Formal verification of exact computations using Newton's method

Nicolas Julien, Ioana Pasca

Formal verification of exact computations using Newton's method

Nicolas Julien, Ioana Paşca

INRIA Sophia Antipolis
[Nicolas.Julien|Ioana.Pasca]@sophia.inria.fr

Abstract. We are interested in the certification of Newton's method. We use a formalization of the convergence and stability of the method done with the axiomatic real numbers of Coq's Standard Library in order to validate the computation with Newton's method done with a library of exact real arithmetic based on co-inductive streams. The contribution of this work is twofold. Firstly, based on Newton's method, we design and prove correct an algorithm on streams for computing the root of a real function in a lazy manner. Secondly, we prove that rounding at each step in Newton's method still yields a convergent process with an accurate correlation between the precision of the input and that of the result. An algorithm including rounding turns out to be much more efficient.

1 Introduction

The Standard Library of the Coq proof assistant [4, 1] contains a formalization of real numbers based on a set of axioms. This gives the real numbers all the desired theoretical properties and makes theorem proving more agreeable and close to "pencil and paper" proofs [16]. However, this formalization has no (or little) computational meaning. During this paper we shall refere to the reals from this implementation as "axiomatic reals". We note that Coq is not an special case and proof assistants in general provide libraries with results from real analysis [5, 7, 8, 10], but with formalizations for real numbers that are not well suited for computations. However, in a proof process, it is often the case that we are interested in computing with the real numbers (or at least approximating such computations), so a considerable effort has been invested in having libraries of exact computations for proof systems [13, 15, 18]. We shall refer to numbers from such implementations as "exact reals". These libraries provide certification of computations for a set of operations and elementary functions on real numbers.

The results in this paper are concerned with Newton's method. Under certain conditions, this method ensures the convergence at a certain speed towards a root of the given function, the unicity of this root in a certain domain and the local stability. But, as the "paper" proof for these results depends on non-trivial theorems from analysis like the mean value theorem and concepts like continuity, derivation etc. the formal development conducted around them is based on the axiomatic reals of Coq. We would like to transfer these "theoretical" properties

to the computations done with exact reals. Our work is thus conducted in two directions. On one side we are interested in proving correct Newton's method on exact reals and having algorithms that are suited for our implementation of the real numbers as co-inductive streams. On the other hand we are concerned in providing appropriate theoretical results to support the correctness of the algorithms and optimizations we make.

The paper is organized as follows: in section 2 we present the theoretical results around Newton's method that have been verified with the axiomatic reals in Coq. This section gives the formalization of well-known results in [6] and presents a new proof that was motivated by our implementation of the method on exact reals. To clarify the need for this proof, in section 3 we present a library of exact real arithmetic implemented with Coq's co-inductive streams and we discuss how computations with Newton's method can be certified in this setting. We also design and prove correct an algorithm for computing the root of a function that is based on Newton's method and is adapted for streams. However, this algorithm is much more efficient when rounding is used during the process. The theorem we present in section 2.1 justifies this optimization, though the optimized algorithm is not completely certified. The applications of our algorithm are given in section 4.4 along with perspectives opened by the suggested improvements. We finish by discussing related work in section 5 as well as conclusions and possible extensions of our work in section 6.

2 Kantorovitch's theorem and related results

Kantorovitch's theorem gives sufficient conditions for the convergence of Newton's method towards the root of a given function and establishes the unicity of this root in a certain domain. A version of this theorem as well as results concerning the speed for the convergence of the process and its stability are discussed in [6]. Preliminary results around a formalization of these theorems inside the Coq proof assistant are described in [19]. At present all the theorems listed in this section are verified in the Coq proof assistant. The formal proof is based on the axiomatic real numbers from Coq's Standard Library. This choice is motivated by the concepts we needed to handle, as the library contains results from real analysis concerning convergence, continuity, derivability etc. The theorems listed bellow illustrate the type of concepts involved in the proof.

Theorem 1 (Existance) *Consider an equation $f(x) = 0$, where $f :]a, b[\to \mathbb{R}$, a, $b \in \mathbb{R}$ $f(x) \in C^{(1)}(]a, b[)$. Let $x^{(0)}$ be a point contained in $]a, b[$ with its closed ε-neighborhood $\overline{U_\varepsilon}(x^{(0)}) = \{|x - x^{(0)}| \leq \varepsilon\} \subset]a, b[$. If the following conditions hold:*

1. *$f'(x^{(0)}) \neq 0$ and $|\frac{1}{f'(x^{(0)})}| \leq A_0$;*
2. *$|\frac{f(x^{(0)})}{f'(x^{(0)})}| \leq B_0 \leq \frac{\varepsilon}{2}$;*
3. *$\forall x, y \in]a, b[, |f'(x) - f'(y)| \leq C|x - y|$*
4. *the constants A_0, B_0, C satisfy the inequality $\mu_0 = 2A_0B_0C \leq 1$.*

then, for an initial approximation $x^{(0)}$, the Newton process

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})},\ n = 0, 1, 2, \ldots \tag{1}$$

converges and $\lim_{n\to\infty} x^{(n)} = x^*$ is a solution of the initial system, so that $|x^* - x^{(0)}| \leq 2B_0 \leq \varepsilon$.

