A certificate-based approach to formally verified approximations

Florent Bréhard, Assia Mahboubi, Damien Pous

A certificate-based approach to formally verified approximations

Florent Bréhard
Plume and AriC teams, LIP, ENS de Lyon, Université de Lyon, Lyon, France
MAC team, LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Assia Mahboubi
Gallinette team, LS2N, INRIA, Université de Nantes, Nantes, France

Damien Pous
Plume team, LIP, CNRS, ENS de Lyon, Université de Lyon, Lyon, France

Abstract

We present a library to verify rigorous approximations of univariate functions on real numbers, with the Coq proof assistant. Based on interval arithmetic, this library also implements a technique of validation *a posteriori* based on the Banach fixed-point theorem. We illustrate this technique on the case of operations of division and square root. This library features a collection of abstract structures that organise the specfication of rigorous approximations, and modularise the related proofs. Finally, we provide an implementation of verified Chebyshev approximations, and we discuss a few examples of computations.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases approximation theory, Chebyshev polynomials, Banach fixed-point theorem, interval arithmetic, Coq

Related Version Except for the appendix, this paper appears in Proc. ITP 2019

Supplement Material https://gitlab.inria.fr/amahboub/approx-models

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).
This work was supported in part by the project FastRelax ANR-14-CE25-0018-01

1 Introduction

While numerical analysis offers sophisticated computational methods to solve various function space problems, the numerical errors caused by floating-point computations, discretisations or finite iterations, are a major concern in domains like safety-critical engineering or computer assisted proofs in mathematics. To address these issues, *rigorous numerics* [37] provides algorithms to compute validated enclosures of the exact solution. However, their correctness is ensured by pen-and-paper mathematical proofs. In particular, there is no guarantee concerning their *implementations*.

In this regard, formal proof offers the highest level of confidence. Several noteworthy works use formally proved rigorous numerics to completely formalise highly nontrivial mathematical results, like the Flyspeck project [18] for the Kepler conjecture or the formal verification [22] of the computer-aided proof of the Lorenz attractor [36]. However, those methods often require intensive computations, which rapidly becomes restrictive inside proof assistants. In the context of formal verification, certificate-based methods is an appealing strategy [19, 12, 1]. It consists in discharging part of the computation work load to external oracles, while correctness remains guaranteed via *a posteriori* validation steps performed inside the proof assistant.

This approach has mostly been used for the purpose of verifying symbolic computations, e.g. primality proofs [17], but we illustrate here how it can also by used in the context of rigorous numerical analysis.

Interval arithmetic. Invented in the 60s by Moore [32] and significantly developed in the 80s by Kulisch *et al.*, interval arithmetic is an essential building block of rigorous numerics. The key idea consists in using real intervals with representable endpoints (e.g., floating-point numbers) as rigorous enclosures of real numbers, and providing operations preserving correctness. For example, from $\pi \in [3.1415, 3.1416]$ and $e \in [2.7182, 2.7183]$, one obtains $\pi + e \in [3.1415, 3.1416] \oplus [2.7182, 2.7183] = [5.8597, 5.8599]$. Efficient implementations are available, as MPFI [33], IntLab [34], C-XSC [27], ARB [23]. The CoqInterval library [31] moreover provides a fully verified implementation inside the Coq proof assistant.

Rigorous Chebyshev approximations. Interval arithmetic is however not a panacea, and replacing all operations on real numbers by interval ones should always be considered with caution: the *dependency phenomenon* may lead to disastrous over-approximations. In such cases, *higher order methods* such as rigorous polynomial approximations (RPAs) are preferable. A pioneer work is that of Berz and Makino on *Taylor models* [4]. Those provide not only a polynomial, but also a *remainder* s.t. the latter contains the difference between the former and the represented function. Since then, efforts were made to clarify the definition of RPAs and extend them to other bases, in particular the Chebyshev basis [10, 25], due to their far better approximation properties than Taylor expansions [35].

On the formal proof side, the CoqInterval library includes an implementation of Taylor models called CoqApprox [30], allowing in particular for an automated rigorous evaluation procedure of definite integrals inside Coq [29]. Unfortunately, an equally accomplished equivalent with Chebyshev approximations does not exist now. Our contribution is a first step towards a formally proved counterpart of the popular Chebfun package [14] for Matlab.

Fixed-point based a posteriori validation. Some operations in function spaces admit straightforward *self-validating* algorithms by replacing all operations in $\mathbb{R}$ by interval ones. Unfortunately, more complicated operations (e.g., division, square root, differential equations) face several obstructions: the intervals may fail to give sufficiently tight enclosures, bounds for the remainders may be unknown, or only asymptotic, or depend on noneffective quantities.

In such cases, *a posteriori* validation techniques are an attractive alternative, widely used in rigorous numerics. They consist in reconstructing afterwards an error bound for a candidate approximation. Dating back from the works of Kantorovich about Newton's method, they gained prominence with the rise of modern computers and were applied to numerous functional analysis problems [26, 39, 38, 28]. Even more recently, those methods were used to compute RPAs for solutions of linear ODEs [2, 8]. Broadly speaking, the function of interest is characterised as a fixed-point of a contracting operator, from which an error bound is recovered thanks to the Banach fixed-point theorem [3, Thm. 2.1]. Such techniques are of special interest for formal verification, for they allow one to rely on efficient but untrusted external tools while keeping the trusted codebase small: it suffices to formalise the theory about contracting operators and provide means of computing with those operators.

Contributions and outline. We present a Coq library that makes it possible to compute rigorous Chebyshev approximations of functions on reals. We support basic operations like multiplication or integration in the standard way. For more complex operations like division

and square root, we resort to *a posteriori* validation techniques, thus making a first step towards a potential cooperation between external numerical tools and Coq.

We use the interval arithmetic provided by CoqInterval, but we design our abstractions for RPAs from scratch: this allows us to experiment with different design choices, with more flexibility. We first describe the main lines of the hierarchy (Section 2): we rely on canonical structures to abstract over the concrete implementation details of interval arithmetic, and we use them to denote both real valued functions and their rigorous approximations. We also abstract away from the concrete basis for approximations, to work in the future with different bases, even non polynomial ones (e.g., Bessel functions). We provide instances for the monomial and Chebyshev bases, the latter being described in Section 3.

The main theorem we need to perform a posteriori validation is the Banach fixed-point theorem, whose formalisation is described in Section 4. We show in Section 5 how to apply this theorem to compute rigorous approximations for division and square root using Newton-like operators. We finally discuss the benefits of our approach on two examples (Section 6): RPAs for the absolute value function, and verified computation of integrals related to the second part of Hilbert's 16th problem.

2 Approximating real numbers and functions

Numerical errors come from the estimation of both *real numbers*, e.g. using floating-point numbers, and *real functions*, e.g. using polynomials. Rigorous estimations must take all these uncertainties into account. For this purpose, *interval arithmetic* provides an explicit enclosure and *rigorous polynomial approximations* attach an interval to a polynomial approximant, which bounds the method error on a given domain. Note that the coefficients of polynomial approximations are usually themselves obtained from evaluations of the function or of its derivatives, and therefore also subject to numerical errors. A formal library about rigorous approximation thus implements several variants of each operation, on real numbers, floats, intervals, mathematical functions, approximants, etc., whose relationships are made precise in the various layers of specifications. Our library features a small hierarchy of structures which formalises and organises the dependencies between these variants.

