2009 19th IEEE International Symposium on Computer Arithmetic

Certified and fast computation of supremum norms of approximation errors

Sylvain Chevillard
sylvain.chevillard@ens-lyon.fr

Mioara Joldeş
mioara.joldes@ens-lyon.fr

Christoph Lauter*
christoph.lauter@ens-lyon.org

LIP (CNRS/ÉNS Lyon/INRIA/Université de Lyon)
Arénaire Project-Team
46, allée d'Italie, 69364 Lyon Cedex 07, France

Abstract

In many numerical programs there is a need for a high-quality floating-point approximation of useful functions f, such as such as exp, sin, erf. *In the actual implementation, the function is replaced by a polynomial p, which leads 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 problem is difficult mainly because that approximation error is very small and the difference $p - f$ is subject to high cancellation. Previous approaches for computing the supremum norm in this degenerate case, have proven to be 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 give timings on several examples.

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

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 include `CRlibm`, `glibc`, Sun `libmcr` and Intel® `MKL`.

In general, these `libm`s 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 new IEEE 754-2008 standard [10] recommends that the functions provided accordingly to the standard be correctly rounded. Under round-to-nearest this means that for a given function f, the function code must always return the floating-point number that is the closest to the exact value $f(x)$.

Particular care is needed for guaranteeing such a property. It can be shown [12] 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. 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 $\|\varepsilon\|_\infty = \sup_{x \in [a, b]} \{|\varepsilon(x)|\}$.

*Chr. Lauter was with ÉNS Lyon at the time this work was completed. He is now with Intel Corporation, SSG-DPD-Numerics team, 2111 NE 25th Avenue, M/S JF1-13, Hillsboro, OR, 97124, USA.

1063-6889/09 $25.00 © 2009 IEEE
DOI 10.1109/ARITH.2009.18

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 $\boldsymbol{r}$ (as thin as desired) such that $\|\varepsilon\|_\infty \in \boldsymbol{r}$.

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, along with $\boldsymbol{r}$, a certificate that can formally be checked using a proof assistant such as COQ* or PVS†. Currently we do not compute such a certificate. We will not address that point, that we consider to be future work.

The polynomial p is often obtained with the Remez algorithm which computes the polynomial that best approximates f, i.e. that minimises the supremum norm of the error. It might seem more interesting to modify the Remez algorithm, making it certified, instead of computing the supremum norm afterwards. This only moves the problem: the Remez algorithm requires to compute supremum norms [4]. Hence a certified algorithm for computing supremum norms is needed in a certified Remez algorithm.

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 provide a general view of previous approaches and we show why they seem inappropriate for the particular problem we address herein. We sketch our algorithm in Section 3.1. In Section 3.3, we show how to compute safely and automatically an approximating polynomial with a certified remainder. It uses classical techniques derived from automatic differentiation that are first recalled in Sections 3.3.1. Then, we show in Section 3.4 how to enclose the zeros of the derivative of the approximation error. Since we reduce to the well-known case of enclosing the roots of a polynomial, we first recall the corresponding classical techniques in Section 3.4.1. Finally, in Section 4 we show experimental results.

*http://coq.inria.fr/

†http://pvs.csl.sri.com/

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. [15]. It is a classical way to perform validated computations with floating-point arithmetic. In the following, we denote an interval $\boldsymbol{x}$ as a pair $\boldsymbol{x} = [\underline{x}, \overline{x}]$ consisting of two numbers $\underline{x}$ and $\overline{x}$ with $\underline{x} \leq \overline{x}$. We define the midpoint of the interval $\boldsymbol{x}$ as: $\operatorname{mid}(\boldsymbol{x}) = {}^{(\underline{x}+\overline{x})}/_2$. The width of an interval is $\overline{x} - \underline{x}$. Moreover, we denote by $\varphi(\boldsymbol{x}) = \{\varphi(x) | x \in \boldsymbol{x}\}$ the exact image of the function φ over the interval $\boldsymbol{x}$.

One fundamental use of interval arithmetic is bounding the image of a function over an interval. However, when using interval calculations, the image of the function is overestimated. As discussed in [17], for a large class of functions, this overestimation is proportional to the width of the evaluation interval. We are therefore interested in using thin intervals for obtaining reasonably tight bounds of the images.

In particular, when evaluating such a function φ over a point interval $\boldsymbol{x} = [x, x]$, the interval enclosure of $\varphi(x)$ can be made arbitrarily tight by increasing the precision used for evaluation [17]. This implies that unless the value of $\varphi(x)$ is zero, the sign of $\varphi(x)$ can always be safely determined. In the sequel we will make this assumption for all the functions that we deal with.

