Efficient and accurate computation of upper bounds of approximation errors

Sylvain Chevillard, John Harrison, Mioara Maria Joldes, Christoph Lauter

Laboratoire de l'Informatique du Parallélisme

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

Efficient and accurate computation of upper bounds of approximation errors

Sylvain Chevillard*,
John Harrison**,
Mioara Joldeş***,
Christoph Lauter**

*INRIA, LORIA
Cacao Project-Team
BP 239, 54506 Vandœuvre-lès-Nancy Cedex, FRANCE

**Intel Corporation
2111 NE 25th Avenue, M/S JF1-13, Hillsboro, OR, 97124, USA

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

January 2010

Research Report N° RR2010-2

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`




Efficient and accurate computation of upper bounds of approximation errors

Sylvain Chevillard*, John Harrison**, Mioara Joldeş***, Christoph Lauter**

*INRIA, LORIA

Cacao Project-Team

BP 239, 54506 Vandœuvre-lès-Nancy Cedex, FRANCE

**Intel Corporation

2111 NE 25th Avenue, M/S JF1-13, Hillsboro, OR, 97124, USA

***LIP (CNRS/ÉNS Lyon/INRIA/Université de Lyon)

Arénaire Project-Team

46, allée d'Italie, 69364 Lyon Cedex 07, FRANCE

January 2010

Abstract

For purposes of actual evaluation, mathematical functions f are commonly replaced by approximation polynomials p. Examples include floating-point implementations of elementary functions, quadrature or more theoretical proof work involving transcendental functions.

Replacing f by p induces a relative error $\varepsilon = p/f - 1$. In order to ensure the validity of the use of p instead of f, the maximum error, i.e. the supremum norm $\|\varepsilon\|_\infty$ must be safely bounded above.

Numerical algorithms for supremum norms are efficient but cannot offer the required safety. Previous validated approaches often require tedious manual intervention. If they are automated, they have several drawbacks, such as the lack of quality guarantees.

In this article a novel, automated supremum norm algorithm with *a priori* quality is proposed. It focuses on the validation step and paves the way for formally certified supremum norms.

Key elements are the use of intermediate approximation polynomials with bounded truncation remainder and a non-negativity test based on a Sum-of-squares expression of polynomials.

The new algorithm was implemented in the tool Sollya. The article includes experimental results on real-life examples.

Keywords: Supremum norm, approximation error, Taylor Models, Sum-of-Squares, validation, certification, formal proof

1 Introduction

Replacing functions by polynomials to ease computations is a widespread technique in mathematics. For instance, handbooks of mathematical functions [1] give not only classical properties of functions, but also convenient polynomial and rational approximations of functions that can be used to approximately evaluate them. These tabulated polynomials have proved to be very useful in the everyday work of mathematicians.

Nowadays, computers are commonly used for computing numerical approximations of functions. Elementary functions, such as exp, sin, arccos, erf, etc., 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 the Intel® `libm` available with the Intel® Professional Edition Compilers and other Intel® Software Products.

When writing handbooks as well as when implementing such functions in a `libm`, it is important to rigorously bound the error between the polynomial approximation p and the function f. In particular, regarding the development of `libms`, IEEE 754-2008 standard [2] recommends that the functions be correctly rounded. Correct rounding of transcendental function requires strict discipline on accuracy bounding and proofs [3].

Currently most `libms` offer strong guarantees: they are made with care and pass many tests before being published. However, in the core of `libms`, the error between polynomial approximations and functions is often only estimated with a numerical application such as Maple that supports arbitrary precision computations. As good as this numerical estimation could be, it is not a mathematical proof. If a library claims to provide correctly rounded results, its implementation should be mathematically proven in the very details.

Given f the function to be approximated and p the approximation polynomial used, the approximation error is given 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. This function is very regular in general: in Figure 1 the approximation error is plotted in a typical case when a minimax approximation polynomial is used [4, Chapter 3].

A numerical algorithm tells us that the maximal absolute value of ε in this case (Figure 1) is approximately $1.1385 \cdot 10^{-6}$. But can we guarantee that this value is actually greater than the real maximal error, i.e. an upper bound for the error committed by the approximation of f by p? This is the problem we address in this article. More precisely, we present an algorithm for computing a tight interval $\boldsymbol{r} = [\ell, u]$, such that $\|\varepsilon\|_\infty^I \in \boldsymbol{r}$. Here, $\|\varepsilon\|_\infty^I$ denotes the infinity or supremum norm over the interval I, defined by $\|\varepsilon\|_\infty^I = \sup_{x \in I}\{|\varepsilon(x)|\}$.

As detailed in Section 3.2 the user can control *a priori* the tightness of $\boldsymbol{r}$ through a simple input parameter. In general $\boldsymbol{r}$ can be made as tight as desired. If the interval $\boldsymbol{r}$ is returned, it is guaranteed to contain $\|\varepsilon\|_\infty^I$ and satisfy the quality required by the user. In some cases, roughly speaking if the function is too complicated, our algorithm will simply fail, but it never lies. We have conducted many experiments challenging our algorithm and in practice it never failed. See Section 6 for details.

As we will see in the following, several approaches exist for bounding $\|\varepsilon\|_\infty^I$. Nevertheless the problem does not have a completely satisfying solution for the moment. In what follows we give an overview of the features needed for our algorithm. This will allow us to briefly present

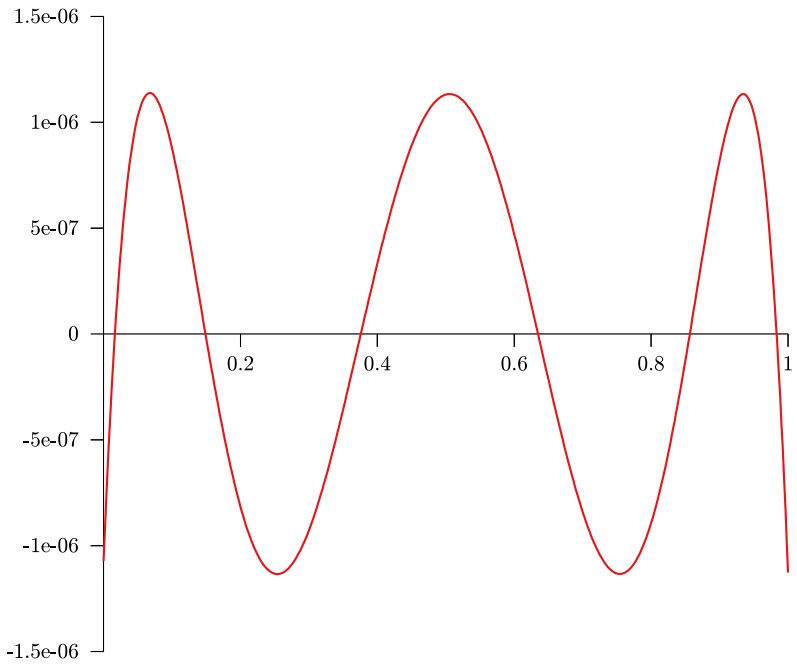

Figure 1: Approximation error in a typical case

and compare some previous works that partially accomplished these goals and highlight the main novelties of our algorithm.

i. We need an algorithm that is fully automated. In practice, a `libm` contains many functions. Each of them contains one or more approximation polynomials whose error must be bounded. It is frequent that new `libms` are developed when new hardware features are available. So our problem should be solved as automatically as possible and should not rely on manual computations.

ii. We would like the algorithm to handle not only simple cases when f is a basic function (such as exp, arcsin, tan, etc.) but also more complicated cases when f is obtained as a composition of basic functions such as $\exp(1+\cos(x)^2)$ for instance. Besides the obvious interest of having an algorithm as general as possible, this is necessary even for implementing simple functions in `libms`. Indeed it is usual to replace the function to be implemented f by another one g in a so-called range reduction process [5, Chapter 11]. The value $f(x)$ is in fact computed from $g(x)$. So, eventually, the function approximated by a polynomial is g. This function is sometimes given as a composition of several basic functions.

   In consequence, the algorithm should accept as input any function f defined by as an expression. The expression is made using basic functions such as exp or cos. The precise list of basic functions is not important for our purpose: we can consider the list of functions defined in software tools like Maple or Sollya [6] for instance.The only requirement for basic functions is that they be differentiable up to a sufficiently high order.

iii. The algorithm should be able to automatically handle a particular difficulty that fre-

quently arises when considering relative errors $\varepsilon(x) = \frac{p(x)}{f(x)} - 1$ in the process of developing functions for `libms`: the problem of removable discontinuities. If the function to be implemented f vanishes at a point x_0 in the interval considered, in general, the approximation polynomial is designed such that it vanishes also at the same point. Hence, although f vanishes, ε may be defined by continuous extension at such points x_0, called removable discontinuities. For example, if p is a polynomial of the form $x\,q(x)$, the function $p(x)/\sin(x) - 1$ has a removable discontinuity at $x_0 = 0$.

iv. The accuracy obtained for the supremum norm should be controlled *a priori* by the user's needs. In our algorithm, a parameter $\overline{\eta}$ controls *a priori* the relative tightness of $\boldsymbol{r} = [\ell, u]$: this means that the algorithm ensures that eventually

$$0 \leq \frac{u - \ell}{\ell} \leq \overline{\eta}.$$

The parameter $\overline{\eta}$ can be chosen as small as desired by the user; in practice, this parameter allows the user to choose the accuracy of ℓ and u seen as approximate values of $\|\varepsilon\|_\infty$.

v. Since complicated algorithms are used, their implementation could contain some bugs. Hence, beside the numerical result, the algorithm should return also a formal proof. This proof can be automatically checked by a computer and gives a high guarantee on the result.

1.0.1 Using generic rigorous global optimization algorithms

It is important to remark that obtaining a tight upper-bound for $\|\varepsilon\|_\infty$ is equivalent to rigorously solving a univariate global optimization problem. This question has already been extensively studied in the literature [7, 8, 9]. However, these algorithms are completely inefficient in our case: this is due to the particular nature of the function ε. Indeed, this function ε is the difference between two functions very close to each other. We will explain this phenomenon in Section 2.

1.0.2 Specific approaches

Since generic algorithms are not able to solve this problem, more specific approaches have been developed in the past fifteen years [10, 11, 12, 13, 14]. In the following we analyze them based on the above-mentioned key features for our algorithm.

Krämer needed to bound approximation errors while he was developing the FI_LIB library [10]. His method was mostly manual: bounds were computed case by case and proofs were made on papers. As explained above, from our point of view, this is a drawback.

In [11] and more recently in [13], one of the authors of the present paper proposed approaches for validating bounds on approximation errors with a formal proof checker. This method presents an important advantage compared to other techniques in what concerns the safety or the results: the proofs of Krämer were made by hand, which is error-prone; likewise, the implementation of an automatic algorithm could contain some bugs.

A formal proof can be checked by a computer and gives a high guarantee of the result. The methods presented in [11] and [13] mainly use the same techniques as Krämer: in particular,

the formal proof is written by hand. This is safer than in [10], since the proof can be automatically checked, but not completely satisfying: we would like the proof to be automatically produced by the algorithm.

Recently in [12, 14], authors of the present article tried to automate the computation of an upper-bound on $\|\varepsilon\|_\infty$. It did not seem obvious to automate the technique used in [10, 11, 13], while correctly handling the particular difficulty of removable discontinuities. The algorithm presented in [12] was a first attempt. The idea was to enhance Interval Arithmetic [15], such that it could correctly handle expressions exhibiting removable discontinuities. This algorithm does not really take into account the particular nature of ε and has mainly the same drawbacks as the generic global optimization techniques mentioned in Section 1.0.1. The second attempt, presented in [14], was more suitable but not completely satisfying: in particular it was impossible to control *a priori* the tightness of the computed interval enclosure for the supremum norm and no formal proof was produced. This algorithm is detailed in Section 2.

1.0.3 Outline of the paper

The present paper unifies the previous approaches and gives a satisfying solution for our problem. Our algorithm furnishes all the features presented above: it is fully automated; it can deal with complicated functions represented as a composition of basic functions; it can handle removable discontinuities in all practical cases; the accuracy obtained for the result is controlled "a priori"; a formal proof is generated. Currently, this formal proof is not complete, but generating a complete formal proof is essentially just a matter of implementation.

In the next section, we explain more precisely why our problem is so specific and why this makes the generic algorithms fail. Furthermore, we explain the main ideas of previous approaches for solving the problem. In Section 3, our algorithm is presented. As we will see, the algorithm relies on automatically computing an intermediate polynomial: Section 4 discusses several existing methods that can be used for computing this intermediate polynomial and we compare their practical results. These methods are not able to handle removable discontinuities; however, we managed to adapt one of the methods for solving this issue. Our modified method is presented in Section 4.4. In Section 5, we explain how a formal proof can be generated by the algorithm and checked by the HOL proof checker [*]. Finally, in Section 6 we show how our new algorithm behaves on real-life examples and compare its results with those obtained with our previous algorithms [12] and [14].

2 Previous Work

2.1 Numerical Methods for Supremum Norms

Firstly, one can consider a simple numerical computation of the supremum norm. In fact, we can reduce our problem to searching for extrema of the error function. These extrema are usually found by searching for the zeros of the derivative of the error function. Well-known numerical algorithms like bisection, Newton's algorithm or the secant algorithm can be used [4]. These techniques offer a good and fast estimation of the needed bound, and implementations are available in most numerical software tools, like Maple or Matlab.

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

Roughly speaking, all the numerical techniques finish by exhibiting a point $x^\star$, more or less near to the actual global extremum of f in the range considered. By evaluating $|f|$ at $x^\star$, an approximation ℓ of the supremum norm's value is obtained. Moreover, by adjusting accordingly the rounding modes we can ensure that $\ell \leq |f(x^\star)|$. Hence $\ell \leq \|\varepsilon\|_\infty$. That final evaluation can be made as accurate as desired. It just suffices to increase the working precision. Moreover, numerical algorithms based on Newton iteration often have *quadratic* convergence [16]. This means that a numerical supremum norm technique can be used to compute a value as close as desired to the value of the function f at *some* point $x^\star$ near to the actual extremum.

Let us stress it again: although fast and relatively easy to implement, these methods are purely numerical and the results they provide are not sufficient in our case.

2.2 Rigorous Global Optimization Methods using Interval Arithmetic

To ensure reliability in computation with finite precision numbers, and validation of the results obtained, one well established technique is Interval Arithmetic [15].

The applications of Interval Arithmetic to rigorous global optimization have been broadly developed in the literature [17, 7, 9]. These methods are based on a general interval branch-and-bound algorithm. It involves an exhaustive search over the initial interval. This interval is subdivided recursively ("branching"), and those subintervals that cannot possibly contain global optima are rejected. The rejection tests are based on using Interval Arithmetic for bounding the image of a function. Many variants for accelerating the rejection process have been implemented. One example is the Interval Newton method [18] used for safely enclosing all the zeros of a univariate function.