2.1 Reals and Intervals

At the bottom of the hierarchy, structure `Ops0` collects the operations available on reals, floats, intervals, but also on polynomials and rigorous approximations. It provides the signature of a ring structure, with symbols $+$, $-$, $*$, 1 and 0 shared by all instances thanks to Coq's system of canonical structures. Yet the ring equational theory is a priori only available for real numbers. These operations are also those trivially self-validating. A super-structure `Ops1` collects other operations required on data-structures used for scalars: reals, floats, interval endpoints, intervals, etc. They are not meant to be implemented on polynomial approximations.

```
Record Ops0 := {
 car:> Type;
 add: car → car → car;
 sub: car → car → car;
 mul: car → car → car;
 zer, one: car }.
```

```
Record Ops1 := {
 ops0:> Ops0;
 fromZ: Z → ops0;
 div: ops0 → ops0 → ops0;
 sqrt, cos, abs: ops0 → ops0;
 pi: ops0 }.
```

Structure `Rel0` specifies the relationship between the operations of `Ops0` on reals and those on intervals. The field `rel` is a relation between the two instances `C` and `D`, which share overloaded notations. The relation will eventually be instantiated with the containment

relation between intervals and reals. When doing so, the requirements on the relation precisely correspond to the fact that interval operations properly approximate real operations. A record `Rel1` is defined in the very same way for `Ops1`.

```
Record Rel0 (C D: Ops0) := {
 rel:> C → D → Prop;
 radd: ∀ x y, rel x y → ∀ x' y', rel x' y' → rel (x+x') (y+y');
 rsub: ∀ x y, rel x y → ∀ x' y', rel x' y' → rel (x-x') (y-y');
 rmul: ∀ x y, rel x y → ∀ x' y', rel x' y' → rel (x*x') (y*y');
 rzer: rel 0 0;
 rone: rel 1 1 }.
```

As much as possible, we will work with polymorphic functions like the following one:

```
Definition f (C: Ops1) (x: C): C := 1 / (1 + sqrt x).
```

First of all, this allows us to define at once a function on real numbers (here, $x \mapsto \frac{1}{1+\sqrt{x}}$) and a function on intervals, whatever the implementation of intervals. Second, and even more importantly, the corresponding approximation correctness theorem will always hold—by a parametricity meta-result, such a function f will always satisfy the following lemma:

```
Lemma rf: ∀ C D (T: Rel1 C D), ∀ x y, T x y → T (f x) (f y).
```

This is only a meta-result: we need to provide a proof for each function f; but the proof is always trivial, and we automatise it.

There are however operations which cannot be implemented at this level of abstraction, even if we were to add some operations to the record `Ops1`. This is typically the case for division and square root of rigorous approximations, which require operations on intervals that do not make sense on real numbers (e.g., computing the range of a function and checking that it is bounded). In order to define those operations while remaining rather agnostic about the choice of interval implementation, we setup an intermediate layer of abstraction using the structure `NBH` (for neighbourhood):

```
Record NBH := {
 II:> Ops1;                (* abstract intervals *)
 contains: Rel1 II ROps1;  (* containment relation; ROps1 is the Ops1 instance on R *)
 convex: ∀ Z x y, contains Z x → contains Z y → ∀ z, x≤z≤y → contains Z z };
 (* additional operations on intervals *)
 bnd: II → II → II;        (* directed convex hull *)
 is_lt: II → II → bool;    (* strict above test *)
 min,max: II → option II;  (* min, max, if any *)
 bot: II;                  (* uninformative, contains all reals *)
 (* specification of the above operations *)
 bndE: ∀ X x, contains X x → ∀ Y y, contains Y y → ∀ z, x≤z≤y →contains (bnd X Y) z;
 is_ltE: ∀ X Y, wreflect (∀ x y, contains X x → contains Y y → x<y) (is_lt X Y);
 minE, maxE, botE: ... }.
```

We will also make use of the two following derived operations:

```
Definition mag (N: NBH) (X: II): option II := max (abs X).
Definition sym (N: NBH) (X: II): II := let X := abs X in bnd (-X) X.
```

The first one approximates the magnitude as an interval, if possible; the second one returns an interval centered in 0 that contains the argument. Note that we assume that intervals are convex. We provide an instance of this structure using the CoqInterval library, using intervals of floating point numbers from the Flocq library [6]. It is actually a family of instances indexed by the desired precision.

2.2 Abstract functions

The structure `FunOps` describes inductively the catalogue of expressions that the library can approximate.

```
Record FunOps (C: Type) := {
 funcar:> Ops0;  (* abstract type for functions, and pointwise basic operations *)
 id: funcar;
 cst: C → funcar;
 eval: funcar → C → C;
 integrate: funcar → C → C → C;
 div': nat → funcar → funcar → funcar;
 sqrt': nat → funcar → funcar }.
```

It is parameterised by a type `C` of ground values (typically, reals or intervals); it packages a set of basic operations on some abstract type for functions (pointwise addition, multiplication…), together with operations specific to functions: identity, constant function, evaluation, integration. It also asks for division and square root operations; those have an additional argument which is used to pass parameters to the oracles used in the implementation of those operations (for now, the degree of the interpolants).

When `C` is `R`, the type of real numbers, this structure is instantiated with the standard operations on `R→R` (ignoring the extra parameters for division and square root); our main goal is to provide instances with intervals for `C`, with which it is possible to perform computations.

Like for ground values, the structure `FunOps` makes it possible to write polymorphic functions like:

```
Definition g (C: Ops1) (F: FunOps C): F :=
  let f: F := div' 33 1 (1 + sqrt' 33 id ) in
  let a: C := integrate f 0 1 in
  pi + id * cst a
```

Such a declaration defines at the same time a function on reals ($x \mapsto \pi + x \int_0^1 \frac{\mathrm{d}t}{1+\sqrt{t}}$) and approximations of it, which will be obvious to prove correct whenever the chosen instance `F` satisfies appropriate properties. Those instances are obtained using *rigorous approximations*.

2.3 Rigorous Approximations

Approximating a function usually consists in projecting this function onto a finite dimension vector space, by expansion on a basis with appropriate properties. For instance, so-called Taylor models [4], are an instance of *rigorous polynomial approximation*. They attach an interval bounding the remainder to a certain polynomial, in this case represented in monomial basis, so as to describe a set of functions containing the one to be approximated. More generally in this section, a *rigorous approximation* refers to a linear combination of elements in a suitable basis, packaged with an interval remainder. In the code, we will also use the shorter term *model*, by analogy with Taylor models.

A basis is described by a family of functions, non necessarily polynomials, indexed by natural numbers, that is a term `T: nat→R→R`. The structure `BasisOps_on` below describes the signature required on a basis `T`. It is parameterised by the type `C` of coefficients; sequences of such coefficients (`seq C`) represent linear combinations of elements of `T`. Linear operations $(+, -, 0)$ need not be provided since they can be implemented independently from the basis. The range operation is important: its role is to bound the range on the given domain; it should be as accurate as possible since it is used at many places to compute error bounds in rigorous approximations (e.g., for multiplication and a posteriori validation). We define `BasisOps` to be a polymorphic function so that we capture with a single object the idealised operations on reals and their concrete implementation with intervals.

```
Record BasisOps_on (C: Type) := {
  lo, hi: C;                      (* bounds for the domain *)
  beval: seq C → C → C;           (* (efficient) evaluation *)
  bmul: seq C → seq C → seq C;    (* multiplication *)
  bone, bid: seq C;               (* constant to 1, identity *)
  bprim: seq C → seq C; }.        (* primitive *)
  brange: seq C → C*C; }.         (* range *)
Definition BasisOps := ∀ C: Ops1, BasisOps_on C.
```

Given such operations, we equip type `seq C` with the basic operations in `Ops0`. Then we can define rigorous approximations:

```
Record Model C := { pol: seq C; rem: C }.
```

Like with `seq C`, we equip `Model C` with the basic operations in `Ops0`, and then with those from `FunOps`. For instance, addition, evaluation and integration are defined as follows:

```
Definition madd (C: Ops1) (M N: Model C): Model C :=
  {| pol := pol M + pol N; rem := rem M + rem N |}.
Definition meval (C: Ops1) (M: Model C) (X: C): C := beval (pol M) X + rem M.
Definition mintegrate (C: Ops1) (M: Model C) (a b: C): C :=
   let N := bprim (pol M) in beval N b - beval N a + (b-a)*rem M.
```

For those relatively simple operations, it suffices to have the basic operations (`Ops1`) on `C`. For other operations like the range of a model, we actually need the additional operations on intervals provided by the structure `NBH`:

```
Definition mrange (N: NBH) (M: Model II) :=
  let (a,b) := brange (pol M) in bnd a b + rem M.
```

This is also the case for division and square root, which we will discuss in Section 5. All in all, we obtain instances `FunOps` through a construction of the following type:

```
Canonical Structure MFunOps (N: NBH) (B: BasisOps): FunOps II.
  (* with carrier [Model II] *)
```

It finally remains to show that those operations defined on rigorous approximations properly match the idealised operations on functions over reals. We fix in the sequel an instance `N: NBH` of neighbourhood and basis operations `B: BasisOps`, and we write `Model` for `Model II`). The central definition to establish this correspondence is the following one, where the function `eval` is the obvious evaluation function for linear combinations of elements of `T`.

```
Definition mcontains (F: Model) (f: R → R) :=
 ∃ p: seq R, scontains (pol F) p /\ ∀ x, lo≤x≤hi → contains (rem F) (f x - eval T p x)
```

Intuitively, a model contains a real-valued function f if it contains a generalised polynomial which is close enough to f on the domain of the basis. (The binary predicate `scontains` denotes the pointwise extension of the relation `contains` to sequences: in the definition, the real coefficients of `p` should be pointwise contained in the interval coefficients of `pol F`.)

