Formal Verifcation for Numerical Methods

Ioana Pasca

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Spécialité : Informatique

Présentée et soutenue par

Ioana PAŞCA

Vérification formelle pour les méthodes numériques

Thèse dirigée par Yves BERTOT

et préparée à l'INRIA Sophia Antipolis - Méditerranée, équipe - projet MARELLE

soutenue le 23 novembre 2010

Jury :

Yves BERTOT	-	Directeur de recherche, INRIA (Directeur)
Thierry COQUAND	-	Professeur, Université de Gothenburg (Rapporteur)
Jean-Christophe FILLIÂTRE	-	Chargé de recherche, CNRS (Examinateur)
Herman GEUVERS	-	Professeur, Université de Nijmegen (Rapporteur)
Carlos SIMPSON	-	Directeur de recherche, CNRS (Examinateur)
Benjamin WERNER	-	Directeur de recherche, INRIA (Président)

Familiei mele.

Acknowledgements

I am very happy I managed to do this work and I am even happier that during this time I had the chance of being surrounded by a bunch of wonderful people. They have influenced me and this work in more ways than one.

Thank you, Yves, for being my supervisor, always available when I needed advice, always supportive and never severe with my faults.

I thank Thierry Coquand and Herman Geuvers for doing me the honour of reviewing this work.

I thank Benjamin Werner, Carlos Simpson and Jean-Christophe Filliâtre for the interest they showed in this work by accepting to be part of my jury.

One of the reasons I started this work was my master's internship that I done in the Marelle team. Everybody here was so nice that I just couldn't imagine a better environment for my thesis.

Merci Laurence de toujours prendre soin de nous, les doctorants.

Merci Laurent et Loïc pour faire en sorte que les repas de midi soient jamais ennuyeux.

Merci Sidi, c'était vraiment cool de t'avoir comme collegue de bureau.

Thank you Tom and Jorge for hosting my coffee breaks and bearing my company even outside work.

Merci Nicolas pour tout ce que j'ai appris en travaillant avec toi.

Thanks Katya, for all your advice and for bringing the chocolate tradition in our office.

Merci Nathalie, Anne, Benjamin, Minh, Sylvain, Michaël, Maxime et Guillaume pour tous les bons moments.

Thank you Cody for always encouraging me in my work.

Merci Georges, Assia et tous les membres Mathematical Components. J'ai beaucoup appris pendant nos réunions.

Mulţumesc tuturor românilor de la INRIA, şi nu numai, pentru pausele de cafea şi ieşirile din timpul liber.

Iţi mulţumesc Alice pentru sprijinul şi prietenia ta.

Thanks to all my friends near and far.

Mulţumesc din tot sufletul familiei mele pentru ca aţi fost mereu alături de mine. Mulţumesc Pierre.

Abstract

My research is about formalizing mathematics in the proof assistant Coq with the purpose of verifying numerical methods. In particular I focus on formalizing concepts involved in solving systems of equations. All the formalizations are carried out inside the Coq proof assistant and the SSReflect extension of Coq. The formalizations involve both linear and non-linear systems of equations.

For linear systems I analyze the case where the associated matrix has interval coefficients. This corresponds to the idea that the value of a coefficient is not known exactly, but rather it is known to lie in an interval of possible values. For solving such systems an important issue is whether the associated matrix is regular (has non-null determinant) for whatever values we choose for the coefficients in their corresponding interval. The results of this work are reusable libraries dealing with intervals, interval matrices and basic operations on them as well as a collection of formally verified criteria for regularity of interval matrices.

In the case of non-linear systems I formalized properties of Newton's method inside the proof assistant Coq. Newton's method is a numerical method widely used for approximating solutions of equations or systems of equations. The goal was to formalize Kantorovitch's theorem which gives the convergence of Newton's method to a solution, the speed of the convergence and the local stability of the method. For the formalization of Kantorovitch's theorem I used the implementation of real numbers provided in the Standard Library of Coq. For the case of one equation the results in the library were sufficient. For the case of a system of equations the proof of Kantorovitch's theorem requires multivariate analysis and Coq does not provide such a library. I formalized concepts of multivariate analysis by using not only the standard library of Coq but also the SSReflect extension of Coq and the libraries developed on it. The formalization covers: real vectors and matrices, norm on vectors and matrices, convergences of sequences

of vectors, relations between a matrix norm and its determinant, limit and continuity for functions of several variables, properties of the partial derivatives of such functions.