However, when using interval calculations, the image of the function is overestimated. As discussed in [18], for a large class of functions, this overestimation is proportional to the width of the evaluation interval. The subdivision process tries to benefit from the fact that for thin intervals the overestimation diminishes.

When evaluating 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 [18]. This can be achieved using multiprecision interval arithmetic libraries, like for example the MPFI Library [19].

2.3 Dependency Problem for Approximation Errors

While the above-mentioned methods can be successfully used in general, when trying to solve our problem, one is faced with the so-called "dependency phenomenon" [20]. Roughly speaking, it is due to the fact that multiple occurrences of the same variable are not exploited by Interval Arithmetic. The computations are performed "blindly", taking into account only the range of a variable independently for each occurrence.

In some cases, by using more clever techniques, rewriting expressions or taking into account the properties of the functions considered, one can reduce this phenomenon. In what follows, we will explain why our problem is prone to this phenomenon and detail its detrimental consequences.

In our case, the dependency phenomenon stems from the subtraction present in the approximation error function $\varepsilon = p - f$. In short, when using Interval Arithmetic, the correlation between f and p is lost. This means that although f and p have very close values, Interval

Arithmetic can not benefit from this fact and will compute an enclosure of ε as the difference of two separate interval enclosures for f and p over the given interval. Even if we could obtain exact enclosures for both f and p over the interval considered, the interval difference would be much wider.

To see what is happening, let us consider two intervals $[u, u+\delta]$ and $[v, v+\delta']$, with $\delta \simeq \delta'$. Their difference is: $[u-v-\delta', u-v+\delta]$, which is an interval approximately centered in $(u-v)$ of width $\delta+\delta' \simeq 2\delta$. We can have two situations: firstly, if $|u-v|$ is much smaller compared to δ, we have almost the interval $[-\delta', \delta]$. On the contrary, if $|u-v|$ is bigger, the interval is very small (almost a point interval) $[u-v, u-v]$.

Analogously, in our situation, given an interval I we want to measure the difference between the exact image of the error $\varepsilon(I)$ and the computed enclosure using the formula: $\varepsilon(I) \subseteq p(I) - f(I)$. For I small enough, we can suppose that p and f have a linear behavior over I. We suppose also that f and p are increasing over I (the case when they are decreasing is analogous). We have $p(I) \simeq [u, u+\delta]$, where $u \simeq p(\inf(I))$ and $\delta = p'(\inf(I)) \operatorname{diam}(I)$. Similarly, we have: $f(I) \simeq [v, v+\delta']$, where $v \simeq f(\inf(I))$ and $\delta' = f'(\inf(I)) \operatorname{diam}(I)$. Moreover, $\|\varepsilon\|_\infty$ over I and $(p-f)(\inf(I))$ have approximately the same magnitude.

So, we can infer two cases: if $\operatorname{diam}(I) \gg \|\varepsilon\|_\infty$, the interval difference is approximately $[-\operatorname{diam}(I), \operatorname{diam}(I)]$. If $\operatorname{diam}(I) \ll \|\varepsilon\|_\infty$, the interval difference gives a small interval around $f(\inf(I)) - p(\inf(I))$, or better said, something representative for the true image of $\varepsilon(I)$.

Hence, we can conclude that in order to obtain a small overestimation for this image, we need to evaluate ε over intervals of the size of $\|\varepsilon\|_\infty$. Note that in some specific cases used in the process of obtaining correctly rounded functions, this is around 2^{-120}. This means that a regular global optimization algorithm would have to subdivide the initial interval into an unfeasible number of small intervals (for example, if the initial interval is $[0, 1]$, 2^{120} intervals have to be obtained) before the actual algorithm is able to suppress some that do not contain the global optima.

In conclusion, rigorous global optimization algorithms based *only* on recursive interval subdivision and interval evaluation for functions are not suitable in our case.

2.4 Methods that Evade the Dependency Phenomenon

In order to bypass the high dependency problem present in $\varepsilon = p - f$, we have to write $p - f$ differently, so that we can avoid the decorrelation between f and p. For this, one widespread technique in the literature [11, 20, 21, 14], firstly informally proposed by David Wheeler, consists in replacing the function f by another polynomial T that approximates it and for which a bound on the remainder $f - T$ is not too difficult to obtain. While choosing the new approximation polynomial T, we have several advantages.

First, we can consider particular polynomials for which the remainder bound is known or it is not to difficult to compute. One such polynomial approximation can be obtained by a Taylor series expansion of f, supposing that it is differentiable up to a sufficient order. If f behaves sufficiently well, this eventually gives a good enough approximation of f to be used instead of f in the supremum norm.

Second, we remove the decorrelation effect between f and p, by transforming it into a cancellation phenomenon between the coefficients of T and those of p. As mentioned above, increasing the precision used is sufficient for dealing with such a cancellation [14].

As will be shown in Section 3.2, the error between T and f is actually not a real issue in terms of accuracy.

Consequently, in the case of absolute errors, we have:

$$\|\varepsilon\|_\infty = \|p - f\|_\infty \leq \|p - T\|_\infty + \|T - f\|_\infty \tag{1}$$

Hence, our problem reduces to:

(1) Finding an easily computable polynomial T. This is well handled in general. Some techniques for obtaining T are detailed in Section 4.

(2) Obtaining a bound Δ for the difference between f and T. Note that we have to ensure that the remainder bound is smaller than $\|\varepsilon\|_\infty$. Roughly speaking, for a smaller remainder, $\|\varepsilon\|_\infty$ can be obtained more accurately.

(3) Obtaining a tight bound for the polynomial difference $T - p$.

Existent works that follow these three steps in trying to solve our problem are detailed below.

On the one hand, a bound Δ for the remainder between f and T is obtained by ad-hoc manual techniques, for particular functions in [10, 11]. Since this process is not automated in these algorithms, we won't detail them further.

On the other hand, there exist approaches that use Taylor forms [20, 21]. Generally speaking, they represent an automatic way of providing both a Taylor polynomial T and a bound for the remainder between f and T. Firstly, in [14] automatic differentiation is used both for computing the coefficients of T and for obtaining a bound Δ for the Taylor remainder, using Lagrange's formula for the remainder. This technique for obtaining the couple (T, Δ) was available since Moore [15]. In [14] it is then used as a basic brick in a more complex algorithm for computing supremum norms of error functions, sketched below.

The central idea of this algorithm is to safely and tightly enclose the zeros of the derivative of the error ε'. One auxiliary function τ is used for this purpose: $\tau = p' - f'$ for absolute error or $\tau = p'f - f'p$ for relative error. We know that all zeros of ε' are zeros of τ. Then τ is replaced 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. A technique for finding polynomial real roots is then employed for obtaining enclosures of zeros of τ based on zeros of the translated polynomials $T + \theta$ and $T - \theta$.

This algorithm is able to handle complicated examples quickly and accurately. However, two limitations were spotted by the authors. One is that there is no control of the accuracy obtained for the supremum norm. In fact, the algorithm uses a polynomial T, of a heuristically determined degree, such that the remainder bound obtained is less than the "hidden" parameter θ, which is "a priori" set heuristically by the user. This signifies in fact, that the approach is not completely automatic.

Another limitation of this algorithm is that no formal proof is provided. This is due mainly to the combination of several techniques that are not currently available in formal proof checkers, like: techniques for surely isolating the zeros of a polynomial or automatic differentiation.

Another way of computing automatically a Taylor form was introduced by Berz and his group [21] under the name of "Taylor models" and used for problems that seemed intractable by Interval Arithmetic. Taylor Models, as defined there, use floating-point coefficients for

the polynomials and all the rounding errors are taken into consideration by appropriately adjusting the remainder.

Although Taylor models proved to have many applications, the available software implementations are scarce. The best known is COSY [22], written in FORTRAN by Berz and his group. Although highly optimized and used for rigorous global optimization problems in [23, 24] and articles referenced therein, currently, for our specific problem, COSY has two major drawbacks. First, it does not provide multiple precision arithmetic, and thus fails to solve the cancellation problem mentioned. Second, it does not deal with the problem of functions that have removable discontinuities. More specifically, for computing a Taylor model for the division of two functions f and g, we need to compute a Taylor model for the function $1/g$. In this case, a removable discontinuity x_0 of f/g is a point where $1/g$ is not defined. So, the straightforward computation of a Taylor model for $1/g$ over any interval that contains x_0 will fail to provide a finite bound for the remainder.

3 Computing a safe and guaranteed supremum norm

3.1 Computing a validated supremum norm vs. validating a computed supremum norm

Previous approaches: computing a validated supremum norm As we have seen, the various previously proposed algorithms for supremum norm evaluation have the following points in common:

- A validated technique is used to compute an upper bound u for a given supremum norm problem $\|\varepsilon\|_\infty$ based on Interval Arithmetic. An enclosure property guarantees that each single step of the computation takes rounding and approximation errors into account. As a result of the composition of all validated steps, the algorithm ensures that $u \geq \|\varepsilon\|_\infty$.

- Along with that upper bound u, most techniques generate a lower bound ℓ on the supremum norm: $\ell \leq \|\varepsilon\|_\infty$. Especially for Interval Arithmetic based algorithms, such a lower bound essentially comes for free. Every quantity appearing in the algorithm is anyway bounded by such lower and upper bounds.

- From the bounds l and u we can safely determine how tightly $\|\varepsilon\|_\infty$ is bounded. In other words, as $\ell \leq \|\varepsilon\|_\infty \leq u$, the quantity $u/\ell - 1$ provides a (validated) bound on the relative error $\eta = \frac{u}{\|\varepsilon\|_\infty} - 1$ *of the computed supremum norm value* u. This can be used as an *a posteriori* check if the computed bound u satisfies given quality requirements on the supremum norm. If the supremum norm bound u does not fulfill the requirements, the computation can be repeated with changed parameters, such as working precision or degree of intermediate polynomial approximation, in the hope of obtaining a more tightly bounded result. This can be a challenging issue because the exact relationship between an algorithm's input parameters and result tightness are often unknown [12, 14].

Let us stress again one fact about the previous techniques: the computation of the very digits of the supremum norm is intermingled with the validation of the fact that u is an upper bound. However, a safe, validated technique may actually be quite inefficient for *computing* these digits.

A good lower bound corresponds to the supremum norm value Our algorithm for validated supremum norms $\|\varepsilon\|_\infty$ is based on a novel approach. It will not compute a result based purely on self-validating arithmetic. It will start with a numerically computed *potential* upper bound u and safely validate that it is *indeed* an upper bound.

As already discussed in Section 2.1, numerical algorithms exist for computing the digits of a supremum norm's value in a way lowering the computational effort. This permits us to state the following key facts for our algorithm:

- Though using a numerical algorithm does not guarantee a validated upper bound, it is possible to guarantee that ℓ is truly a lower bound for $\|\varepsilon\|_\infty$:

$$\ell \leq \|\varepsilon\|_\infty .$$

  This is important in order to be able to make valid assertions about η:

$$\eta = \frac{u}{\|\varepsilon\|_\infty} - 1 \leq \frac{u}{\ell} - 1.$$

- The lower bound ℓ can be made as close as necessary to the actual value of the supremum norm, even though its actual accuracy with respect to the true value of $\|\varepsilon\|_\infty$ cannot be guaranteed. We hence assume that for any given $\overline{\eta_1} \in \mathbb{R}^+$, a value ℓ can numerically be computed such that there is reason to believe that $\ell = \|\varepsilon\|_\infty / (1 + \eta_1)$, where $0 < \eta_1 \leq \overline{\eta_1}$.

- With ℓ the very digits of the supremum norm are already known – at least up to some extent. That extent corresponds to $\overline{\eta_1}$.

Computing a potential upper bound with *a priori* error Suppose henceforth that ℓ is a sharp lower bound to $\|\varepsilon\|_\infty$ such that there is reason to believe that its distance from $\|\varepsilon\|_\infty$ is no greater than $\ell\,\overline{\eta_1}$. Consequently, it is possible to compute, out of ℓ and $\overline{\eta_1}$, a *potential upper* bound m for the supremum norm. It suffices to add that maximal distance $\ell\,\overline{\eta_1}$; see Figure 2 for illustration. Let

$$m = \ell\,(1 + \overline{\eta_1}) \tag{2}$$

.

So there is as much reason to believe that m is an upper bound for $\|\varepsilon\|_\infty$ as there is reason to believe that ℓ is an approximation of the supremum norm value with a relative error no greater than $\overline{\eta_1}$. We remark that m may actually be unreasonably tight to the supremum norm value. As a matter of fact, the lower bound ℓ and the potential upper bound $m = \ell\,(1 + \overline{\eta_1})$ are both pretty accurate approximations to the supremum norm, with a relative error not larger than $\overline{\eta_1}$.

In contrast, even though the *potential* upper bound m may be pretty accurate, it is often not required that the final, *validated* upper bound u be very accurate [3]. So it is reasonable to inflate the potential upper bound m. Take for example $u = m\,(1 + \overline{\eta_2})$ for some positive $\overline{\eta_2}$. As m is a potential upper bound, u is also a (potential) upper bound. However, the headroom $m\,\overline{\eta_2}$ between m and u makes it easier to *prove* that u actually is an upper bound.

