Formal Proofs for Theoretical Properties of Newton's Method

Ioana Pasca

Formal Proofs for Theoretical Properties of Newton's Method

Ioana Paşca

N° 7228

February 2010

ISSN 0249-6399 ISRN INRIA/RR--7228--FR+ENG

Formal Proofs for Theoretical Properties of Newton's Method

Ioana Paşca

Thème : Systèmes symboliques
Équipe-Projet Marelle

Rapport de recherche n° 7228 — February 2010 — 33 pages

Abstract: We discuss a formal development for the certification of Newton's method. We address several issues encountered in the formal study of numerical algorithms: developing the necessary libraries for our proofs, adapting paper proofs to suit the features of a proof assistant, and designing new proofs based on the existing ones to deal with optimizations of the method. We start from Kantorovitch's theorem that states the convergence of Newton's method in the case of a system of equations. To formalize this proof inside the proof assistant Coq we first need to code the necessary concepts from multivariate analysis. We also 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. This proof is based on Kantorovitch's theorem but it represents an original result. An algorithm including rounding is a more accurate model for computations with Newton's method in practice.

Key-words: proof assistants, formalization of mathematics, multivariate analysis, Kantorovitch's theorem, Newton's method with rounding

Preuves formelles pour les propriétés théoriques de la méthode de Newton

Résumé : Ce rapport présente un développement formel pour la certification de la méthode de Newton. On s'intéresse à plusieurs problèmes rencontrés dans l'étude formelle des algorithmes numériques : développer les bibliothèques nécessaires pour nos preuves, adapter des preuves papier aux caractéristiques d'un assistant à la preuve, concevoir des nouvelles preuves basées sur les preuves existantes pour certifier des optimisations de la méthode. Notre point de départ est le théorème de Kantorovitch qui établit la convergence de la méthode de Newton dans le cas d'un système d'équations. Pour formaliser ce théorème dans l'assistant à la preuve Coq on a besoin d'abord de coder les concepts nécessaires d'analyse multivariée. On démontre aussi qu'arrondir à chaque itération de la méthode de Newton donne lieu à un processus qui est encore convergent, avec une corrélation bien determinée entre la précision des données d'entrée et celle du résultat. Un algorithme avec des arrondis est un modèle plus fidèle pour les calculs pratiques par la méthode de Newton.

Mots-clés : assistants à la preuve, formalisation des mathématiques, analyse multivariée, théorème de Kantorovitch, méthode de Newton avec arrondis

1 Formal systems and numerical methods

Often, in verifying mathematical theorems in proof assistants we start with a paper proof of some (famous) theorem and try to obtain a formal model of the theorem inside the system. The concepts are coded in a manner that keeps the balance between mathematical accuracy and handiness of use. This encoding process is not always trivial as the mathematical concepts, expressed in general in set theory, need to be translated into type theory or higher order logic. The limitations and benefits of the formal framework need to be taken into account. Once the concepts inside the system, we try to reproduce the reasoning steps to get the desired proof. Automatization is often possible for some (small) parts of the problem, depending on the field. A good example of a field where mechanization is wide spread is algebra while calculus is less prone to automatization. As a consequence formal developments in real or numerical analysis are more tedious. This is a setback for proof assistants in comparison to computer algebra system which support a wide variety of numerical methods. However, it is sometimes the case that these systems produce erroneous output [16, 6]. So, when a high level of correctness is required, choosing a proof assistant for the task could be a good solution. The aim of this paper is to describe several aspects of a formal development around a numerical algorithm. We discuss the problems encountered, possible solutions and potential applications, in order to allow a better understanding of what such a development entails. Among others, we address the following issues:

- providing the proof assistant with the necessary concepts to support all reasoning steps that we are interested in;
- formalizing a piece of mathematics stating the desired properties for our algorithms;
- designing new proof based on the existing ones to offer theoretical basis for optimizations of our algorithms.

We do a case study on Newton's method and detail all the points above. Widely used as an approximation method to determine the root of a given function or, equivalently, the solution of a system of equations, Newton's method has good performance with respect to the speed of convergence and the stability of the process. These performances are backed by theoretical results in numerical analysis establishing sufficient conditions for the convergence of the method. Among such results we have Kantorovitch's theorem, which we chose as a basis for our formal development. The statement of the theorem according to [8] is as follows:

Theorem 1 (Kantorovitch)**.** *Consider a system of non-linear algebraic or transcendent equations $f(x) = 0$, where the vector function $f : \mathbb{R}^p \to \mathbb{R}^p$ has continuous first and second partial derivatives in a certain domain ω, i.e. $f(x) \in C^{(2)}(\omega)$. Let x_0 be a point with its closed ε-neighborhood $\overline{U_\varepsilon}(x_0) = \{\|x - x_0\| \leq \varepsilon\}$ included in ω. If the following conditions hold:*

1. *the Jacobian matrix $W(x) = \left[\dfrac{\partial f_i(x)}{\partial x_j}\right]$ has an inverse for $x = x_0$, $\Gamma_0 = W^{-1}(x_0)$ with $\|\Gamma_0\| \leq A_0$;*

2. $\|\Gamma_0 f(x_0)\| \leq B_0 \leq \frac{\varepsilon}{2}$*;*

3. $\sum_{k=1}^{p} \left| \frac{\partial^2 f_i(x)}{\partial x_j \partial x_k} \right| \leq C$ *for* $i, j = 1, 2, ..., p$ *and* $x \in \overline{U_\varepsilon}(x_0)$*;*

4. the constants A_0, B_0, C *satisfy the inequality* $2pA_0B_0C \leq 1$*.*

then, for the initial approximation x_0*, the Newton process*

$$x_{n+1} = x_n - W^{-1}(x_n) f(x_n) \tag{1}$$

($n = 1, 2, ...$*) converges and the limit vector* $x^* = \lim_{n\to\infty} x_n$ *is a solution of the initial system, so that* $\|x^* - x_0\| \leq 2B_0 \leq \varepsilon$*. Moreover, in the domain* $\{\|x - x_0\| \leq 2B_0\}$ *the solution is unique.*

A quick look at the theorem reveals that a formal model of the problem needs to include formalizations of real numbers and real analysis in one and several dimensions. The background results in analysis need to deal with continuity, differentiation, convergence, and so on. Properties on matrices are also used as concepts of inverse or Jacobian matrix need to be handled.

For our work we chose the proof assistant Coq [1, 4] which provides a library of real analysis that is developed enough to cover the concepts needed in the proof of Kantorovitch's theorem in the simplified case of a real function. For the multivariate case, however, Coq does not offer a library, so we need to encode all the necessary concepts. We provide a reusable formalization of multivariate analysis concepts and we present the details in section 2. Section 3.1 discusses the formalization of the Kantorovitch theorem, insisting on the relation between paper proofs and formal proofs. In particular, we discuss an optimized version of Newton's method, where we perform rounding at each step. Based on the proof of Kantorovitch's theorem we prove in section 3.2 that this optimized version converges to the root. The last section presents the conclusions and perspectives of our work.

The paper also contains a special type of section called "Technical Details" that end with "***", in an attempt to make the reading interesting to both experts and non-experts. These technical sections are clearly delimited so they can be easily skipped by the reader not interested in such details. They usually contain implementation details and issues specific to Coq and SSReflect.

2 Formalized mathematical theories for numerical methods

The successful verification of numerical methods depends on having the appropriate mathematical theories formalized in our proof assistant. We need notions of real analysis, linear algebra or multivariate analysis. Coq already contains a library of real analysis as well as an implementation of algebraic structures and matrices in the SSReflect extension of Coq. In our study of properties for Newton's method we need to extend the matrix library on general matrices over a ring with specific notions on real matrices. We also need multivariate analysis concepts. To formalize these concepts we use extensively the standard Coq library on real numbers and real analysis and the SSReflect libraries on matrices. To better understand our formalization we begin by giving a brief description of these existing libraries, in the following section.

2.1 Existing libraries in COQ and SSReflect

2.1.1 COQ's Standard Library Reals

The proof assistant Coq provides an axiomatic definition of the real numbers. The formalization is based on 17 axioms which introduce the reals as a complete, archimedean, ordered field that satisfies the least upper bound principle. This choice of implementation has as positive effect the fact that we can handle real numbers in a manner similar to that of math books on classical real analysis. In particular, we can reason on cases thanks to the trichotomy axiom: for two real numbers x, y exactly one of the following relations holds: $x < y$ or $x = y$ or $x > y$.

The library contains a bunch of results for real analysis: sequences and series, transcendental function, concepts of limit, continuity, differentiation, integration, calculus theorems like the mean value theorem, the fundamental theorem of calculus etc. Details on this implementation can be found in [21].