Theorem 2 (Uniqness) *Under the conditions of Theorem 1 the root x^* of the function f is unique in the interval $[x^{(0)} - 2B_0, x^{(0)} + 2B_0]$.*

Theorem 3 (Speed of convergence) *Under the conditions of Theorem 1 the speed of the convergence of Newton's method is given by $|x^{(n)} - x^*| \leq \frac{1}{2^{n-1}}\mu_0^{2^p-1}B_0$.*

Theorem 4 (Local stability) *If the conditions of Theorem 1 are satisfied and if, additionally, $0 < \mu_0 < 1$ and $[x^{(0)} - \frac{2}{\mu_0}B_0, x^{(0)} + \frac{2}{\mu_0}B_0] \subset]a, b[$, then for any initial approximation $x'^{(0)}$ that satisfies $|x'^{(0)} - x^{(0)}| \leq \frac{1-\mu_0}{2\mu_0}B_0$ the associated Newton's process converges to the root x^*.*

The convergence of the process ensures that Newton's method is indeed appropriate for determining the root of the function. The unicity of the solution in a certain domain is used in practice for isolating the roots of the function. The result on the speed of the convergence means we know a bound for the distance between a given element of the sequence and the root of the function. This represents the precision at which an element of the sequence approximates the root. In practice this theorem is used to determine the number of iterations needed in order to achieve a certain precision for the solution. The result on the stability of the process will help with efficiency issues as it allows the use of an approximation rather than an exact real.

We do not present here the proofs of the theorems, we just give a few elements of these proofs that are needed in understanding the next section. For details on the proofs we refere the reader to [6]. The central element of the proof is an induction process that establishes a set of properties for each element of the Newton sequence. The proof introduces the auxiliary sequences $\{A_n\}_{n\in\mathbb{N}}$, $\{B_n\}_{n\in\mathbb{N}}$ and $\{\mu_n\}_{n\in\mathbb{N}}$:

$$A_n = 2A_{n-1} \tag{2}$$

$$B_n = A_{n-1}B_{n-1}^2C = \frac{1}{2}\mu_{n-1}B_{n-1} \tag{3}$$

$$\mu_n := 2A_nB_nC = \mu_{n-1}^2 \tag{4}$$

For each element of the Newton sequence, we are able to verify properties that are similar to those for $x^{(0)}$. Reasoning by induction we get the following:

- $f'(x^{(n)}) \neq 0$ *and* $|\frac{1}{f'(x^{(n)})}| \leq A_n$
- $|f(x^{(n)})/f'(x^{(n)})| \leq B_n \leq \frac{\varepsilon}{2^{n+1}}$

- $\mu_n \leq 1$

Notice that hypothesis 3. is a property of the function and it does not depend on the elements of Newton's sequence.
From the above relations we get the convergence, unicity and speed of convergence for the sequence.

For Theorem 4 (local stability) we prove that the new initial approximation $x'^{(0)}$ satisfies similar hypotheses as those for $x^{(0)}$. The new constants are $A' = \frac{4}{3+\mu_0}A_0$ and $B' = \frac{3+\mu_0}{4\mu_0}B_0$. This makes that $\mu' = 2A'B'C = 1$ and we can verify that

- $f'(x'^{(0)}) \neq 0$ *and* $|\frac{1}{f'(x'^{(0)})}| \leq A'$
- $|f(x'^{(0)})/f'(x'^{(0)})| \leq B'$
- $\mu' \leq 1$

We are thus in the hypotheses of Theorem 1 and by applying this theorem we conclude that the process converges to the same root x^*.

Notice, however, that for the new constants we get $\mu' = 1$. If we do a Newton iteration, we would get the new $\mu'' = \mu'^2 = 1$ (cf. equation (4)) and we would not be able to do an approximation again, because Theorem 4 requires $\mu'' < 1$. To correct this, we impose a finer approximation $|x_0 - x'_0| \leq \frac{(1-\mu_0)}{4\mu_0}B_0$. This new approximation yields the following formulas for the constants:

$$A' = \frac{8}{7+\mu_0}A_0 \tag{5}$$

$$B' = \frac{\mu_0^2 + 46\mu_0 + 17}{8(7+\mu_0)\mu_0}B_0 \tag{6}$$

this makes that

$$\mu' = \frac{\mu_0^2 + 46\mu_0 + 17}{(7+\mu_0)^2} < 1 \tag{7}$$

We summarize these results in:

Corollary 1 *If the conditions of Theorem 1 are satisfied and if, additionally,* $0 < \mu_0 < 1$ *and* $[x^{(0)} - \frac{2}{\mu_0}B_0, x^{(0)} + \frac{2}{\mu_0}B_0] \subset]a, b[$*, then for any initial approximation* $x'^{(0)}$ *that satisfies* $|x'^{(0)} - x^{(0)}| \leq \frac{1-\mu_0}{4\mu_0}B_0$ *the associated Newton's process converges to the root* x^**.*

2.1 Newton's method with rounding

We now have all the necessary tools to state and prove a theorem on the behavior of Newton's method if we consider rounding at each step. The rounding we do is just good enough to ensure the convergence. This theorem is particularly interesting for computations in arbitrary or multiple precision, as it relates number of iterations with the precision of the input and that of the result. This means that for the first iterations we need a lower precision, as we are not close to the root. We will later increase the precision of our input with the desired precision for the result.

Theorem 5 *We consider a function $f :]a, b[\to \mathbb{R}$ and an initial approximation $x^{(0)}$ satisfying the conditions in Theorem 1.*
We also consider a function $rnd : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ that models the approximation we will make at each step in the perturbed Newton sequence:
$t^{(0)} = x^{(0)}$ *and* $t^{(n+1)} = rnd_{n+1}(t^{(n)} - f(t^{(n)})/f'(t^{(n)}))$
If

1. $\forall n \forall x, x \in]a, b[\Rightarrow rnd_n(x) \in]a, b[$
2. $\frac{1}{2} \leq \mu_0 < 1$
3. $[x^{(0)} - 3B_0, x^{(0)} + 3B_0] \subset]a, b[$
4. $\forall n \forall x, |x - rnd_n(x)| \leq \frac{1}{3^n} R_0$, *where* $R_0 = \frac{1-\mu_0^2}{8\mu_0} B_0$

then

a. *the sequence* $\{t^{(n)}\}_{n \in \mathbb{N}}$ *converges and* $\lim_{n \to \infty} t^{(n)} = x^*$ *where* x^* *is the root of the function* f *given by Theorem 1*
b. $\forall n, |x^* - t^{(n)}| \leq \frac{1}{2^{n-1}} B_0$

The first hypothesis makes sure that the new value will also be in the range of the function. The second and third hypotheses come from the use of the stability property of the Newton sequence (see Corollary 1). The fourth hypothesis controls the approximation we are allowed to make at each iteration. The conclusion gives us the convergence of the process to the same limit as Newton's method without approximations. Also we give an estimate of the distance from the computed value to the root at each step.