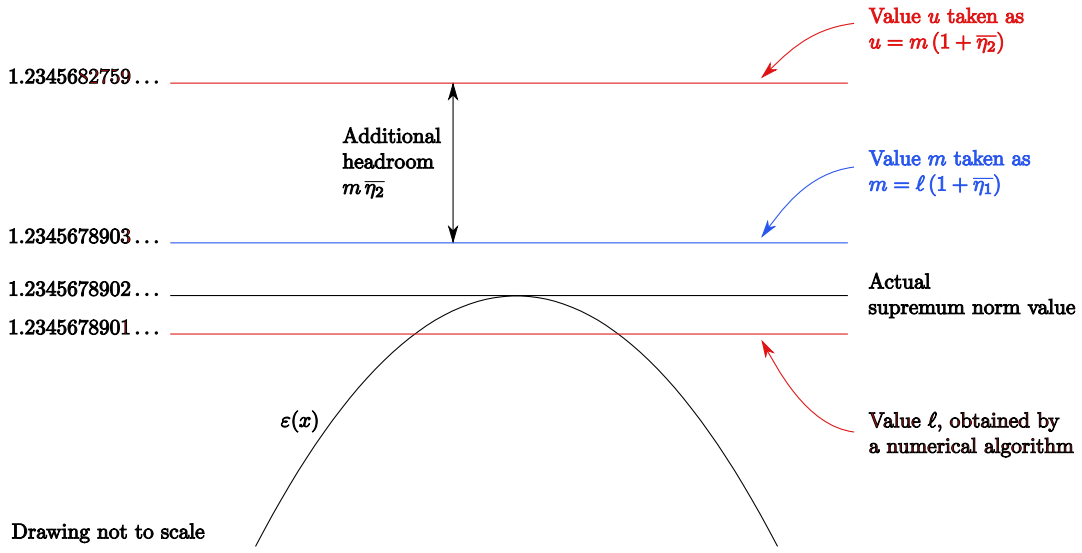

Figure 2: Lower and upper bounds

However, if $\overline{\eta_2}$ is kept reasonably small, u will also correspond, in its leading digits, to the supremum norm value $\|\varepsilon\|_\infty$. Precisely, we have

$$\eta = \frac{u}{\|\varepsilon\|_\infty} - 1 \leq \frac{u}{\ell} - 1 = \frac{\ell\,(1+\overline{\eta_1})\,(1+\overline{\eta_2})}{\ell} - 1 = \overline{\eta_1} + \overline{\eta_2} + \overline{\eta_1}\,\overline{\eta_2}. \tag{3}$$

The question that remains is how to choose $\overline{\eta_2}$ in order to keep η smaller than a bound $\overline{\eta}$, representing the maximal relative error that would be tolerated for a supremum norm (upper bound) result. Neglecting the quadratic term, we discover from equation (3) above that $\overline{\eta_1}$ and $\overline{\eta_2}$ sum up in their contribution to the final error η. Since it is desirable to have as much headroom for the validation as possible and since the use of most numerical algorithms permits one to make $\overline{\eta_1}$ pretty small, we could choose for example

$$\begin{aligned} \overline{\eta_1} &= {}^{1}\!/_{32}\,\overline{\eta} \\ \overline{\eta_2} &= {}^{30}\!/_{32}\,\overline{\eta}. \end{aligned}$$

We would obtain that

$$\eta \leq {}^{1}\!/_{32}\,\overline{\eta} + {}^{30}\!/_{32}\,\overline{\eta} + {}^{1}\!/_{32}\,\overline{\eta} \cdot {}^{30}\!/_{32}\,\overline{\eta} \leq \overline{\eta}$$

(with the easy assumption that $\overline{\eta} \leq 1$).

More formally, we can choose $\overline{\eta_2} = \beta_2\,\overline{\eta}$ and then set $\overline{\eta_1} = \beta_1\,\overline{\eta}$ with

$$\beta_1 = \frac{1-\beta_2}{1+\beta_2\,\overline{\eta}}. \tag{4}$$

We obtain in any case $\eta \leq \overline{\eta}$.

This means that it is easy to compute a potential upper bound u which, once we can validate it, will necessarily give a supremum norm approximation with the *a priori* error $\overline{\eta}$. That is a novelty of our approach. Previous methods could only give an *a posteriori* error bound.

Novel approach: validation of a computed result at algorithm's core This means that our algorithm separates the steps of computation and validation, contrary to previous approaches. We start with a numerically obtained result ℓ, determined with an *a priori* error. From ℓ we deduce a potential upper bound $u = \ell\,(1+\overline{\eta_1})\,(1+\overline{\eta_2})$. The core task of our algorithm is only to *validate* u as an upper bound, i.e. show that $\|\varepsilon\|_\infty \leq u$.

The following point is worth a remark: any *a priori* error bound $\overline{\eta}$ is by nature a bound on a *relative error* η. In contrast, all techniques for validation work with *absolute* error (or overestimation) quantities. This has been an issue in previous work; for example [14] suffers exactly from this issue. By taking $m\,\overline{\eta_2}$, we have actually transformed a relative error bound into an absolute quantity. This usefully guides the validation process, as discussed in the next Section. It enables the accuracy of intermediate polynomial approximations to be adapted, as will be described in Section 4.

3.2 Validating an upper bound on $\|\varepsilon\|_\infty$ for absolute error problems $\varepsilon = p - f$

Let us first consider the absolute error case. The relative error case is quite similar in nature but slightly more intricate technically. It will be described in Section 3.4.

We have $u = m\,(1+\overline{\eta_2})$. We seek to validate the assertion that u is an upper bound:

$$\|\varepsilon\|_\infty = \|p - f\|_\infty \leq m\,(1+\overline{\eta_2})\,. \tag{5}$$

As $\varepsilon = p - f$ contains the non-rational function f, this validation cannot be performed directly. What we can do is to use the triangle inequality

$$\|p - f\|_\infty \leq \|p - T\|_\infty + \|T - f\|_\infty \tag{6}$$

where we introduce a polynomial T that approximates f with an error $\delta = f - T$ that we can bound as $\|\delta\|_\infty = \|T - f\|_\infty \leq \overline{\delta}$ with some value $\overline{\delta}$. In Section 4 we will see how to implement a procedure `findMinimalPoly` that computes such a T, given f and $\overline{\delta}$.

The validation goal then transforms to showing that

$$\|p - T\|_\infty \leq b \tag{7}$$

where $b = m\;(1+\overline{\eta_2}) - \overline{\delta}$. Also there is reason to believe that:

$$\|p - T\|_\infty \leq \|p - f\|_\infty + \|f - T\|_\infty \leq m + \overline{\delta}. \tag{8}$$

So we only have reason to believe that the bound 7 is fulfilled if:

$$m + \overline{\delta} \leq m\;(1+\overline{\eta_2}) - \overline{\delta}. \tag{9}$$

This gives us a heuristic for choosing $\overline{\delta}$.

$$\overline{\delta} \leq m\,\frac{\overline{\eta_2}}{2}.$$

As a matter of course, to ease validation, we want $\overline{\delta}$ to be as large as possible. This means we must take $\overline{\delta} = m\,\frac{\overline{\eta_2}}{2}$.

Let us finally see how the final bounding validation of $p - T$ by b can be implemented.

- Both p and T are polynomials. This means that their difference polynomial $d = p - T$ can explicitly be expressed. This explicit subtraction – coefficient by coefficient – yields a cancellation on every power. In other words, the correlation between p and T, which we had already observed for p and f without being able to solve it, has successfully been transformed into a cancellation. It is hence often possible to show that $\forall x \in I, |d(x)| \leq b$ by classical rigorous global optimization techniques (see Section 2).

- The bound $\forall x \in I, |d(x)| \leq b$ can also be shown by proving that the two polynomials

$$s_1 = d + b \text{ and} \tag{10}$$
$$s_2 = -d + b \tag{11}$$

  are positive on the whole domain I under consideration.

  Such a positivity test can be implemented by rewriting it as a Sum-of-Squares or by exhibiting a point where s_i is positive and showing that it has no zero in I. The first technique proves to be perfectly adapted for certification through formal proofs. It will be explained in detail in Section 5. The second technique, establishing the absence of zeros, is easy to implement using traditional polynomial real roots counting techniques, such as Sturm sequences or the Descartes test [25]. Our current implementation computes a Sturm sequence.

3.3 The complete supremum norm algorithm for absolute error problems $\varepsilon = p - f$

When combining all results presented in Sections 3.1 and 3.2, we obtain the complete description of our supremum norm algorithm, as presented in Algorithm 3.1.

1 **Algorithm:** `supremumNormAbsolute`
Input: p, f, I, $\overline{\eta}$
Output: ℓ, u such that $\ell \leq \|p - f\|_\infty^I \leq u$ and $|u/\ell - 1| \leq \overline{\eta}$
2 $\beta_2 \leftarrow {}^{30}/_{32}$; $\beta_1 \leftarrow \frac{1-\beta_2}{1+\overline{\eta}\,\beta_2}$; $\overline{\eta_1} \leftarrow \beta_1\,\overline{\eta}$; $\overline{\eta_2} \leftarrow \beta_2\,\overline{\eta}$;
3 $\ell \leftarrow$ `computeLowerBound`$(p - f, \overline{\eta_1})$;
4 $m \leftarrow \ell\,(1 + \overline{\eta_1})$; $u \leftarrow m\,(1 + \overline{\eta_2})$; $\overline{\delta} \leftarrow m\,\frac{\overline{\eta_2}}{2}$;
5 $T \leftarrow$ `findMinimalPoly`$(f, I, \overline{\delta})$;
6 $b \leftarrow m\,(1 + \overline{\eta_2}) - \overline{\delta}$;
7 $s_1 \leftarrow p - T + b$; $s_2 \leftarrow -p + T + b$;
8 **if** `showPositivity`$(s_1, I) \wedge$ `showPositivity`(s_2, I) **then return** (ℓ, u);
9 **else return** $\perp$; `/* Numerically obtained bound` ℓ `not accurate enough */`

Algorithm 3.1: Complete supremum norm algorithm for $\|\varepsilon\|_\infty = \|p - f\|_\infty$

Actually, as indicated in the listing in Figure 3.1, our algorithm is not proven to produce a result in all cases. However, it will never lie, i.e. if it returns an interval $[\ell, u]$, this bound will safely contain the supremum norm and satisfy the quality requirement $\overline{\eta}$. The algorithm can fail in the case when the *belief* that the accuracy of the lower bound ℓ, computed by a numerical algorithm, was not actually verified by facts. Also, as explained in Section 4,

in some cases, if the input parameter $\overline{\delta}$ in the procedure `findMinimalPoly` is too small, this procedure fails to return an approximation polynomial T satisfying this requirement. However, we have conducted many challenging experiments on our algorithm in practice, and it never failed – see Section 6 for details.

3.4 Relative error problems $\varepsilon = p/f - 1$ without removable discontinuities

In Sections 3.2 and 3.3, we have seen how to handle problems of absolute approximation error, i.e. problems where $\varepsilon = p - f$. In practice it is often required to consider relative errors $\varepsilon = p/f - 1$. This is particularly important if f and p vary over I by several orders of magnitude.

In order to ease the issue with relative approximation error functions $\varepsilon = p/f - 1$, let us start with the assumption that f does not vanish in the interval I under consideration for $\|\varepsilon\|_\infty^I$. That assumption actually implies that $p/f - 1$ does not have any removable discontinuity. We will eliminate this initial assumption in Section 3.5.

When deriving an algorithm for the relative error case, we can work by analogy with the absolute error case. A key step in Section 3.2 consists in introducing a polynomial T approximating f such that its error $\delta = f - T$ be bounded by $\|\delta\|_\infty \leq \overline{\delta}$ and to express the absolute error $p - f$ as the sum of the absolute errors $p - T$ and $T - f$ in the triangle inequality (6). For the relative error $\varepsilon = p/f - 1$ a similar formula for error composition exists:

$$\begin{aligned}
\frac{p}{f} - 1 &= \frac{p-f}{f} \\
&= \frac{p-T}{T} + \frac{T-f}{f} + \frac{p-T}{T} \cdot \frac{T-f}{f} \\
&= \frac{p-T}{T} + \frac{T-f}{f}\left(1 + \frac{p-T}{T}\right) \\
&= \frac{p-T}{T} + (T-f)\,\frac{1}{f}\left(1 + \frac{p-T}{T}\right)
\end{aligned} \tag{12}$$

Assuming the following notation

$$\alpha = \frac{1}{f}\left(1 + \frac{p-T}{T}\right) \tag{13}$$

we obtain the following triangle inequality

$$\|\varepsilon\|_\infty \leq \left\|\frac{p}{T} - 1\right\|_\infty + \|\alpha\|_\infty \cdot \|T - f\|_\infty . \tag{14}$$

That triangle inequality closely resembles inequality (6) that we used for the absolute error:

$$\|p - f\|_\infty \leq \|p - T\|_\infty + \|T - f\|_\infty . \tag{6}$$

A technique for computing and validating a bound $\overline{\alpha}$ such that $\|\alpha\|_\infty \leq \overline{\alpha}$ will be presented below. Then we can adapt our supremum norm algorithm 3.1 as follows:

- Instead of taking $\overline{\delta} = m\frac{\overline{\eta_2}}{2}$, we take

$$\overline{\delta} = \frac{1}{\overline{\alpha}}\, m\, \frac{\overline{\eta_2}}{2}.$$

- Instead of setting the bound for the error between p and T to $b = m\ (1 + \overline{\eta_2}) - \overline{\delta}$, we compensate for the new factor in $\overline{\delta}$ and set the bound b to

$$b = m\ (1 + \overline{\eta_2}) - \overline{\alpha}\,\overline{\delta}.$$

  We remark that actually, $\overline{\alpha}$ cancels out completely and we still have:

$$b = m\left(1 + \frac{\overline{\eta_2}}{2}\right).$$

- We adapt the test validating the bound b for the error between p and T. It is now formulated to show that

$$\left\|{}^{p}/_{T} - 1\right\|_\infty \leq b.$$

  We do so by modifying the expressions defining s_1 and s_2.

- We add the computation and validation of $\overline{\alpha}$ to the algorithm. We remark that all the other previous modifications are pretty "cosmetic".

Just for completeness, if all validation steps pass, we still have

$$\begin{aligned} \|\varepsilon\|_\infty &\leq \left\|\frac{p}{T} - 1\right\|_\infty + \|\alpha\|_\infty \cdot \|T - f\|_\infty && (15)\\ &\leq m\ (1 + \overline{\eta_2}) - \overline{\alpha}\,\overline{\delta} + \overline{\alpha}\,\overline{\delta} \\ &= m\ (1 + \overline{\eta_2}) = u. && (16)\end{aligned}$$

So it merely remains to solve how $\overline{\alpha}$ can be determined and validated. We have

$$\alpha = \frac{1}{f}\left(1 + \frac{p - T}{T}\right).$$

Firstly, we remark that the term $\frac{p-T}{T} = \frac{p}{T} - 1$ also appears as the main term of the supremum norm result. A validated bound $b = m\left(1 + \frac{\overline{\eta_2}}{2}\right)$ is proved anyway. Hence we get the following bound for free:

$$\left\|1 + \frac{p - T}{T}\right\|_\infty \leq 1 + b. \tag{17}$$

Secondly, we remark that our assumption that f does not vanish on the domain I comes in handy: a finite bound can be obtained for $\left\|{}^{1}/_{f}\right\|_\infty$. We consider a simple interval evaluation $\boldsymbol{J}$ of f over I.

Frequently, since f does not vanish over I, we have $0 \notin \boldsymbol{J}$. Consequently, $\overline{F} = \frac{1}{\min(|\inf \boldsymbol{J}|, |\sup \boldsymbol{J}|)}$ yields a validated, finite upper bound for $\left\|{}^{1}/_{f}\right\|_\infty$. In practice, the accuracy of $\overline{F}$ with respect to $\left\|{}^{1}/_{f}\right\|_\infty$ is sufficient for our purpose. As a matter of fact, any overestimate will only lead to an overestimate of $\overline{\alpha}$. This purely has influence on $\overline{\delta} = \frac{1}{\overline{\alpha}}\, m\, \frac{\overline{\eta_2}}{2}$ forcing it to be smaller than it needs to be.This means the degree of the intermediate polynomial T will have to increase as a better accuracy in the approximation of f by T is required to satisfy a smaller $\overline{\delta}$. However, the accuracy of the supremum norm will not be impacted as expected for an *a priori* accuracy. Indeed, $\overline{\alpha}$ completely cancels out in equation (16).