2.1.2 SSReflect Libraries

SSReflect (**S**mall **S**cale **Reflect**ion) is an extension of Coq that offers new syntax features for the proof shell and basic libraries that make use of small scale reflection in various respects. An extended presentation for the tactics of SSReflect can be found in [13]. The proof of the Four Color Theorem [12] and the on-going effort to prove Feit-Thompson theorem illustrate the power of SSReflect. For example, the Feit Thompson theorem is of major importance in group theory and it states that every finite group of odd order is solvable. The initial paper proof for the Feit-Thompson theorem is 255 pages long and covers many mathematical theories. The formalization in SSReflect is organized in a modular way. This organization allows the libraries to be reused in various other branches of mathematics, in spite of the fact that the main goal is a formalization in group theory.

The basic SSReflect libraries implement types with decidable equality, finite types, lists, finite sets, finite functions, natural numbers, countable types (and more). They also define a hierarchy of algebraic structures: monoid, group, abelian group, ring, unit ring, commutative unit ring, field. The SSReflect libraries provide a formalization of matrices with elements of an arbitrary type T. For operations on rows and columns (for example, deleting a row, swapping two rows etc.) no additional properties are required for T. Once one starts talking about operations on matrices like addition or multiplication, the type of elements T has to be a ring. The library provides all the basic operations and their properties, the notions of determinant and inverse. Details on the matrix library can be found in [11, 3].

2.1.3 Mixing COQ and SSReflect

To get real matrices we use the real numbers in the standard Coq library. They can be endowed with a field structure in the sense of the SSReflect algebraic structures. These structures can be defined on the reals in a way that is transparent for the user and that will be explained in the following section. Once these definitions in place, we can have real matrices and all the generic results on matrices will be available without any further effort. We gathered all the technical details in the following section. The user not interested in the implementation details may completely skip this technical section.

Technical Details 1 (Coq and SSReflect)**.** The key idea in SSReflect is having a mechanism that provides dual views for decidable propositions. This mechanism is called *reflection* and it allows us to link a decidable proposition to a boolean, more precisely and as expected, we link the proposition to the boolean true when the decision procedure says the proposition is true, and to false otherwise.

The propositional version is appropriate when doing structured proofs while the boolean view is used for computing. The user can move from one view to the other by a simple rewrite. This framework is particularly appropriate for working with structures equipped with a decidable equality, as various properties can be reflected by boolean values.

We will analyze in detail the example of types with decidable equality, as this will allow us to illustrate some features of our framework, like *coercions* and *canonical structures*.

Equality in Coq is a syntactic equality, also called Leibniz equality. A decidable equality is a binary boolean relation equivalent to the Leibniz equality. In SSReflect, a type with decidable equality is implemented as a type sort together with a relation eq: sort→sort→bool that reflects the standard Coq equality on that type, that is eq x y is true exactly when x = y in the Leibniz equality sense. Here is the definition of the structure for a type with decidable equality. For didactic reasons we give a simplified definition. The actual SSReflect definition is the same in essence, but more complex in form, due to technical reasons that come from having a very large development and explained in detail in [11].

```
Structure eqType : Type := EqType {
 sort : Type;
 eq : sort → sort → bool;
 eqP : forall x y, reflect (x = y) (eq x y)
}.
Coercion sort : eqType ↣ Type.
```

The coercion mechanism implemented in Coq allows us to say a certain type is a subtype of another type. A coercion is a function from the subtype to the supertype. The coercion is automatically inserted by the system. In our example, the subtype is eqType and the supertype is Type and our coercion is sort. Now, every time the system expects a Type but gets a eqType instead, it will automatically insert this coercion to get a Type. A coercion is not displayed by the pretty-printer, so its use is mostly transparent to the user. This form of explicit subtyping allows any T : eqType to be used as a Type.

There are cases where we would like the system to see a certain concrete type, say the type of natural numbers, as an eqType. This is a normal request, as the equality on natural numbers is decidable. To achieve this we use Coq's *Canonical Structure mechanism*. We illustrate the way it works on the case of natural numbers. In Coq natural numbers are defined as Peano integers. This definition is inductive, that is, we give all possible constructions of a natural number: we either have zero or we have the successor of another natural number. The type of natural numbers is called nat. Based on the definition we can build a decidable equality predicate eqn : nat→nat→bool that reflects the Leibniz equality.

```
Lemma eqnP : forall x y : nat, reflect (x = y) (eqn x y).
```

Now we can declare an eqType structure on our natural numbers.

```
Canonical Structure nat_eqType := EqType eqnP.
```

The Canonical Structure declaration will make that every time an expression requires an eqType, but gets a nat instead, Coq will automatically infer the type nat_eqType for the expected argument. The expression will type-check without intervention from the user. This means the generic theorems and notations for eqTypes can directly be applied to natural numbers.

In a similar manner to the definition for an eqType, the SSReflect libraries define other structures. We will briefly describe some of them in what follows, as they played a role in our development.

A **choiceType** is a type T with a choice function choose that returns a canonical representative of any non-empty subset of elements of type T. By canonical we mean that for two extensionally equal sets and two proofs that the sets are non-empty the function will return the same representative. Natural numbers, for example, are a choiceType as we can define a function nat_choose that starts from zero and checks all numbers until it finds an element of the given non-empty set. The set being non-empty the function will only need a finite number of steps to return a representative of the set. The representative returned is the first one found and therefore canonical. This construction is more general, any countable type can be endowed with a canonical choice function.

Finite types play a central role in the development. A finType contains the list of all its elements and the property that in this list each element appears exactly once. As an example in the library we have the type of natural numbers smaller than p, called ordinal p with notation 'I_p.

Functions with a finType as the definition domain are called **finite functions** or finfun and they benefit from a special treatment in the library. Such a function is represented by the list of all its values and then coerced to the corresponding arrow type. We thus have a dual view for finfuns, as a function and as the function's graph. To define a finfun we use the notation {ffun aT→rT}. If the return type rT is an eqType then the finite function type will also be an eqType as the extensional equality on functions will reflect the Leibniz equality. Similarly, if rT is a choiceType, then {ffun aT→rT} will also be a choiceType.

Once these basic structures are in place, the SSReflect library develops an **algebraic structure hierarchy**. In version 1.2 of SSReflect the hierarchy contains groups, abelian groups, rings, commutative rings and fields. The elements of these structures also have an eqType and choiceType structure. The algebraic structures are defined using the same Structure construct as the eqType.

This means we can use in the same fashion the Canonical Structure mechanism to endow various types with a given algebraic structure. For example, we can make our standard reals a field in the sense of the SSReflect algebraic structures.

SSReflect also contains a library that treats in a general fashion **indexed operations**. By this, we mean we have a uniform way of writing:

$$\sum_{i=0}^{n} x_i \quad \text{or} \quad \prod_{i \in I} v_i \quad \text{or} \quad \max_{i, v_i \neq w} \|v_i - w\|$$

Formally, the general notation is:

\big[op/nil]_(i ← r | P i) F

where r represents the list of indexes i for which the operation op is to be repeated; nil is the value to be return for the empty list of indexes (usually the neutral element for the operation, if it exists) while P is the property that the indexes have to respect; F is the expression over which the operation is iterated.

When translating the above formulas in Coq, in the first case we write:

\big[+/0]_(i < n) x i

Supposing that I and r are lists of indexes, the second formula is:

\big[∗/1]_(i ← I) v i

and the third:

\big[Rmax/0]_(i ← r | v i != w) (norm (v i) − w)

We note that sums and products indexed over natural numbers or over elements of a ring can be written with a more natural \sum or \prod notation. For example, the first formula can alternatively be written as: \sum_(i < n) x i .

The lemmas in the library of indexed operations are organized according to the properties of the operator op. Some lemmas work for any operator, others work only if op is a monoid law, others require an abelian monoid law and so on. Canonical structures and coercions play an important role here also. Details can be found in [2].

Making use of the indexed operations, a formalization of **matrices** with elements of type R is given. Matrices in $M_{p \times q}$(T) are represented as finite functions {ffun 'I_p ∗ 'I_q→T}. Notations are provided in order to simplify the work with matrices, for example the matrix

$$A \in M_{m \times n}(T), \quad A = [a_{ij}], \quad i \in \{1, \ldots, m\}, \quad j \in \{1, \ldots, n\}$$

is given by

Definition A : 'M[T]_(m,n) := \matrix_(i < m, j < n) a i j.

Operations on matrices are defined when the base type has a ring structure, so in order to get real matrices, we have to declare a ring structure on our standard Coq real numbers, denoted R. The hierarchy of algebraic structures is built on types with decidable equality and with a choice operator, so we have to begin by defining an eqType and a choiceType for R. To have the eqType structure on real numbers, we will base ourselves on the trichotomy axiom in the standard library `Reals` which implies that we can reason on cases on whether two reals are equal or not. But first we will go even further in our technical details and explain how this fits in Coq's formalism.

In Coq the type of logical propositions is Prop and it is a type with special features. To make it clearer, in Coq we have data which are in type Type and logical propositions on these data which are in type Prop. Data and propositions do not live at the same level, more precisely we can use data to build another data or a proposition but we cannot build a piece of data from a proposition, we can only build other propositions. In particular, if we have a disjunction P∨Q in Prop we cannot build a function that returns a certain piece of data based on whether P or Q is satisfied. This corresponds to a disjunction that is not necessarily decidable.