In a joint work with Nicolas Julien, we use the formalization of Kantorovitch's theorem done on the 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. This work has two main parts. First, 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. Second, 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. This proof is designed on the previous proofs formalized for Kantorovitch's theorem but it is an original result.

Résumé

Ma recherche s'articule autour de la formalisation de mathématiques dans l'assistant à la preuve Coq dans le but de vérifier des méthodes numériques. Plus précisément, je me concentre sur la formalisation de concepts qui apparaissent dans la résolution de systèmes d'équations. Toutes les formalisations sont faites dans l'assistant à la preuve Coq et l'extension SSReflect de Coq. Ces formalisations concernent à la fois des systèmes linéaires et non-linéaires d'équations.

Pour les systèmes linéaires, j'analyse le cas où la matrice associée a des coefficients intervalles. Ceci correspond à l'idée que la valeur d'un coefficient n'est pas connue exactement, mais plutôt qu'elle se trouve dans un intervalle de valeurs possibles. Pour résoudre de tels systèmes, un problème important qui se pose est de savoir si la matrice associée est régulière (a un déterminant non nul) pour tous les choix possibles des valeurs des coefficients dans leurs intervalles respectifs. Mon travail sur cette problématique a produit des bibliothèques réutilisables sur les intervalles et les matrices d'intervalles avec les opérations de base, ainsi qu'une vérification formelle d'une collection de critères de régularité pour les matrices d'intervalles.

Dans le cas des systèmes non-linéaires, j'ai formalisé les propriétés de la méthode de Newton dans l'assistant à la preuve Coq. La méthode de Newton est une méthode numérique couramment utilisée pour approcher les solutions d'une équation ou d'un système d'équations. Le but a été de formaliser le théorème de Kantorovitch qui montre la convergence de la méthode Newton vers une solution, l'unicité de la solution dans un voisinage, la vitesse de la convergence et la stabilité locale de la méthode. Pour la formalisation du théorème de Kantorovitch j'ai utilisé l'implémentation des nombres réels fournie dans la bibliothèque standard

de Coq. Dans le cas d'une unique équation, les résultats déjà formalisés dans cette bibliothèque ont suffi. Dans le cas d'un système d'équations, la preuve du théorème de Kantorovitch utilise des concepts d'analyse multivariée et Coq ne fournit pas de telle bibliothèque. J'ai donc formalisé des concepts d'analyse multivariée en utilisant la biblithèque standard de Coq et les bibliothèques developpées sur l'extension SSReflect de Coq. Ma formalisation comprend : vecteurs et matrices réels, normes sur les vecteurs et les matrices, convergence d'une suite de vecteurs, relations entre norme et déterminant d'une matrice, limite et continuité d'une fonction à plusieurs variables, propriétés des derivées partielles d'une telle fonction.

Dans un travail commun avec Nicolas Julien, nous avons utilisé ma formalisation du théorème de Kantorovitch faite sur les réels de la bibliothèque standard de Coq pour valider des calculs avec la méthode de Newton menés dans le cadre d'une bibliothèque d'arithmétique réelle exacte basée sur des suites co-inductives. Ce travail a deux parties importantes. D'abord, en se basant sur la methode de Newton, on implémente et on montre la correction d'un algorithme sur des suites qui calcule de manière paresseuse la racine d'une fonction réelle. Ensuite, on montre qu'arrondir à chaque étape (de la méthode de Newton) préserve la convergence, avec une corrélation bien determinée entre la précision des données d'entrée et celle du résultat. Cette preuve est inspirée par les preuves précédemment formalisées pour le théorème de Kantorovitch, mais elle constitue un résultat original.