In the case when $\boldsymbol{J}$ actually contains zero, it suffices to split the interval $\boldsymbol{I}$. However we never observed this phenomenon in our experiments.

Combining this bound $\overline{F}$ for $\left\|1/f\right\|_\infty$ with the bound for $\left\|1 + \frac{p-T}{T}\right\|_\infty$ given by equation (17), we obtain an appropriate value for $\overline{\alpha}$:

$$\overline{\alpha} = \overline{F}\ (1 + b)\,. \tag{18}$$

As seen, under the hypothesis that f does not vanish over I, our supremum norm algorithm for relative approximation error functions $\varepsilon = p/f - 1$ is not much different from our algorithm for absolute approximation error functions. In the next Section 3.5, let us see how we can overcome that final issue which will allow f to vanish over I.

3.5 Handling removable discontinuities

Suppose the function f vanishes over I on some points z_i. Mathematically speaking this is not a problem for the approximation error function $\varepsilon = p/f - 1$. It may stay defined – and bounded – by *continuity*. As seen in the Introduction, the matter is not purely theoretical but quite common in practice.

As a matter of course, the supremum norm $\left\|\varepsilon\right\|_\infty$ of the approximation error function $\varepsilon = p/f - 1$ will only have a finite value if p vanishes also at the zeros z_i of f and if the zeros of p are of at least the same order. Otherwise ε is not bounded.

The following heuristic is quite common in manual supremum norm computations [12]. The technique consists in presuming that the supremum norm is actually finite and basing matters on the following observations:

- As p is a polynomial, it can have only a finite number of zeros z_i with orders k_i. As already stated, f can have only these zeros.
- Since p is supposedly an approximation of f used in some floating-point environment, the overall structure of that floating-point code will try to ensure that the zeros of f are actually floating-point numbers.

This means that it is possible to determine a list of s presumed floating-point zeros z_i of f and to transform the relative approximation error function ε as follows:

$$\begin{aligned}
\varepsilon(x) &= \frac{p(x)}{f(x)} - 1 \\
&= \frac{\frac{p(x)}{(x-z_0)^{k_0} \ldots (x-z_{s-1})^{k_{s-1}}}}{\frac{f(x)}{(x-z_0)^{k_0} \ldots (x-z_{s-1})^{k_{s-1}}}} - 1 \\
&= \frac{q(x)}{g(x)} - 1.
\end{aligned}$$

In this transformation, two types of rewritings are used:

- The division of p by $(x - z_0)^{k_0} \ldots (x - z_{s-1})^{k_{s-1}}$ is performed by long division using exact, rational arithmetic. The remainder's being zero indicates whether the presumed zeros of f are actual zeros of p, as expected. If the remainder is not zero, the heuristic fails; otherwise q is a polynomial.
- The division of f by $(x - z_0)^{k_0} \ldots (x - z_{s-1})^{k_{s-1}}$ is performed symbolically, i.e. an expression representing g is constructed.

With the transformation performed, q is a polynomial and g is presumably a function not vanishing on I. These new functions are hence fed into the algorithm for supremum norms on relative error functions presented in Section 3.4. Even though the function g might actually still vanish on I as the zeros of f are only numerically determined, the supremum norm result will be safe. In the case when g does vanish, the algorithm presented in Section 3.4 will not be able to determine a safe and validated *finite* bound for $\|1/g\|_\infty$. It will eventually fail, returning a non-finite bounding for the supremum norm. This indicates the limits of our heuristic which, in practice, just works fine. See Section 6 for details on examples stemming from practice.

We remark that the heuristic algorithm just discussed actually only moves the problem elsewhere: into the function f for which an intermediate polynomial T must eventually be computed. This is not an issue. The expression defining f may anyway have removable discontinuities. This can even happen for supremum norms of absolute approximation error functions. An example would be $f(x) = \frac{\sin x}{\log(1+x)}$ approximated by $p(x) = 1 + 1/2 \cdot x$ on an interval I containing 0. We will address in Section 4.4 the problem of computing an intermediate approximation polynomial T with a finite bound $\overline{\delta}$ when the expression of f contains a removable discontinuity.

4 Obtaining the intermediate polynomial T and its remainder

Let us recall the terms of our problem: we are given a function f, a polynomial p, an interval I and a quality bound $\overline{\eta}$ and we want to automatically compute two values ℓ and u such that

$$\begin{cases} \|\varepsilon\|_\infty^I \in [\ell, u] \\ \frac{u-\ell}{\ell} \leq \overline{\eta}, \end{cases}$$

where $\varepsilon = p - f$ or $\varepsilon = p/f - 1$. As we have seen in Algorithm 3.1, this goal is achievable provided that we make use of a procedure `findMinimalPoly` satisfying the following specification:

$$\texttt{findMinimalPoly} : (f, I, \overline{\delta}) \mapsto T \quad \text{such that we are sure that } \|T - f\|_\infty \leq \overline{\delta}.$$

We shall now examine how this procedure can be implemented.

Let us fix some notations: in the following n denotes the degree of the intermediate polynomial T. We denote by R the remainder $f - T$ and by $\mathbf{\Delta}$ an interval enclosure of $R(I)$: $\mathbf{\Delta} \supseteq R(I)$. In practice, all the algorithms we are going to describe take the form of a procedure `findPoly` where n is an input parameter:

$$\texttt{findPoly} : (f, I, n) \mapsto (T, \mathbf{\Delta}) \quad \text{such that } \deg(T) = n \text{ and } R(I) \subseteq \mathbf{\Delta}.$$

Hence, the procedure `findMinimalPoly` needs somehow to iterate over n: a simple bisection strategy like the one used in `findMinimalPoly` (see Algorithm 4.1) suffices in practice.

It is important to note that the polynomial T will eventually be used to prove polynomial inequalities in a formal proof checker. In such an environment, the cost of computations is highly related to the degree of the polynomials involved. So we want the degree of T to be as small as possible. In that respect, the choice of the algorithm used for implementing `findPoly` is crucial: indeed due to several sources of overestimation, the remainder bound $\mathbf{\Delta}$ given by `findPoly` will often be a crude overestimation of the actual remainder image $R(I)$.

In practice, this overestimation will lead us to overestimate the necessary degree for T. Hence, we would like to choose an algorithm that minimizes this overestimation phenomenon.

For all the methods presented in what follows, it is possible to find situations where the remainder is significantly overestimated. The overestimation depends on the function f, the interval I, and sometimes on internal parameters of the algorithm. From our experiments, there is no algorithm clearly dominating all the others. It is worth mentioning that Taylor models often give relatively accurate remainders, so they might be preferred in practice. This is just a remark based on a few experiments and it probably should not be considered as a universal statement.

```
1  Algorithm: findMinimalPoly
   Input: f, I, δ̄
   Output: T such that ‖T − f‖_∞^I ≤ δ̄ while trying to minimize the degree of T
2  n ← 1;
3  do
4  |  (T, δ_try) ← findPoly(f, I, n);
5  |  n ← 2n;
6  while δ_try > δ̄ ;
7  n_min ← n/2 ;                          /* we know that n_min is too small */
8  n_max ← n ;                            /* we know that n_max achieves δ̄ */
9  while n_min + 1 < n_max do
10 |  n ← ⌊(n_min + n_max)/2⌋;
11 |  (T, δ_try) ← findPoly(f, I, n);
12 |  if δ_try ≤ δ̄ then n_max ← n else n_min ← n ;
13 end
14 (T, δ_try) ← findPoly(f, I, n_max);
15 return T ;
```

Algorithm 4.1: Bisection over n for finding a polynomial T such that $\|T - f\|_\infty \leq \overline{\delta}$ making n as small as possible.

In the following, we describe available methods on which the implementation of `findPoly` can be based. We only present the main ideas of the methods and do not get into all the formal details. In particular, we suppose that the arithmetic operations ($+$, $-$, $\times$, $\div$) can be performed exactly. Of course, in practice this is not the case and technical work is needed to guarantee safety while using floating-point arithmetic. Though technical, these details are not difficult to settle in general. It is not the purpose of this article to enter these details.

We distinguish two families of methods:

- methods that compute first the polynomial T and that bound the remainder afterwards. These methods will be described in Section 4.1;
- methods that simultaneously compute both T and a bound on the remainder. These methods are grouped under the name of *Taylor models*. We will describe them in Section 4.2

4.1 Computing T before bounding the remainder

4.1.1 Using an interpolation polynomial

For the polynomial T, we can choose an interpolation polynomial: if the interpolation points are well-chosen, such a polynomial can usually provide a near-optimal approximation of f. The advantage of interpolation polynomials is that an explicit bound on the remainder exists [4]; namely, if T interpolates f at points $x_0, \ldots, x_n$, the remainder R satisfies

$$\exists\, \xi \in I,\ R(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \tag{19}$$

where $f^{(n+1)}$ denotes the $(n+1)$-st derivative of f.

For the interpolation points, it is usually advisable to take the roots of the $(n+1)$-st Chebyshev polynomial:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(n-i+1/2)\,\pi}{n+1}\right) \qquad \text{where } I = [a, b]. \tag{20}$$

Indeed, this is the set of points that minimizes $\max\{\prod_{i=0}^{n} |x - x_i|,\ x \in I\}$.

The interpolation polynomial itself is easy to compute (we refer the reader to any book on numerical analysis for a review of existing techniques). For bounding the remainder, the only difficulty is to bound the term $f^{(n+1)}(\xi)$ for $\xi \in I$. This can be achieved using a technique often called *automatic differentiation* [26, 27, 15]*, *differentiation arithmetic* [27], *algorithmic differentiation* [28] or *Taylor arithmetic* [26].

Automatic differentiation allows for evaluating the first n derivatives of f in a point x_0 without doing any symbolic differentiation. For bounding the $f^{(n+1)}(\xi)$ when $\xi \in I$, it suffices to apply the same algorithm using Interval Arithmetic, replacing x_0 by I.

The general idea is to replace any function g by the array $G = [g_0, \ldots, g_n]$ where $g_i = g^{(i)}(x_0)/i!$. It is easy to see that, given two arrays U and V (corresponding to two functions u and v), the array corresponding to $w = u + v$ is simply given by $\forall i, w_i = u_i + v_i$. It is also easy to see (using Leibniz formula) that the array corresponding to $w = uv$ is given by

$$\forall i, w_i = \sum_{k=0}^{i} u_k\, v_{i-k}.$$

There exist formulæ for computing W from U where $w = \exp(u)$, $w = \sin(u)$, $w = \arctan(u)$, etc. See [15, 26]. More generally, if one knows the array U formed by the values $u^{(i)}(x_0)/i!$ and the array V formed by the values $v^{(i)}(y_0)/i!$ where $y_0 = u(x_0)$, it is possible to compute from U and V the array corresponding to $w = (v \circ u)$ in x_0. Hence, given any expression `expr` representing a function f, a straightforward recursive procedure computes the array F:

- if `expr` is a constant c, return $[c, 0, 0, \ldots]$;
- if `expr` is the variable, return $[x_0, 1, 0, \ldots]$;
- if `expr` is of the form `expr1 + expr2`, compute recursively the arrays U and V corresponding to `expr1` and `expr2` and returns $[u_0 + v_0, \ldots, u_n + v_n]$;

*More precisely, we should say that it is inspired by automatic differentiation, since automatic differentiation is usually a code-transformation process, intended to deal with functions of several variables.

- etc.

Indeed, manipulating these arrays is nothing but manipulating truncated formal series. There exist fast algorithms for multiplying, composing or inverting formal series [29, 30]. For instance, computing the first n terms of the product of two series can be performed in $\mathcal{O}(n\,\log(n))$ operations only (instead of the $\mathcal{O}(n^2)$ operations required when using Leibniz formula). These sophisticated techniques are not mandatory for our purpose since we intend to deal with degrees n not larger than a few hundreds.

4.1.2 Advantage and drawback of interpolation polynomials

As we mentioned, using an interpolation polynomial for T has an interesting advantage: T is often a near-best approximation polynomial. So, if one is looking for a polynomial T of minimal degree achieving $\|T - f\|_\infty \leq \overline{\delta}$, it is often the case that an interpolation polynomial is almost the best we can expect. However, this advantage can seriously be damaged by the overestimation of the remainder.

Actually, evaluating an expression by Interval Arithmetic usually leads to overestimatation of the result. This is the *dependency phenomenon* that we already described in Section 2.3. This phenomenon is especially notable when the intervals manipulated are quite large and when the same variable occurs many times in the expression. Of course, this phenomenon appears when we use automatic differentiation with Interval Arithmetic. The resulting interval can be so greatly overestimated that we lose all the benefit of using an interpolation polynomial.

4.1.3 Using a Taylor polynomial

The previous remarks suggest that it could be interesting to use a polynomial T with worse approximation quality but for which the remainder is less overestimated. Consider a point $x_0 \in I$. We suppose that for each x in I, $f(x)$ is the sum of its Taylor series:

$$\forall x \in I,\ f(x) = \underbrace{\left(\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}\,(x-x_0)^i\right)}_{T(x)} + \underbrace{\left(\sum_{i=n+1}^{+\infty} \frac{f^{(i)}(x_0)}{i!}\,(x-x_0)^i\right)}_{R(x)}. \tag{21}$$

Technically, this means that the function is *analytic* on a complex disc $\mathcal{D}$ containing I and centered on x_0 in the complex plane. It is not necessary to know complex analysis to understand the general ideas of what follows. The reader interested by this subject is advised to consult a reference book (e.g. [31]) for a general background on complex analysis.

The functions of interest for us are generally analytic on the whole complex plane, except maybe in a given list of points (called the *singularities* of the function). Most of the time, in practice, the singularities are far enough from the interval for the hypothesis to hold. When singularities are too close to I, it is not possible to find a disc containing I avoiding singularities. However, even in this case, it is possible to cover I with several small discs $\mathcal{D}_1, \ldots, \mathcal{D}_k$ satisfying the property (see figure 3 for an illustration). Then it suffices to reason on each $\mathcal{D}_k$ separately.

