consequently, this yields to a decrease in the average bit size of the coefficients and thus to the cost of arithmetic operations.

3.3 Fast and certified bounding of the zeros of a derivative

3.3.1 Reducing absolute and relative error terms to polynomials

For computing the supremum norm of the approximation error ε, we use the same general idea as the one developed in [6]: we localize as accurately as possible the zeros of ε'. Actually, we know that if x is an extremum of ε over $[a, b]$, then $x = a$ or $x = b$ or $\varepsilon'(x) = 0$.

Hence we want to compute a list of arbitrarily small intervals $\mathcal{Z}$, such that we are sure that every zero of ε' lies in one of the intervals $\mathbf{z}$. Furthermore we just have to evaluate ε by means of interval arithmetic on each $\mathbf{z}$ for obtaining a safe enclosure of the extrema.

Let us remark that some $\mathbf{z}$ may possibly contain several zeros of ε'. Likewise some $\mathbf{z}$ may not contain any zero of ε'. What is important here is that each zero of ε' actually lies in one of the $\mathbf{z}$'s and that the intervals $\mathbf{z}$ are thin enough for the interval evaluation of ε on $\mathbf{z}$ to be an accurate upper-bound of the exact image interval $\varepsilon(\mathbf{z})$.

In the case of the absolute error $\varepsilon = p - f$, so $\varepsilon' = p' - f'$. Let $\tau = p' - f'$.

The case of the relative error $\varepsilon = p/f - 1$ is more tricky. It holds that $\varepsilon' = (p'f - f'p)/f^2$. A problem will occur if f has a zero in $[a, b]$. As a matter of fact, ε may be well-defined and infinitely differentiable, although the function f vanishes at some point z. This happens when p also has a zero in z.

We face a problem when it comes to compute the Taylor expansion of ε': it requires a division by f^2. Straightforward interval arithmetic ignores the fact that p and f vanish simultaneously and returns something like $[-\infty, +\infty]$. In [6] a method based on L'Hôpital's rule has been proposed that makes it possible to evaluate such a division by interval arithmetic and with a relevant result. However, this method supposes that the function being evaluated is given by an expression. This cannot be used here, since automatic differentiation precisely avoids using expressions.

In consequence, in the case of the relative error, we define τ by $\tau = p'f - f'p$. Thus, each zero of ε' is also a zero of τ. The reciprocal is not true: we may have introduced new zeros. Anyway, there is no harm in having these superfluous zeros, if the intervals $\mathbf{z}$ in which they are enclosed are thin enough.

How can we make sure that we bound every zero of τ without forgetting any? We reduce the problem to the polynomial case. This case is easy to solve as we saw it in Section 3.2. For this purpose, suppose that we know a polynomial T that approximates τ with a remainder bounded by θ: $\forall x \in [a,\, b], |\tau(x) - T(x)| \leq \theta$. We will see in Section 3.3.3 how we can compute such a couple $(T,\, \theta)$. Of course, θ is chosen to be small with respect to $\|\varepsilon\|_\infty$. The smaller θ will be, the thinner the enclosure of the zeros of τ will be.

3.3.2 Tight bounding of the zeros of a function

We have reduced the initial problem to the following one: let τ be a function defined on an interval $[a,\, b]$. We suppose that τ is approximated by a polynomial T with a remainder bounded in magnitude by θ. We want to compute a list $\mathcal{Z}$ that contains intervals $\mathbf{z}$ (of arbitrarily small width) such that

$$\forall z \in [a,\, b], \quad \tau(z) = 0 \Longrightarrow \exists\, \mathbf{z} \in \mathcal{Z},\, z \in \mathbf{z}.$$

Remark that $\tau(z) = 0$ implies that $|T(z)| \leq \theta$. Thus, it suffices to find the points $z \in [a,\, b]$ such that $T(z)$ is included in the strip delimited by $-\theta$ and θ (see Figure 1). Formally, we look for the points z such that both $T(z) \leq \theta$ and $T(z) \geq -\theta$ hold.