Equipped with this definition, we prove lemmas like:

```
Lemma rmmul: ∀ F f G g,
   mcontains F f → mcontains G g → mcontains (F*G) (f*g).
Lemma rmdiv: ∀ n F f G g,
   mcontains F f → mcontains G g → mcontains (div' n F G) (div' n f g).
Lemma rmintegrate: ∀ F f A a B b,
   (∀ x, lo≤x≤hi → continuous_at f x) → lo≤a≤hi → lo≤b≤hi →
   mcontains F f → contains A a → contains B b →
   contains (integrate F A B) (integrate f a b).
```

Of course, we need assumptions on the basis operations in order to do so. Those assumptions are summarised in the following structure. Recall that a `B: BasisOps` provides us with operations `B ROps1` on reals and operations `B II` on intervals. The structure assumes: 1) the expected properties on the operations on reals, i.e, efficient evaluation corresponds to evaluation with `T`, multiplication indeed corresponds to pointwise multiplication under evaluation, etc.; and 2) a relationship between the operations on reals and on intervals. This separation of concerns is very convenient: the latter containment lemmas are always proved in a trivial way (i.e., automatically), and the former properties do not involve intervals at all, but only real numbers and functions, for which usual mathematical intuitions apply.

```
Record ValidBasisOps (N: NBH) (B: BasisOps) := {
 (* properties of operations on reals (B ROps1) *)
 lohi: lo < hi;
 bevalE: ∀ p x, beval p x = eval T p x;
 eval_cont: ∀ p x, continuity_pt (eval T p) x;
 eval_mul: ∀ p q x, eval T (bmul p q) x = eval T p x * eval T q x;
 eval_prim: ∀ p a b, eval T (bprim p) b - eval T (bprim p) a = RInt (eval T p) a b;
 ...
 (* relationship between operations on intervals (B II) and on reals (B ROps1) *)
 rbeval: ∀ P p X x, scontains P p → contains X x → contains (beval P X) (beval p x);
 rbmul: ∀ P p Q q, scontains P p → scontains Q q → scontains (bmul P Q) (bmul p q);
 rbprim: ∀ P p, scontains P p → scontains (bprim P) (bprim p);
 ... }.
```

3 Arithmetic on Chebyshev polynomials

In order to use the previously described rigorous approximations, it remains to provide implementation of operations (`BasisOps`) for certain families `T` of functions. We provide two instances of them: one for the standard monomial basis, where `T n x = x^n`, and one described in this section for Chebyshev basis, where `T n` is the n-th Chebyshev polynomial.

Chebyshev polynomials are defined by the following recurrence, which immediately translates to a recursive definition in Coq.

$$T_0 = 1 \qquad T_1 = X \qquad T_{n+2} = 2XT_{n+1} - T_n$$

We can then prove simple properties of those polynomials, for instance:

$$T_nT_m = (T_{n+m} + T_{m-n})/2 \qquad (n \le m) \tag{1}$$

$$T_0 = T_1' \qquad T_1 = \frac{T_2'}{4} \qquad T_{n+3} = \frac{T_{n+3}'}{2(n+3)} - \frac{T_{n+1}'}{2(n+1)} \tag{2}$$

$$T_n(\cos t) = \cos(nt) \tag{3}$$

Those are proved in a few lines using existing lemmas about derivation and cosine.

3.1 Clenshaw's evaluation algorithm

The first operation we must implement for `BasisOps` is the evaluation function (`beval`). This operation should be polymorphic and as efficient as possible: it will be executed repeatedly when constructing and using rigorous approximations. We use Horner evaluation scheme for the monomial basis, and Clenshaw's algorithm [15] for Chebyshev, which are both linear in the number of elementary operations. The latter is usually presented as a dynamic programming routine. We fix abstract operations `C: Ops1` for the remaining Coq snippets in this section, and we translate this routine into a recursive function with two accumulators:

```
Fixpoint Clenshaw b c (p: seq C) x :=
  match p with
  | [] => c - x*b
  | a::q => Clenshaw c (a + 2*x*c - b) q x
  end.
Definition beval (p: seq C) x := Clenshaw 0 0 (rev p) x.
```

This code might look mysterious; it is justified by the following invariant on real numbers:

```
Lemma ClenshawR b c p x: Clenshaw b c p x = eval T (catrev p [c - 2*x*b; b]) x.
```

In the right-hand side, `catrev` is the function that reverses its first argument and catenate it with the second one. The proof is done by induction in just three lines, using the Coq tactic for ring equations [16]. Correctness (i.e., field `bevalE` from structure `ValidBasisOps`) follows.

Note that while the definition of `beval` can be used with any `Ops1` structure, its correctness is proved only on reals: the lemma `ClenshawR` does not hold in every `Ops1` structure. The behaviour of `beval` on those structures is specified only through the fact that it respects containments (field `rbeval` from structure `ValidBasisOps`, which is proved automatically.)

3.2 Multiplication

Another important operation is multiplication. Again, this operation should be polymorphic, and efficient. A difficulty here is that due to Equation (1), the n-th coefficient of a multiplication potentially depends on all coefficients of its arguments, not only on the coefficient of smaller rank. We use the following definition, with two auxiliary recursive functions:

```
Fixpoint mul_pls (p q: seq C): seq C :=
  match p,q with
  | [],_ | _,[] => []
  | a::p', b::q' => sadd (a*b::(sadd (sscal a q') (sscal b p'))) (0::0::mul_pls p' q')
  end.
Fixpoint mul_mns (p q: seq C): seq C :=
  match p,q with
  | [],_ | _,[] => []
  | a::p', b::q' => sadd (a*b::(sadd (sscal a q') (sscal b p'))) (mul_mns p' q')
  end.
Definition smul (p q: seq C): seq C := sscal (1/2) (sadd (mul_mns p q) (mul_pls p q))
```

where `sscal` is the scalar multiplication for polynomials— we cannot yet use the standard notation for this operation since we are in the process of defining an `Ops0` structure on `seq C`. The function `mul_pls` actually corresponds to multiplication in the monomial basis, it covers the first summand in the right-hand side of (1). The function `mul_mns` differs only in the fact that the recursive call is not pushed away using two 'cons' operations; it covers the second summand in the right-hand side of (1). Like previously, that `smul` preserves containments (field `rbmul` of structure `ValidBasisOps`) is obvious: this operation only performs a finite sequence of operations preserving containments. Proving that it behaves correctly on reals numbers is more interesting; the key invariant is the following one:

```
Lemma eval_mul_: ∀ (p q: seq R) n x,
  eval_ n p x * eval_ n q x = (eval (mul_mns p q) x + eval_ (n+n) (mul_pls p q) x)/2.
```

Here, `eval_ n p` evaluates `P` padded with n zeros in front of it. Again, the difficulty is to find the lemma: it is proved in six lines using (1), and correctness of `smul` on reals immediately follows. Taking primitives in Chebyshev basis follows the same pattern (see Appendix A).

3.3 Range

As mentioned above, we need accurate estimations of the range of a given polynomial in order to be able to compute precise rigorous approximations. This range can always be estimated by evaluating the polynomial on the interval representing the domain (i.e., given `p: seq C`, compute `beval p (bnd lo hi)`). This technique is however not sufficient in practice: this tends to produce largely over-estimated bounds. With Chebyshev basis we can proceed differently: indeed, thanks to Equation (3), T_n ranges over $[-1;1]$ on $[-1;1]$. Therefore, the range of a polynomial on $[-1;1]$ can be estimated by using the sum of the absolute values of the coefficients in Chebyshev basis (and actually, we do not need to take the absolute value of the first coefficient since $T_0 = 1$).

```
Definition range_: seq C → C := foldr (fun A X => abs A + X) 0.
Definition range (P: seq C): C*C :=
  match p with
  | [] => (0,0)
  | A::Q => let R := range_ Q in (A-R,A+R)
  end.
```

3.4 Rescaling

Putting everything together, we obtain the polymorphic operations `chebyshev.basis: BasisOps`, which can readily be used to construct rigorous approximations, with the instance `MFunOps` from Section 2.3. This basis however requires to work on the domain $[-1;1]$ (for estimating the range as explained in the previous section, but also to perform interpolation, see Section 5.1). In order to use it on other domains, we provide a rescaling function that takes a `B: BasisOps` and rescales it to a given interval $[a;b]$ using the obvious affine function. We show that this operation preserves validity of basis operations, so that we can use it whenever needed.