We use the `MPFI` library‡ for multiprecision interval arithmetic. It allows for an arbitrarily high precision for the interval bounds.

When evaluating functions over intervals, the algorithm presented in [5] can be used. It is based on a Taylor expansion of order 1 and allows for obtaining smaller overestimates. 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 optimisation problem [7].

‡http://gforge.inria.fr/projects/mpfi/

There are efficient floating-point techniques for finding the global extremum of a function (see [2] 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 tools like Maple and Matlab typically suffer from this issue [5]. 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 [11], [14], [7]. 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 f very well, the subtraction $p(x) - f(x)$ is subject to high cancellation. In floating-point arithmetic, cancellation can be overcome by increasing working precision. Interval arithmetic, in contrast, does not consider p and f in one point x but on a whole interval $\boldsymbol{x}$. Hence, as p and f vary on $\boldsymbol{x}$, even if they cancel in each point $x \in \boldsymbol{x}$, interval subtraction does not yield a smaller value. That phenomenon, called decorrelation [6], cannot be suppressed by mere usage of arbitrary precision. It leads to very loose resulting interval bounds. For instance, GlobSol [11] is not tailored for our specific problem and produces completely loose bounds.

There are approaches addressing that specificity. Decorrelation can be overcome if the variations of p and f over an interval can explicitly be expressed and that way transformed into benign cancellation of the bounds of thin intervals, that behave like floating-point numbers. A common way for achieving that transformation is the use of a high order Taylor expansion T of the function $\varepsilon = p - f$ (resp. $\varepsilon = {}^{(p-f)}/_{f}$).

As the Taylor polynomial is affected by some error, 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 [9]. However, his method has the disadvantage that no formal proof can be produced, the results are not very tight if they come near the machine precision [9] and the remainder bound is computed manually.

Harrison also uses 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 [8]. Usually SOS algorithms are either very slow or present numerical problems [8]. The particular SOS algorithm presented in [8] solves some of these issues. With that approach results can be verified in the formal proof checker HOL*. Nevertheless, the second problem of automatically bounding the Taylor remainder is not addressed.

For automatic bounding of Taylor remainders solutions based on Taylor models have been proposed [6, 16]. 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 [21]. Taylor models checked by formal tools like PVS may require expensive computations [6].

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 [5]. They formally differentiate the function ε and look for the zeros of ε'. All zeros x_i of this derivative are enclosed by intervals $\boldsymbol{x_i}$, such that the diameters of $\boldsymbol{x_i}$ are less than a predefined small bound d, which is a parameter of their algorithm. These tight intervals can be computed by bisection or with interval Newton iteration. Then the function ε is evaluated on each interval $\boldsymbol{x_i}$ using the algorithm mentioned in Section 2.1. The resulting inner and outer enclosures [5] are combined to obtain an interval enclosure of $\|\varepsilon\|_\infty$ over $[a, b]$. As the enclosures $\boldsymbol{x_i}$ can be made small by adjusting the bound d, decorrelation issues are alleviated by their algorithm up to some point. However, when the approximation polynomial p starts to become too correlated to f, their approach breaks down because of computation time.

3. A polynomial-based approach

3.1. The algorithm at a glance

Our algorithm follows the same principal scheme as that algorithm presented in [5]. A list $\mathcal{Z}$ of thin intervals safely enclosing the zeros of the derivative ε' is computed first. Then ε is evaluated on each interval in $\mathcal{Z}$, which yields $\|\varepsilon\|_\infty$. The main novelty lies in how the zeros of the derivative are bounded.

Our algorithm takes as input f and p, an interval $[a, b]$ and an argument that indicates if ε is an absolute or relative error. It sets $\tau = p' - f'$ for absolute error or $\tau = p'f - f'p$ for relative error. All zeros of ε' are zeros of τ. This choice is explained in Section 3.2.

*http://www.cl.cam.ac.uk/~jrh13/hol-light/

†http://bt.pa.msu.edu/index.htm

It then replaces τ by a Taylor polynomial T with a known remainder bound $\|T - \tau\|_\infty \leq \theta$. The value θ is an additional parameter of the algorithm. The polynomial T, of a heuristically determined order $n-1$, is obtained using an automatic differentiation based technique, that will be discussed in Section 3.3.

The algorithm then safely encloses the zeros of the translated polynomials $T+\theta$ and $T-\theta$ using existing root finding techniques for polynomials, discussed in Section 3.4.1. A technique explained in Section 3.4.2 then allows for deriving enclosures for the zeros of τ from the enclosures of the zeros of these translated polynomials. That yields the list $\mathcal{Z}$ of enclosures of the zeros of the derivative.