Suppose for a moment we could compute a list $\mathcal{L}_u$ of intervals representing the points where T is below θ. The same way, let $\mathcal{L}_\ell$ be a list of intervals representing the points where T is above $-\theta$. It is clear that out of both lists we can compute intervals $\mathbf{z_i}$ for which T is in the strip delimited by $-\theta$ and θ. Indeed it suffices to intersect the intervals in $\mathcal{L}_u$ and $\mathcal{L}_\ell$ (see Figure 1). Formally we have

$$\mathcal{Z} = \left(\bigcup_{\mathbf{r} \in \mathcal{L}_u} \mathbf{r} \right) \bigcap \left(\bigcup_{\mathbf{s} \in \mathcal{L}_\ell} \mathbf{s} \right).$$

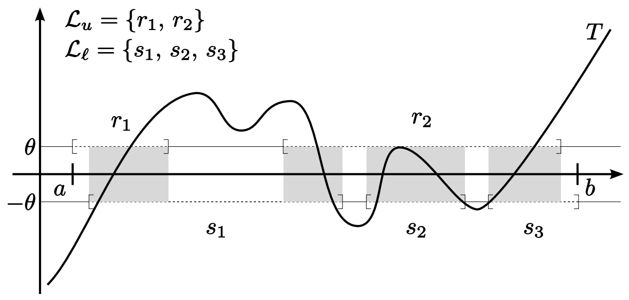
$\mathcal{L}_u = \{r_1,\, r_2\}$
$\mathcal{L}_\ell = \{s_1,\, s_2,\, s_3\}$
T
r_1
r_2
θ
$-\theta$
a
b
s_1
s_2
s_3

Figure 1: How to compute $\mathcal{Z}$

Actually the list $\mathcal{L}_u$ can be computed with the following technique, that also applies for $\mathcal{L}_\ell$.

$\mathcal{L}_u$ represents the set of points x where $T(x) \leq \theta$, i.e. $T(x) - \theta \leq 0$. $T - \theta$ is a polynomial: hence it has a finite number of roots in the interval $[a,\, b]$. Moreover, the areas where $T - \theta \leq 0$ are delimited by the zeros of $T - \theta$. Thus, it suffices to find the zeros of $T - \theta$ and look at its sign on the left and on the right of each zero, in order to know where $T - \theta$ is positive and where it is negative.

Since we want certified results, we must be rigorous. We use a technique of root isolation for polynomials (Section 3.2). It returns the list $\mathcal{Z}_u = \{\mathbf{z_1}, \cdots, \mathbf{z_k}\}$ of the roots of $T - \theta$ (the $\mathbf{z_i}$s are in fact thin enclosing intervals).

For each $\mathbf{z} \in \mathcal{Z}_u$ we must determine the sign of $T - \theta$ at $\underline{z}$ and at $\overline{z}$. For this purpose we compute by interval arithmetic $(T - \theta)([\underline{z},\, \underline{z}])$ (we do the same for $\overline{z}$). Since $[\underline{z},\, \underline{z}]$ is degenerated, we can make the resulting interval as tight as we want by increasing the precision (see Section 2.1). In particular, we eventually determine the sign of $(T - \theta)(\underline{z})$ with safety.