4 Formalisation of Banach fixed-point theorem

Banach fixed-point theorem is the cornerstone of the method discussed here.

▶ **Theorem 1** (Banach fixed-point). *Let $(X, \|\cdot\|)$ be a Banach space, an operator $\boldsymbol{F} : X \to X$, $h^\circ \in X$, and $\mu, b, r \in \mathbb{R}_+$, satisfying the following conditions:*

(i) $\|h^\circ - \boldsymbol{F} \cdot h^\circ\| \leq b$;
(ii) $\boldsymbol{F}$ *is μ-Lipschitz over the closed ball* $\overline{B}(h^\circ, r) := \{h \in X \mid \|h - h^\circ\| \leq r\}$:

$$\forall h_1, h_2 \in X, \quad h_1 \in \overline{B}(h^\circ, r) \wedge h_2 \in \overline{B}(h^\circ, r) \;\Rightarrow\; \|\boldsymbol{F} \cdot h_1 - \boldsymbol{F} \cdot h_2\| \leq \mu \|h_1 - h_2\|;$$

(iii) $\mu < 1$: $\boldsymbol{F}$ *is contracting over* $\overline{B}(h^\circ, r)$;
(iv) $b + \mu r \leq r$.

Then $\boldsymbol{F}$ admits a unique fixed-point h^* in $\overline{B}(h^\circ, r)$.

This classic result has been formalised in various flavours of logic and proof assistants. In particular, Boldo et al. have provided a formal proof of a version of this fixed-point theorem, based on the Coquelicot library, for the purpose of the formalisation of the Lax-Milgram theorem [5]. Using the same backbone library, we formalise an alternative version of the theorem: our version is significantly more concise, and closer to the computational content of the result. We describe below this formalisation.

The Coquelicot library formalises topological concepts using *filters* [7, 20], which we briefly recall here. A filter on a type T is a collection of collections of inhabitants of T which is non-empty, upward closed and stable under finite intersections:

```
Record Filter (T : Type) (F : (T → Prop) → Prop) := {
  filter_true : F (fun _ => True) ;
  filter_and : ∀ P Q : T → Prop, F P → F Q → F (fun x => P x /\ Q x) ;
  filter_imp : ∀ P Q : T → Prop, (∀ x, P x → Q x) → F P → F Q }.
```

While filters are used to formalise neighbourhoods, *balls* allow for expressing the relative closeness of points in the space. Balls are formalised using a ternary relation between two points in the carrier type, and a real number, with the following axioms:

```
  ball : M → R → M → Prop ;
  ax1 : ∀ x (e > 0), ball x e x ;
  ax2 : ∀ x y e, ball x e y → ball y e x ;
  ax3 : ∀ x y z e1 e2, ball x e1 y → ball y e2 z → ball x (e1 + e2) z
```

Two points are called *close* when they cannot be separated by balls:

```
Definition close (x y : M) : Prop := ∀ eps > 0, ball x eps y.
```

A filter is called a *Cauchy filter* when it contains balls of arbitrary (small) radius:

```
Definition cauchy (T : UniformSpace) (F : (T → Prop) → Prop) :=
  ∀ eps > 0, ∃ x, F (ball x eps).
```

Finally, a *uniform space* is a type equipped with a ball relation and a *complete space* moreover has a limit operation on filters, which ensures the convergence of Cauchy sequences via the following axioms (where `ProperFilter F` is equivalent to `Filter F /\ ∀ P, F P → ∃ x, P x`):

```
  lim : ((T → Prop) → Prop) → T ;
  ax1 : ∀ F, ProperFilter F → cauchy F → ∀ eps > 0, F (ball (lim F) eps) ;
  ax2 : ∀ F1 F2, F1 ⊆  F2 → F2 ⊆ F1 → close (lim F1) (lim F2)
```

The above formal definition of balls does not enforce closedness nor openness. We thus introduced the relation associated with the *closure* of balls, so as to model *closed* neighbourhoods:

```
Definition cball x r y := ∀ e > 0, ball x (r+e) y.
```

Equipped with this definition, hypothesis (ii) of Theorem 1 is formalised as follows:

```
Definition lipschitz_on (F : U → U) (mu : R) (P : U → Prop) :=
∀ x y : U, ∀ r ≥ 0, P x → P y → cball x r y → cball (F x) (mu*r) (F y).
```

We now sketch our formalised proof, using mathematical notations. We consider a complete space X and we write $y \in B(x,r)$ for the formal `(ball x r y)`, and $y \in \overline{B}(x,r)$ for `(cball x r y)`. The key notion is that of *strongly stable ball* (see Figure 1):

▶ **Definition 2** (Strongly stable ball). *A ball $\overline{B}(u,r)$ is μ-*strongly stable *for $\boldsymbol{F}$ if $\boldsymbol{F}$ is μ-Lipschitz on $\overline{B}(u,r)$ and if there is a non-negative real number s, called the* offset*, s.t.:*

$$\boldsymbol{F}\cdot u \in \overline{B}(u,r) \qquad \textit{and} \qquad s+\mu r \;\leq\; r.$$

▶ Remark 3 (Stability). For any x in $\overline{B}(u,r)$, a strongly stable ball for $\boldsymbol{F}$, $\boldsymbol{F}\cdot x \in \overline{B}(u,r)$.

▶ Remark 4 (Contracting case). When $0 \leq \mu < 1$, for any μ-strongly stable ball $\overline{B}(v,\rho)$, with offset σ, $\overline{B}(\boldsymbol{F}\cdot v, \mu\rho)$ is also a strongly stable ball, with offset $\mu\sigma$. Moreover, $\overline{B}(\boldsymbol{F}\cdot v, \mu\rho)$ is included in $\overline{B}(v,\rho)$.

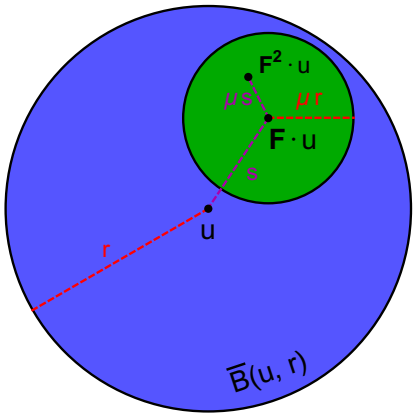


Figure 1 Balls $\overline{B}_0$ and $\overline{B}_1$.

Assume that $\boldsymbol{F}$ has a μ-strongly stable ball $\overline{B}(u,r)$ of offset s, with $\mu < 1$. In particular, $\boldsymbol{F}$ is contracting on $\overline{B}(u,r)$. Consider the sequence of balls defined as follows:

$$\overline{B}_n = \overline{B}(u_n, r_n) \quad \text{with} \quad u_n = \boldsymbol{F}^n \cdot u \quad \text{and} \quad r_n = r\mu^n$$

where $\boldsymbol{F}^n \cdot u$ denotes the iterated images of u under $\boldsymbol{F}$. By Remark 4, $(\overline{B}_n)_{n\in\mathbb{N}}$ is a nested sequence of μ-strongly stable ball for $\boldsymbol{F}$, with offset $s\mu^n$. Let $\mathcal{F}$ be the family of collections of points in U defined as:

$$\mathcal{F} = \{P \subseteq U \mid \exists\, n, \overline{B}_n \subseteq P\}.$$

It is a proper filter: $\mathcal{F}$ contains U, it is obviously upward closed, and for $P, Q \in \mathcal{F}$, $P \cap Q$ is also in $\mathcal{F}$ because $(\overline{B}_n)_{n\in\mathbb{N}}$ is decreasing for inclusion. Thus $\mathcal{F}$ has a limit w, such that for any $\varepsilon > 0$, balls $\overline{B}_n$ are eventually included in $B(w,\varepsilon)$. We provide a formal proof of Theorem 5, a reformulation of Theorem 1 using the vocabulary of the Coquelicot library:

▶ **Theorem 5.** *The limit w of the filter $\mathcal{F}$ is in $\overline{B}_0$, and w is a fixed point of $\boldsymbol{F}$. Moreover, w is close to every other fixed point of $\boldsymbol{F}$ in $\overline{B}_0$.*

Proof. In this statement "w is a fixed point of $\boldsymbol{F}$" means "w is close to $\boldsymbol{F}\cdot w$". First, $w \in \overline{B}_n$ for all n. Indeed, for any $\varepsilon > 0$, there is an $m \geq n$ s.t. $\overline{B}_m \subseteq B(w,\varepsilon)$, and since $\overline{B}_m \subseteq \overline{B}_n$, $u_m \in \overline{B}_n \cap B(w,\varepsilon)$. In particular, $w \in \overline{B}_0$. It is also clear by stability that $\boldsymbol{F}\cdot w \in \overline{B}_n$ for all n. Moreover, w is close to any point v s.t. $v \in \overline{B}_n$ for all n (for any $\varepsilon > 0$, choose n s.t. $2\mu r^n < \varepsilon$). Taking $v := \boldsymbol{F}\cdot w$ proves that w is a fixed point of $\boldsymbol{F}$.