Proof. Our proof is based on those for theorems 1 - 4 and corollary 1. To give the intuition behind the proof, we decompose Newton's perturbed process $t^{(n)}$ as follows:

1. set $t^{(0)} := x^{(0)}$
2. do a Newton iteration to get $x^{(1)} := t^{(0)} - \frac{f(t^{(0)})}{f'(t^{(0)})}$
3. do an approximation of the result to get $t^{(1)} := rnd(x^{(1)})$
4. set $t^{(0)} := t^{(1)}$ and go to step 2.

Now let's look at these steps individually:

- At step 1. we start with the initial $x^{(0)}$ that satisfies the conditions in Theorem 1. This means that Newton's method from this initial point converges to the root x^* (cf. Theorem 1).
- At step 2. we consider a Newton sequence starting with $x^{(1)}$. This sequence is the same as the sequence at step 1. except that we "forget" the first element of the sequence and start with the second. It is trivial that this sequence converges to the root x^*. We note that (cf. proof of Theorem 1) we can associate the constants A_1, B_1 to the initial iteration of this sequence and get the corresponding hypotheses from Theorem 1.

- At step 3. we consider Newton's sequence starting from $t^{(1)}$. This initial point is just an approximation of the initial point of the previously considered sequence. From Corollary 1 we get the convergence of the new sequence to the same root x^*. Moreover, the proof of Corollary 1 gives us the constants A', B' associated to the initial point that also satisfy the hypotheses of Theorem 1. This means we can start the process over again.

If we take $x^{(0)}$ and then all the initial iterations of the sequences formed at step 3. we get back our perturbed Newton's sequence. But decomposing the problem as we did gives the intuition of why this sequence should converge. However, just having a set of sequences that all converge to the same root does not suffice to prove that the sequence formed with all initial iterations of these sequences will also converge to the same root. The reason is simple, the approximation at step 3. could bring us back to the initial point $x^{(0)}$ which would still yield a convergent Newton's sequence, but which would not make the new element of the perturbed sequence any closer to the root than the previous one. To get the convergence of the perturbed sequence we need to control the approximation we make. We will see in what follows that hypothesis 4. suffices to ensure the convergence of the new process.

To make the intuitive explanation more formal we consider the sequence of sequences of real numbers $\{Y_p\}_{p\in\mathbb{N}}$ defined as follows:

$Y_0^n = x^{(n)}$ is the original Newton's sequence;

Y_1 is given by

$Y_1^0 = rnd_1(x^{(1)})$;

$Y_1^{n+1} = Y_1^n - f(Y_1^n)/f'(Y_1^n)$ is the Newton's sequence associated to the initial iteration Y_1^0;

we continue in the same manner and for an arbitrary p we define Y_p as follows

$Y_{p+1}^0 = rnd_{p+1}(Y_p^1)$;

$Y_{p+1}^{n+1} = Y_{p+1}^n - f(Y_{p+1}^n)/f'(Y_{p+1}^n)$.

We notice that taking the first element in each of these sequences forms our perturbed Newton's process:

$$Y_0^0 = x^{(0)} = t^{(0)} \text{ and}$$

$$Y_{n+1}^0 = rnd_{n+1}(Y_n^0 - f(Y_n^0)/f'(Y_n^0)) = rnd_{n+1}(t^{(n)} - f(t^{(n)})/f'(t^{(n)})) = t^{(n+1)}$$

Following our plan, we now show that for each p the sequence $\{Y_p^n\}_{n\in\mathbb{N}}$ converges to x^* and ensures a certain bound in the error.

- We start with sequence $\{Y_0^n\}_{n\in\mathbb{N}}$. Since it coincides with the initial sequence, the properties from Theorem 1 are trivially satisfied. For the initial point Y_0^0 we have the associated constants A_0, B_0. Applying Theorem 1 we get that $\lim_{n\to\infty} Y_0^n = x^*$ and $|x^* - Y_0^0| \leq 2B_0$.
- Before considering $\{Y_1^n\}_{n\in\mathbb{N}}$, we note that the sequence $\overline{Y}_0^n = Y_0^{n+1}$ (i.e. the previously considered sequence where we start from the second element) also satisfies the conditions, with initial point $\overline{Y}_0^0 = Y_0^1$ and constants $\overline{A}_0 = 2A_0$ and $\overline{B}_0 = A_0B_0^2C$. The laws for these constants are deduced from relations

(2), (3). We get that $\lim_{n\to\infty} \overline{Y}_0^n = x^*$ and $|x^* - \overline{Y}_0^0| = |x^* - Y_0^1| \leq 2\overline{B}_0 = 2(A_0 B_0^2 C)$.