So, whenever we want to be able to distinguish two cases we use a similar construction under Type. This construction is {P} + {Q}, where P and Q are under type Prop but {P} + {Q} is under type Type. We can see it as a set with one element such that we can determine if this element is P or Q. This corresponds to a disjunction that is effectively decidable. In particular we can build functions that return a certain data based on whether P or Q is true.

In the `Reals` library, the trichotomy axiom is stated using this disjunction under Type. This means we can define a function R→R→bool that returns true if the two numbers are equal and false if they are not. This will be the boolean equality function in our eqType.

```
(* the trichotomy axiom *)
Axiom total_order_T : forall r1 r2:R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
(* lemma derived from the trichotomy axiom *)
Lemma Req_case : forall x y: R, {x = y} + {x <> y}.
(* definition for the boolean equality function *)
Definition eqr (x y : R) : bool := match (Req_case x y) with
  | left _ ⇒ true | right _ ⇒ false end.
(* lemma proving the equivalence between boolean and Leibniz equality
   *)
Lemma eqrP : forall x y, reflect (x = y) (eqr x y).
(* the canonical type for reals with a decidable equality *)
Canonical Structure real_eqType := EqType eqrP.
```

In order to endow R with a choiceType structure we need additional axioms in our logic, i.e. a version of the axiom of choice and the axiom of functional extensionality. The latter is needed because the choice operator on R needs to produce the same canonical element for two sets that are extensionally equal and for two proofs that the set is non-empty. Though present in our context, we limit the use of these axioms to this only instance.

Now we have the base properties on R needed to define the algebraic hierarchy. We endow the real numbers with canonical structures for group, ring, commutative ring and field. These Canonical Structure declarations make all theorems regarding the algebraic structures directly available for the reals. The use of canonical structures will also allow us to use freely all the existing theorems on the real numbers. We will be able to have real matrices and have all the results on SSReflect matrices available.

***

2.2 Real Matrices

Though all results in the generic SSReflect matrix library can directly be used for real matrices, there are still other notions, specific to real matrices that are not part of the generic library. We note that the development on real matrices was done to cover the concepts needed in proofs for numerical methods. It does not treat all concepts on real matrices one would expect to have.

If $\mathbb{R}$ denotes the set of real numbers, the set of real matrices with m lines and n columns is $M_{m\times n}(\mathbb{R})$. A matrix in this set is

$$A = [A_{ij}]_{m\times n}, \quad A_{ij} \in \mathbb{R}, \quad i \in \{1,\dots,m\}, \quad j \in \{1,\dots,n\}$$

We need to generalize some basic real number concepts to matrices. This is done in a componentwise manner. We define the absolute value function $|A| = [|A_{ij}|]$. In Coq, where Rabs is the absolute value of a real number, we get

```
Definition Mabs (A: 'M[R]_(m, n)) := \matrix_(i, j) Rabs (A i j).
```

Similarly, a comparison relation $\omega \in \{\leq, <, \geq, >\}$ for two matrices A and B is given by $A\ \omega\ B \Leftrightarrow \forall ij, A_{ij}\ \omega\ B_{ij}$.

We formalize canonical norms for matrices. A canonical matrix norm according to [8] is an operator $\|\cdot\| : M_{m\times n}(\mathbb{R}) \to \mathbb{R}$ with the following properties

- $\forall A, 0 \leq \|A\|$
- $\forall \alpha \in R, \forall A, \|\alpha A\| \leq |\alpha|\|A\|$
- $\forall AB, \|A+B\| \leq \|A\| + \|B\|$
- $\forall AB, \|AB\| \leq \|A\|\|B\|$
- $\forall Aij, |A_{ij}| \leq \|A\|$
- $\forall AB,$ if $\forall ij, |A_{ij}| \leq |B_{ij}|$ then $\|A\| \leq \|B\|$

We are interested in relating the value of the norm of the matrix to the value of its determinant, for a square matrix of size p. The relation is the following

$$\|A\| < 1 \Rightarrow \det(E_p - A) \neq 0 \tag{2}$$

where E_p denotes the identity square matrix of size p. In order to do this proof we need to talk about sequences and series of matrices. Given a sequence of matrices $(A_k)_{k\in\mathbb{N}}$

$$A_k = \left[a_{ij}^{(k)}\right], \ (k = 1, 2, \dots)$$

we define the limit of this sequence componentwise.

$$A = \lim_{k\to\infty} A_k = \left[\lim_{k\to\infty} a_{ij}^{(k)}\right]$$

We get the following relation between canonical matrix norms and convergence:

$$\lim_{k\to\infty} A_k = A \Leftrightarrow \lim_{k\to\infty} \|A - A_k\| = 0$$

Series of matrices are defined as

$$\sum_{k=0}^{\infty} A_k = \lim_{N\to\infty} \sum_{k=0}^{N} A_k \tag{3}$$

If the above limit exists, the series is called convergent.

Let us now detail some issues that arise when dealing with indexed sums in Coq.

Technical Details 2 (Finite sums)**.** The implementation of series is not complicated. A series is just a limit of a sequence, where the elements of the sequence are finite sums. We already saw in technical section 1 that SSReflect has a library for dealing with indexed operations. Finite sums are a special case of such operations. The notion of series of real numbers is formalized in Coq's standard library `Reals`. But in this formalization, we do not have the SSReflect notion of finite sums. Instead we have a less general notion of sum: sum of real numbers indexed over natural numbers, denoted in the code below by sum_f_R0. All properties of series are proved using this formalization. However, we would like to benefit from the formalization on indexed operations in SSReflect, denoted in the code below by \sum. Based on the two formalizations of sums, we give two definitions of convergence of matrix series, that we prove equivalent.

```
(* convergence of a matrix series according to the definition 3 *)
Definition cv_mat_ser (Ak: nat → 'M_(p, q)) (A: 'M_(p, q)) :=
  limit_m (fun N ⇒ \sum_(i < N.+1) Ak i) A.
(* limit_m is the definition for the convergence of a real matrix
    sequence *)

(* convergence of a matrix series as convergence on components *)
Definition cv_mat_ser_comp (Ak: nat → 'M_(p, q)) (A: 'M_(p, q)) :=
  forall i j, Un_cv (fun N ⇒ sum_f_R0 (fun n => Ak n i j) N) (A i j).
(* Un_cv is the standard library definition for the convergence of a
    sequence of real numbers *)
```

In the first definition the sum is a sum of matrices, so this definition will help us when we need to manipulate indexed sums. The second definition uses concepts of convergence of sequences of real numbers, so this definition will help us when we need to prove results on the convergence of series of matrices by reducing it to convergence of series of real numbers.

***

A series of matrices is absolutely convergent if the following series is convergent.

$$\sum_{k=1}^{\infty} |A_k| = \left[\sum_{k=1}^{\infty} \left| a_{ij}^{(k)} \right| \right]$$

We get the following relation between norm and absolute convergence:

$$\sum_{k=1}^{\infty} \|A_k\| \text{ convergent } \Rightarrow \sum_{k=1}^{\infty} A_k \text{ absolutely convergent}$$

We note that we do not have a formalization of absolute convergence for a series of real numbers in Coq's standard library. We needed to prove the above statement in the case of real numbers before generalizing it to matrices.

We are interested in the special case of series of the form

$$\sum_{k=1}^{\infty} A^k$$

We show that such a series converges if $\|A\| < 1$.
We consider the associated partial sum which verifies the equality:

$$(E_p + A + A^2 + \ldots + A^k)(E_p - X) = E_p - A^{k+1}$$

Passing at the limit in this identity gives

$$S(E_p - A) = E_p, \text{ where } S = \sum_{k=1}^{\infty} A^k$$

Therefore

$$\det S * \det(E_p - A) = \det E_p = 1$$

and we conclude

$$\det(E_p - A) \neq 0$$

To summarize, we have accomplished our goal and proved equality 2.

```
Lemma matr_inv_norm: forall A, norm A < 1 → \det (1 − A) ≠ 0.
```

All the above definitions and results are formalized using an abstract canonical norm for matrices. We instantiate this abstract norm to the following maximum norm.

$$\|A\| = \max_i \sum_j |a_{ij}| \tag{4}$$

2.2.1 Related formalizations

Developments on matrices exist in several proof assistants. The quantity of results formalized varies. Most of them implement matrices with elements from a ring. All developments treat operations on matrices and their properties. In Isabelle/HOL [22] implements matrices in order to deal with linear programs and treats the special case of sparse matrices. In ACL2 we have [5] where matrices are implemented in a way that insures computation efficiency. In HOL Light we have [15]. In Coq there are several formalizations for matrices and linear algebra. We cite [20] and [25] as standard Coq contributions, and [3] as the implementation of matrices using the SSReflect extension of Coq that we used as a basis of our development.