Finally, if $w' \in \overline{B}_0$ is another fixed point of $\boldsymbol{F}$, then it follows from an easy induction that $w' \in \overline{B}_n$ for all n. Hence, the foregoing shows that w is close to w'. ◀

Strongly stable balls model the requirements set on the untrusted data to be formally verified. They can also be seen as balls centered at the initial point, and large enough to include all its successive iterates, i.e. as instances of the locus at stake in classical presentations of the proof. The version proved by Boldo et al. has a slightly more technical wording, which seems to be made necessary by its further usage in the verification of the Lax-Milgram theorem. Our proof script is significantly shorter, partly because we automate proofs of positivity conditions (for radii of balls) using canonical structures for manifestly positive expressions. But the key ingredient for concision is to make most of the filter device in the proof, and to refrain from resorting to low-level properties of geometric sequences. To

the best of our knowledge, the other libraries of formalised analysis featuring a proof of this result, notably Isabelle/HOL and HOL-Light, are based on variant of proof strategy closer to the approach of Boldo et al. than to ours.

5 Newton-like validation operators

The purpose of this section is twofold. We first present the general principle of fixed-point based *a posteriori* validation methods, and more particularly, the use of Newton-like validation operators. Then we apply it to the division and square root of models.

Throughout this section, let $(X, \|\cdot\|)$ denote a Banach space, and h^* the exact solution of an equation in X. In this article, X stands for the space $\mathcal{C}(I)$ of real-valued continuous functions defined over a compact segment $I = [a, b]$, with the uniform norm $\|h\| := \sup_{x\in I} |h(x)|$. The division and square root of functions are simple examples of solutions of equations in $\mathcal{C}(I)$, but there are also differential equations, integral equations, delay equation, etc. The general scheme for Banach fixed-point based *a posteriori* validation methods follows two steps:

1. **Approximation step.** A numerical approximation $h^\circ \in X$ of h^* is obtained by an oracle, which may resort to any approximation method. In particular, this step requires no mathematical assumption and can be executed purely numerically outside the proof assistant, good approximation properties being only desirable for efficiency. In our setting with $X = \mathcal{C}(I)$, interpolation at Chebyshev nodes (Section 5.1) is an efficient and accurate oracle for a wide range of function space problems.
2. **Validation step.** The initial problem is rephrased in such a way that h^* is a fixed point of a (locally) contracting operator $\boldsymbol{F} : X \to X$. An *a posteriori* error bound on $\|h^\circ - h^*\|$ is deduced from the Banach fixed-point theorem (Theorem 1).

We thus need to find a contracting operator $\boldsymbol{F}$ of which h^* is a fixed point. To this end, we use Newton-like validation methods, which transform an equation $\boldsymbol{M} \cdot h = 0$ into an equivalent fixed-point equation $\boldsymbol{F} \cdot h = h$ with $\boldsymbol{F}$ contracting. More specifically, suppose that $\boldsymbol{M} : X \to Y$ is differentiable; we use a Newton-like operator $\boldsymbol{F} : X \to X$ defined as:

$$\boldsymbol{F} \cdot h = h - \boldsymbol{A} \cdot \boldsymbol{M} \cdot h, \qquad h \in X,$$

with $\boldsymbol{A} : Y \to X$ an *injective bounded linear* operator, intended to be close to $(\mathcal{D}\boldsymbol{M}_{h^\circ})^{-1}$. The operator $\boldsymbol{A}$ may be given by an oracle and does not need to be this exact inverse, which anyway might be non representable on computers exactly. The mean value theorem yields a Lipschitz ratio μ for $\boldsymbol{F}$ over any convex subset S of X:

$$\forall h_1, h_2 \in S,\ \|\boldsymbol{F}\cdot h_1 - \boldsymbol{F}\cdot h_2\| \le \mu \|h_1 - h_2\|, \quad \text{with } \mu = \sup_{h\in S} \|\mathcal{D}\boldsymbol{F}_h\| = \sup_{h\in S} \|\mathbf{1}_X - \boldsymbol{A}\cdot\mathcal{D}\boldsymbol{M}_h\|,$$

which is expected to be small over some neighbourhood of h°.

Concretely, in order to apply Theorem 1, one needs to compute the following quantities:

- a bound $b \ge \|\boldsymbol{A} \cdot \boldsymbol{M} \cdot h^\circ\| = \|h^\circ - \boldsymbol{F} \cdot h^\circ\|$;
- a bound $\mu_0 \ge \|\mathbf{1}_X - \boldsymbol{A} \cdot \mathcal{D}\boldsymbol{M}_{h^\circ}\| = \|\mathcal{D}\boldsymbol{F}_{h^\circ}\|$;
- a bound $\mu'(r) \ge \|\boldsymbol{A} \cdot (\mathcal{D}\boldsymbol{M}_h - \mathcal{D}\boldsymbol{M}_{h^\circ})\| = \|\mathcal{D}\boldsymbol{F}_h - \mathcal{D}\boldsymbol{F}_{h^\circ}\|$ valid for any $h \in B(h^\circ, r)$, and parameterised by a radius $r \in \mathbb{R}_+$.

If we are able to find a radius $r \in \mathbb{R}_+$ satisfying:

$$\mu(r) := \mu_0 + \mu'(r) < 1, \qquad \text{and} \qquad b + r\mu(r) \ \le\ r, \tag{4}$$

then Theorem 1 guarantees the existence and uniqueness of a root h^* of $\boldsymbol{M}$ in $B(h^\circ, r)$.

▶ Remark 6. Finding an r as small as possible while satisfying (4) may be an nontrivial task for automated validation procedures. For many problems, $\mu'(r)$ is polynomial, hence conditions (4) are polynomial inequalities over r: this is called the *radii polynomial approach* [21] in rigorous numerics. In our case, division (resp. square root) induces an affine (resp. quadratic) equation, which admits closed form solutions.

5.1 Approximation step: interpolation

Since they are certified a posteriori, (non-rigorous) approximations for division and square root of given models can be obtained using arbitrary numerical techniques. We use interpolation at Chebyshev nodes of the second kind for its efficiency and excellent approximation properties [35].

Ideally, we would implement this operation outside of the proof assistant, in order not to pay the price of an interpreted language. This would however require a lot of work in order to design a proper interface between Coq values and external values (e.g., converting Coq representation of floating points numbers into machine level floating points, and back). Instead, and for now, we implement the oracles inside Coq, as unspecified functions. To this end, we add a field to the structure `BasisOps_on`, to compute interpolants of a given degree:

```
interpolate: nat → (C → C) → seq C;
```

We implement this operation for Chebyshev basis using the discrete orthogonality relations on Chebyshev polynomials.

To reduce the price of staying inside Coq for those computations, we exploit the polymorphism built in our framework to perform those computations on floating-point numbers rather than intervals. To this end, we add the following fields to the structure `NBH`:

```
FF: Ops1;          (* abstract type for floating points and their operations *)
I2F: II → FF;      (* conversion from intervals to floating points to (midpoint) *)
F2I: FF → II;      (* conversion from floating points to intervals (singleton) *)
```

Equipped with these operations, we can define conversion operations between models (on intervals) and polynomials with floating point coefficients:

```
Definition mcf (M: Model): seq FF := map I2F (pol M).
Definition mfc (p: seq FF): Model := {| pol := map F2I p; rem := 0 |}.
```

The field `FF`, of type `Ops1`, will make it possible to call the functions `interpolate` and `beval` from the basis with `C` as `FF`, i.e., to let them operate on floating point numbers. By doing so we do not have to reimplement Clenshaw's evaluation scheme on floating point numbers.

5.2 Validation step for division

For $f, g \in \mathcal{C}(I)$ with g nonvanishing over I, the quotient f/g is the unique root of $\boldsymbol{M} : h \mapsto gh - f$. Let h° be a candidate approximation given by the approximation step. Constructing the Newton-like operator $\boldsymbol{F}$ requires an approximation $\boldsymbol{A}$ of $(\mathcal{D}\boldsymbol{M}_{h^\circ})^{-1} : k \mapsto k/g$. For that purpose, suppose $w \approx 1/g \in \mathcal{C}(I)$ is also given by an oracle, and define:

$$\boldsymbol{F} \cdot h = h - w(gh - f). \tag{5}$$

The next proposition computes an upper bound for $\|h^\circ - f/g\|$; it is implemented in `div.newton`.