Computing T itself is easy: the coefficients can efficiently be computed using automatic differentiation. The only problem is to bound the remainder R. A first idea could be to use

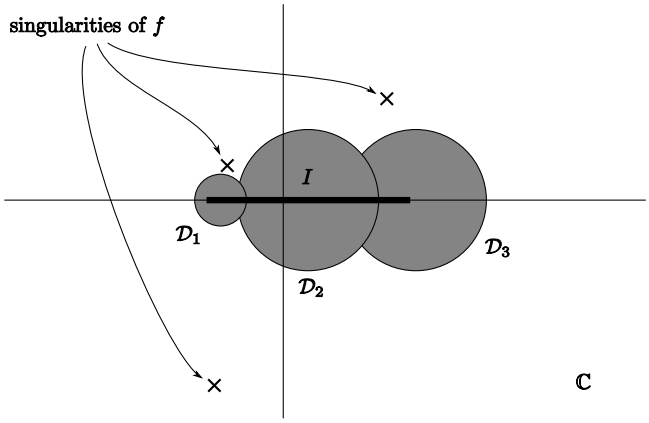

Figure 3: When singularities are too close to I, the interval can be covered by discs that avoid the singularities.

the Lagrange formula

$$\exists\, \xi \in I,\ R(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,(x - x_0)^{n+1}. \tag{22}$$

However, this formula requires to bound the same quantity $f^{(n+1)}(I)$ that we already encountered when bounding the remainder of an interpolation polynomial. Hence, it will lead to exactly the same problem of overestimation as before, but without the advantages exhibited by the interpolation polynomials.

Actually, in equation (21), R is expressed as a series. This expression turns out to give an accurate way of bounding R. Suppose for instance that we can find values M and d such that we can prove that

$$\forall i > n,\ \left|\frac{f^{(i)}(x_0)}{i!}\right| \leq \frac{M}{d^i} := b_i. \tag{23}$$

Thus, we can obviously bound R the following way:

$$\forall x \in I,\quad |R(x)| = \left|\sum_{i=n+1}^{+\infty} \frac{f^{(i)}(x_0)}{i!}\,(x - x_0)^i\right| \leq \sum_{i=n+1}^{+\infty} b_i\ |x - x_0|^i\,. \tag{24}$$

The series $\sum_{i=n+1}^{+\infty} b_i\,|x - x_0|^i$ is called a *majorizing series* of R. Since it is a geometric series, its sum is easy to bound uniformly on I:

$$\text{if } \gamma := \max_{x\in I} \frac{|x - x_0|}{d} < 1, \qquad \text{it holds that} \quad \forall x \in I,\ |R(x)| \leq \frac{M\,\gamma^{n+1}}{1-\gamma}.$$

Of course the principle of majorizing series is not limited to geometric series: b_i can take other forms than M/d^i, provided that the series $\sum_{i=n+1}^{+\infty} b_i\,|x - x_0|^i$ can easily be bounded. Neher and Eble proposed a software tool called ACETAF [32] for automatically computing suitable majorizing series. ACETAF uses different techniques, all based on a theorem of complex analysis called *Cauchy's estimate*. We refer to [32] for the details on this subject.

As an alternative to methods based on Cauchy's estimate, it is sometimes possible to compute a majorizing series by using direct information on the values $(f^{(i)}(x_0)/i!)$. For

instance, if f is solution of a linear differential equation with polynomial coefficients, the sequence $(c_i)_{i\in\mathbb{N}}$ defined by $c_i = f^{(i)}(x_0)/i!$ satisfies a finite recurrence: there exists an integer r such that, for all i, c_{i+r} is completely determined by the values $c_i, \ldots, c_{i+r-1}$. Such a function is called *differentially finite* [33] (or also *D-finite* or *holonomic*). As a direct consequence, the whole sequence $(c_i)_{i\in\mathbb{N}}$ depends only on the first r terms $c_0, \ldots, c_{r-1}$. Hence, in principle, the behavior of the whole sequence can be known just by studying the r first values. This principle can be made rigorous and can be transformed into an algorithm: Mezzarobba and Salvy show in [33] how to compute an accurate majorizing series of a D-finite function f and deduce from it an accurate bound on the remainder.

4.1.4 Advantage and drawback of Taylor polynomials

Given a Taylor polynomial T, we experimentally observed that the bound $\mathbf{\Delta}$ given by ACETAF is generally a fairly tight overestimation of the actual remainder image $R(I)$. Consider the following situation: given a degree n, we denote by T_{taylor} a Taylor polynomial of degree n, R_{taylor} its remainder and $\mathbf{\Delta}_{\text{taylor}}$ an enclosure computed with help of ACETAF. Accordingly, we denote by $T_{\text{interpolate}}$ and $R_{\text{interpolate}}$ an interpolation polynomial of the same degree n and its remainder. We denote by $\mathbf{\Delta}_{\text{interpolate}}$ an enclosure computed by automatic differentiation as explained in Section 4.1.1. In general, it holds that $R_{\text{interpolate}}(I) \ll R_{\text{taylor}}(I)$. However since the overestimation is much smaller with ACETAF than with automatic differentiation, we frequently observed the opposite for the bounds: $\mathbf{\Delta}_{\text{taylor}} \ll \mathbf{\Delta}_{\text{interpolate}}$.

However, the methods used in ACETAF are currently not completely satisfying for our purpose. The main problem comes from the fact that the algorithms based on Cauchy's estimate depend on many parameters. Choosing the parameters is currently done by hand and is mainly a matter of feeling. If the parameters are badly chosen, very poor bounds are produced. Though promising, the method cannot straightforwardly be used for designing a completely automatic algorithm. Furthermore, these techniques require that the function is analytic on a complex disc. As we already said it, this hypothesis can be considered as practically always true. However, in our context it is not sufficient to *believe* that the function satisfies this hypothesis, but we will actually have to *prove* it automatically. This could be an issue.

On the other hand, the method by Mezzarobba and Salvy seems to give fairly accurate bounds as well. The problem here comes from the limitation to D-finite functions. Indeed, many elementary or special functions are D-finite (exp, sin, arctan, Bessel functions, etc. are). Besides, the set of D-finite functions has strong closure properties. However, not all commonly used functions are D-finite: for instance tan is not. More generally, the set of D-finite functions it is not closed under composition, which is a serious limitation for our purpose where we would like to consider any function given defined by an expression. This limitation is even more an issue if an automatic approach is required.

4.2 Simultaneously computing T and Δ

The methods described above for implementing the procedure `findPoly` are all composed of two separate steps:

1. Compute a polynomial T (Taylor or interpolation polynomial) approximating f;
2. Bound the remainder $R = f - T$.

Another possibility consists of computing both the polynomial and a bound on the remainder simultaneously. Such a technique has been introduced by Berz and Makino [34] under the name of *Taylor model*. A good introduction to this subject is [21]. As the name indicates, Taylor Models have been designed to compute Taylor polynomials. Their advantage is that the remainder is automatically and systematically computed. The method applies to any function given by an expression; there is no parameter to manually adjust.

There is a major difference with the methods of the previous section, concerning the hypothesis of analyticity of f. Of course, since Taylor models compute a Taylor polynomial, the method gives interesting results only if f is analytic on a complex disc containing I. However, there is no need to check the analyticity beforehand: if f is not analytic, the method will find a polynomial and a valid bound. This bound will just be very much overestimated. In contrast, the methods described in the previous section were *based* on the fact that the series converges to f: hence for using them it was *necessary* to check the analyticity beforehand. We stress it again: in general f *is* analytic (or it suffices to split the interval into several ones for this condition to be fulfilled). But automatically proving it may be challenging.

4.2.1 Informal description of Taylor models

Taylor models are inspired by automatic differentiation. As we have seen, automatic differentiation permits us to compute the first n derivatives of a function by applying simple rules recursively on the structure of f. Following the same idea, Taylor models compute a polynomial together with a bound by applying simple rules recursively on the structure of f. However, the bound computed with Taylor models is usually much tighter than the one obtained by automatic differentiation. In order to understand this phenomenon, consider the Taylor development of a composite function $v \circ u$ (we use the Lagrange formula that we already recalled in equation (22)):

$$(v \circ u)(x) = \left(\sum_{i=0}^{n} \frac{(v \circ u)^{(i)}(x_0)}{i!} (x - x_0)^i \right) + \frac{(v \circ u)^{(n+1)}(\xi)}{(n+1)!} (x - x_0)^{n+1}.$$

When bounding the remainder by means of automatic differentiation, an interval $\boldsymbol{J}$ enclosing $(v \circ u)^{(n+1)}(I)/(n+1)!$ is obtained by performing many operations involving enclosures of $u^{(i)}(I)/i!$ and $v^{(i)}(I)/i!$. These enclosures themselves are obtained by recursive calls. Due to the dependency phenomenon, these values are already overestimated and this overestimation increases at each step of the recursion.

In contrast, we can consider $(v \circ u)(x)$ as function v evaluated at point $u(x)$. Its Taylor development in point $u(x_0)$ is

$$(v \circ u)(x) = \left(\sum_{i=0}^{n} \frac{v^{(i)}(u(x_0))}{i!} (u(x) - u(x_0))^i \right) + \frac{v^{(n+1)}(u(\xi))}{(n+1)!} (u(x) - u(x_0))^{n+1}. \tag{25}$$

In this formula, the only derivatives involved are the derivatives of v which is a basic function (such as exp, sin, arctan, etc.). Fairly simple formulæ exist for the derivatives of such functions and evaluating them by Interval Arithmetic does not lead to serious overestimation.

There are two questions to solve when using formula (25): how to extract from it a polynomial and how to bound the remainder. Suppose that a recursive call gives us a polynomial $T_u \simeq u$ together with a bound on the remainder $\boldsymbol{\Delta_u}$. Since T_u is a polynomial, many methods exist [35] for efficiently computing an accurate enclosure $\boldsymbol{J}$ of $T_u(I)$.

Reasoning informally, the polynomial part of $v \circ u$ is simply obtained by replacing $u(x)$ by $T_u(x)$ in the sum of equation (25). In fact only the coefficients of order 0 to n are computed. The contribution of other coefficients is rigorously bounded and will eventually be included in the remainder bound of the Taylor model. Of course, u is not exactly equal to T_u: their difference must be taken into account when replacing u by T_u. We do not explain these technical details: a wide literature exists on the subject and we refer the reader to e.g. [21] or [36, Chapter 5.1] for proofs and details.

For bounding the remainder of formula (25), it suffices to bound the terms $u(x)$ and $u(\xi)$ that appear in it, i.e. compute an enclosure of $u(I)$. This enclosure is simply given by

$$u(I) \subseteq \boldsymbol{J} + \boldsymbol{\Delta_u} =: \boldsymbol{K}.$$

If $\boldsymbol{\Delta_u}$ does not overestimate $(u - T_u)(I)$ too much, then $\boldsymbol{K}$ does not overestimate $u(I)$ too much. As we mentioned, when v is a basic function, an accurate enclosure for $v^{(n+1)}(\boldsymbol{K})$ can be computed. In conclusion, the overestimation does not grow too much during the recursion process.

4.2.2 Advantage and drawback of Taylor models

In the previous section we just gave a rough idea of Taylor models. In practice, several optimizations are possible and implementations vary in speed and quality depending on choices made for the implementation [21, 36]. It is also possible to use the technique of Taylor models for computing polynomials that are not Taylor polynomials [36].

However even a straightforward out-of-the box implementation of Taylor models gives satisfying results. In practice the bound computed with Taylor models is much less overestimated than a bound computed by automatic differentiation. The techniques presented in Section 4.1.3 give often a tighter estimation of the remainder than Taylor models but the gap is not very big in general. Moreover, as we mentioned, the Taylor models have the advantage of being completely automatic and not requiring us to check further assumptions (such as analyticity).

4.3 Practical comparison of the different methods

Table 4.3 shows the quality of some bounds obtained by the methods that we presented. Each row of the table represents one example. The function f, the interval I and the degree n of T are given in the first column. The interpolation was performed at Chebyshev points (see equation (20)) and the corresponding remainder R was bounded using automatic differentiation with Interval Arithmetic as explained in Section 4.1.1. It leads to an enclosure $\boldsymbol{\Delta} \subseteq R(I)$. The second column of the table is $\sup|\boldsymbol{\Delta}|$ while the third column is a numerical estimation of $\sup|R(I)|$.

The Taylor polynomial was developed in the midpoint of I. The resulting remainder R was bounded using the software tool ACETAF presented in Section 4.1.3. The result is related in the fourth column of the table. The sixth column of the table is a numerical estimation of $\sup|R(I)|$. The fifth column corresponds to the bound given by Taylor models.

Note that ACETAF actually proposes four different methods. All depend on one or several parameters. These parameters must be adjusted by hand, so it was impossible in practice to systematically test so many possibilities. In the table, we only show the results obtained with the first method of ACETAF (with this method, there is only one parameter to set). This

is somehow unfair since the other methods generally give more accurate bounds. However, as can be seen in Table 4.3, even this first simple method gives results often as accurate as Taylor models (and sometimes much better as in example 3).

We tried to make the examples representative of several situations. The first example is a basic function which is analytic on the whole complex plane. There is almost no overestimation in this case, whatever method we use. The second is also a basic function but it has singularities in the complex plane (in $\pm i$, where $i^2 = -1$). However, in this example, the interval I is relatively far from the singularities. All the methods present a relatively small overestimation. The third example is the same function but over a larger interval: so the singularities are closer and Taylor polynomials are not very good approximations. The fourth and fifth example are composite functions on fairly large intervals. This challenges the methods. One can see that the overestimation in the interpolation method becomes very large while it stays reasonable with ACETAF and Taylor models.

In each row, the method that leads to the tightest bound is written in bold. One can observe that there is no method better than the others in all circumstances. However, Taylor models seem to offer a good trade-off, in particular for composite functions.

$f(x)$, I, n	Interpolation	Exact bound	ACETAF	TM	Exact bound
$\sin(x)$, [3, 4], 10	**1.19e−14**	1.13e−14	6.55e−11	1.22e−11	1.16e−11
$\arctan(x)$, [−0.25, 0.25], 15	**7.89e−15**	7.95e−17	1.00e−9	2.58e−10	3.24e−12
$\arctan(x)$, [−0.9, 0.9], 15	**5.10e−3**	1.76e−8	6.46	1.67e2	5.70e−3
$\exp(1/\cos(x))$, [0, 1], 14	0.11	6.10e−7	9.17e−2	**9.06e−3**	2.59e−3
$\frac{\exp(x)}{\log(2+x)\cos(x)}$, [0, 1], 15	0.18	2.68e−9	1.76e−3	**1.18e−3**	3.38e−5

Table 1: Examples of bounds obtained by several methods

4.4 The problem of removable discontinuities

As explained in Section 3.5, it is very important in practice to correctly handle functions with removable discontinuities. More precisely, we have seen how to handle the computation of $\|p/f - 1\|_\infty$ when f vanishes in the interval. The expression $p/f - 1$ stays bounded in the interval only if f has a finite number of zeros $z_0, \ldots, z_{s-1}$ and if they are all zeros of p. Once these zeros have been numerically found, the problem of computing $\|p/f - 1\|_\infty$ is replaced by the equivalent problem of computing $\|q/g - 1\|_\infty$ where q is obtained by a long division performed exactly and g is an expression of the form

$$g(x) = \frac{f(x)}{(x - z_0)^{k_0} \cdots (x - z_{s-1})^{k_{s-1}}}.$$

So, the supremum norm algorithm will eventually call the procedure `findPoly` with g as argument.