2.3 Multivariate Analysis

For the purposes of our formalization of numerical methods we need concepts from the field of multivariate analysis like, for example, functions of several variables that are partially derivable and properties of the partial derivatives. However, Coq's libraries only deal with real analysis, so we need to develop the notions from multivariate analysis ourselves. We start by formalizing real vectors in Coq.

2.3.1 Real vectors

The choice of implementation for vectors in the Coq - SSReflect framework seems obvious: do the same as for matrices. We can "do the same" in two ways:

- say that vectors are a special kind of matrices (either $1 \times p$ or $p \times 1$ matrices, depending on whether we consider row vectors or column vectors), and use the existing implementation on matrices,
- do a specific implementation for vectors, inspired by that of matrices.

We chose to do a specific implementation for vectors, as it feels like we should be able to consider vectors apart from matrices. We might revise our choice for future developments.

We implement a vector of length p with elements of a certain type T as a function from the finite domain $\{0, 1, \ldots, p-1\}$ to T and we call this type vec T p. For a vector v: vec T p, we write v i for the i-th component of the vector. It is similar to the mathematical use of having a vector $v = (v_0, v_1, \ldots, v_{p-1})$.

As specific notions for real vectors, we define operations on vectors componentwise: addition, opposite, multiplication by a scalar. Here is the example of addition, where the notation \vec is used for building a vector.

```
Definition add_v (u v: vec R p) := \vec_(i < p) u i + v i.
```

We use the notations $+\hat{}$, $-\hat{}$ and $*\hat{}$ for addition, opposite and multiplications on vectors. Equality on vectors is equivalent to the equality on components.

```
Lemma vecP : forall u v: vec R p, (∀ i, u i = v i) ↔ u = v.
```

As operations on real vectors are defined componentwise, properties of these operations are proved by simply reducing them to properties on the real numbers. As we remarked previously, for proofs involving operations on real numbers we benefit from tactics like ring, field and fourier provided by Coq, which automatically solve a large variety of equalities and inequalities on the reals.

As vectors and matrices do not have the same type in our implementation, we need to define how they interact, in particular we need to define multiplication of a matrix by a vector. We suppose we have column vectors, so we have multiplication to the right:

$$(Av)_i = \sum_j A_{ij} * v_j$$

On the long term, vectors will be endowed with a vector space structure. For the moment this structre is not available in the SSReflect libraries.

We define the notion of norm for real vectors as an operator $\|\cdot\|$: vec R p→R that respects:

- positive definedness $\forall v, 0 \leq \|v\|$
- positive homogeneity $\forall av, \|av\| \leq |a|\|v\|$
- triangle inequality $\forall uv, \|u+v\| \leq \|u\| + \|v\|$

We prove properties on this abstract norm, that we then instantiate to the maximum norm

$$\|v\| = \max_i |v_i|$$

Technical Details 3 (Maximum as an indexed operation)**.** Defining this norm is straightforward using the library on indexed operations:

```
Definition norm_max (v: vec R p) := \big[Rmax/0]_(i < p) Rabs (v i).
```

Proving the good properties for the norm is done by using properties already proved for \big. For example, a lemma stating the positivity of the norm

```
Lemma norm_max_pos : forall v, 0 ≤ norm_max v.
```

can be easily proved by applying a generic lemma named big_prop. It states that if we have an operator (here, Rmax - the maximum of two real numbers) and a property P(x) (here, $0 \leq x$) which is

- closed with respect to the operator op (here, $\forall xy, 0 \leq x \wedge 0 \leq y \Rightarrow 0 \leq$ Rmax x y)
- satisfied by the default value (here, $0 \leq 0$)
- satisfied by the formula for every index (here $\forall i, 0 \leq$ Rabs (v i))

then the property P is also satisfied by the indexed operation (here, exactly what we need to prove, that is $0 \leq$ big[Rmax /0]_(i < p) Rabs (v i)).

Nevertheless, the use of the maximum as an indexed operation posed some difficulties. As stated before (see technical details 1), the lemmas on indexed operations are organized in a sort of hierarchy following the algebraic structure given by the operator. In the case of the maximum, we have associativity and commutativity, but we do not have a neutral element on the type of real numbers. Since we work only with positive numbers (and the maximum on this subset has 0 for neutral element), we would like to be able to use the lemmas that deal with an abelian monoid structure, as we know that this is the case on the subset we work on.

There are two possible solutions for this problem. The first is to have a new type for positive reals. We can define the canonical structure of abelian monoid on this new type, manipulate the indexed operation as desired and inject the result in the original type. The second solution is to define a new operator that gives the type the desired structure. This operator has to be equal to the original one on the target subset (here, the positive reals). We can then move freely between the two operators thanks to the lemmas in the indexed operations library. We adopted this second approach, as we had a construction at hand:

$$\max{}'\ x\ y = \begin{cases} \max\ x\ y & \text{if } x > 0 \vee y > 0; \\ \min\ x\ y & \text{if } x \leq 0 \wedge y \leq 0 \end{cases}$$

Basic lemmas on the norm are proved by moving to this equivalent operator and using the indexed operation library.

***

Other norms can be instantiated without difficulty. This maximum norm is compatible with the matrix norm defined in section 2.2 (relation 4), in the sense that

$$\|Av\| \leq \|A\|\|v\|$$

We can define a distance on $\mathbb{R}^P$ based on the norm operator.

$$\text{dist\_Rp}\ (u, v) = \|u - v\|$$

The properties for the distance follow naturally from those of the norm to ensure that, in our representation, $\mathbb{R}^p$, equipped with the above defined distance, is a metric space.

2.3.2 Metric spaces: convergence, limit, continuity

To fix concepts, we recall that a metric space is a set M with a function dist : $M \times M \to \mathbb{R}$ that satisfies the following:

- $\forall xy, 0 \leq \text{dist}\ (x, y)$ and $\text{dist}\ (x, y) \leftrightarrow x = y$
- $\forall xy, \text{dist}\ (x, y) = \text{dist}\ (y, x)$ (symmetric)
- $\forall xyz, \text{dist}\ (x, z) \leq \text{dist}\ (x, y) + \text{dist}\ (y, z)$ (triangle inequality)

In a metric space $(M,\ \text{dist})$, a sequence $(X_n)_{n\in\mathbb{N}} \subseteq M$ is called convergent to the limit l and we note $\lim\limits_{n\to\infty} X_n = l$ if:

$$\forall\varepsilon \in \mathbb{R}, 0 < \varepsilon \Rightarrow \exists N \in \mathbb{N} \text{ such that } \forall n \in \mathbb{N}, N \leq n \Rightarrow \text{dist}\ (X_n, l) < \varepsilon$$

This is straightforwardly translated in Coq

```
Definition conv (M: Metric_Space) (Xn: nat → M) (l: M) :=
  forall eps: R, 0 < eps → exists N: nat, (forall n:nat, N ≤ n → dist (Xn n) l <
      eps).
```

We also define what it means for the sequence $(X_n)_{n\in\mathbb{N}}$ to satisfy Cauchy's criterion:

$$\forall\varepsilon \in \mathbb{R}, 0 < \varepsilon \Rightarrow \exists N \in \mathbb{N} \text{ such that } \forall m, n \in \mathbb{N}, N \leq m, N \leq n \Rightarrow \text{dist}\ (X_m, X_n) < \varepsilon)$$

We formally show that in all metric spaces, the limit of a sequence is unique and a convergent sequence satisfies Cauchy's criterion.

A metric space where all Cauchy sequences are convergent is called a complete metric space. We prove completeness in the case of the metric space $\mathbb{R}^p$. We also prove that convergence in $\mathbb{R}^p$ according to the above definition is equivalent to the convergence on components.

$$\lim_{n\to\infty} X_n = l \Leftrightarrow \forall i \in \{0, \ldots, p-1\}, \lim_{n\to\infty} (X_n)_i = l_i$$

In similar terms we talk about limits of functions between metric spaces. If we have two metric spaces $(M,\ \text{dist}_M)$ and $(M',\ \text{dist}_{M'})$, we say that the limit of a function $f : M \to M'$ at a point $x_0 \in M$ is $l \in M'$ in the following manner:

$$\forall \varepsilon > 0, \exists \alpha > 0, \forall x \in M, 0 < \text{dist}_M(x, x_0) < \alpha \Rightarrow \text{dist}_{M'}(f(x), l) < \varepsilon \tag{5}$$

We also define what it means for a function $f : M \to M'$ to be continuous at a point x_0: the limit in x_0 is equal to the value of the function at x_0.

$$\forall \varepsilon > 0, \exists \alpha > 0, \forall x \in M, \text{dist}_M(x, x_0) < \alpha \Rightarrow \text{dist}_{M'}(f(x), f(x_0)) < \varepsilon$$

In the special case of $\mathbb{R}^p$ our development contains basic results like: the limit of the sum of two functions is the sum of the two limits, the limit in $\mathbb{R}^p$ is unique, relations between convergence and continuity in $\mathbb{R}^p$, the limit of a function is a limit on components.