- Now we consider $\{Y_1^n\}_{n\in\mathbb{N}}$. The initial point of this sequence is $Y_1^0 = rnd_1(Y_0^0 - f(Y_0^0)/f'(Y_0^0)) = rnd_1(\overline{Y}_0^0)$. We are in the situation of Corollary 1, where we have a converging sequence ($\{\overline{Y}_0^n\}_{n\in\mathbb{N}}$) and we introduce an approximation in the initial iteration. To be able to apply this corollary we need to verify $0 < \overline{\mu}_0 < 1$, $[\overline{Y}_0^0 - \frac{2}{\overline{\mu}_0}\overline{B}_0, \overline{Y}_0^0 + \frac{2}{\overline{\mu}_0}\overline{B}_0] \subset]a, b[$ and $|rnd_1(\overline{Y}_0^0) - \overline{Y}_0^0| \leq \frac{1-\overline{\mu}_0}{4\overline{\mu}_0}\overline{B}_0$. We will show later on that under our hypotheses these three conditions are indeed verified. From Corollary 1 we get the new constants according to relations (5), (6). This makes that we find ourselves again in the conditions of Theorem 1 and we can deduce that $\lim_{n\to\infty} \overline{Y}_1^n = x^*$ and $|x^* - Y_1^0| \leq 2B' = 2\frac{\overline{\mu}_0^2 + 46\overline{\mu}_0 + 17}{8(7+\overline{\mu}_0)\overline{\mu}_0}\overline{B}_0$.

We are in the appropriate conditions to start this process again and explain in the same manner the properties for $\{Y_2^n\}_{n\in\mathbb{N}}$, $\{Y_3^n\}_{n\in\mathbb{N}}$, etc. The auxiliary sequences are given by the following relations:

$$A'_0 = A_0 \; and \; A'_{n+1} = \frac{8}{7 + \overline{\mu}_n(2A'_n)}$$

$$B'_0 = B_0 \; and \; B'_{n+1} = \frac{{\overline{\mu}_n}^2 + 46\overline{\mu}_n + 17}{8(7+\overline{\mu}_n)\overline{\mu}_n}(A'_n {B'_n}^2 C)$$

$$\overline{\mu}_n = 2(2A'_n)(A'_n {B'_n}^2 C)C = (2A'_n B'_n C)^2$$

we also consider

$$\mu'_{n+1} = 2A'_{n+1}B'_{n+1}C = \frac{\overline{\mu}_n^2 + 46\overline{\mu}_n + 17}{(7+\overline{\mu}_n)^2} = \frac{{\mu'^2_n}^2 + 46\mu'^2_n + 17}{(7+\mu'^2_n)^2}$$

$$R_n = \frac{1-\overline{\mu}_n}{4\overline{\mu}_n}\overline{B}_n = \frac{1-\mu'^2_n}{4\mu'^2_n}(\frac{1}{2}\mu'_n B'_n) = \frac{1-\mu'^2_n}{8\mu'_n}B'_n$$

Using the above reasoning steps, we get by induction that $|Y_n^0 - x^*| \leq 2B'_n$ and we also manage to show $\forall n, B'_{n+1} \leq \frac{1}{2}B'_n \leq \frac{1}{2^{n-1}}B_0$. The latter relations is deduced from the above formulas by basic manipulations. It trivially implies the convergence of the perturbed sequence to the root x^*.

We need some auxiliary results to ensure that Corollary 1 is applied in the appropriate conditions each time we make a rounding. These results are as follow:

- $0 < \frac{1}{2} \leq \mu_0 = \mu'_0 \leq \mu'_n \leq \mu'_{n+1} \leq \ldots < 1$
- $R_{n+1} \leq \frac{1}{3}R_n \leq \ldots \leq \frac{1}{3^n}R_0 = \frac{1}{3^n}\frac{1-\overline{\mu}_0}{4\overline{\mu}_0}\overline{B}_0 = \frac{1}{3^n}\frac{1-\mu_0^2}{8\mu_0}B_0$
- $|Y_{n+1}^0 - Y_n^0| \leq \frac{1}{2^n}B_0 + \frac{1}{3^n}R_0$
- $[\overline{Y}_n^0 - \frac{2}{\overline{\mu}_n}\overline{B}_n, \overline{Y}_n^0 + \frac{2}{\overline{\mu}_n}\overline{B}_n] \subseteq [Y_0^0 - 3B_0, Y_0^0 + 3B_0] \subset]a, b[$

We do not discuss all the details as they are elementary reasoning steps concerning inequalities, second degree equations or geometric series. All these results have been formalized in Coq to ensure that no steps are overlooked.

Remarks. Independent of certification of exact computations, this proof has an interest from a proof engineering point of view. We were able to come up with the proof because we had formalized theorems 1 - 4 inside a proof assistant. Such a formalization forces the user to understand the structure of the proof on one hand and to handle details with care on the other. Thus, an assisted proof is usually more structured and more detailed than a paper proof (especially in domains where automatic techniques are difficult to implement, like real analysis). For example, while on paper the auxiliary sequences $\{A_n\}_{n\in\mathbb{N}}$, $\{B_n\}_{n\in\mathbb{N}}$ appear during the proof, on the computer they are defined appart from the proof, allowing the user to better understand their importance and use similar sequences in the new proof. A proof assistant is also helpful with syntactic aspects like properly constructing the induction hypothesis and doing the bookkeeping to make sure all needed details are taken into consideration.

3 A Coq library for exact real arithmetic

The exact real library we are considering represents a real in the interval $[-1, 1]$ as a lazy infinite sequence of signed digits of an arbitrary integer base. The signed digits of a base β are the integers in $[-\beta + 1, \beta - 1]$. We denote $s_1 :: s$ the infinite sequence beginning by the digit s_1 and followed by the infinite sequence s. The real number r represented by such an infinite sequence s in base β is :

$$r = [\![s]\!]_\beta = [\![s_1 :: s_2 :: s_3 :: \ldots]\!]_\beta = \sum_{i=1}^{\infty} \frac{s_i}{\beta^i}.$$

A real number represented by a stream and for which we know the first digit can be written as: $r = [\![s_1 :: s]\!]_\beta = \frac{s_1 + [\![s]\!]_\beta}{\beta}$.