The technique of Taylor models, as usually described, is not able to handle such functions g. The usual way of performing a division u/v is to compute a Taylor model for v, compute its inverse, and then multiply by a Taylor model of u. The inversion fails if v vanishes in the interval (more precisely, it does not fail but the computed remainder bound is infinite). We propose an adaptation of Taylor models that allows for performing such a division.

Consider an expression of the form u/v. We suppose that a numerical heuristic has already found a point z_0 where u and v seem to vanish simultaneously. If there are several zeros of

v in the interval, we split it into several ones. So, we will assume that z_0 is presumably the unique zero of v in the interval. There are two important remarks:

- We additionally assume that z_0 is exactly representable as a floating-point number. This hypothesis may seem restrictive at first sight. However this is usually the case in the practice of floating-point code development. Moreover, it is hopeless to handle non-representable singularities without introducing symbolic methods;
- The numerical procedure that found z_0 might have missed a zero of v. This is not a problem: in this case, our variant of Taylor models will simply return an infinite remainder bound the same way as the classical Taylor models do. We do not pretend to have a technique that computes models for every possible function u/v: we only try to improve the classical algorithms in order to handle the most common practical cases. In all other cases, our algorithm is permitted to return useless (though correct) infinite bounds.

Our method is based on what everyone would do manually in such a situation: develop both u and v in Taylor series with center z_0 and factor out the leading part $(z-z_0)^{k_0}$ in both expressions. Let us take an example: $v(x) = \log(1+x)$ vanishes at $z_0 = 0$ and its Taylor series with center z_0 is

$$\log(1+x) = 0 + \sum_{i=1}^{+\infty} (-1)^{i+1} \frac{(x-z_0)^i}{i}.$$

Note that the constant term is exactly 0: this is a direct consequence of the fact the $\log(1+x)$ vanishes at z_0. More generally, if z_0 is a zero of order k_0 of a function v, the first k_0 coefficients of the Taylor series of v are exactly zero. If we take for instance $u(x) = \sin(x)$, we have

$$\sin(x) = 0 + \sum_{i=0}^{+\infty} \frac{(-1)^i}{(2i+1)!} (x-z_0)^{2i+1}.$$

So, we can factor out $(x-z_0)$ both from the denominator and the numerator and hence obtain

$$\frac{\sin(x)}{\log(1+x)} = \frac{\displaystyle\sum_{i=0}^{+\infty} \frac{(-1)^i}{(2i+1)!} (x-z_0)^{2i}}{\displaystyle\sum_{i=1}^{+\infty} \frac{(-1)^{i+1}}{i} (x-z_0)^{i-1}}.$$

The denominator of this division does not vanish anymore, so this division can be performed using the usual division of Taylor models.

The first difficulty encountered when trying to automate this scheme is that we need to be sure that the leading coefficients of u and v are exactly zero: it is not sufficient to approximately compute them. A classical solution to this problem is to represent the coefficients by small enclosing intervals instead of floating-point numbers. Interval Arithmetic is used throughout the computations, which ensures that the true coefficients actually lie in the corresponding small intervals. For basic functions, we know which coefficients are exactly zero, so we can replace them by the point-interval [0, 0]. This point interval has a nice property: it propagates well during the computations, since any interval multiplied by [0, 0] leads to [0, 0]

itself. So, in practice, the property of *being exactly zero* is well propagated when composing functions: hence we can expect that for any reasonably simple functions u and v vanishing in z_0, the leading coefficients of their Taylor development will be [0, 0] from which we can surely deduce that the true coefficients are exactly 0.

The second difficulty is that, in fact, we do not compute series: we use Taylor models, i.e. truncated series together with a bound on the remainder. If we use classical Taylor models, the function u/v will be replaced by something of the form

$$\frac{T_u + \boldsymbol{\Delta_u}}{T_v + \boldsymbol{\Delta_v}}. \tag{26}$$

Hence, even if both T_u and T_v have a common factor $(x - z_0)^{k_0}$, we cannot divide both the numerator and denominator by $(x - z_0)^{k_0}$ because the division will not simplify in the remainder bounds.

The solution to this problem is easy: if T_u is the Taylor polynomial of degree n of a function u in z_0, we know that the remainder is of the form $R_u = (x - z_0)^{n+1}\,\widetilde{R_u}$. The usual Taylor models propagate an enclosure $\boldsymbol{\Delta_u}$ of $R_u(I)$ through the computation. Instead, we can propagate an enclosure $\boldsymbol{\Delta_u}$ of $\widetilde{R_u}(I)$. In this case, equation (27) becomes

$$\frac{T_u + (x - z_0)^{n+1}\,\boldsymbol{\Delta_u}}{T_v + (x - z_0)^{n+1}\,\boldsymbol{\Delta_v}}, \tag{27}$$

and it becomes possible to factor out the term $(x - z_0)^{k_0}$ from the numerator and the denominator.

This leads to the following definition. Note that in this definition, z_0 itself is replaced by a small interval enclosing it. This is convenient when composing our modified Taylor models. Of course, since we assume that z_0 is a floating-point number, we can replace it by the point-interval $[z_0, z_0]$.

Definition (modified Taylor models): *a modified Taylor model of degree n consists of:*

- *an interval (usually a point-interval) $\boldsymbol{z_0}$ representing the development point;*
- *a list of (usually small) interval coefficients $\boldsymbol{a_0}, \ldots, \boldsymbol{a_n}$;*
- *an interval $\boldsymbol{\Delta}$ representing a bound on the (scaled) remainder.*

A modified Taylor model represents *a function f over I when*

$$\forall \xi_0 \in \boldsymbol{z_0}, \exists \alpha_0 \in \boldsymbol{a_0}, \ldots, \exists \alpha_n \in \boldsymbol{a_n}, \forall x \in I,\ \exists \delta \in \boldsymbol{\Delta},\ f(x) = \left(\sum_{i=0}^{n} \alpha_i\,(x - \xi_0)^i\right) + (x - \xi_0)^{n+1}\,\delta.$$

All the classical operations on Taylor models (addition, multiplication, composition) translate easily to modified Taylor models. We can now modify the division rule, in order to handle correctly the case when the numerator and denominator both vanish at z_0. To obtain a modified Taylor model of degree n of u/v in $[z_0, z_0]$, we proceed in two steps:

- first, a numerical heuristic is applied to determine if both u and v vanish in z_0. If this is the case, the order k_0 of z_0 as a zero of v is determined;

- second, modified models of order $n + k_0$ of u and v in z_0 are recursively computed. We check that the first k_0 coefficients of T_u and T_v are all exactly the point-interval [0, 0]. If this is the case, it validates *a posteriori* the order of the zero the numerical heuristic already discovered. We can then factor out the term $(x - z_0)^{k_0}$, which leads to models of order $n + k_0 - k_0 = n$ and proceed with the division using the classical rule of Taylor models. If one of the leading coefficients of T_u or T_v is not exactly [0, 0], we fail to give an accurate model: a infinite remainder bound is returned.

We implemented these modified Taylor models in the development version of the Sollya software tool and used them for handling functions with removable discontinuities. We report on several such examples in the next section.

A modified Taylor model is easy to convert into a classical model: it mainly suffices to multiply the bound $\mathbf{\Delta}$ by $(I - \boldsymbol{z_0})^{n+1}$. We observe in practice that the enclosures obtained this way are generally a bit larger than the one obtained with the usual Taylor models. This comes from the fact that the remainder bounds in our modified approach are larger intervals than in the classical approach, hence more subject to the dependency phenomenon. However, the gap between both methods stayed reasonably small in all examples we tried.

5 Certification and formal proof

Our approach is distinguished from many others in the literature in that we aim to give a validated and guaranteed error bound, rather than merely one 'with high probability' or 'modulo rounding error'. Nevertheless, since both the underlying mathematics and the actual implementations are fairly involved, the reliability of our results, judged against the very highest standards of rigor, can still be questioned by a determined skeptic. The most satisfying way of dealing with such skepticism is to use our algorithms to generate a complete formal proof that can be verified by a highly reliable proof-checking program. This is doubly attractive because such proof checkers are now often used for verifying floating-point hardware and software [11, 37, 38, 39, 40, 41, 42, 43], In such cases bounds on approximation errors often arise as key lemmas in a larger formal proof, so an integrated way of handling them is desirable.

There is a substantial literature on using proof checkers to verify the results of various logical and mathematical decision procedures [44]. In some cases, a direct approach seems necessary, where the algorithm is expressed logically inside the theorem prover, formally proved correct and 'executed' in a mathematically precise way via logical inference. In many cases, however, it is possible to organize the main algorithm so that it generates some kind of 'certificate' that can be formally checked, i.e. used to generate a formal proof, without any formalization of the process that was used to generate it. This can often be both simpler and more efficient than the direct approach. (In fact, the basic observation that 'result checking' can be more productive than 'proving correctness' has been emphasized by Blum [45] and appears in many other contexts such as computational geometry [46].) The two principal phases of our approach illustrate this dichotomy quite well:

- In order to bound the difference $|f - T|$ between the function f and its Taylor series T, there seems to be no shortcut beyond formalizing the theory underlying the Taylor models inside the theorem prover and instantiating it for the particular cases used.

- Bounding the difference between the Taylor series T and the polynomial p that we are interested in reduces to polynomial nonnegativity on an interval, and this admits several

potentially attractive methods of certification, with 'sum of squares' techniques being perhaps the most convenient.

We consider each of these in turn.

5.1 Formalizing Taylor models

Fully formalizing this part inside a theorem prover is still work in progress. For several basic functions such as *sin*, versions of Taylor's theorem with specific, computable bounds on the remainder have been formalized in HOL Light, and set up so that formally proven bounds for any specific interval can be proven automatically. For example, in this interaction example from [41], the user requests a Taylor series for the cosine function such that the absolute error for $|x| \leq 2^{-2}$ is $\leq 2^{-35}$. The theorem prover not only returns the series $1 - x^2/2 + x^4/24 - x^6/720 + x^8/40320$ but also a theorem, formally proven from basic logical axioms, that indeed the desired error bound holds:

$$\forall x.\, |x| \leq 2^{-2} \Rightarrow |\cos(x) - (1 - x^2/2 + x^4/24 - x^6/720 + x^8/40320)| \leq 2^{-35}$$

```
#MCLAURIN_COS_POLY_RULE 2 35;;
it : thm =
 |- ∀x. abs x <= inv (&2 pow 2)
        ⇒ abs(cos x - poly [&1; &0; --&1 / &2; &0; &1 / &24; &0;
                            --&1 / &720; &0; &1 / &40320] x)
            <= inv(&2 pow 35)
```

However, this is limited to a small repertoire of basic functions expanded about specific points, in isolation, often with restrictions on the intervals considered. Much more work of the same kind would be needed to formalize the general Taylor models framework we have described in this paper, which can handle a wider range of functions, expanded about arbitrary points and nested in complex ways. This is certainly feasible, and related work has been reported [47, 48], but much remains to be done, and performing the whole operation inside a formal checker appears to be very time-consuming.

5.2 Formalizing polynomial nonnegativity

Several approaches to formally proving polynomial nonnegativity have been reported, including a formalization of Sturm's theorem [11] and recursive isolation of roots of successive derivatives [41]. However, perhaps the best approach for formal proof is to generate certificates involving sum-of-squares (SOS) decompositions. In order to prove that a polynomial $p(x)$ is everywhere nonnegative, a SOS decomposition $p(x) = \sum_{i=1}^{k} a_i s_i(x)^2$ for rational $a_i > 0$ is an excellent certificate: it can be used to generate an almost trivial formal proof, mainly involving the verification of an algebraic identity. For the more refined assertions of nonnegativity over an interval $[a, b]$, slightly more elaborate 'Positivstellensatz' certificates involving sums of squares and multiplication by $b - x$ or $x - a$ work well.

In a more general context of multivariate polynomials, Parrilo [49] pioneered the approach of generating such certificates using semidefinite programming (SDP). However, the main high-performance SDP solvers involve complicated nonlinear algorithms implemented in floating-point arithmetic. While they can invariably be used to find *approximate* SOS decompositions, it can be problematic to get *exact* rational decompositions, particularly if the

original coefficients have many significant bits and the polynomial has relatively low variation. Unfortunately these are just the kinds of problems we are concerned with. But if we restrict ourselves to *univariate* polynomials, which still covers our present application, more direct methods can be based only on complex root-finding, which is easier to perform in high precision. In what follows we correct an earlier description of such an algorithm [13] and extend it to the generation of full Positivstellensatz certificates.

The basic idea is simple. Suppose that a polynomial $p(x)$ is everywhere nonnegative. Roots always occur in conjugate pairs, and any real roots must have even multiplicity, otherwise the polynomial would cross the x-axis instead of just touching it. Thus, if the roots are $a_j \pm ib_j$, we can imagine writing the polynomial as:

$$\begin{aligned} p(x) &= [(x - [a_1 + ib_1])(x - [a_2 + ib_2]) \cdots (x - [a_m + ib_m])] \cdot \\ &\quad [(x - [a_1 - ib_1])(x - [a_2 - ib_2]) \cdots (x - [a_m - ib_m])] \\ &= (q(x) + ir(x))(q(x) - ir(x)) \\ &= q(x)^2 + r(x)^2 \end{aligned}$$

This well-known proof that any nonnegative polynomial can be expressed as a sum of two squares with arbitrary real coefficients can be adapted to give an exact rational decomposition algorithm, compensating for the inevitably inexact representation of the roots $a_j \pm ib_j$. This is done by finding a small initial perturbation of the polynomial that is still nonnegative. The complex roots can then be located sufficiently accurately using the excellent arbitrary-precision complex root finder in PARI/GP, which implements a variant of an algorithm due to Schönhage.

5.2.1 Squarefree decomposition

Since the main part of the algorithm introduces inaccuracies that can be made arbitrarily small but not eliminated completely, it is problematic if the polynomial is ever *exactly* zero. However, if the polynomial touches the x-axis at $x = a$, there must be a root $x - a$ of even multiplicity, say $p(x) = (x - a)^{2k} p^*(x)$. We can factor out all such roots by a fairly standard squarefree decomposition algorithm that uses only exact rational arithmetic and does not introduce any inaccuracy. The modified polynomial $p^*(x)$ can then be used in the next stage of the algorithm and the resulting terms in the SOS decomposition multiplied appropriately by the $(x - a)^k$.

Suppose, hypothetically, that the initial polynomial $p(x)$ has degree n and splits as

$$p(x) = c \prod_k (x - a_k)^{m_k}$$

We use the standard technique of taking the greatest common divisor of a polynomial and its own derivative to separate out the repeated roots, applying it recursively to obtain the polynomials $r_i(x)$ where $r_0(x) = p(x)$ and then $r_{i+1} = \gcd(r_i(x), r_i'(x))$ for each $0 \le i \le n-1$. We then obtain

$$r_i(x) = c \prod_k (x - a_k)^{\max(m_k - i, 0)}$$

note that each $m_k \le n$, so $r_i(x) = c$ for each $i \ge n$.