2.3.3 Derivatives

We begin our study of derivatives by implementing partial derivatives for functions from $\mathbb{R}^p$ to $\mathbb{R}$. We say a function f is partially derivable at a point a with respect to the i-th component if the following limit exists:

$$\lim_{t \to 0} \frac{f(a + t \cdot e_i) - f(a)}{t}$$

where e_i is the i-th vector of the canonical base, that is the vector with all zeros and a one in the i-th position. The value of this limit is denoted $\dfrac{\partial f(a)}{\partial x_i}$ and is called the partial derivative of f in a with respect to variable x_i. This is equivalent to having a function where we fixed all other variables except for x_i and we derive this real function.

The implementation of partial derivatives follows the implementation of derivatives in the Coq standard library.

Technical Details 4 (Implementation of derivatives)**.** We define the partial derivative of a function in three steps. We first express the property "the function f is partially derivable at a point a with respect to the i-th component and the value of the partial derivative is dp" by using the concept of limit on real functions.

```
Definition part_deriv_pt_1 (f: vec R p → R)(a: vec R p)(i: 'I_p)(dp: R) : Prop:=
  limit (fun t ⇒ (f (a +^ t *^ (base_v i)) − f a) / t) 0 dp.
```

Then, we define a real number with the above property.

```
Definition dbl_pt_1 (f: vec R p → R)(i: 'I_p)(a: vec R p):=
  {dp | part_deriv_pt_1 f a i dp}.
```

And we obtain the real number that is the value of the partial derivative by taking the first projection of the above data structure.

```
Definition dp_1 f i a (pr: dbl_pt_1 f i a) := projT1 pr.
```

By using the function dp_1, every time we have a proof that a function is partially derivable we can get the value of the partial derivative.

In the same manner we define partial derivatives for functions vec R p → vec R p.

Second order partial derivatives are defined as the derivative of the first order derivative, so the definition takes in argument a proof that the first order partial derivatives are partially derivable.

```
Definition part_deriv_pt_1_2 f a i j (pr1: forall v, dbl_pt_1 f i v) dp2 :=
 part_deriv_pt_1 (fun v ⇒ dp_1 f i v (pr1 v)) a j dp2.
```

We can get the value of the second order derivative using the same three step construction as above.

***

We show basic properties of the derivation operator like linearity. We also relate the different notions between them and to derivation in one dimension. For instance, a function that has second order partial derivatives will trivially have first order partial derivatives; the partial derivative of a vectorial function is the vector of the partial derivatives of the component functions. An elegant example of a "paper" proof is the following:

$$\begin{aligned} f(x_1,\ldots,x_p) &- f(y_1,\ldots,y_p) = f(x_1,\ldots,x_p) - f(y_1,x_2,\ldots,x_p) + \\ &+ f(y_1,x_2,\ldots,x_p) - f(y_1,y_2,x_3,\ldots,x_p) + \ldots + \\ &+ f(y_1,\ldots,y_{p-1},x_p) - f(y_1,\ldots,y_p) = \\ &= \sum_{i=1}^{p}(x_i-y_i)\frac{\partial f(y_1,\ldots,y_{i-1},c_i,x_{i+1},\ldots,x_p)}{\partial x_i} \end{aligned}$$

It is also an example of the implicit or intuitive reasoning a human reader makes to replace the ... or to realize that the indexes $i-1$, $i+1$ are only used where they make sense. Another implicit view is interpreting the difference $f(x_1,\ldots,x_p)-f(y_1,x_2,\ldots,x_p)$ of a vector function varying in the first argument as a real function. All these are non-trivial reasoning steps for a mechanized system.

The most involved result we needed for our development is Taylor's formula for functions of class $C^{(2)}$. The statement and the proof are as follow:

Lemma 1 (Taylor second degree)**.** *Let $f : \mathbb{R}^p \to \mathbb{R}$ be twice partially derivable with continuous first and second partial derivatives, then for all $a \in \mathbb{R}^p$ and $v \in \mathbb{R}^p$ there exists $c \in (a, a+v)$ such that* $f(a+v) = f(a) + \sum_{i=1}^{p}\frac{\partial f(a)}{\partial x_i}v_i + \frac{1}{2!}\sum_{i,j=1}^{p}\frac{\partial^2 f(c)}{\partial x_i \partial x_j}v_i v_j$.

Proof. Consider

$$g : [0,1] \to \mathbb{R},\ \ g(t) = f(a+tv)$$

then g is twice derivable on $[0,1]$ and

$$g'(t) = \sum_{i=1}^{p}\frac{\partial f(a+tv)}{\partial x_i}v_i \tag{6}$$

$$g''(t) = \sum_{i,j=1}^{p}\frac{\partial^2 f(a+tv)}{\partial x_i \partial x_j}v_i v_j \tag{7}$$

From the Taylor formula in one dimension we get that there exists $\eta \in (0,1)$ so that

$$g(1) = g(0) + g'(0) + \frac{1}{2!}g''(\eta)$$

which gives us the desired result for $c = a + t\eta \in (a, a+v)$. □

The proof of this theorem is based on the proof of the Taylor formula in one dimension, which we also formalized. Also, an important issue for this proof is to show some relations between various concepts of differentiability, i.e. to prove equalities (6) and (7).

We also associate to a function

$$f : \mathbb{R}^p \to \mathbb{R}^p, \quad f(X) = f(x_1, \ldots, x_p) = (f_1(x_1, \ldots, x_p), \ldots, f_p(x_1, \ldots, x_p))$$

a matrix called the Jacobian and given by

$$J_f(X) = J_f(x_1, x_2, \ldots, x_p) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(X) & \frac{\partial f_1}{\partial x_2}(X) & \ldots & \frac{\partial f_1}{\partial x_p}(X) \\ \frac{\partial f_2}{\partial x_1}(X) & \frac{\partial f_2}{\partial x_2}(X) & \ldots & \frac{\partial f_2}{\partial x_p}(X) \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial f_p}{\partial x_1}(X) & \frac{\partial f_p}{\partial x_2}(X) & \ldots & \frac{\partial f_p}{\partial x_p}(X) \end{pmatrix}$$

2.3.4 Related formalizations

The only proof assistant that already has a formalization of multivariate analysis is HOL Light [15]. In this formalization, vectors are also implemented as functions from a finite type to real numbers. The topics covered include linear algebra: operators, matrices, determinants; topology: open, closed, compact, convex sets; sequences, continuity, differentiability; basic calculus theorems: mean value theorem, inverse function theorem.

3 Newton's Method

Newton's method is one of the well known methods from numerical analysis for finding successively better approximations for the roots of a function f. Given a staring point x_0 we compute the sequence of approximations in the following manner:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Let's look at an example:

$$f(x) = x^3 - 7$$

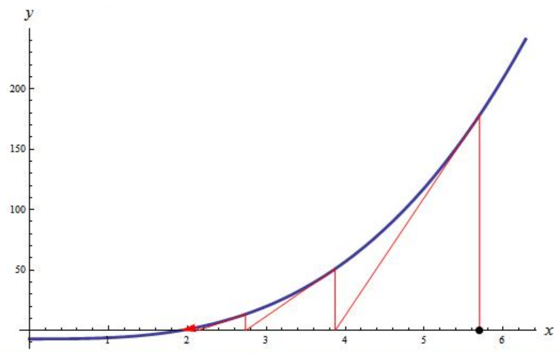

Figure 1: Newton's method for $f(x) = x^3 - 7$

The root of f is $\sqrt[3]{7} \approx 1.9129$. Let's start the search for the root using Newton's method at

$$x_0 = 5.7$$

for the first five approximations we get:

$$5.7,\ 3.871816969,\ 2.736860604,\ 2.136083331,\ 1.935431626,\ 1.913191749$$

It seems that the sequence converges indeed to the root of f. To better see what is going on, look at Figure 1.

The formula for Newton's method can be deduced from the first terms of the Taylor series of the function f at a point x.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0) + \dots$$

Keeping only the first order terms we get:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \tag{8}$$

From equation 8 we get precisely the equation of the tangent line to the curve at point $(x_0, f(x_0))$

$$y = f(x_0) + f'(x_0)(x - x_0)$$

This tangent line intersects the $x-$axis at point $(x_1, 0)$ given by

$$0 = f(x_0) + f'(x_0)(x_1 - x_0) \Leftrightarrow$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

For a well chosen x_0, the computed x_1 is a better approximation of the root of f. Again, the graph gives us an intuitive idea that this is the case, for our example. We can repeat the process from x_1 in order to get finer approximations.