Having signed digits makes our representation redundant. For example we can represent $\frac{1}{3}$ as $[\![3::3::3::3\ldots]\!]_{10}$ but also as $[\![4::-7::4::-7\ldots]\!]_{10}$. For each digit k the set of real numbers that admit a representation beginning by this digit is: $[\frac{k-1}{\beta}, \frac{k+1}{\beta}]$. The sets associated to consecutive digits overlap with a constant magnitude of $\frac{2}{\beta}$. The main benefit of this redundancy is that we are able to design algorithms for which we can decide a possible first digit of the output. Without redundancy this is in general undecidable. Take the example of addition: $[\![0::3\ldots]\!]_{10} + [\![0::6\ldots]\!]_{10}$ may need infinite precision to decide whether the first digit is 0 or 1. In the case of signed digits we give 1 as a first digit knowing we can always go back to a smaller number by using a negative digit. We also note that in our example it was sufficient to know two digits of the input to decide the first digit of the output and this is true for addition in general.

Designing an algorithm therefore requires approximating the result to a precision that is sufficient to determine a possible first digit. In the library this is supported by the function `make_digit` and the required precision is $\frac{\beta-2}{2\beta^2}$. Also, since our real numbers are infinite streams, the algorithms need to be designed

in such a way that we are always able to provide an extra digit of the result. This is done by co-recursive calls on our co-inductive streams.

In Coq, co-induction [9] provides a way to describe potentially infinite datatypes as our infinite sequences of digits. It offers both efficient lazy evaluation and nice proof schemes. The type of infinite sequences of objects of some type `A` is define as follows

CoInductive stream (A : Set) : Set := | Cons : A → stream A → stream A.

`Cons` should not be understood as a way to construct an infinite stream from another since we cannot build an initial infinite stream, but as a way to decompose an infinite stream into a finite part and an infinite part that could be described again with a new `Cons` and so on.

Our real numbers will be streams of signed digits, so we also need to create a model for the digits. They are abstracted with respect to the base and the implementation of integers, so both the base and the type of integers can be chosen by the user. Here we will denote their type by `digit`.

We can define new streams using co-recursive functions, for instance the stream of 0s, which obviously represents the real number 0.

CoFixpoint zero : stream digit := Cons 0 zero.

To prove the correctness of the algorithms on streams of digits we first define a relation between these streams and the axiomatic reals of Coq. This relation is based on the relation between the real value of a sequence, its first digit and the real value of the following sequence as noted previously. We formalize this relation as a co-inductive predicate :

CoInductive represents ($\beta : \mathbb{Z}$) : stream digit → $\mathbb{R}$ → Prop :=
| rep : $\forall\, s\ r\ k, -\beta < k < \beta \rightarrow \quad -1 \leq r \leq 1 \rightarrow$
represents $\beta\ s\ r \rightarrow$ represents β (Cons $k\ s$) $\frac{k+r}{\beta}$.

This relation also makes sure that streams only represent reals in $[-1, 1]$ and that the digits are in the set of the allowed signed digits.

The correctness of our algorithms is verified when we manage to express a `represents` relation between our implementation and the standard in the Coq library. For instance the proof that the multiplication is correct is formulated in this way :

Theorem mult_correct :
$\forall$ x y vx vy, represents x vx → represents y vy → represents (x ⊗ y) (x ∗ y).

In means that every time we have an exact real (i.e. a stream of digits) x that represents an axiomatic real vx and an y that represents a vy than our multiplication of streams x and y (here denoted $\otimes$) will represent the multiplication of axiomatic reals vx and vy.

For further details on algorithms and proofs for this library we refer the reader to [13].

4 Newton's method on exact reals

4.1 Correctness of Newton's method

We want to prove correctness of computation with Newton's method on exact reals in the same manner we proved correctness of multiplication in section 3. We code Newton's algorithm for both exact reals and axiomatic reals. For simplification we use a function g on exact reals to represent the ratio $\frac{f(x)}{f'(x)}$ of axiomatic reals.

Fixpoint EXn g ex0 n {struct n}: stream digit:= **match** n **with**
| 0 ⇒ ex0 | S n ⇒ **let** exn:= (EXn g x0 n) **in** exn ⊖ g exn **end**.

Fixpoint Xn f f' x0 n {struct n}: R:= **match** n **with**
| 0 ⇒ x0 | S n ⇒ **let** xn:= (Xn x0 f f' n) **in** xn − f xn / f' xn **end**.

The relation between elements of the same rank in the two sequences:

```
∀ n, represents (EXn g EX0 n) (Xn X0 f f' n)
```

is almost trivial, if we have a `represents` relation for the initial iteration and for the function.

Theorem EXn_correct : ∀ g ex0 f f' x0 n, represents ex0 x0 →
(∀ x vx, represents x vx → represents (g x) (f vx / f' vx)) →
(∀ n, −1 ≤Xn x0 f f' n ≤1) → represents (EXn g ex0 n) (Xn x0 f f' n).

The proof follows from the correction of the subtraction on streams with respect to the subtraction on axiomatic reals.

This theorem allows us to transfer properties proved for Newton's method on axiomatic reals to the method implemented on exact reals. If we satisfy the conditions of Theorem 1 for the function f and the initial iteration $X0$, then we can compute the root of the function at an arbitrary accuracy, given by Theorem 3 (speed of convergence). From the same theorem we get the rank to which we need to compute for a given accuracy to be obtained. However, if we wanted to increase this accuracy, we would need to redo all the computation for the new rank. We want to avoid this and take advantage of the lazy evaluation characteristic for streams: we can design an algorithm that uses Newton's method to compute an arbitrary number of digits for the root of a given function, under certain conditions for this function.

Remark In the implementation we use tail recursive version of the method to improve efficiency. The equivalent definitions given above help in better understanding the problem.