▶ **Proposition 7.** *Let $f, g, h^\circ, w \in \mathcal{C}(I)$, and $\mu, b \in \mathbb{R}_+$ such that:*

(7 i) $\|w(gh^\circ - f)\| \le b$, (7 ii) $\|1 - wg\| \le \mu$, (7 iii) $\mu < 1$.

Then g does not vanish over I and $\|h^\circ - f/g\| \le b/(1-\mu)$.

Proof. Conditions (7 ii) and (7 iii) imply that $\boldsymbol{F}$ (Equation (5)) is contracting over $\mathcal{C}(I)$ with ratio μ. The radius $r := \frac{b}{1-\mu}$ makes the ball $\overline{B}(h^\circ, r)$ strongly stable with offset b (7 i), since $b + \mu r = r$. Therefore, h^* is the (global) unique root of $\boldsymbol{M}$, and $\|h^\circ - h^*\| \le r$.

Finally, w and g do not vanish because $\|1 - wg\| \le \mu < 1$. Hence, $h^* = f/g$ over I. ◀

The concrete division of models is implemented as follows:

```
Definition mdiv_aux (F G H W: Model): Model :=
 let K1 := 1-W*G in
 let K2 := W*(G*H - F) in
 match mag (mrange K1), mag (mrange K2) with
 | Some mu, Some b when is_lt mu 1 => {| pol := pol H; rem := rem H + sym (b/(1-mu)) |}
 | _ => mbot
 end.
Definition mdiv n (F G: Model): Model :=
  let p, q := mcf F, mcf G in
  mdiv_aux F G (mfc (interpolate n (fun x => beval p x / beval q x)))
               (mfc (interpolate n (fun x => 1 / beval q x))).
```

Note that we use the trivial model `mbot={|pol:=[];rem:=bot|}` as a default value, when the concrete computations fail to validate the guess of the oracle (either because this guess is just wrong, or because of over-approximations in the computations). The correctness lemmas use the properties of operations on models to prove the assumptions of `div.newton`.

```
Lemma rmdiv_aux F f G g H h W w:
  mcontains F f → mcontains G g → mcontains H h → mcontains W w →
  mcontains (mdiv_aux F G H W) (f/g).
Lemma rmdiv n F f G g: mcontains F f → mcontains G g → mcontains (mdiv' n F G) (f/g).
```

5.3 Validation step for square root

Let $f \in \mathcal{C}(I)$ be strictly positive over I. The square root $\sqrt{f}$ is one of the two roots of the quadratic equation $\boldsymbol{M} \cdot h := h^2 - f = 0$ (the other being $-\sqrt{f}$). Let h° be a candidate approximation. Since $\mathcal{D}\boldsymbol{M}_h : k \mapsto 2hk$, one also needs an approximation $w \approx 1/(2h^\circ) \approx 1/(2\sqrt{f}) \in \mathcal{C}(I)$ in order to define $\boldsymbol{A} : k \mapsto wk$, approximating $(\mathcal{D}\boldsymbol{M}_{h^\circ})^{-1}$. Then:

$$\boldsymbol{F} : \quad h \quad \mapsto \quad h - w(h^2 - f). \tag{6}$$

The next proposition (implemented by `sqrt.newton`), computes an upper bound for $\|h^\circ - \sqrt{f}\|$.

▶ **Proposition 8.** *Let $f, h^\circ, w \in \mathcal{C}(I)$, $\mu_0, \mu_1, b \in \mathbb{R}_+$ and $t_0 \in I$ such that:*

(8 i) $\left\| w\left(h^{\circ 2} - f\right)\right\| \le b$, (8 ii) $\|1 - 2wh^\circ\| \le \mu_0$, (8 iii) $\|w\| \le \mu_1$,

(8 iv) $\mu_0 < 1$, (8 v) $(1-\mu_0)^2 - 8b\mu_1 \ge 0$, (8 vi) $w(t_0) > 0$.

Then $f > 0$ over I and $\left\|h^\circ - \sqrt{f}\right\| \le r^*$, where:

$$r^* := \frac{1 - \mu_0 - \sqrt{(1-\mu_0)^2 - 8b\mu_1}}{4\mu_1}.$$

Proof. First, since $\|1 - 2wh^\circ\| \leq \mu_0 < 1$ (by (8 ii) and (8 iv)) and $w(t_0) > 0$ (8 vi), w and h° are strictly positive over I, by continuity. Using (8 iii), $\mu_1 > 0$.

If $b = 0$, then $r^* = 0$ and $h^\circ = \sqrt{f}$ over I, because $w(h^{\circ 2} - f) = 0$ (8 i) and $w, h^\circ > 0$. Hence the conclusion holds.

From now on, we assume $b > 0$. $\boldsymbol{F}$ is Lipschitz of ratio $\mu(r) := \mu_0 + 2\mu_1 r$ over $\overline{B}(h^\circ, r)$ for any $r \in \mathbb{R}_+$, because:

$$\boldsymbol{F}\cdot h_1 - \boldsymbol{F}\cdot h_2 = (h_1 - h_2) - w(h_1^2 - h_2^2) = [(1 - 2wh^\circ) + w(h^\circ - h_1) + w(h^\circ - h_2)]\,(h_1 - h_2).$$

Therefore, satisfying $b + \mu(r)r \leq r$ is equivalent to the quadratic inequality:

$$2\mu_1 r^2 + (\mu_0 - 1)r + b \leq 0. \tag{7}$$

Condition (8 v) implies that (7) admits solutions, and r^* is the smallest one. Moreover, since $b, \mu_1 > 0$, we get $r^* > 0$, so that $b + \mu(r^*)r^* = r^*$ also implies $\mu(r^*) < 1$.

Now, all the assumptions of Theorem 1 are fulfilled. Hence, $\boldsymbol{F}$ has a unique fixed point h^* in $\overline{B}(h^\circ, r^*)$. To obtain $h^* = \sqrt{f}$ over I, it remains to show that $h^* > 0$. This follows from $w > 0$ and:

$$\|1 - 2wh^*\| \leq \|1 - 2wh^\circ\| + \|2w(h^* - h^\circ)\| \leq \mu_0 + 2\mu_1 r^* = \mu(r^*) < 1. \qquad \blacktriangleleft$$

▶ Remark 9. Contrary to the case of division where continuity was not needed at all, it is here used for w. Therefore, `sqrt.newton` requires w to be continuous over I.

The Coq code for the corresponding operations on models `msqrt_aux` and `msqrt`, together with the statements of their correctness lemmas, are given in Appendix B.

6 Examples

6.1 Playing with approximations of the absolute value function

Consider the function $f_\varepsilon : x \mapsto \sqrt{\varepsilon + x^2}$ over $[-1, 1]$, with $\varepsilon > 0$. When $\varepsilon \to 0$, f_ε converges uniformly to the absolute value function $x \mapsto |x|$ (which is not analytic at 0), with:

$$|f(x) - |x|| = \left|\sqrt{\varepsilon + x^2} - \sqrt{x^2}\right| = \left|\frac{\varepsilon}{\sqrt{\varepsilon + x^2} + \sqrt{x^2}}\right| \leq \sqrt{\varepsilon}. \tag{8}$$

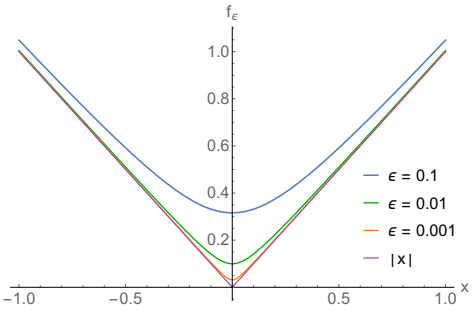
Rigorous uniform approximations. Approximating f_ε with polynomials becomes harder for small ε, due to the complex singularities $\pm i\sqrt{\varepsilon}$ getting closer to the interval $[-1, 1]$. Nevertheless, Chebyshev interpolation still works and our implementation computes rigorous approximations as accurate as desired (see Figure 2b), of exponential convergence with ratio determined by ε. Note that for too small degree, the computed approximation of the square root is too far from the solution, and the a posteriori validation returns an infinite remainder.