However, it is not always the case that the new point will be closer to the root than the old one. Consider for example the function:

$$f(x) = 1 - x^2$$

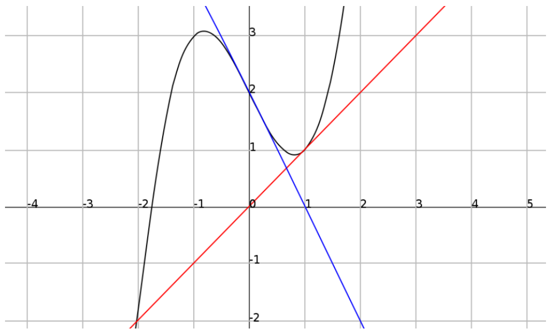

Figure 2: Newton's method oscillates for $f(x) = x^3 - 2x + 2$ and $x_0 = 0$

If we start the iteration at $x_0 = 0$ we get

$$x_1 = 0 - \frac{f(0)}{f'(0)} = 0 - \frac{1}{0}$$

which is undefined.
We look at a second example:

$$f(x) = x^3 - 2x + 2$$

with starting point $x_0 = 0$. Then we have

$$x_1 = 0 - \frac{f(0)}{f'(0)} = 0 - \frac{2}{-2} = 1$$

$$x_2 = 1 - \frac{f(1)}{f'(1)} = 1 - \frac{1}{-1} = 0$$

We get an oscillating sequence of 0 and 1 without converging to the root, as illustrated in Figure 2.
For one last example we take

$$f(x) = \sqrt[3]{x}$$

and the initial approximation $x_0 = 1$. We compute the general formula for Newton's sequence

$$x_{n+1} = x_n - \frac{{x_n}^{\frac{1}{3}}}{\frac{1}{3}{x_n}^{-\frac{2}{3}}} = x_n - 3x_n = -2x_n$$

The root of the function is 0, but the terms of the sequence will get further and further away from the root

$$x_0 = 1,\ x_1 = -2,\ x_2 = 4,\ x_3 = -8,\ x_4 = 16, \ldots$$

These examples show that Newton's method is not always convergent. Using this method with inappropriate functions and initial values can give undesired results. In order to get the expected behavior the function and the initial point need to satisfy some conditions that we will discuss in detail in section 3.1.

Newton's method can be generalized to find approximations for roots of a function

$$f : \mathbb{R}^p \to \mathbb{R}^p$$

In this case we have

$$f(X) = (f_1(X), \dots, f_p(X)),\ X = (x_1, \dots, x_p)$$

Finding a root of f means solving the following system of equations:

$$\begin{cases} f_1(x_1, x_2, \dots, x_p) = 0 \\ f_2(x_1, x_2, \dots, x_p) = 0 \\ \dots \\ f_p(x_1, x_2, \dots, x_p) = 0 \end{cases}$$

To express Newton's method in this case, we need an equivalent of the derivative in several dimensions. This is the Jacobian matrix defined as defined in section 2.3.3. Then Newton's method becomes:

$$X_{n+1} = X_n - J_f(X_n)^{-1} f(X_n)$$

For Newton's method in higher dimensions the same issues arise as in the one dimensional case. Though the method is used to determine roots of functions, it is sometimes the case that the sequence does not converge. The convergence of the sequence is determined by properties of the function and the initial point. Several studies by Willers, Stéinine, Ostrowski, Kantorovitch and others are concerned with establishing sufficient conditions for the convergence of Newton's method. We are interested in Kantorovitch's theorem. We gave the statement in the case of a function of several variables in section 1 (theorem 1). We will now present the details of the proof and some related results. We start with the case of a real function.

3.1 Kantorovitch's theorem in the case of a real function

We present here Kantorovitch's theorem and some related results in the case of a function with one variable. This one dimensional case is a simplified model of the problem. The interest of treating it separately is to allow a better understanding for the structure of the proof. The one dimensional case reveals the key points of the proof as well as the places where the reasoning in the paper proof is difficult to pass on a machine.

Theorem 2 (Convergence)**.** *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 $\overline{U_\varepsilon}(x_0) = \{|x - x_0| \le \varepsilon\} \subset (a, b)$. If the following conditions hold:*

1. *$f'(x_0) \neq 0$ and $\left|\dfrac{1}{f'(x_0)}\right| \le A_0$;*
2. *$\left|\dfrac{f(x_0)}{f'(x_0)}\right| \le B_0 \le \dfrac{\varepsilon}{2}$;*
3. *$\forall x, y \in (a, b), |f'(x) - f'(y)| \le C|x - y|$*

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

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1, 2, \ldots \tag{9}$$

converges and $\lim\limits_{n\to\infty} x_n = x^*$ is a solution of the initial system, so that $|x^* - x_0| \leq 2B_0 \leq \varepsilon$.

Theorem 3 (Uniqueness)**.** *Under the conditions of Theorem 2 the root x^* of the function f is unique in the interval $[x_0 - 2B_0, x_0 + 2B_0]$.*

Theorem 4 (Speed of convergence)**.** *Under the conditions of Theorem 2 the speed of the convergence of Newton's method is given by*

$$|x_n - x^*| \leq \frac{1}{2^{n-1}}\mu_0^{2^n-1}B_0$$

Theorem 5 (Local stability)**.** *If the conditions of Theorem 2 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 $|x'_0 - x_0| \leq \dfrac{1-\mu_0}{2\mu_0}B_0$ the associated Newton's process converges to the root x^*.*

The theorem of convergence of Newton's method shows that this 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 convergence means we know a bound for the distance between a given element of the sequence and the root of the function. This distance 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 method helps with efficiency issues as it allows the use of an approximation instead of the exact value as we shall see in section 3.2.

We do not present here the proofs of the theorems, we just give a few elements of these proofs that will help understand how paper proofs relate to formal proofs and how formalized proofs can help discover new proofs. For detailed proofs we refer the reader to [8]. The outline of the proof for theorem 2 as presented in [8] is as follows:

- prove a collection of properties for each element of the Newton sequence, more precisely, show that hypotheses 1 - 4 are verified, with different constants, for every element of the sequence ;
- infer that Newton's sequence is a Cauchy sequence and, by the completeness of $\mathbb{R}$, a convergent sequence;
- prove that the limit of the sequence is a root of the given function.

The proof introduces the auxiliary sequences $\{A_n\}_{n\in\mathbb{N}}$, $\{B_n\}_{n\in\mathbb{N}}$ and $\{\mu_n\}_{n\in\mathbb{N}}$:

$$A_n = 2A_{n-1} \tag{10}$$

$$B_n = A_{n-1}B_{n-1}^2 C = \frac{1}{2}\mu_{n-1}B_{n-1} \tag{11}$$

$$\mu_n := 2A_nB_nC = \mu_{n-1}^2 \tag{12}$$

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:

$$\overline{U}_\varepsilon(x_0) \supset \overline{U}_{\frac{\varepsilon}{2}}(x^{(1)}) \supset \ldots \supset \overline{U}_{\frac{\varepsilon}{2^n}}(x_n) \supset \ldots \tag{13}$$

furthermore

$$f'(x_n) \neq 0 \text{ and } \left|\frac{1}{f'(x_n)}\right| \leq A_n \tag{14}$$

$$\left|\frac{f(x_n)}{f'(x_n)}\right| \leq B_n \leq \frac{\varepsilon}{2^{n+1}} \tag{15}$$

$$\mu_n \leq 1 \tag{16}$$

Notice that hypothesis 3 is a property of the function and it does not depend on the elements of Newton's sequence.

From (13) we can infer that x_n is a Cauchy sequence:

$$x_{n+m} \in \overline{U}_{\frac{\varepsilon}{2^n}}(x_n) \Rightarrow \|x_{n+m} - x_n\| \leq \frac{\varepsilon}{2^n}$$

The latter quantity can be made arbitrary small for $n > N$ and $m \in \mathbb{N}$, which is equivalent to Cauchy's criterion. We use the result that $\mathbb{R}$ is a complete metric space to deduce that the sequence converges. By taking the limit in (9) we get that the limit of the sequence is a root of function f.

To prove the uniqueness of the solution, we suppose that there exists another solution of the equation and prove that it is also the limit of the sequence. By uniqueness of this limit we have the desired result.

For Theorem 5 (local stability) we prove that the new initial approximation x'_0 satisfies similar hypotheses as those for x_0. The new constants are $A' = \dfrac{4}{3+\mu_0}A_0$ and $B' = \dfrac{3+\mu_0}{4\mu_0}B_0$. This implies $\mu' = 2A'B'C = 1$ and we can verify that

- ∘ $f'(x'_0) \neq 0$ and $\left|\dfrac{1}{f'(x'_0)}\right| \leq A'$
- ∘ $\left|\dfrac{f(x'_0)}{f'(x'_0)}\right| \leq B'$
- ∘ $\mu' \leq 1$

We are thus in the hypotheses of Theorem 2 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 (12)) and we would not be able to do an approximation again, because Theorem 5 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:

$$A' = \frac{8}{7+\mu_0} A_0 \tag{17}$$

$$B' = \frac{\mu_0^2 + 46\mu_0 + 17}{8(7+\mu_0)\mu_0} B_0 \tag{18}$$

this implies

$$\mu' = \frac{\mu_0^2 + 46\mu_0 + 17}{(7+\mu_0)^2} < 1 \tag{19}$$

We summarize these results in:

Corollary 1. *If the conditions of Theorem 2 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^**.*

3.2 Newton's method with rounding

In our description of Newton's method up to here we assumed that the computations are made with "true" real numbers. By this we mean that no rounding is performed during the computation. However, in actual applications the method is implemented on floating point numbers or on some other machine representable subset of real numbers. So rounding is performed at each step of Newton's method. The method we are actually performing is not Newton's method as described before, but a method that looks like:

$$t_0 = \ \mathrm{rnd}_0(x_0)$$

$$t_{n+1} = \ \mathrm{rnd}_{n+1}(t_n - \frac{f(t_n)}{f'(t_n)})$$

where rnd_n is the rounding performed at step n in the classical Newton's method.