Now for each $1 \leq i \leq n+1$, let $l_i(x) = r_{i-1}(x)/r_i(x)$, so

$$l_i(x) = \prod_k (x - a_k)^{(\text{if } a_k \geq i \text{ then } 1 \text{ else } 0)}$$

and then similarly for each $1 \leq i \leq n$ let $f_i(x) = l_i(x)/l_{i+1}(x)$, so that

$$f_i(x) = \prod_k (x - a_k)^{(\text{if } a_k = i \text{ then } 1 \text{ else } 0)}$$

We have now separated the polynomial into the components $f_i(x)$ where the basic factors $(x - a_k)$ appear with multiplicity i, and we can then extract a maximal 'squarable' factor by

$$s(x) = \prod_{1 \leq i \leq n} f_i(x)^{\lfloor i/2 \rfloor}$$

We can then obtain a new polynomial $p^*(x) = p(x)/s(x)^2$ without repeated roots, for the next step, and subsequently multiply each term inside the SOS decomposition by $s(x)$.

5.2.2 Perturbation

From now on, thanks to the previous step, we can assume that our polynomial is strictly positive definite, i.e. $\forall x \in \mathbb{R}.\, p(x) > 0$. Since all polynomials of odd degree have a real root, the degree of the polynomial (and the original polynomial before the removal of squared part) must be even, say $\deg(p) = n = 2m$, and the leading coefficient of $p(x) = \sum_{i=0}^{n} a_i x_i$ must also be positive, $a_n > 0$. Since $p(x)$ is *strictly* positive, there must be an $\varepsilon > 0$ such that the perturbed polynomial

$$p_\varepsilon(x) = p(x) - \varepsilon(1 + x^2 + ... + x^{2m})$$

is also (strictly) positive. For provided $\varepsilon < a_n$, this is certainly positive for sufficiently large x, say $|x| > R$, since the highest term of the difference $p(x) - \varepsilon(1 + x^2 + ... + x^{2m})$ will eventually dominate. And on the compact set $|x| \leq R$ we can just also choose $\varepsilon < \inf_{|x| \leq R} p(x) / \sup_{|x| \leq R}(1 + x^2 + ... + x^{2m})$.

To find such an ε algorithmically we just need to test if a polynomial has real roots, which we can easily do in PARI/GP using Sturm's method; we can then search for a suitable ε by choosing a convenient starting value and repeatedly dividing by 2 until our goal is reached; we actually divide by 2 again to leave a little margin of safety. (Of course, there are more efficient ways of doing this.) We have been tacitly assuming that the initial polynomial *is* indeed nonnegative, but if it is not, that fact can be detected at this stage by checking the $\varepsilon = 0$ case, ensuring that $p(x)$ has no roots and that $p(c) > 0$ for any convenient value like $c = 0$.

5.2.3 Approximate SOS of perturbed polynomial

We now use the basic 'sum of two real squares' idea to obtain an approximate SOS decomposition of the perturbed polynomial $p_\varepsilon(x)$, just by using approximations of the roots. Recall from the discussion above that with exact knowledge of the roots $a_j \pm ib_j$ of $p_\varepsilon(x)$, we could

obtain a SOS decomposition with two terms. Assuming l is the leading coefficient of $p_\varepsilon(x)$ we would have:

$$\begin{aligned} p_\varepsilon(x) &= l[(x-[a_1+ib_1])(x-[a_2+ib_2])\cdots(x-[a_m+ib_m])]\cdot \\ &\quad [(x-[a_1-ib_1])(x-[a_2-ib_2])\cdots(x-[a_m-ib_m])] \\ &= l(s(x)+it(x))(s(x)-it(x)) \\ &= ls(x)^2+lt(x)^2 \end{aligned}$$

Using only approximate knowledge of the roots as obtained by PARI/GP, we obtain instead $p_\varepsilon(x) = ls(x)^2 + lt(x)^2 + u(x)$ where the coefficients of the remainder $u(x)$ can be made as small as we wish. We determine how small this needs to be in order to make the next step below work correctly, and select the accuracy of the root-finding accordingly.

5.2.4 Absorption of remainder term

We now have $p(x) = ls(x)^2 + lt(x)^2 + \varepsilon(1 + x^2 + ... + x^{2m}) + u(x)$, so it will suffice to express $\varepsilon(1 + x^2 + ... + x^{2m}) + u(x)$ as a sum of squares. Note that the degree of u is $< 2m$ by construction (though the procedure to be outlined would work with minor variations even if it were exactly $2m$). Let us say $u(x) = a_0 + a_1x + \ldots + a_{2m-1}x^{2m-1}$. Note that

$$\begin{aligned} x &= (x+1/2)^2 - (x^2+1/4) \\ -x &= (x-1/2)^2 - (x^2+1/4) \end{aligned}$$

and so for any $c \geq 0$:

$$\begin{aligned} cx^{2k+1} &= c(x^{k+1}+1/2\,x^k)^2 - c(x^{2k+2}+1/4\,x^{2k}) \\ -cx^{2k+1} &= c(x^{k+1}-1/2\,x^k)^2 - c(x^{2k+2}+1/4\,x^{2k}) \end{aligned}$$

Consequently we can rewrite the odd-degree terms of u as

$$a_{2k+1}x^{2k+1} = |a_{2k+1}|(x^{k+1}+\mathrm{sgn}(a_{2k+1})/2\,x^k)^2 - |a_{2k+1}|(x^{2k+2}+1/4\,x^{2k})$$

and so:

$$\begin{aligned} \varepsilon(1+x^2+...+x^{2m})+u = {} & \textstyle\sum_{k=0}^{m-1}|a_{2k+1}|(x^{k+1}+\mathrm{sgn}(a_{2k+1})/2\,x^k)^2 + \\ & \textstyle\sum_{k=0}^{m}(\varepsilon + a_{2k} - |a_{2k-1}| - |a_{2k+1}|/4)\,x^{2k} \end{aligned}$$

where by convention $a_{-1} = a_{2m+1} = 0$. This already gives us the required SOS representation, provided each $\varepsilon + a_{2k} - |a_{2k-1}| - |a_{2k+1}|/4 \geq 0$, and we can ensure this by computing the approximate SOS sufficiently accurately. We finally recover a SOS decomposition for the original polynomial by incorporating the additional factor $x - 2$ into each square:

5.2.5 Finding Positivstellensatz certificates

By a well-known trick, we can reduce a problem of the form

$$\forall x.\, a \leq x \leq b \Rightarrow 0 \leq p(x)$$

where $p(x)$ is a univariate polynomial, to the unrestricted polynomial nonnegativity problem $\forall y \in \mathbb{R}.\, 0 \leq q(y)$ by the change of variable

$$x = \frac{a + by^2}{1 + y^2}$$

and clearing denominators:

$$q(y) = (1 + y^2)^{\deg(p)} p\left(\frac{a + by^2}{1 + y^2}\right)$$

To see that this change of variables works, note that as y ranges over the whole real line, y^2 ranges over the nonnegative reals and so $x = (a + by^2)/(1 + y^2)$ ranges over $a \leq x < b$, and although we do not attain the upper limit b, the two problems

$$\forall x.\, a \leq x \leq b \Rightarrow 0 \leq p(x)$$

and

$$\forall x.\, a \leq x < b \Rightarrow 0 \leq p(x)$$

are equivalent, since $p(x)$ is a continuous function and therefore if $p(b) < 0$ there would be an open set containing b on which the polynomial is negative.

If we now use the algorithm from the previous subsections to obtain a SOS decomposition of $q(y) = \sum_i c_i s_i(y)^2$ for nonnegative rational numbers c_i, it is useful to be able to transform back to a Positivstellensatz certificate [49] for the nonnegativity on $[a, b]$ of the original polynomial $p(x)$. So suppose we have

$$q(y) = (1 + y^2)^{\deg(p)} p\left(\frac{a + by^2}{1 + y^2}\right) = \sum_i c_i s_i(y)^2$$

Let us separate each $s_i(y)$ into the terms of even and odd degree

$$s_i(y) = r_i(y^2) + y t_i(y^2)$$

This gives us the decomposition

$$q(y) = \sum_i c_i (r_i(y^2)^2 + y^2 t_i(y^2)^2 + 2y r_i(y^2) t_i(y^2))$$

However, note that by construction $q(y)$ is an even polynomial, and so by comparing the odd terms on both sides we see that $\sum_i y r_i(y^2) t_i(y^2) = 0$. By using this, we obtain the simpler decomposition arising by removing all those terms:

$$q(y) = \sum_i c_i r_i(y^2)^2 + c_i y^2 t_i(y^2)^2$$

Inverting the change of variable we get

$$y^2 = \frac{x - a}{b - x}$$

and

$$1 + y^2 = \frac{b - a}{b - x}$$

Therefore we have, writing $d = \deg(p)$:

$$\left(\frac{b-a}{b-x}\right)^d p(x) = \sum_i c_i r_i \left(\frac{x-a}{b-x}\right)^2 + c_i \frac{x-a}{b-x} t_i \left(\frac{x-a}{b-x}\right)^2$$

and so

$$p(x) = \sum_i \frac{c_i}{(b-a)^d}(b-x)^d r_i \left(\frac{x-a}{b-x}\right)^2 + \frac{c_i}{(b-a)^d}(x-a)(b-x)^{d-1} t_i \left(\frac{x-a}{b-x}\right)^2$$

We can now absorb the additional powers of $b-x$ into the squared terms to clear their denominators and turn them into polynomials. We distinguish two cases, according to whether $d = \deg(p)$ is even or odd.

- If d is even, we have

$$\begin{aligned} p(x) = \; & \textstyle\sum_i \frac{c_i}{(b-a)^d} \left[(b-x)^{\frac{d}{2}} r_i \left(\frac{x-a}{b-x}\right)\right]^2 + \\ & (x-a)(b-x) \textstyle\sum_i \frac{c_i}{(b-a)^d} \left[(b-x)^{\frac{d}{2}-1} t_i \left(\frac{x-a}{b-x}\right)\right]^2 \end{aligned}$$

- If d is odd, we have

$$\begin{aligned} p(x) = \; & (b-x) \textstyle\sum_i \frac{c_i}{(b-a)^d} \left[(b-x)^{\frac{d-1}{2}} r_i \left(\frac{x-a}{b-x}\right)\right]^2 + \\ & (x-a) \textstyle\sum_i \frac{c_i}{(b-a)^d} \left[(b-x)^{\frac{d-1}{2}} t_i \left(\frac{x-a}{b-x}\right)\right]^2 \end{aligned}$$

In either case, this gives a certificate that makes clear the nonnegativity of $p(x)$ on the interval $[a, b]$, since it constructs $p(x)$ via sums and products from squared polynomials, nonnegative constants and the expressions $x-a$ and $b-x$ in a simple and uniform way.

6 Experimental results

We have implemented our novel algorithm for validated supremum norms in a modified version of the Sollya software tool*. The Sum-of-Square decomposition necessary for the certification step has been implemented using the PARI/GP software tool†. The formal certification step has been performed using the HOL light theorem prover‡.

During the computation step before formal verification, the positivity of difference polynomials s_1 and s_2 (see Section 3) is shown using an Interval Arithmetic based implementation of the Sturm Sequence algorithm [50].The implementation has a fall-back to rational arithmetic if Interval Arithmetic fails to give an unambiguous answer because the enclosure is not sufficiently tight [12] . In the examples presented, this fall-back had to be used only once. However, beside this method, other well known techniques exist [25].

The intermediate polynomial T has been computed using Taylor models. Our implementation supports both absolute and relative remainder bounds. Relative remainder bounds are

*http://sollya.gforge.inria.fr/
†http://pari.math.u-bordeaux.fr/
‡http://www.cl.cam.ac.uk/~jrh13/hol-light/

used by the algorithm only when strictly necessary, i.e. when a removable discontinuity is detected (see Section 4.4). Our implementation of Taylor models integrates also some optimizations for computing remainder bounds found in literature [47]. We did not explicitly discuss those optimizations in this article for the sake of brevity.

We have compared the implementation of our novel supremum norm algorithm on 9 examples with implementations of the following existing algorithms discussed in Section 2:

- A pure numerical algorithm for supremum norms available in the Sollya tool through the command `dirtyinfnorm` [6]. The algorithm mainly samples the zeros of the derivative of the approximation error function ε and refines them with a Newton iteration [6]. We will refer to this algorithm as `Ref1`. As a matter of course, this algorithm does not offer the guarantees we address in this article. Its purpose is only to give reference timings corresponding to the kind of algorithms commonly used by people for computing supremum norms.

- A rigorous, Interval Arithmetic based supremum norm available through the Sollya command `infnorm` [6]. The algorithm performs a trivial bisection until Interval Arithmetic shows that the derivative of the approximation error function ε no longer contains any zero or some threshold is reached. The algorithm is published in [12]. We will refer to this algorithm as `Ref2`. We were not able to obtain a result in reasonable time (less than 10 minutes) using this algorithm for some instances. The cases are marked "N/A" below.

- A rigorous supremum norm algorithm based on Automatic Differentiation and rigorous bounds of the zeros of a polynomial. The algorithm is published in [14]. It gives an *a posteriori* error. We will refer to this algorithm as `Ref3`.

We will refer to the implementation of our new supremum norm algorithm as `Supnorm`. We made sure all algorithms computed a result with comparable final accuracy. This required choosing suitable parameters by hand for algorithms `Ref2` and `Ref3`. The time required for this manual adaptation is not accounted for but, of course, it exceeds the computation time by a huge factor. Our novel algorithm achieves this automatically by its *a priori* quality property.

We used the example instances for supremum norm computations published in [14]. More precisely, we can classify the examples as follows:

- The two first examples are somehow "toy" examples also presented in [12].

- 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.

- In examples 4 through 9, p is the minimax polynomial, i.e. the polynomial p of a given degree that minimizes 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.

The implementation of our novel supremum norm algorithm as well as the three reference algorithms are based on the Sollya tool. That tool was compiled using `gcc` version 3.4.6. The

timings were performed on an Intel® Core™ 2 Quad based system clocked at 2.66 GHz (1.66 GHz FSB) running Redhat* EL4 Update 4 Linux 2.6.9-42. All timings in Table 2 are in milliseconds (ms). The "mode" indicates whether the absolute or relative error between p and f was considered.