In order to provide a comparison with CoqApprox's Taylor models, we used the tactic `interval with (i_depth 1, i_bisect_taylor x N, i_prec p)` to build a Taylor model of degree N with precision p. Timings given in Table 2c reveal a significant advantage of our implementation (there we use $\varepsilon = 2$ to avoid convergence issues of Taylor models). Concerning accuracy, our experiments tend to show that when $\varepsilon \leq 1$, CoqApprox fails to compute *converging* Taylor models. Indeed, even with large L, a goal like:

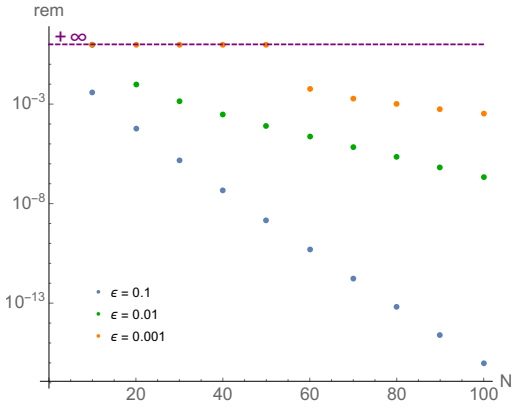
```
Goal Fail : ∀ x : R, -1 ≤ x ≤ 1 → sqrt (1/100+x*x) ≤ L
```

is not solved when the degree N becomes too large, probably indicating that the Taylor models *diverge* due to complex singularities inside the unit disk. (Note that the `interval` tactic can solve this goal, but only by resorting to subdivision techniques.)

Error bounding. We want to bound $|f_\varepsilon(x) - |x||$ for $x \in [-1, 1]$ without making use of any symbolic manipulation like (8). At first glance, one can choose to use the rigorous approximations over $[-1, 1]$ obtained previously, and evaluate $f_\varepsilon(x) - x$ (resp. $f_\varepsilon(x) + x$) over $[0, 1]$ (resp. $[-1, 0]$) using Clenshaw's algorithm. However, even if the approximations are quite good, this evaluation strategy gives huge overestimations because $[0, 1]$ and $[-1, 0]$ are not small intervals. Instead, we compute separately two approximations for f_ε: one over $[0, 1]$ and one over $[-1, 0]$, and we evaluate $f_\varepsilon(x) - x$ (resp. $f_\varepsilon(x) + x$) over $[1, 0]$ (resp. $[-1, 0]$) using the Chebyshev `range` function. This approach yields bounds that are rather close to the optimal $\sqrt{\varepsilon}$ (see Figure 2d). However, this does not allow for arbitrary accuracy: a subdivision procedure would be necessary here.



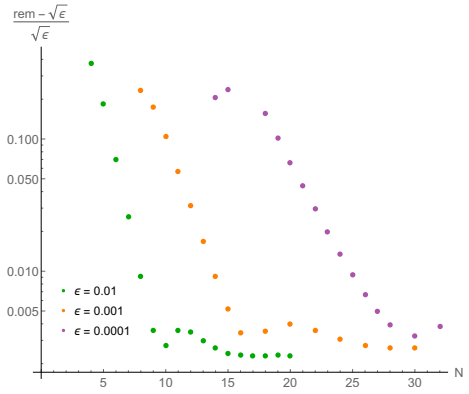
(a) Functions f_ε and $x \mapsto |x|$ over $[-1, 1]$.



(b) Magnitude of the remainders of degree-N Chebyshev models approximating f_ε over $[-1, 1]$.

N	time (in seconds)	
	Chebyshev	CoqApprox
10	0.11	0.10
20	0.16	0.12
30	0.22	0.14
40	0.31	0.20
50	0.42	0.29
60	0.56	0.44
70	0.71	0.64
80	0.89	0.93
90	1.08	1.33
100	1.31	1.84
120	1.80	3.34
140	2.43	5.60
160	2.98	8.89
180	3.86	13.47
200	4.75	19.71

(c) Timings of degree-N models for f_2.



(d) Overapproximation ratio of the remainder of the degree-N Chebyshev model for $f_\varepsilon - 1$ over $[0, 1]$, compared to the optimal bound $\sqrt{\varepsilon}$.

Figure 2 Approximating functions f_ε and $x \mapsto |x|$ with Chebyshev models.

6.2 Evaluating an Abelian integral

Abelian integrals naturally appear when computing the number of limit cycles bifurcating from a Hamiltonian polynomial vector field in the plane. Indeed, the number of sign alternations of those contour integrals (parameterised by the energy level of the potential function) gives a lower bound on the number of limit cycles of the perturbed system, which

is a hard question related to Hilbert's 16th problem.

In [24], the author claims to prove the existence of 26 limit cycles for a well-constructed quartic system, whereas the previous record for degree 4 was 22 [13]. However, the implementation with which the Abelian integrals were "rigorously" computed was erroneous, which led to apparently more sign alternations than in reality. By tuning the coefficients and computing the integrals with another rigorous numerics library, the authors of the ongoing work [9] obtain 24 limit cycles, which, if not 26, is still greater than the current record 22.

To conclude this article, we rigorously evaluate some of these integrals inside Coq to show how our implementation behaves on non-crafted examples. Below are the formulas defining a family of integral $\mathfrak{I}_{ij}(r)$ which need to be computed precisely for several values of r. Table 1 summarises the results of our computational experiments. In each line, we chose parameters that were enough to obtain the desired precision. These encouraging results give us hope that it will be possible to fully verify the critical computations involved in recent work of the first author [9].

$$\mathfrak{I}_{ij}(r) = \int_{x_-}^{x_+} x^i (y^+(x)^{j-1} - y^-(x)^{j-1})\,\mathrm{d}x + \int_{y_-}^{y_+} (x^-(y)^{i-1} + x^+(y)^{i-1}) y^j \frac{y^2 - y_0}{\delta_x(y)}\,\mathrm{d}y.$$

$$x_0 = \frac{9}{10}, \quad x_\pm = \sqrt{x_0 \pm r/\sqrt{2}}\ , \quad \delta_y(x) = \sqrt{r^2 - (x^2 - x_0)^2}\ , \quad x^\pm(y) = \sqrt{x_0 \pm \delta_x(y)}\ ,$$
$$y_0 = \frac{11}{10}, \quad y_\pm = \sqrt{y_0 \pm r/\sqrt{2}}\ , \quad \delta_x(y) = \sqrt{r^2 - (y^2 - y_0)^2}\ , \quad y^\pm(x) = \sqrt{y_0 \pm \delta_y(x)}\ .$$

r	N	p	time (s)	$\mathfrak{I}_{00}$	$\mathfrak{I}_{20}$	$\mathfrak{I}_{22}$	$\mathfrak{I}_{40}$	$\mathfrak{I}_{04}$
0.5	13	32	0.38	2,4e-05	2,9e-05	4,1e-05	3,0e-05	4,8e-05
0.78	15	32	0.47	4,6e-05	2,0e-05	2,7e-05	2,4e-05	1,1e-04
0.88	65	128	17.34	2,5e-08	5,0e-11	8,5e-11	5,3e-11	6,3e-08
0.89	95	128	35.13	2,0e-08	1,8e-11	2,9e-11	2,0e-11	5,1e-08
0.895	135	300	173.23	2,6e-08	1,7e-11	1,8e-11	1,3e-11	6,7e-08

Table 1 Reached precision for $\mathfrak{I}_{ij}(r)$ for different values of r, computed with degree-N Chebyshev models and floating-point precision p (in each cell we display the width of the computed enclosure).

7 Conclusion and future work

The Coq development is available online [11]. It consists of around 1300 lines of specifications and 1500 lines of proofs. We leave several directions for future work: integrate it with CoqInterval to benefit from its automatic subdivision techniques; interface the library with external tools for the approximation steps; implement other bases; address higher-dimensional problems. Applying this approach to verify solutions of linear ODEs in a systematic way [2, 8] is also a longer-term perspective.

References

1 Henk Barendregt and Erik Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28(3):321–336, Apr 2002.

2 Alexandre Benoit, Mioara Joldeş, and Marc Mezzarobba. Rigorous uniform approximation of D-finite functions using Chebyshev expansions. *Math. Comp.*, 86(305):1303–1341, 2017. URL: https://doi.org/10.1090/mcom/3135.

3 Vasile Berinde. *Iterative approximation of fixed points*, volume 1912 of *Lecture Notes in Mathematics*. Springer, Berlin, 2007.

4 Martin Berz and Kyoko Makino. Verified integration of odes and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4):361–369, 1998.