It is reasonable to ask ourselves "Do the convergence results on the classical Newton's method remain true when using rounding in the computation? If so, under which conditions?" As empirical data suggests, Newton's method with rounding will still converge, but under stronger conditions. With the theorems presented so far, we have all the necessary tools to state and prove a theorem on the behavior of Newton's method with rounding at each step. The result presented in what follows is based on theorems 2 - 5 but it is an original result. We control the rounding at each step so the precision is just good enough to ensure the convergence. This result is particularly interesting for computations in arbitrary or multiple precision, as it relates the 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 intermediate values with the desired precision for the result.

Theorem 6 (Convergence with rounding)**.** *We consider a function $f : (a, b) \rightarrow \mathbb{R}$ and an initial approximation x_0 satisfying the conditions in theorem 2. We also consider a function $rnd : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{R}$ that models the approximation we will make at each step in the perturbed Newton sequence:*

$$t_0 = x_0 \textit{ and } t_{k+1} = rnd_{k+1}\left(t_k - \frac{f(t_k)}{f'(t_k)}\right)$$

If

1. $\forall k \forall x, x \in (a, b) \Rightarrow rnd_k(x) \in (a, b)$
2. $\frac{1}{2} \leq \mu_0 < 1$
3. $[x_0 - 3B_0, x_0 + 3B_0] \subset (a, b)$
4. $\forall k \forall x, |x - rnd_k(x)| \leq \frac{1}{3^k} R_0$, *where* $R_0 = \frac{1 - \mu_0^2}{8\mu_0} B_0$

then

a. *the sequence $\{t_k\}_{k\in\mathbb{N}}$ converges and $\lim_{k\rightarrow\infty} t_k = x^*$ where x^* is the root of the function f given by theorem 2*

b. $\forall k, |x^* - t_k| \leq \frac{1}{2^{k-1}} B_0$

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 2 - 5 and corollary 1. To give the intuition behind the proof, we decompose Newton's perturbed process t_n as follows:

i. set $t_0 := x_0$

ii. do a Newton iteration to get $x_1 := t_0 - \frac{f(t_0)}{f'(t_0)}$

iii. do an approximation of the result to get $t_1 :=$ rnd(x_1)

iv. set $t_0 := t_1$ and go to step ii

Now let's look at these steps individually:

- At step i we start with the initial x_0 that satisfies the conditions in theorem 2. This means that Newton's method from this initial point converges to the root x^* (see theorem 2).

- At step ii we consider a Newton sequence starting with x_1. This sequence is the same as the sequence at step i 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 (see proof of theorem 2) we can associate the constants A_1, B_1 to the initial iteration of this sequence and get the corresponding hypotheses from theorem 2.

- At step iii 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 2. 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 iii 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 iii 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. Hypothesis 4 suffices to ensure the convergence of the new process. For the proof we use the idea from theorem 2 and associate to each element of the perturbed sequence t_k the constants A'_k, B'_k, μ'_k. The behaviour of these associated constants help us prove the results we need.

To make the intuitive explanation more formal we consider the sequence of sequences of real numbers $\{Y_k\}_{k\in\mathbb{N}}$ defined as follows:
$Y_0^n = x_n$ is the original Newton's sequence;
Y_1 is given by
$Y_1^0 = \text{ rnd}_1(x_1)$;
$Y_1^{n+1} = Y_1^n - \dfrac{f(Y_1^n)}{f'(Y_1^n)}$ is the Newton's sequence associated to the initial iteration Y_1^0;
we continue in the same manner and for an arbitrary k we define Y_{k+1} as follows
$Y_{k+1}^0 = \text{ rnd}_{k+1}(Y_k^1)$;
$Y_{k+1}^{n+1} = Y_{k+1}^n - \dfrac{f(Y_{k+1}^n)}{f'(Y_{k+1}^n)}$.
We notice that taking the first element in each of these sequences forms our perturbed Newton's process:

$$Y_0^0 = x_0 = t_0 \text{ and}$$

$$Y_{k+1}^0 = \text{ rnd}_{k+1}\left(Y_k^0 - \frac{f(Y_k^0)}{f'(Y_k^0)}\right) = \text{ rnd}_{k+1}\left(t_k - \frac{f(t_k)}{f'(t_k)}\right) = t_{k+1}$$

We want to show that for each k, Y_k^0 is at a certain distance form the root x^*, which ensures the convergence of the perturbed Newton sequence at the desired speed. We start by explaining what happens when we do one step with the perturbed Newton's sequence. We go from t_0 to t_1 or, in the vocabulary we introduced in order to explain the proof), we go from Y_0^0 to Y_1^0, by following looking at what happens at each if the steps i -iii.

- We start with sequence $\{Y_0^n\}_{n\in\mathbb{N}}$. Since it coincides with the initial sequence, the properties from theorem 2 are trivially satisfied. For the initial point Y_0^0 we have the associated constants A_0, B_0, μ_0. For uniform notation we rename these constants $A'_0 = A_0, B'_0 = B_0, \mu'_0 = \mu_0$. By applying theorem 2 we get that
$$|x^* - Y_0^0| \leq 2B'_0$$

- We consider the sequence $\overline{Y}_0^n = Y_0^{n+1}$ (that is, the previously considered sequence where we start from the second element). This sequence also satisfies the conditions of theorem 2 where ew have as initial point $\overline{Y}_0^0 = Y_0^1$ and the associated constants $\overline{A}_0 = 2A'_0,\ \overline{B}_0 = A'_0{B'_0}^2C,\ \overline{\mu_0} = (\mu'_0)^2$. The laws for these constants are deduced from relations 10, 11 and 12 in section 3.1. We get that
$$|x^* - \overline{Y}_0^0| = |x^* - Y_0^1| \leq 2\overline{B}_0 = 2(A'_0{B'_0}^2C)$$

- Now we consider $\{Y_1^n\}_{n\in\mathbb{N}}$. The initial point of this sequence is $Y_1^0 = \mathrm{rnd}_1(\overline{Y}_0^0)$. We are in a situation where we have a converging sequence ($\{\overline{Y}_0^n\}_{n\in\mathbb{N}}$) and we introduce an approximation in the initial iteration. Such situation is described by corollary 1 in section 3.1. We verify that in our case, the hypotheses of corollary 1 are indeed satisfied. This consists in showing:
$$0 < \overline{\mu}_0 < 1$$
$$\left[\overline{Y}_0^0 - \frac{2}{\overline{\mu}_0}\overline{B}_0\ ,\ \overline{Y}_0^0 + \frac{2}{\overline{\mu}_0}\overline{B}_0\right] \subset (a,b)$$
$$\left|\ \mathrm{rnd}_1(\overline{Y}_0^0) - \overline{Y}_0^0\right| \leq \frac{1-\overline{\mu}_0}{4\overline{\mu}_0}\overline{B}_0$$

  The proof of these properties uses rather basic manupulations of the quantities involved.

  By applying corollary 1 we obtain the constants A'_1, B'_1, μ'_1 associated to the point Y_1^0 according to relations (17) and (18) in section 3.1
$$A'_1 = \frac{8}{7+\overline{\mu}_0}\overline{A}_0 = \frac{8}{7+\overline{\mu}_0}(2A'_0)$$
$$B'_1 = \frac{{\overline{\mu}_0}^2 + 46\overline{\mu}_0 + 17}{8(7+\overline{\mu}_0)\overline{\mu}_0}\overline{B_0} = \frac{{\overline{\mu}_0}^2 + 46\overline{\mu}_0 + 17}{8(7+\overline{\mu}_0)\overline{\mu}_0}(A'_0{B'_0}^2C)$$
$$\mu'_1 = 2A'_1B'_1C = \frac{\overline{\mu}_0^2 + 46\overline{\mu}_0 + 17}{(7+\overline{\mu}_0)^2} = \frac{{\mu'^2_0}^2 + 46{\mu'}_0^2 + 17}{(7+{\mu'}_0^2)^2}$$

  We find ourselves again in the conditions of theorem 2 and we can deduce that
$$|x^* - Y_1^0| \leq 2B'_1$$

We are also able to show that

$$B'_1 \leq \frac{1}{2} B'_0$$

We managed to deduce for Y_1^0 the same kind of properties as for Y_0^0 with different associated constants. This means we are in the appropriate conditions to start this process again for $\{Y_2^n\}_{n\in\mathbb{N}}$, $\{Y_3^n\}_{n\in\mathbb{N}}$, etc. We reason by induction on k. For each Y_k^0 we have the associated constants:

$$A'_0 = A_0 \text{ and } A'_{k+1} = \frac{8}{7+\overline{\mu}_k}(2A'_k)$$

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

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

where

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

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:

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

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.