4.2 An algorithm for exact computation of roots

We consider a function $f : [-1, 1] \to \mathbb{R}$ with x^* the root of f and a suitable initial approximation $x^{(0)}$ for Newton's process. We have to find a possible first digit of the result x^* in base β. For this we use `make_digit` which requires a precision of $\frac{\beta-2}{2\beta^2}$ of the result to make the appropriate choice of the first digit.

To determine the number of Newton iterations that ensures this precision we use Theorem 3 (speed of convergence), which gives us n s.t. $|x^{(n)} - x^*| \leq \frac{\beta-2}{2\beta^2}$. We choose as a first digit for x^* the first digit d_1 of a representation of $x^{(n)}$. This gives us $x^* = \frac{d_1+x_1^*}{\beta}$, where x_1^* is the number formed from the remaining digits of x^*. Since $f(x^*) = 0$, we get $f(\frac{d_1+x_1^*}{\beta}) = 0$. This means we can define a new function $f_1(x) := f(\frac{d_1+x}{\beta})$, and x_1^* is the root of f_1. Determining the second digit of x^* is equivalent to determining the first digit of x_1^*. We repeat the previous steps for function f_1 and we take as the initial approximation the remaining digits of $x^{(n)}$, given by $\overline{x}^{(n)} = \beta x^{(n)} - d_1$. Now we have a co-recursive process to produce the digits of the root of our function one by one. If we simplify our algorithm by using $g = \frac{f}{f'}$, when we transform g in g_1 we get

$$g_1(x) := \frac{f_1(x)}{f_1'(x)} = \frac{f(\frac{d_1+x}{\beta})}{\frac{1}{\beta}f'(\frac{d_1+x}{\beta})} = \beta \times g(\frac{d_1+x}{\beta})$$

For the exact real implementation in Coq we express the algorithm on streams of digits, so we remind that for the stream $d_1 :: x$, we have $[\![d_1 :: x]\!]_\beta = \frac{d_1+[\![x]\!]_\beta}{\beta}$

```
CoFixpoint exact_newton (g: stream digit →stream digit) ex0 n:=
  match (make_digit (EXn g ex0 n) with
    |d1::x' ⇒ d1::exact_newton (fun x ⇒ (β ⊙ g (d1::x))) x' n
  end.
```

We note that the multiplication by the base is done by a specific function, for efficiency.

The formal certification of this algorithm means we have to prove that the output of this algorithm `represents` the root of the function f. For this we use Theorems 1 - 3 (see section 2) on axiomatic reals and the theorem `EXn_correct` (section 4.1) that links Newton's method on exact reals to Newton's method on streams. We need to show that if the initial function f satisfies the hypotheses of Theorem 1 then the function f_1 built at the co-recursive call will also satisfy these hypotheses, thus yielding a correct algorithm.

The hypotheses of Theorem 1 impose that

1. $f \in C^{(1)}(]-1,1[)$
2. $\forall x, y \in]-1,1[, |f'(x) - f'(y)| \leq C|x-y|$
3. $f'(x^{(0)}) \neq 0$ and $|\frac{1}{f'(x^{(0)})}| \leq A_0$;
4. $|\frac{f(x^{(0)})}{f'(x^{(0)})}| \leq B_0 \leq \frac{\varepsilon}{2}$;
5. $\mu_0 = 2A_0B_0C \leq 1$.

We analyze $f_1(x) := f(\frac{d_1+x}{\beta})$ for which we have $f_1' = \frac{1}{\beta}f'$ and the new initial iteration $\overline{x}^{(n)} = \beta x^{(n)} - d_1$

1. the class of the function is obviously the same, so $f \in C^{(1)}(]-1,1[)$
2. $|f_1'(x) - f_1'(y)| = |\frac{1}{\beta}f'(\frac{d_1+x}{\beta}) - \frac{1}{\beta}f'(\frac{d_1+y}{\beta})| \leq \frac{1}{\beta}C|\frac{d_1+x}{\beta} - \frac{d_1+y}{\beta}| = \frac{1}{\beta^2}C|x-y|$

3. $f_1'(\overline{x}^{(n)}) = f'(x^{(n)}) \neq 0$ and $|\frac{1}{f_1'(\overline{x}^{(n)})}| = |\beta\frac{1}{f'(x^{(n)})}| \leq \beta A_n$;
4. $|\frac{f_1(\overline{x}^{(n)})}{f_1'(\overline{x}^{(n)})}| = |\beta\frac{f(x^{(n)})}{f'(x^{(n)})}| \leq \beta B_n$;
5. $\overline{\mu}_n = 2\beta A_n \beta A_n \frac{1}{\beta^2} C = 2A_n B_n C \leq 1$.

Relations 3. - 5. are given by the proof of Theorem 1. We are now able to prove by co-induction the following correctness theorem

Theorem exact_newton_correct:
∀ g ex0 f f' x0 n,
(hypotheses on f, f', x0 from Theorem 1 *)*
(hypothesis on the number of iterations n*)*
(∀ x vx, represents x vx → represents (g x) (f vx / f' vx)) →
represents ex0 x0 → represents (exact_newton g ex0 n) x^*.

4.3 Improvements of the algorithm

Though short, elegant and certified, the algorithm presented in this section is not usable in practice as it is very slow. There are two main reasons for this:

1. The certified computations from the library require a precision of the operands higher than that of the result. We saw that in the case of addition on extra digit is required, but for other operations and function this precision can be higher. When we have an expression where we perform several operations, the precision demanded for each individual operand is a lot higher than the precision of the output. In the case of Newton's method, each iteration only brings a certain amount of information, so using a higher precision will not improve the result.
2. This approach relies on the higher-order capabilities of the functional programming language: the first argument of the `exact_newton` function is itself a function that becomes more and more complex as `exact_newton` calls itself recursively. The management of this function is somehow transparent to the programmer, but it has a cost: a new closure is built at every recursive call to `exact_newton` and when the function `g` is called, all the closures built since the initial call have to be unraveled to obtain the operations that really need to be performed. This cost can be avoided by building directly a first older data structure.