5 Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq formal proof of the Lax–Milgram theorem. In *6th ACM SIGPLAN Conference on Certified Programs and Proofs*, Paris, France, January 2017. URL: https://hal.inria.fr/hal-01391578, doi:10.1145/3018610.3018625.

6 Sylvie Boldo and Guillaume Melquiond. *Verifying Floating-point Algorithms with the Coq System*. Elsevier, 2017.

7 Nicolas Bourbaki. *General Topology*. Springer, 1995. Original French edition published by MASSON, Paris, 1971. doi:10.1007/978-3-642-61701-0.

8 Florent Bréhard, Nicolas Brisebarre, and Mioara Joldes. Validated and numerically efficient Chebyshev spectral methods for linear ordinary differential equations. *ACM Transactions on Mathematical Software*, 2018.

9 Florent Bréhard, Nicolas Brisebarre, Mioara Joldes, and Warwick Tucker. A New Lower Bound on the Hilbert Number for Quartic Systems, 2019. URL: http://www.jncf2019.uvsq.fr/program/abs-brehard.pdf.

10 Nicolas Brisebarre and Mioara Joldeş. Chebyshev interpolation polynomial-based tools for rigorous computing. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 147–154. ACM, 2010.

11 Florent Bréhard, Assia Mahboubi, and Damien Pous. Web appendix to the present paper. https://gitlab.inria.fr/amahboub/approx-models.

12 Olga Caprotti and Martijn Oostdijk. Formal and efficient primality proofs by use of computer algebra oracles. *J. Symb. Comput.*, 32(1/2):55–70, 2001.

13 Colin Christopher. Estimating limit cycle bifurcations from centers. In *Differential equations with symbolic computation*, Trends Math., pages 23–35. Birkhäuser, Basel, 2005. doi:10.1007/3-7643-7429-2_2.

14 Tobin A Driscoll, Nicholas Hale, and Lloyd N Trefethen. Chebfun guide, 2014.

15 L. Fox and I. B. Parker. *Chebyshev polynomials in numerical analysis*. Oxford University Press, London-New York-Toronto, Ont., 1968.

16 Benjamin Grégoire and Assia Mahboubi. Proving equalities in a commutative ring done right in coq. In Joe Hurd and Thomas F. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005. URL: https://doi.org/10.1007/11541868_7, doi:10.1007/11541868_7.

17 Benjamin Grégoire and Laurent Théry. A purely functional library for modular arithmetic and its application to certifying large prime numbers. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2006.

18 Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015. URL: http://arxiv.org/abs/1501.02155.

19 John Harrison and Laurent Théry. A skeptic's approach to combining HOL and maple. *J. Autom. Reasoning*, 21(3):279–294, 1998.

20 Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in isabelle/hol. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 279–294, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

21 Allan Hungria, Jean-Philippe Lessard, and Jason D. Mireles James. Rigorous numerics for analytic solutions of differential equations: the radii polynomial approach. *Math. Comp.*, 85(299):1427–1459, 2016.

22 Fabian Immler. A verified ODE solver and the Lorenz attractor. *Journal of automated reasoning*, pages 1–39, 2018.

23 Fredrik Johansson. Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, 66(8):1281–1292, 2017.

24 Tomas Johnson. A quartic system with twenty-six limit cycles. *Exp. Math.*, 20(3):323–328, 2011. doi:10.1080/10586458.2011.565252.

25 Mioara Joldeş. *Rigorous Polynomial Approximations and Applications*. PhD thesis, École normale supérieure de Lyon – Université de Lyon, Lyon, France, 2011. URL: https://tel.archives-ouvertes.fr/tel-00657843.

26 Edgar W Kaucher and Willard L Miranker. *Self-validating numerics for function space problems: Computation with guarantees for differential and integral equations*, volume 9. Elsevier, 1984.

27 Rudi Klatte, Ulrich Kulisch, Andreas Wiethoff, and Michael Rauch. *C-XSC: A C++ class library for extended scientific computing*. Springer Science & Business Media, 2012.

28 Jean-Philippe Lessard and Christian Reinhardt. Rigorous numerics for nonlinear differential equations using Chebyshev series. *SIAM J. Numer. Anal.*, 52(1):1–22, 2014.

29 Assia Mahboubi, Guillaume Melquiond, and Thomas Sibut-Pinote. Formally verified approximations of definite integrals. *Journal of Automated Reasoning*, 62(2):281–300, Feb 2019. URL: https://doi.org/10.1007/s10817-018-9463-7, doi:10.1007/s10817-018-9463-7.

30 Érik Martin-Dorel and Guillaume Melquiond. Proving tight bounds on univariate expressions with elementary functions in coq. *Journal of Automated Reasoning*, 57(3):187–217, Oct 2016. URL: https://doi.org/10.1007/s10817-015-9350-4, doi:10.1007/s10817-015-9350-4.

31 Guillaume Melquiond. Proving bounds on real-valued functions with computations. In *International Joint Conference on Automated Reasoning*, pages 2–17. Springer, 2008.

32 Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

33 Nathalie Revol and Fabrice Rouillier. Motivations for an arbitrary precision interval arithmetic and the mpfi library. *Reliable computing*, 11(4):275–290, 2005.

34 Siegfried M Rump. Intlab—interval laboratory. In *Developments in reliable computing*, pages 77–104. Springer, 1999.

35 Lloyd Nicholas Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2013. See http://www.chebfun.org/ATAP/. URL: http://www.chebfun.org/ATAP/.

36 Warwick Tucker. A rigorous ODE solver and Smale's 14th problem. *Found. Comput. Math.*, 2(1):53–117, 2002. URL: http://www.math.cornell.edu/~warwick/main/rodes/JFoCM.pdf.

37 Warwick Tucker. *Validated numerics: a short introduction to rigorous computations*. Princeton University Press, 2011.

38 Jan Bouwe Van Den Berg and Jean-Philippe Lessard. Chaotic braided solutions via rigorous numerics: Chaos in the Swift–Hohenberg equation. *SIAM Journal on Applied Dynamical Systems*, 7(3):988–1031, 2008.

39 Nobito Yamamoto. A numerical verification method for solutions of boundary value problems with local uniqueness by Banach's fixed-point theorem. *SIAM J. Numer. Anal.*, 35(5):2004–2013, 1998.

A Coq code for primitive in Chebyshev basis

```
Fixpoint prim_ (C: Ops1) (n: nat) (p: seq C): seq C :=
  match P,n with
    | [],_ => []
    | a::q,0 => sadd [0; a] (prim_ 1 q)
    | a::q,1 => 0 :: (sadd [0; a/4] (prim_ 2 q))
    | a::q,_ => sadd [-a/(2(n-1)); 0; a/(2(n+1))] (0 :: prim_ (n+1) Q)
  end.
Definition prim (C: Ops1) (P: seq C) := prim_ 0 P.
```

The key lemma is the following one:

```
Lemma eval_prim_ n (p: seq R) x: Derive (eval_ T (n-1) (prim_ n P)) x = eval_ T n P x.
```

B Coq code for the square root of a model

```
Let msqrt_aux (F H W: Model) (x: II): Model :=
  let Wx := meval W x in
  if ~~ (is_lt lo x && is_lt x hi && is_lt 0 Wx) then mbot else
  let K1 := 1 - 2*W*H in
  let K2 := W*(H*H-F) in
  match mag (mrange K1), mag (mrange W), mag (mrange K2) with
  | Some mu0, Some mu1, Some b =>
    let delta := (1 - mu0)^2 - 8*b*mu1 in
    let rmin := (1 - mu0 - sqrt delta)/(4*mu1) in
    let mu := mu0 + 2*mu1*rmin in
    if is_lt mu0 1 && is_lt 0 delta && is_lt mu' 1 then
    {| pol := pol H; rem := rem H + sym rmin' |}
    else mbot
  | _ => mbot
  end.

Let msqrt n (F: Model): Model :=
  let p: seq FF := mcf F in
  let h: seq FF := interpolate n (fun x => sqrt (beval p x)) in
  msqrt_aux M (mfc h) (mfc (interpolate n (fun x => 1/(2*beval h x)))) ((lo+hi)/2).

Lemma rmsqrt_aux (F H W: Model) (X: II) (f h w : R → R) (x: R):
  mcontains F f → mcontains H h → mcontains W w → contains X x → lo≤x≤hi →
  (∀ x, lo≤x≤hi → continuity_pt w x) →
  mcontains (msqrt_aux F H W X) (sqrt f).

Lemma rmsqrt n F f: mcontains F f → mcontains (msqrt' n F) (sqrt f).
```