The list $\mathcal{L}_u$ is then easy to obtain. If $\mathbf{z_i}$ is a zero characterized by the signs $(+,\, -)$, it means that $T - \theta$ becomes negative after $\mathbf{z_i}$. It becomes positive again with the first following $\mathbf{z_j}$ characterized

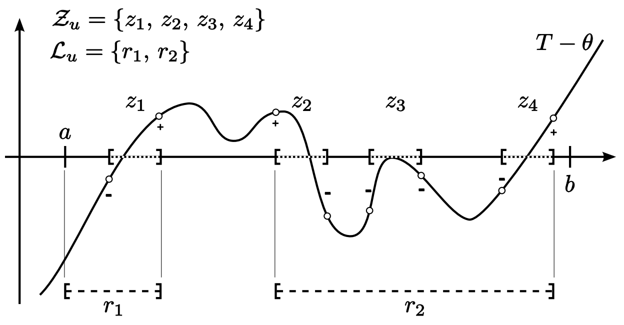
$\mathcal{Z}_u = \{z_1, z_2, z_3, z_4\}$
$\mathcal{L}_u = \{r_1, r_2\}$
$T-\theta$
z_1
z_2
z_3
z_4
a
b
r_1
r_2

Figure 2: How to compute $\mathcal{L}_u$

by the signs $(-, +)$. Consequently we add $[\underline{\mathbf{z_i}}, \overline{\mathbf{z_j}}]$ to the list $\mathcal{L}_u$. A special rule applies for the first and the last zero. This is illustrated in Figure 2.

3.3.3 Certified computation of approximating polynomials

Taylor expansions with a certified remainder Let $n \in \mathbb{N}$. Any n times differentiable function τ over an interval $[a, b]$ around x_0 can be written as

$$\tau(x) = \sum_{i=0}^{n-1} \frac{\tau^{(i)}(x_0)}{i!} \cdot (x - x_0)^i + \Delta_n(x, \xi),$$

where $\Delta_n(x, \xi) = \dfrac{\tau^{(n)}(\xi)(x - x_0)^n}{n!}$, $x \in [a, b]$ and ξ lies strictly between x and x_0.

We face two problems. First we need to bound the remainder $\Delta_n(x, \xi)$. This bound does not need to be tight: it suffices that we prove it to be small enough. The second problem comes from the fact that the coefficients $c_i = \tau^{(i)}(x_0)/i!$ of the Taylor polynomial are not exactly computable. Thus, we need to bound tightly the difference between the polynomial we compute and the actual Taylor polynomial.

a) It is easy to bound the term $(x - x_0)^i$, where $x, x_0 \in [a, b]$ and $i \in \{0, ..., n\}$. Let $\mathbf{x_i}$ be an enclosing interval: $(x - x_0)^i \in \mathbf{x_i}$.

b) Bounding $\Delta_n(x, \xi)$. We compute an interval enclosure

$$\boldsymbol{\tau}_{\mathbf{n}}^{[\mathbf{a},\mathbf{b}]} \ni \frac{\tau^{(n)}(\xi)}{n!}, \quad \text{where } \xi \in [a, b]$$

calling `AutoDiff(`τ`,`n`,`$[a, b]$`)`. Let $\boldsymbol{\Delta} = \boldsymbol{\tau}_n^{[a,b]} \cdot \mathbf{x_n}$
Thus, $\Delta_n(x, \xi) \in \boldsymbol{\Delta}$.

c) Enclosing the coefficients c_i, $i \in \{0, ..., n - 1\}$. We obtain interval enclosures $\mathbf{c_i} \ni c_i$ by calling `AutoDiff(`τ`,`$n-1$`,`$[x_0, x_0]$`)`.
Moreover, the resulting intervals $\mathbf{c_i}$ can be made arbitrarily tight by increasing the precision of the interval arithmetic since the automatic differentiation process is applied on a point interval (see Section 2.1).

d) Computation of the approximation polynomial. Consider $t_i = \mathrm{mid}\,(\mathbf{c_i})$ and the polynomial $T(x)$ of degree $n - 1$,

$$T(x) = \sum_{i=0}^{n-1} t_i (x - x_0)^i.$$

The difference between T and the actual Taylor polynomial can be easily bounded using interval arithmetic. We have $c_i \in \mathbf{c_i}$, and thus $(c_i - t_i) \in [\underline{\mathbf{c_i}} - t_i, \overline{\mathbf{c_i}} - t_i]$, which leads to

$$\sum_{i=0}^{n-1}(c_i - t_i)(x - x_0)^i \in \sum_{i=0}^{n-1}[\underline{\mathbf{c_i}} - t_i, \overline{\mathbf{c_i}} - t_i] \cdot \mathbf{x_i} = \boldsymbol{\delta}.$$