We discuss two possible improvements of this algorithm, dealing with these two issues. For the first point the solution is simple, just use the significant digits in the stream. Determining which are these significant digits and certifying the result is still possible thanks to Theorem 5. We implement a `truncate` function that given a stream s returns the stream containing the first n digits of s and sets the rest to zero. This function represents the `rnd` function on axiomatic reals (see Theorem 5).

```
Fixpoint truncate s n {struct n} :=
  match n with | 0 ⇒ zero | S n ⇒
    match s with | d :: s' ⇒ d :: truncate s' n end
  end.
```

The perturbed Newton's method becomes:

```
Fixpoint Etn g ex0 (n : nat) {struct n} : stream digit := match n with
  | 0 ⇒ ex0 | S n ⇒ let tn := (Etn g ex0 n) in (truncate (tn ⊖ (g tn)) (ϕ n)) end.
```

The function ϕ controls the approximation we can make at each iteration and follows the constrains imposed by Theorem 5. The `exact_newton` algorithm will work in the same way with this sequence as with the original method.

```
CoFixpoint exact_newton_rnd (g: stream digit →stream digit) ex0 n:=
  match (make_digit (Etn g ex0 n) with
    |d1::x' ⇒ d1::exact_newton_rnd (fun x ⇒ (β ⊙ g (d1::x))) x' n
  end.
```

Though the proof for this new algorithm is not finalized yet, we feel there is no real difficulty in obtaining it as both the algorithm and the optimization we make are certified.

To tackle the second point in our list of possible improvements we make explicit the construction of the new function g in the co-recursive call.

```
CoFixpoint exact_newton_aux
  (g : stream digit → stream digit) (Xn : stream digit) k n :=
    let Xn' := make_k_digits x0 (EXn g x0 n) k in
      (nth k Xn') :: exact_newton_aux g Xn' (S k) n.

Definition exact_newton2 (g : stream digit → stream digit)
  (X0 : stream digit) n := exact_newton_aux g X0 0 n.
```

The function `make_k_digits` takes three arguments: two streams x and y and an integer k and produces $k+1$ digits of y by copying the first k digits in x and computing the digit $k+1$ by using `make_digit`. For the function to perform correctly we must ensure that the first k digits in y can indeed be the same as those in x. In our case this results from the theorem on the speed of convergence of Newton's method, which make sure that a certain element is close enough to the root. The way the algorithm works is that it does iterations always for the same function g. It produces digits one at a time. Once it reached enough precision to certify an extra digit, the $(k+1)$th, it gets this digit by using the function `nth` and it continues to compute where it left of.

This algorithm performs better than the previous one, but the optimizations performed in this case seem more difficult to certify. At the time of writing this paper we have the good properties on `make_k_digit` and the proof of correctness of the algorithm is in progress.

4.4 Applications to the square root

Newton's method is commonly used for the implementation of nth root function or division. We discuss the example of the square root to illustrate the behaviour of our algorithms. The square root of a positive real number a is the root of the function $f_{sqrt}(x) = x^2 - a$. The corresponding function g_{sqrt} is $\frac{f_{sqrt}(x)}{f'_{sqrt}(x)} = \frac{x}{2} - \frac{a}{2x}$. Due to restrictions about implementing the inverse function of exact reals, the

library provides functions of the family $x \mapsto \frac{1}{\beta^n x}$ where $n > 0$. So we chose instead the function $f_{sqrt}(x) = \beta^2 x^2 - a$ which corresponds to $g_{sqrt}(x) = \frac{x}{2} - \frac{a}{2\beta^2 x}$. The root of this function is $\frac{\sqrt{a}}{\beta}$. So a final multiplication by the base will give the expected result. We apply the algorithm to this function g_{sqrt} and the user provide a suitable initial approximation. The process will actually converge for any positive initial approximation. We prove in Coq that the resulting function actually computes a representation of the square root function on axiomatic reals divided by the base.

Definition Ssqrt (a : stream digit) ex0 n := exact_newton (g_sqrt a) ex0 n.
Theorem sqrt_correct : ∀ (a : stream digit) (va : R),
represents a va → represents (Ssqrt a) ((sqrt va)/β).

The original algorithm is slow. For example the computation of the first digit of $\sqrt{\frac{1}{2}}$ in base 2^{124} using the original algorithm blocks the system, while for the same algorithm improved with approximations we get the equivalent precision of 37 decimal digits in 12 seconds. The second algorithm `exact_newton2` brings an improvement at each new digit we want to obtain making the algorithm run in average twice as fast. We should also take into consideration that using $f(x) = \frac{a}{x^2} - 1$ can improve our execution times considerably as there is only one division involved. Nevertheless, our intention here was not to implement an efficient square root, but to test the capabilities of the previous presented algorithms.

5 Related work

This work presents different angles in the formal verification of a numerical algorithm. For the moment such developments are not very popular. A lot of work is being done concerning formally verified exact real arithmetic libraries. Besides the library presented here, the development [15] for PVS [21] and [18] also for Coq are two of the most recent such implementations. These two libraries have computations that are certified with respect to the real analysis formalizations in PVS and C-CoRN [5], respectively. A significant part of the work presented here could be reproduced in any of these libraries. In the case of [18] the exact reals operations and functions are certified via an isomorphism between the exact reals and the C-CoRN real structure; there is also an isomorphism between C-CoRN reals and Standard Library reals (see [14]), so in theory it should be possible to certify computations by using the presented proofs and the two isomorphisms.