Example	f	$[a, b]$	$\deg(p)$	mode	quality $-\log_2 \overline{\eta}$	`Supnorm` time	`Ref1` time	`Ref2` time	`Ref3` time
#1	$\exp(x) - 1$	$[-0.25, 0.25]$	5	rel.	37.6	145	62	3138	427
#2	$\log_2(1 + x)$	$[-2^{-9}, 2^{-9}]$	7	rel.	83.3	349	224	N/A	2151
#3*	$\arcsin(x + m)$	$[a_3, b_3]$	22	rel.	15.9	1397	855	N/A	3881
#4	$\cos(x)$	$[-0.5, 0.25]$	15	rel.	19.5	687	215	N/A	1723
#5	$\exp(x)$	$[-0.125, 0.125]$	25	rel.	42.3	1059	697	N/A	4346
#6	$\sin(x)$	$[-0.5, 0.5]$	9	abs.	21.5	158	97	5419	520
#7	$\exp(\cos^2 x + 1)$	$[1, 2]$	15	rel.	25.5	22000	279	N/A	11789
#8	$\tan(x)$	$[0.25, 0.5]$	10	rel.	26.0	461	188	97000	1169
#9	$x^{2.5}$	$[1, 2]$	7	rel.	15.5	408	130	7582	1049

Table 2: Execution times (in ms) of our supremum norm algorithm compared to three other algorithms

The implementation of our novel algorithm, compared with the other validated supremum algorithms `Ref2` and `Ref3`, exhibits the best performance shown. As a matter of course, counterexamples can be constructed but are hard to find.

We note that with our new supremum norm algorithm, the overhead in using a validated technique for supremum norms of approximation error functions with respect to an unsafe, numerical technique `Ref1` drops to a factor around 3 to 5. This positive effect is reinforced by the fact that the absolute execution times for our supremum norm algorithm are less than 1 second in most cases. Hence supremum norm validation need no longer be a one time - one shot overnight task as previous work suggests [12].

Even certification in a formal proof checker comes into reach with our supremum norm algorithm. Table 3 gives the execution times for the post-computational rewriting of the difference polynomials s_i as a Sum-of-Squares. All timing values are given in seconds (s).

Example	f	$[a, b]$	$\deg(p)$	mode	`SOS` time
#1	$\exp(x) - 1$	$[-0.25, 0.25]$	5	rel.	2.7
#2	$\log_2(1 + x)$	$[-2^{-9}, 2^{-9}]$	7	rel.	11.3
#3*	$\arcsin(x + m)$	$[a_3, b_3]$	22	rel.	82.9
#4	$\cos(x)$	$[-0.5, 0.25]$	15	rel.	11.2
#5	$\exp(x)$	$[-0.125, 0.125]$	25	rel.	202.7
#6	$\sin(x)$	$[-0.5, 0.5]$	9	abs.	3.83
#7	$\exp(\cos(x)^2 + 1)$	$[1, 2]$	15	rel.	90.9
#8	$\tan(x)$	$[0.25, 0.5]$	10	rel.	9.1
#9	$x^{2.5}$	$[1, 2]$	7	rel.	6.1

Table 3: Execution times (in s) for showing non-negativity by rewriting s_i as a Sum-of-Squares

If even though the execution times for Sum-Of-Squares decomposition may seem high compared to the actual supremum norm computation times, they are quite reasonable. Our

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

implementation is still not at all optimized and most of the time, the final certification step, requiring the computation of a Sum-of-Squares decomposition is run only once per supremum norm instance in practice.

7 Conclusion

Every time a transcendental function f is approximated using a polynomial p, there is a need to determine the maximum error induced by this approximation. Several domains where this bound for the error is needed are: floating-point implementation of elementary functions, some cases of validated quadrature as well as in more theoretical proof work, involving transcendental functions.

Computing a rigorous, i.e. not underestimated, upper bound on the supremum norm of an approximation error function ε has long been considered a difficult task. While fast numerical algorithms exist, there was a lack of a validated algorithm. Expecting certified results was out of sight. Several previous proposals in the literature had many drawbacks. The computational time was too high, hence not permitting one to tackle complicated cases involving composite functions or high degree approximation polynomials. Moreover, the quality of the supremum norm's output was difficult to control. This was due either to the unknown influence of parameters or simply because the techniques required too much manual work.

The supremum norm algorithm proposed in this article seems to solve most of the problems. It is able to compute, in a validated way, a rigorous upper bound for the supremum norm of an approximation error function – in both absolute and relative error cases – with an *a priori* quality. Execution time, measured on real-life examples, is more than encouraging. There is no longer an important overhead in computing a validated supremum norm instead of a numerical approximation to it. In fact, the overhead factor is between 3 and 5 only and the absolute execution time is often less than 1 second on a current machine.

The algorithm presented is based on two important validation steps: the computation of an intermediate polynomial T with a validated bound for the remainder and the proof that some polynomials s_i are non-negative. In this article, several ways of automatically computing an intermediate polynomial with a remainder bound were revised. Special attention was given to how non-negativity of a polynomial could be shown rewriting it as a sum of squares. This technique already permits us not only to validate a non-negativity result but actually to certify it by formally proving it in a formal proof checker.

One point in certification is still outstanding: certification of the intermediate polynomial's remainder bound in a formal proof checker. The algorithms for Taylor models, revised in this article and implemented in a validated way, will have to be "ported" to the environment of a formal proof checker. Previous works like [47] are encouraging and the task does not seem to be technically difficult, the algorithms being well understood. The challenge is in the sheer number of base remainder bounds to be formally proved for all basic functions, each proof requiring a case-by-case study and "collateral" proofs in the field of analysis. However, we will continue to work on this point in the future.

Another small issue in the proposed validated supremum norm algorithm also needs to be addressed and understood. As detailed in Section 3, the algorithm consists of two steps: first, a numerical computation of a potential upper bound and second, a validation of this upper bound. A detailed timing analysis shows that the first step actually takes more than half of the execution time. On the one hand, this observation is encouraging as it means that computing a

validated result for a supremum norm is not much more expensive than computing a numerical approximation. On the other hand, this means that our hypothesis that a lower bound for a supremum norm could be found in negligible time has to be reconsidered. Future work should address that point, finding a way to start with a very rough and quickly available lower bound approximation that gets refined in the course of alternating computation and validation.

Acknowledgments

The authors would like to thank Nicolas Brisebarre, Martin Berz, Guillaume Hanrot, Joris van der Hoeven, Kyoko Makino, Frédéric Messine, Ned Nedialkov and Markus Neher for their precious advice and help.

References

[1] M. Abramowitz, I. A. Stegun, Handbook of Mathematical Functions, Dover, 1965.

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

[3] F. de Dinechin, C. Q. Lauter, G. Melquiond, Assisted verification of elementary functions using Gappa, in: P. Langlois, S. Rump (Eds.), Proceedings of the 21st Annual ACM Symposium on Applied Computing - MCMS Track, Vol. 2, Association for Computing Machinery, Inc. (ACM), Dijon, France, 2006, pp. 1318–1322.

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

[5] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torrès, Handbook of Floating-Point Arithmetic, Birkhauser Boston, 2009. doi:10.1007/978-0-8176-4705-6.
URL http://www.springer.com/birkhauser/mathematics/book/978-0-8176-4704-9

[6] S. Chevillard, C. Lauter, N. Jourdan, M. Joldes, Users manual for the Sollya tool, Release 1.1, http://gforge.inria.fr/frs/download.php/7055/sollya.pdf (September 2008).

[7] F. Messine, Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution de problèmes avec contraintes, Ph.D. thesis, INP de Toulouse (1997).

[8] B. Kearfott, Rigorous Global Search: Continuous Problems, Kluwer, Dordrecht, Netherlands, 1996.

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

[10] W. Krämer, Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen, Tech. rep., Institut für angewandte Mathematik, Universität Karlsruhe (1996).

[11] J. Harrison, Floating point verification in HOL light: the exponential function, Technical Report 428, University of Cambridge Computer Laboratory (1997).
URL http://www.cl.cam.ac.uk/users/jrh/papers/tang.ps.gz

[12] S. Chevillard, C. Lauter, A certified infinite norm for the implementation of elementary functions, in: Proc. of the 7th International Conference on Quality Software, 2007, pp. 153–160.

[13] 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, Springer-Verlag, 2007, pp. 102–118.

[14] S. Chevillard, M. Joldes, C. Lauter, Certified and fast computation of supremum norms of approximation errors, in: 19th IEEE SYMPOSIUM on Computer Arithmetic, 2009, pp. 169–176.

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

[16] W. Press, B. Flannery, S. Teukolsky, W. Vetterling, Numerical Recipes in C, The Art of Scientific Computing, 2nd edition, Cambridge University Press, 1992.

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

[18] N. Revol, Newton's algorithm using multiple precision interval arithmetic, Numerical Algorithms 34 (2) (2003) 417–426.
URL http://www.inria.fr/rrrt/rr-4334.html

[19] N. Revol, F. Rouillier, The MPFI library, http://gforge.inria.fr/projects/mpfi/.

[20] A. Neumaier, Taylor forms–use and limits, Reliable Computing 9 (1) (2003) 43–79.

[21] K. Makino, M. Berz, Taylor models and other validated functional inclusion methods, International Journal of Pure and Applied Mathematics 4 (4) (2003) 379–456.
URL http://bt.pa.msu.edu/pub/papers/TMIJPAM03/TMIJPAM03.pdf

[22] M. Berz, K. Makino, COSY INFINITY Version 9.0, http://cosyinfinity.org.

[23] M. Berz, K. Makino, Rigorous global search using taylor models, in: SNC '09: Proceedings of the 2009 conference on Symbolic numeric computation, ACM, New York, NY, USA, 2009, pp. 11–20. doi:http://doi.acm.org/10.1145/1577190.1577198.

[24] M. Berz, K. Makino, Y.-K. Kim, Long-term stability of the tevatron by verified global optimization, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 558 (1) (2006) 1 – 10, proceedings of the 8th International Computational Accelerator Physics Conference - ICAP 2004. doi:DOI:10.1016/j.nima.2005.11.035.

[25] F. Rouillier, P. Zimmermann, Efficient isolation of polynomial's real roots, Journal of Computational and Applied Mathematics 162 (1) (2004) 33–50. doi:http://dx.doi.org/10.1016/j.cam.2003.08.015.

[26] C. Bendsten, O. Stauning, TADIFF, a Flexible C++ Package for Automatic Differentiation Using Taylor Series, Tech. Rep. 1997-x5-94, Technical University of Denmark (April 1997).

[27] L. B. Rall, The arithmetic of differentiation, Mathematics Magazine 59 (5) (1986) 275–282.

[28] A. Griewank, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, no. 19 in Frontiers in Appl. Math., SIAM, Philadelphia, PA, 2000.

[29] R. P. Brent, H. T. Kung, $\mathcal{O}\left((n \log n)^{3/2}\right)$ algorithms for composition and reversion of power series, in: J. F. Traub (Ed.), Analytic Computational Complexity, Academic Press, New York, 1975, pp. 217–225.

[30] R. P. Brent, H. T. Kung, Fast Algorithms for Manipulating Formal Power Series, Journal of the ACM 25 (4) (1978) 581–595.

[31] L. V. Ahlfors, Complex analysis. An introduction to the theory of analytic functions of one complex variable., 3rd Edition, McGraw-Hill New York, 1979.

[32] I. Eble, M. Neher, ACETAF: A Software Package for Computing Validated Bounds for Taylor Coefficients of Analytic Functions, ACM Transactions on Mathematical Software 29 (3) (2003) 263–286.

[33] M. Mezzarobba, B. Salvy, Effective Bounds for P-Recursive Sequences, Tech. Rep. abs/0904.2452, arXiv (2009).

[34] K. Makino, Rigorous analysis of nonlinear motion in particle accelerators, Ph.D. thesis, Michigan State University, East Lansing, Michigan, USA (1998).

[35] V. Stahl, Interval methods for bounding the range of polynomials and solving systems of nonlinear equations, Ph.D. thesis, Johannes Kepler University Linz, Linz, Austria (1995).

[36] R. Zumkeller, Global optimization in type theory, Ph.D. thesis, École polytechnique (2008).

[37] J. S. Moore, T. Lynch, M. Kaufmann, A mechanically checked proof of the correctness of the kernel of the $AMD5_K86$ floating-point division program, IEEE Transactions on Computers 47 (1998) 913–926.

[38] D. Russinoff, A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions, LMS Journal of Computation and Mathematics 1 (1998) 148–200, available on the Web at http://www.russinoff.com/papers/k7-div-sqrt.html.

[39] J. O'Leary, X. Zhao, R. Gerth, C.-J. H. Seger, Formally verifying IEEE compliance of floating-point hardware, Intel Technology Journal 1999-Q1 (1999) 1–14, available on the Web as http://download.intel.com/technology/itj/q11999/pdf/floating_point.pdf.

[40] R. Kaivola, M. D. Aagaard, Divider circuit verification with model checking and theorem proving, in: M. Aagaard, J. Harrison (Eds.), Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000, Vol. 1869 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 338–355.

[41] J. Harrison, Formal verification of floating point trigonometric functions, in: W. A. Hunt, S. D. Johnson (Eds.), Formal Methods in Computer-Aided Design: Third International Conference FMCAD 2000, Vol. 1954 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 217–233.

[42] C. Jacobi, Formal verification of a fully IEEE compliant floating point unit, Ph.D. thesis, University of the Saarland, available on the Web as http://engr.smu.edu/~seidel/research/diss-jacobi.ps.gz (2002).

[43] S. Boldo, Preuves formelles en arithmétiques à virgule flottante, Ph.D. thesis, ENS Lyon, available on the Web from http://www.ens-lyon.fr/LIP/Pub/Rapports/PhD/PhD2004/PhD2004-05.pdf (2004).

[44] R. J. Boulton, Efficiency in a fully-expansive theorem prover, Technical Report 337, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, author's PhD thesis (1993).

[45] M. Blum, Program result checking: A new approach to making programs more reliable, in: A. Lingas, R. Karlsson, S. Carlsson (Eds.), Automata, Languages and Programming, 20th International Colloquium, ICALP93, Proceedings, Vol. 700 of Lecture Notes in Computer Science, Springer-Verlag, Lund, Sweden, 1993, pp. 1–14.

[46] K. Mehlhorn, S. Nher, M. Seel, R. Seidel, T. Schilz, S. Schirra, C. Uhrig, Checking geometric programs or verification of geometric structures, in: Proceedings of the 12th Annual Symposium on Computational Geometry (FCRC'96), Association for Computing Machinery, Philadelphia, 1996, pp. 159–165.

[47] R. Zumkeller, Formal Global Optimization with Taylor Models, in: Proc. of the 4th International Joint Conference on Automated Reasoning, 2008, pp. 408–422.

[48] F. Cháves, M. Daumas, A library of taylor models for pvs automatic proof checker, CoRR abs/cs/0602005.

[49] P. A. Parrilo, Semidefinite programming relaxations for semialgebraic problems, Mathematical Programming 96 (2003) 293–320.

[50] F. Broglia (Ed.), Lectures in Real Geometry, Vol. 23 of Expositions in Mathematics, de Gruyter, 1996, Ch. Basic algorithms in real algebraic geometry and their complexity: from Sturm's theorem to the existential theory of reals, pp. 1–67.