Finally, the error between the function τ and its approximation polynomial T is bounded by $\boldsymbol{\delta} + \boldsymbol{\Delta}$:

$$\forall x \in [a, b], \tau(x) - T(x) \in \boldsymbol{\delta} + \boldsymbol{\Delta}.$$

Let $\theta \in \mathbb{R}^+$ minimal such that $\boldsymbol{\delta} + \boldsymbol{\Delta} \subseteq [-\theta,\, \theta]$. Thus, $\forall x \in [a,\, b],\ |\tau(x) - T(x)| \leq \theta$.

Chebyshev approximations for a tighter remainder As we have seen, a Taylor polynomial is an approximation to τ that is easy to compute. Moreover, we know how to safely bound the remainder. However, the Taylor polynomial of a given degree is in general a poor approximation to the function among other polynomials of the same degree. In particular, as the domain $[a,\, b]$ grows, this technique may require a very high order. To solve this issue, the domain $[a,\, b]$ might be cut into smaller intervals on which the algorithm is applied.

Another idea (that does not exclude the other solution) is to replace Taylor polynomials by polynomials with a better approximation quality and for which we also know an explicit bound of the remainder. A natural idea is to choose a polynomial that interpolates τ. It is possible to effectively bound the remainder of such a polynomial [5, Chap. 3]. Chebyshev points

$$\cos\left(\frac{\pi(k + 1/2)}{n+1}\right) \cdot \frac{b-a}{2} + \frac{a+b}{2}, \quad (k = 0,\, \ldots,\, n)$$

form a classical choice of interpolation points and offer in general a very good quality of approximation [5].

In order to use such polynomials, there are some issues to solve. We did not implement this technique yet and technical details are still future work.

4 Experimental results

We have implemented a prototype of the method using the Sollya software tool[‡]. This prototype is available on request and will be distributed as a free software. Our prototype includes an implementation of automatic differentiation in interval arithmetic with arbitrary precision. For isolating the roots of polynomials, we currently use an algorithm that uses interval arithmetic and based on Sturm's method. For comparison purpose, we plan to implement an algorithm based on Descartes' rule of sign in a close future.

Our algorithm takes in input a function f, a polynomial p, an interval $[a,\, b]$ and an argument that indicates if the relative or absolute error is to be considered. Moreover, our algorithm relies on two parameters: the precision `prec` used in the interval arithmetic and the maximal error θ allowed for the higher order polynomial approximation. We currently use a Taylor polynomial.

We set $\tau = p' - f'$ or $\tau = p' \cdot f - f' \cdot p$ as explained in Section 3.3.1. This step currently requires a formal differentiation of p and f. We plan to use automatic differentiation here as well in the future.

We begin by heuristically finding an order $n - 1$ such that the Taylor polynomial T obtained by automatic differentiation (see Section 3.3.3) has a remainder bounded by θ. Then we enclose the zeros of τ using the polynomial T, as explained in Section 3.3.2. This gives us a list of intervals $\mathcal{Z}$ to which we add $[a,\, a]$ and $[b,\, b]$. Thus the list contains every possible extremum of τ.

Finally, we evaluate the approximation error $p - f$ or $p/f - 1$ (using interval arithmetic) for each interval of $\mathcal{Z}$. We combine the resulting intervals and obtain an enclosure of $\|\varepsilon\|_\infty$.