Using the same reasoning steps as for Y_1^0, we get that

$$|Y_k^0 - x^*| \leq 2B'_k$$

$$B'_k \leq \frac{1}{2^k} B_0$$

These two relations trivially imply the convergence of the perturbed sequence to the root x^* at the desired speed, thus concluding our proof.

$\square$

3.3 Implementation in COQ for the real case

In this section we will show how we formalized the theorems from the previous section inside the Coq proof assistant. We first discuss the issues raised by working with derivatives inside a proof assistant.

3.3.1 Derivation

Let $f : \mathbb{R} \to \mathbb{R}$ be a derivable function on (a, b). "On paper" we can write the corresponding Newton's sequence $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, without worrying whether the term x_n is in the interval (a, b) where the function is derivable. The Coq formalization of derivatives requires that when we talk about the derivative of a function in a point, we provide a proof that the function is derivable at that point. The goal is to ensure that derivatives are properly used. In the case if Newton's sequence, for writing the definition of the sequence in Coq we would have to provide a proof that f is derivable in x_n. We can prove this for every n, but we need to define the sequence before being able to do this proof.

We worked around this impediment by defining a total function f' to use in the definition of Newton's sequence. We then imposed that on the interval (a, b) f' is equal to the derivative of f. This is equivalent to using Hilbert's epsilon operator. We are now able to define our sequence and at the same time we are prevented from using properties of the derivative in a point before proving the function is derivable there.

3.3.2 Formal proofs

The proofs consist in basically reproducing the reasoning presented in section 3.1 inside the proof assistant Coq. Most of the real analysis needed in the proof was already available in Coq's standard library. Only simple lemmas were needed in addition. When translating paper proofs on a machine we sometimes need to adapt the structure of the proof to the features of the formal system. For Kantorovitch's theorem, in the paper proof we have the sequences $\{A_n\}_{n\in\mathbb{N}}$, $\{B_n\}_{n\in\mathbb{N}}$ and $\{\mu_n\}_{n\in\mathbb{N}}$ (see section 3.1) that are defined in an ad-hoc manner during the proof of the theorem. In the proof assistant Coq we need to define them separately, before starting the proof of the theorem. This allows the user to better understand their importance and use similar sequences in proving new results like theorem 6.

3.4 Moving to several dimensions

The formalization of multivariate analysis concepts presented in section 2.3 was done to cover the background mathematics necessary for the proof of Kantorovitch's theorem. The structure of the proof in several dimensions is the same as in one dimension (see section 3.1). The formalization done for the real case is a good guide in our development as the hypothesis and lemmas needed are just a generalization of the ones for the real case.

For example, the properties of the absolute value generalize to those of the vector and matrix norm, as the absolute value is a norm for the real numbers. The inverse of a real number finds an equivalent in the inverse of a matrix, so a result like

$$|1 - t| \le \frac{1}{2} \Rightarrow t \neq 0$$

generalizes to

$$\|E_p - A\| \le \frac{1}{2} \Rightarrow \det A \neq 0.$$

This adaptation is sometimes far from trivial as we saw for the latter example which has been detailed in section 2.2. For Kantorovitch's theorem, however, the most difficult work of generalization concerns concepts and results from multivariate analysis that have been covered in sections 2.2, 2.3.

4 Conclusion

We presented a complete formal study of theoretical properties of Newton's method, which covers general theories involved in the proofs, well-known results on the method as well as new results for a modified version of Newton's method, for computations in finite precision. The entire formal development is available online: `http://www-sop.inria.fr/marelle/Ioana.Pasca/code`

A preliminary version of the formalization on multivariate analysis and the Kantorovitch's theorem has been presented by the author in [24]. At that time the formalization of multivariate analysis was not complete. The paper [18] shows how theorem 6 can be used in the context of verification of exact real number computations.

Our development is carried out inside the proof assistant Coq and the SSReflect extension of Coq. However, some issues discussed here may apply to other proof assistants. For example, implementations of multivariate analysis are not available in proof assistants with the exception of HOL Light. But, most proof assistants do have a real analysis library, comparable to that of Coq and that can serve as a basis for multivariate analysis. For the interested reader we point out the following work on real analysis: in Isabelle [9], in PVS [7], in HOL [14], in ACL2 [10]. Once embarked on the formalization of multivariate analysis we need to carefully choose a representation of vectors well suited for the type system we work in. Derivatives are also delicate to handle in a proof assistant, because we will always need to prove that a function is derivable in a point before talking about the derivative at that point. Multivariate analysis cannot be approached without talking about matrices. Here also, proof assistants provide more or less complex libraries (see section 2.2.1).

For the reader interested in the implementation details of our formalization we provide special technical sections. These include explanations for Coq features, description for Coq and SSReflect libraries and implementation choices in the described development.

We also present an original result concerning the convergence of Newton's method with rounding at each step in theorem 6. This proof of convergence has an interest from a proof engineering point of view. We were able to come up with the proof because we had formalized theorems 2 - 5 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). 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.

To the best of the author's knowledge, the result and proof of Theorem 6 are new, though the author is not an expert 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.

Since we are talking about a numerical method, it seems natural to have computations with Newton's method. Real number computations can be performed inside proof assistants by using exact real arithmetic libraries like [17, 23, 19]. Some results around exact computation with Newton's method are discussed by Julien and the author in [18], where Theorem 6 played a crucial role, as Newton's method with rounding at each step is a lot more efficient. In the future we could also investigate the use of Theorem 6 in the validation of computation with Newton's method on floating point numbers.

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, Georges Gonthier, Sidi Ould Biha, and Ioana Paşca. Canonical Big Operators. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, August 2008.

[3] Sidi Ould Biha. Formalisation des mathématiques : une preuve du théorème de Cayley-Hamilton. In *Journées Francophones des Langages Applicatifs*, pages 1–14, 2008.

[4] Coq development team. *The Coq Proof Assistant Reference Manual, version 8.2*, 2009.

[5] John Cowles, Ruben Gamboa, and Jeff Van Baalen. Using ACL2 Arrays to Formalize Matrix Algebra. In *ACL2 Workshop*, 2003.

[6] David Delahaye and Micaela Mayero. Quantifier Elimination over Algebraically Closed Fields in a Proof Assistant using a Computer Algebra System. In *Proceedings of Calculemus 2005*, volume 151(1) of *ENTCS*, pages 57–73, 2006.

[7] B. Duerte. Elements of Mathematical Analysis in PVS. In *Proceedings of the Ninth International Conference on Theorem Proving in Higher-Order Logics (TPHOL '96)*, 1996.

[8] B. Démidovitch et I. Maron. *Éléments de calcul numérique.* Mir - Moscou, 1979.

[9] 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.

[10] R. Gamboa and M. Kaufmann. Nonstandard Analysis in ACL2. *Journal of automated reasoning*, 27(4):323–428, November 2001.

[11] François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging mathematical structures. In *TPHOLs*, pages 327–342, 2009.

[12] Georges Gonthier. Notation of the Four Colour Theorem proof. Available at `http://research.microsoft.com/~gonthier/4colnotations.pdf`.

[13] Georges Gonthier and Assia Mahboubi. A small scale reflection extension for the coq system. INRIA Technical report, available at `http://hal.inria.fr/inria-00258384`.

[14] John Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998.

[15] John Harrison. A HOL Theory of Euclidian Space. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *LNCS*, pages 114–129. Springer, 2005.

[16] John Harrison and Laurent Théry. A Skeptic's Approach to Combining HOL and Maple. *J. Autom. Reasoning*, 21(3):279–294, 1998.

[17] Nicolas Julien. Certified exact real arithmetic using co-induction in arbitrary integer base. In *Functional and Logic Programming Symposium (FLOPS)*, LNCS. Springer, 2008.

[18] Nicolas Julien and Ioana Pasca. Formal Verification of Exact Computations Using Newton's Method. In *TPHOLs 2009*, volume 5674 of *LNCS*, pages 408–423, 2009.

[19] 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.

[20] Nicolas Magaud. Ring properties for square matrices. `http://coq.inria.fr/contribs-eng.html`.

[21] Micaela Mayero. *Formalisation et automatisation de preuves en analyses reelle et numerique.* PhD thesis, Université de Paris VI, 2001.

[22] Steven Obua. Proving bounds for real linear programs in isabelle/hol. In *Theorem Proving in Higher-Order Logics*, pages 227–244, 2005.

[23] 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.

[24] Ioana Paşca. A Formal Verification for Kantorovitch's Theorem. In *Journées Francophones des Langages Applicatifs*, pages 15–29, 2008.

[25] J. Stein. Documentation for the formalization of Linerar Agebra. `http://www.cs.ru.nl/~jasper/`.

Contents

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399