3.2. Reducing error terms to polynomials

The key of our approach for computing the supremum norm $\|\varepsilon\|_\infty$ is the tight localisation of 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 $\varepsilon'(x) = 0$. We want to compute a list $\mathcal{Z}$ of arbitrarily small $\boldsymbol{z}$, such that we are sure that every zero of ε' lies in one of the intervals $\boldsymbol{z}$. Afterwards we just have to evaluate ε by means of interval arithmetic on each $\boldsymbol{z}$ for obtaining a safe enclosure of the extrema.

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

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 occurs if f has a zero in $[a, b]$. Actually ε may be bounded and even infinitely differentiable, although the function f vanishes at some point z. This happens when p has also 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 [5] 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. As explained in Section 3.3, this would hinder the use of automatic differentiation and hence have a huge negative performance impact.

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 $\boldsymbol{z}$ 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. For this purpose, we compute a polynomial T that approximates τ with a remainder bounded by θ: $\forall x \in [a, b], |\tau(x) - T(x)| \leq \theta$. We explain in the next Section 3.3 how we can compute such a couple (T, θ). 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. Once the case is reduced to a polynomial, the zeros can easily be bounded, as we will see in Section 3.4.

3.3. Computation of approximating polynomials

3.3.1 Some classical 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 Taylor polynomial T such that $\|T - \tau\|_\infty \leq \theta$. In practice, τ is given as an expression tree that represents the function. An order $n-1$ must be chosen for T.

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 $\|T - \tau\|_\infty \leq \theta$. Moreover, we need to compute the Taylor approximation itself. This implies to compute $\tau^{(i)}/i!$ for each integer $i \in \{0, \ldots, n-1\}$.

The naïve approach consists in formally differentiating n times function τ and successively evaluating the expressions obtained this way. This is utterly inefficient because the size of the expressions grows too fast with n.

Automatic differentiation is a technique that makes it possible to evaluate the derivative of a numerical function at some point, without actually differentiating it in a formal way. Even if the interest in automatic differentiation usually lies on multivariate functions, some authors focused on the problem of efficiently computing the coefficients of the Taylor expansion of a univariate function [1], [16].

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. Instead of writing the memory-intensive expressions of the successive derivatives, we work with compact arrays $[\tau_0, \ldots, \tau_n]$ where $\tau_i = \tau^{(i)}(x_0)/i!$.

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 $(u+v)_i = u_i + v_i$.

Considering Leibniz' formula, it is also 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 a complete introduction to this domain in [15, Chap. 3.4] or [1].

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.

These procedures may be used with interval arithmetic: if x_0 is replaced by an interval $\boldsymbol{x}$, and if the procedures are applied with interval arithmetic, the result is an interval $\boldsymbol{\tau_i}$ that encloses the scaled image ${}^{\tau^{(i)}(\boldsymbol{x})}/_{i!}$ of the i-th derivative of τ on the interval $\boldsymbol{x}$. In the following, we denote by `AutoDiff(`τ`,`n`,`$\boldsymbol{x}$`)` a procedure that returns an array $[\boldsymbol{\tau_0^x}, \boldsymbol{\tau_1^x}, \ldots, ..., \boldsymbol{\tau_n^x}]$, where ${}^{\tau^{(i)}(\boldsymbol{x})}/_{i!} \subseteq \boldsymbol{\tau_i^x}$.

By applying that procedure to the interval $\boldsymbol{x} = [x_0, x_0]$ and using a multiprecision interval arithmetic, we also obtain a safe and arbitrarily tight enclosure of ${}^{\tau^{(i)}(x_0)}/_{i!}$. This will be useful for safely computing the Taylor coefficients of τ.

3.3.2 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 $\boldsymbol{x_i}$ be an enclosing interval: $(x - x_0)^i \in \boldsymbol{x_i}$.

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

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

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

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

d) Computation of the approximation polynomial. Consider $t_i = \operatorname{mid}(\boldsymbol{c_i})$ 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 easily be bounded using interval arithmetic. We have $c_i \in \boldsymbol{c_i}$, and thus $(c_i - t_i) \in [\underline{\boldsymbol{c_i}} - t_i, \overline{\boldsymbol{c_i}} - t_i]$, which leads to

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

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, $\forall x \in [a, b]$, $|\tau(x) - T(x)| \leq \theta$.