Concerned with exact real arithmetic and also with co-inductive aspects we mention the work of Niqui [17]. This works aims to obtain all field operations on real numbers via the Edalat-Potts algorithm for lazy exact arithmetic.

Results of the convergence of Newton's method with rounding have been proved for some special cases like the the inverse and the square root [3]. Of course, in these cases the speed of convergence is better than in the general case.

The certification of a square root algorithm has been the subject of several formal developments. We mention [2] for the certification of the GMP square

root algorithm, [11] for an Intel architecture square root algorithm and [20] for the verification of the square root algorithm in an IBM processor. A general algorithm using Newton's method was developped by Hur and Davenport [12] on a different representation of exact reals but not in a certified setting.

6 Conclusions and perspectives

As a case study of theorem proving in numerical analysis, this work tries to underline three aspects of such a development: how do design and formalize the necessary proofs from "paper" mathematics, how to prove correct numerical methods implemented on exact reals and provide certified computations, how to design and verify specific algorithms for an implementation of exact reals. The Coq development can be found at `http://www-sop.inria.fr/marelle/Exact_Newton`.

To the best of the authors' knowledge, the result and proof of Theorem 5 are new, though the authors are not experts in numerical analysis. Using only a predetermined precision for our computation makes it that our formalization can be seen as an (imperfect) model of computation in multiple or arbitrary precision, thus validating Newton's method in such a context. The proof of Theorem 5 was motivated by the need to improve the algorithm discussed in section 4.2. The contribution of proof assistants in obtaining this proof is twofold: the proof was motivated by a formal development and the proof was constructed inside the proof assistant, following the pattern of existing proofs.

The algorithms presented in section 4 are also new. The certified algorithm, though it is not of practical use, it can serve as a model in obtaining proofs for our optimized algorithms. The next step is to put together the two proofs we presented here and get a formally verified algorithm for computing roots in a (more) efficient manner. With the results presented here we see no difficulties in obtaining this proof.

The axiomatic formalization on Newton's method contains the multivariable version of the theorems 1 - 4. This means we can solve systems of (non-)linear equations using this method. Exact real arithmetic libraries do not yet treat such cases, so it would be an interesting experiment to see to what extent the results presented here can be obtained in the multivarite setting. We note that the proof of Theorem 5 has the same structure for the multivarite case. We presented here the real case to make it easier for the reader to follow and to show the correspondence with results from section 4.2.

Acknowledgments

We thank Yves Bertot for his help and constructive suggestions.

References

1. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art:the Calculus of Inductive Constructions*. Springer-Verlag, 2004.

2. Yves Bertot, Nicolas Magaud, and Paul Zimmermann. A Proof of GMP Square Root. *J. Autom. Reasoning*, 29(3-4):225–252, 2002.
3. R.P. Brent and P. Zimmermann. *Modern Computer Arithmetic.* 2006. In preparation. Available at http://www.loria.fr/zimmerma/mca/pub226.html.
4. Coq development team. *The Coq Proof Assistant Reference Manual, version 8.1*, 2006.
5. L. Cruz-Filipe, H. Geuvers, and F. Wiedijk. C-CoRN: The Constructive Coq Repository at Nijmegen. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *Mathematical Knowledge Management, Third International Conference, MKM*, volume 3119 of *LNCS*, pages 88–103. Springer-Verlag, 2004.
6. B. Démidovitch et I. Maron. *Éléments de calcul numérique.* Mir - Moscou, 1979.
7. Jacques D. Fleuriot. On the mechanization of real analysis in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 146–162. Springer-Verlag, 2000.
8. R. Gamboa and M. Kaufmann. Nonstandard Analysis in ACL2. *Journal of automated reasoning*, 27(4):323–428, November 2001.
9. Eduardo Giménez. Codifying guarded definitions with recursive schemes. In Peter Dybjer, Bengt Nordström, and Jan Smith, editors, *Types for proofs and Programs*, volume 996 of *LNCS*, pages 39–59. Springer Verlag, 1994.
10. John Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998.
11. John Harrison. Formal verification of square root algorithms. *Formal Methods in System Design*, 22(2):143–153, 2003.
12. Namhyun Hur and James H. Davenport. A generic root operation for exact real arithmetic. 2064:82–??, 2001.
13. Nicolas Julien. Certified exact real arithmetic using co-induction in arbitrary integer base. In *Functional and Logic Programming Symposium (FLOPS)*, LNCS. Springer, 2008.
14. Cezary Kaliszyk and Russell O'Connor. Computing with classical real numbers. *CoRR*, abs/0809.1644, 2008.
15. David R. Lester. Real Number Calculations and Theorem Proving. In Otmane Aït Mohamed, César Muñoz, and Sofiène Tahar, editors, *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2008.
16. Micaela Mayero. *Formalisation et automatisation de preuves en analyses reelle et numerique.* PhD thesis, Université de Paris VI, 2001.
17. Milad Niqui. Coinductive formal reasoning in exact real arithmetic. *Logical Methods in Computer Science*, 4(3:6):1–40, September 2008.
18. Russell O'Connor. Certified Exact Transcendental Real Number Computation in Coq. In *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada*, pages 246–261, 2008.
19. Ioana Paşca. A Formal Verification for Kantorovitch's Theorem. In *Journées Francophones des Langages Applicatifs*, pages 15–29, 2008.
20. Jun Sawada and Ruben Gamboa. Mechanical verification of a square root algorithm using taylor's theorem. In Mark Aagaard and John W. O'Leary, editors, *FMCAD*, volume 2517 of *Lecture Notes in Computer Science*, pages 274–291. Springer, 2002.
21. Natarajan Shankar, Sam Owre, and John M. Rushby. *The PVS Proof Checker: A Reference Manual.* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.