[‡]available at `http://sollya.gforge.inria.fr`

Example	f	$[a,\, b]$	$\deg(p)$	mode	$\deg(T)$	quality	time
#1	$\exp(x) - 1$	$[-0.25,\, 0.25]$	5	relative	11	37.6	0.4 s
#2	$\log_2(1 + x)$	$[-2^{-9},\, 2^{-9}]$	7	relative	23	83.3	2.2 s
#3	$\arcsin(x + m)$	$[a_3,\, b_3]$	22	relative	37	15.9	5.1 s
#4	$\cos(x)$	$[-0.5,\, 0.25]$	15	relative	28	19.5	2.2 s
#5	$\exp(x)$	$[-0.125,\, 0.125]$	25	relative	41	42.3	7.8 s
#6	$\sin(x)$	$[-0.5,\, 0.5]$	9	absolute	14	21.5	0.5 s
#7	$\exp(\cos(x)^2 + 1)$	$[1,\, 2]$	15	relative	60	25.5	11.0 s
#8	$\tan(x)$	$[0.25,\, 0.5]$	10	relative	21	26.0	1.1 s
#9	$x^{2.5}$	$[1,\, 2]$	7	relative	26	15.5	1.4 s

Table 1: Results of our algorithm on several examples

In Table 1 we present nine examples of various situations. Experiments were made on an Intel Pentium D 3.00 GHz with a 2 GB RAM running GNU/Linux and compiling with gcc. The two first examples are those presented in [6]. The authors say that the second example is handled in about 320 seconds on a 2.5 GHz Intel Pentium 4. Thus our algorithm has a speed-up factor of about 120 compared to their algorithm.

The third example is a polynomial taken from the source code of `CRlibm`[§]. In this example,

$$\begin{aligned} m &= 770422123864867 \cdot 2^{-50}, \\ a_3 &= -205674681606191 \cdot 2^{-53}, \\ b_3 &= 205674681606835 \cdot 2^{-53}. \end{aligned}$$

This is typically the kind of problems that developers of `libms` address. The degree of p is 22, which is quite high in this domain. Our algorithm only needs 5 seconds to handle it.

In the examples 4 to 9, the polynomial p is the minimax, i.e. the polynomial of a given degree that minimizes the supremum norm of the error. These examples involve more or less complicated functions on intervals of various width. Examples 7 and 9 should be considered as quite hard for our algorithm since the interval $[a,\, b]$ has width 1: this is large when using Taylor polynomials and it requires a high degree. All examples are handled in less than 15 seconds. This is reasonable for a computation that is made once, when implementing a function in a `libm`. Most of our examples cannot be handled with a global optimization software like GlobSol because arbitrary precision is intrinsically needed for avoiding the cancellations effects.

For each example presented in Table 1, we give the computation time and the quality of the computed bound. More precisely, if the algorithm computes a bound $[\ell,\, u]$, the quality of the result is $-\log_2({}^{(u-\ell)}/_{\ell})$. It indicates roughly the accuracy (in bits) given when considering u as an approximated value of $\|\varepsilon\|_\infty$. Accuracy specifications, as for correct rounding, generally simply ask for knowing the order of magnitude of the error. Knowing about 15 significant bits is hence quite accurate.

5 Conclusion and future work

Bounding the approximation error $\varepsilon = {}^{p-f}/_{f}$ between a function f and an approximating polynomial p is required as a basic block in implementing correctly rounded elementary functions for mathematical libraries. However, previous approaches prove to be unsatisfactory in what concerns safety, automation or computation time.

In this paper, we presented a safe and fast algorithm for bounding the supremum norm of the approximation errors. We combined and reused several existing techniques and designed a new algorithm which overcomes previous shortcomings. Our algorithm can handle absolute errors as well as relative errors.

[§] `CRlibm` is distributed under the LGPL and available at `http://lipforge.ens-lyon.fr/www/crlibm/`

Our algorithm uses automatic differentiation techniques and interval arithmetic for a fast and safe computation of high order derivatives over intervals. This lets us compute automatically Taylor polynomials with a safely bounded remainder. It will also permit to use Chebyshev approximation polynomials in the future. This should lead to a significant improvement of the performances of the algorithm: a speed-up of the computations and the possibility of tackling error functions defined over larger intervals.