Other approximations for a tighter remainder As seen, a Taylor polynomial approximating τ is easy to compute and its remainder can safely be bound. However, as the remainder may rapidly grow for larger domains $[a, b]$, a very high order may be required or the interval to be cut into smaller pieces.

Other approximation techniques, giving a tighter, though safe remainder bound, might be used in the future. We are considering the following two techniques.

Interpolation in Chebyshev points [4] commonly yields approximations of superior quality compared to Taylor polynomials. An explicit remainder bound is known [4, Chap. 3]. Some technical issues must be overcome for using that technique.

In practice, Taylor Models yield Taylor-like approximations that seem to have much tighter remainder interval bounds. They are equally safe. That better tightness of the remainder bounds can be explained by the fact that Taylor Models directly propagate them through the automatic differentiation process.

3.4. Fast and certified bounding of the zeros of ε'

3.4.1 How to isolate the roots of polynomials

Given a univariate real polynomial T, we now want to compute a list $\mathcal{Z}$ of disjoint thin intervals $\boldsymbol{z}$ such that each root of T lies in one of the intervals $\boldsymbol{z}$. We may use general methods such as the interval Newton method [17]. But we can also take advantage of the fact that T is a polynomial and use a specific method for isolating the real roots of a polynomial. There are two main classes of such specific methods.

On the one hand we have the methods based on "Descartes' rule of signs". Descartes based strategies are well developed in the literature (see [19]). Roughly speaking, the algorithm is based on bounding the number of positive real roots by the number of sign changes in the sequence of 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.

Although the computation of the number of roots in an interval is replaced by an upper-bound, the termination is guaranteed when the width of the search interval becomes sufficiently small [19].

On the other hand, another class of algorithms for isolating the real roots of a polynomial are the algorithms based on Sturm's method (see [3] for an overview). Specifically, the exact number of distinct real roots is computed using an algorithm based on counting the sign changes in the Sturm sequence. Sturm's 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 refining the bounding of the roots, both classes of algorithms use bisection [19].

However, the computation of the Sturm polynomials poses more numerical problems [3] and has a higher complexity compared to Descartes based process [18]. In contrast, Descartes' method may return wrong results if the polynomial is not square-free. In general, this condition can be ensured by computing the greatest common divisor between the polynomial and its first derivative, which is costly.

According to [19], 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 a decrease in the average bit size of the coefficients and thus to the cost of arithmetic operations.

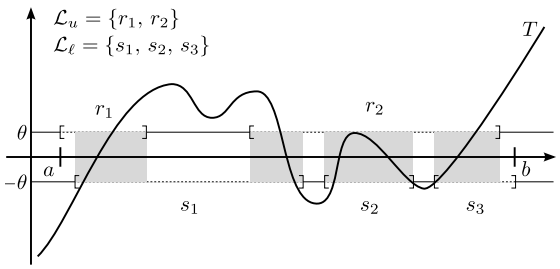
3.4.2 Tight bounding of the zeros of a function

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 $\boldsymbol{z}$ (of arbitrarily small width) such that

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

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 temporarily we could compute a list $\mathcal{L}_u$ of intervals representing the points $x \in [a, b]$ where $T(x)$ is below θ. The same way, let $\mathcal{L}_\ell$ be a list of intervals representing the points x where $T(x)$ is above $-\theta$. It is clear that out of both lists we can compute intervals $\boldsymbol{z_i}$ 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).



(Intervals r_i and s_i not to scale for illustration purposes)

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 one of the techniques of root isolation for polynomials (Section 3.4.1). It returns the list $\mathcal{Z}_u = \{\boldsymbol{z_1}, \cdots, \boldsymbol{z_k}\}$ of the roots of $T - \theta$ (the $\boldsymbol{z_i}$s are in fact thin enclosing intervals).

For each $\boldsymbol{z} \in \mathcal{Z}_u$ we must determine the sign of $T-\theta$ at the left (and at the right) of $\boldsymbol{z}$. Let $\boldsymbol{z'}$ be the previous

interval in $\mathcal{Z}_u$ (take $\boldsymbol{z'} = [a,\, a]$ if $\boldsymbol{z}$ is the first one). We know that $T - \theta$ does not change its sign between $\overline{\boldsymbol{z'}}$ and $\underline{\boldsymbol{z}}$. So, we simply need to determine the sign of $T - \theta$ at any point x between them (e.g. the middle of both values). This is achieved by evaluating $(T - \theta)$ by interval arithmetic on the point-interval $[x,\, x]$ with enough precision (see Section 2.1).

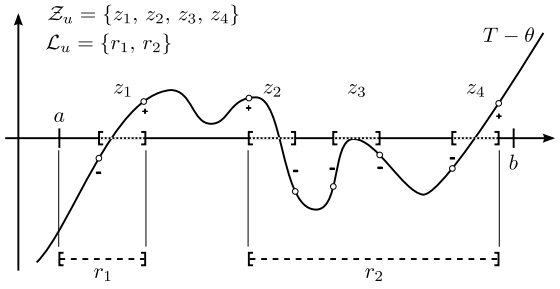


Figure 2. How to compute $\mathcal{L}_u$

The list $\mathcal{L}_u$ is then easy to obtain. If $\boldsymbol{z_i}$ is a zero characterised by the signs $(+,\, -)$, it means that $T - \theta$ becomes negative after $\boldsymbol{z_i}$. It becomes positive again with the first following $\boldsymbol{z_j}$ characterised by the signs $(-,\, +)$. Consequently we add $[\underline{\boldsymbol{z_i}},\, \overline{\boldsymbol{z_j}}]$ to the list $\mathcal{L}_u$. A special rule applies for the first and the last zero. This is illustrated in Figure 2.

4. Experimental results

We have implemented a prototype of the method using the free Sollya tool*, into which we will integrate it in the future. 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 is based on Sturm's method. For comparison, we plan to implement an algorithm based on Descartes' rule of sign or on the interval Newton algorithm in the near future.

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

The two first examples are those presented in [5]. With that technique, the second example was handled in about 320 seconds on a 2.5 GHz Pentium 4. The new algorithm has a speed-up factor of about 120.

The third example is a polynomial taken from the source code of `CRlibm`†. It is the typical problem that developers of `libms` address. The degree of p is 22, which is quite high in this domain. Our algorithm needs only 5 seconds to handle it.

In examples 4 through 9, the polynomial p is the minimax, i.e. the polynomial of a given degree that minimises the supremum norm of the error. These examples involve more or less complicated functions over 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`.

5. Conclusion and future work

Bounding the approximation error $\varepsilon = p/f - 1$ between a function f and an approximating polynomial p is indispensable when implementing correctly rounded functions for mathematical libraries. Previous approaches prove to be unsatisfactory concerning safety, automation or computation time.

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

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 in the performance of the algorithm: a speed-up of the computations and the possibility of tackling error functions defined over larger intervals. We also consider using Taylor Models as a replacement. Other techniques based on Cauchy's formula allow for computing certified polynomial approximations of analytic functions [16, 20]. Using them is future work.

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

*`http://sollya.gforge.inria.fr`

†`http://lipforge.ens-lyon.fr/www/crlibm/`

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

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 accurately than in other related approaches currently available.

A current 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 [13] and is no longer an issue.

Acknowledgements

The authors would like to thank Nicolas Brisebarre, Guillaume Hanrot, John Harrison, Frédéric Messine and Ned Nedialkov for their precious advice and help.

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] R. P. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, 1973.

[3] F. Broglia, editor. *Lectures in Real Geometry*, volume 23 of *Expositions in Mathematics*, pages 1–67. de Gruyter, 1996.

[4] E. W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, 1966.

[5] S. Chevillard and C. Lauter. A certified infinite norm for the implementation of elementary functions. In *Proc. of the 7th International Conference on Quality Software*, pages 153–160, 2007.

[6] M. Daumas, G. Melquiond, and C. Muñoz. Guaranteed proofs using interval arithmetic. In *Proc. of the 17th IEEE Symp. on Comp. Arithmetic*, pages 188–195, 2005.

*Values for example #3: $m = 770422123864867 \cdot 2^{-50}$, $a_3 = -205674681606191 \cdot 2^{-53}$, $b_3 = 205674681606835 \cdot 2^{-53}$

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

[8] J. Harrison. Verifying nonlinear real formulas via sums of squares. In *Proc. of the 20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007*, pages 102–118. Springer-Verlag, 2007.

[9] 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, Universität Karlsruhe, 1998.

[10] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754™-2008*, Aug. 2008.

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

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

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

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

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

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

[17] N. Revol. Newton's algorithm using multiple precision interval arithmetic. *Numerical Algorithms*, 34(2):417–426, 2003.

[18] F. Rouillier and P. Zimmermann. Efficient isolation of a polynomial real roots. Technical Report 4113, INRIA, 2001.

[19] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial's real roots. *Journal of Computational and Applied Mathematics*, 162(1):33–50, 2004.

[20] J. van der Hoeven. On effective analytic continuation. *Mathematics in Computer Science*, 1(1):111–175, 2007.

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