We explained how we can use high order approximation polynomials and root isolation technique for finding tight enclosure of the zeros of ε'.

The implementation of our algorithm has successfully been used for various examples, including an example really used in the code of `CRlibm`. All examples are safely handled faster and more accurate than in other related approaches currently available.

Currently, a limitation of our algorithm is that no formal proof is provided. In order to solve this issue, we must see if it is possible to use one of the techniques for surely isolating the zeros of a polynomial, in a formal proof checker. Automatic differentiation must also be available in the proof checker. Finally, arbitrary precision interval arithmetic must be performed with the proof checker. That point has been solved recently [20] and is no longer an issue.

References

[1] C. Bendsten and O. Stauning. TADIFF, a Flexible C++ Package for Automatic Differentiation Using Taylor Series. Technical Report 1997-x5-94, Technical University of Denmark, April 1997.

[2] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: the Calculus of Inductive Constructions*. Springer, 2004.

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

[4] Fabrizio Broglia, editor. *Lectures in Real Geometry*, volume 23 of *De Gruyter Expositions in Mathematics*, pages 1–67. Walter de Gruyter, 1996.

[5] Elliot W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill Book Co., New York, 1966.

[6] S. Chevillard and Ch. Lauter. A certified infinite norm for the implementation of elementary functions. In *Proceedings of the Seventh International Conference on Quality Software*, pages 153–160, 2007.

[7] M. Daumas, G. Melquiond, and C. Muñoz. Guaranteed proofs using interval arithmetic. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 188–195, 2005.

[8] Laurent Fousse. *Intégration numérique avec erreur bornée en précision arbitraire*. PhD thesis, Université Henri Poincaré, Nancy 1, 2006.

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

[10] M. Grimmer, K. Petras, and N. Revol. Multiple precision interval packages: Comparing different approaches. In *Lecture Notes in Computer Science*, volume 2991, pages 64–90, 2004.

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

[12] John Harrison. Hol light: A tutorial introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

[13] John Harrison. Verifying nonlinear real formulas via sums of squares. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118. Springer-Verlag, 2007.

[14] 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.

[15] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754™-2008 (Revision of IEEE Std 754-2008)*, August 2008. Sponsored by the Microprocessor Standards Committee, IEEE, 3 Park Avenue, New York, NY, USA.

[16] Baker Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer, Dordrecht, Netherlands, 1996.

[17] R. Baker Kearfott. Globsol: History, composition, and advice on use. In *In Global Optimization and Constraint Satisfaction, Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, 2003.

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

[19] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding of the elementary functions in double precision. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 111–118, 2001.

[20] G. Melquiond. Floating-point arithmetic in the Coq system. In *RNC 8 Proceedings, 8th Conference on Real Numbers and Computers*, pages 93–102, 2008.

[21] F. Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes*. PhD thesis, Institut National Polytechnique de Toulouse, 1997.

[22] Ramon E. Moore. *Methods and Applications of Interval Analysis*. Society for Industrial Mathematics, 1979.

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

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

[25] Arnold Neumaier. Taylor forms–use and limits. *Reliable Computing*, 9(1):43–79, 2003.

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

[27] F. Rouillier and P. Zimmermann. Efficient isolation of a polynomial real roots. Technical Report 4113, INRIA, 2001. `http://www.inria.fr/rrrt/rr-4113.html`.

[28] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of polynomial's real roots. *Journal of Computational and Applied Mathematic*, 162(1):33–50, 2004.

[29] R. Zumkeller. Formal Global Optimization with Taylor Models. In *IJCAR 2008 - Proceedings of the 4th International Joint Conference on Automated Reasoning*, pages 408–422, 2008.