Contents

Contents **vi**

1 Context **1**

1.1 Numerical methods and proof assistants 1

1.2 How we work in a proof assistant 4

1.3 Soft Introduction to Concepts to Be Formalized 12

1.3.1 Newton's Method 12

1.3.2 Newton's method with rounding 17

1.3.3 Exact real arithmetic 17

1.3.4 Interval analysis 18

1.4 Formalizing a Numerical Method 20

2 Formalized Mathematical Theories for Numerical Methods **21**

2.1 Existing formalizations 21

2.1.1 Real analysis 21

2.1.2 Matrices 23

2.2 Mixing COQ and SSReflect 24

2.3 Real Matrices 30

2.4 Multivariate Analysis 36

2.4.1 Vectors in $\mathbb{R}^p$ 36

2.4.2 Metric spaces: convergence, limit, continuity 40

2.4.3 Derivatives 44

2.4.4 Related formalizations 48

2.5 Interval Analysis 48

2.5.1 Description 49

2.5.2 Rounded interval arithmetic 50

2.5.3 Implementation 51

2.5.4 Interval matrices 56

2.5.5 Related formalizations 58
2.6 Conclusion 59

3 Solving Equations and Systems of Equations with Newton's Method 61
3.1 Proofs for properties of Newton's method 62
3.1.1 Statements 62
3.1.2 Formalization issues 66
3.1.3 Moving to several dimensions 67
3.2 Newton's method with rounding 69
3.3 Newton's method and exact real computations 74
3.3.1 A Coq library for exact real arithmetic 75
3.3.2 Correctness of Newton's method 78
3.3.3 An algorithm for exact computation of roots 79
3.3.4 Applications to the square root 81
3.4 Related work 82
3.5 Conclusion and future work 82

4 Regularity of Interval Matrices 85
4.1 The solution set of a system of linear interval equations 87
4.2 Basic regularity criteria 88
4.3 Efficient regularity criteria 90
4.4 Conclusion and future work 93

5 Conclusions and Perspectives 95

A Mathematical Proofs for Newton's Method 101

B COQ Statements 115
B.1 Newton's method in one dimension 115
B.2 Newton's method with rounding 117
B.3 Newton's method in several dimensions 118
B.4 Criteria for regularity of interval matrices 119

Bibliography 129

Chapter 1

Context

This work is an attempt to bring closer numerical methods and formal methods by doing a formal study on elements of algorithms for numerically solving systems of equations.

1.1 Numerical methods and proof assistants

What is a numerical method? We call **numerical method** a method that uses numerical approximations in order to provide the solution of a problem. Such numerical methods are used when there are no known algorithms for providing the exact solution or when the exact algorithms are very costly. For example, when we talk about the value of sin(1), we cannot give it exactly, so we use Taylor series approximations in order to give the result at a certain precision:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$
$$\sin(1) \approx 1 - \frac{1^3}{3!} + \frac{1^5}{5!} - \frac{1^7}{7!} = 0.861904762$$

But we also use numerical approximations when we are unable to represent some quantity in an exact manner. This is the case when we deal with real numbers on computers, where they are usually approximated by floating point numbers. For irrational numbers like π or $\sqrt{2}$ we do not expect to represent them exactly on computer, but there are examples of rational numbers like $\frac{1}{3}$ or 0.1 that cannot be represented exactly as binary floating point numbers only because of the limitations of the floating point format.

So, when computing the solution of some problem, we have to take into account, in general, two kinds of errors: method errors and rounding errors. Method errors are intrinsic to numerical methods, as the result will usually be an approximation of the “true” result. Rounding errors appear because real numbers cannot be represented exactly on machines, so, again, we only have an approximation of the “true” result. In order to develop reliable software, it is important to deal with both types of errors and get an estimate of the total error that has been committed during the computation of a result.

Numerical methods are nowadays present in all types of software. We deal with them whenever we need to compute the value of a function, to solve equations or systems of equations, to integrate a function, to solve differential equations, optimization problems and many other types of problems.

What is a proof assistant? We call **proof assistant** or **interactive theorem prover** a piece of software that allows the user to encode concepts corresponding to a certain problem (for example, mathematical theories), to state properties of these concepts and to verify that the properties hold by building a proof. The construction of the proof is assisted by the software, but directed by the user. If we think of a mathematical theorem, the user of the proof assistant will give the reasoning steps involved in the proof, and the assistant will verify that these steps are correct. Proof assistant should not be mistaken for automated theorem provers, whose purpose is to find the proofs of the statements by themselves. However, in some cases it is possible to automate (part of) the proofs done in a proof assistant.

The user interacts with the proof assistant by giving a series of commands, that correspond to the logical reasoning steps made in the proof of a statement. These commands consist in applying some previously proved theorem, reasoning by induction, doing a case analysis, reasoning by contradiction, and so on.

We will use the term formalization when we talk about encoding something in a proof assistant, and the term formal proof when we talk about a proof done inside a proof assistant.

Since the purpose of proof assistants is to make sure all reasoning steps are logically correct, a statement proved in a proof assistant will have a higher guarantee of correctness than a paper proof of the same statement.

The motivation for doing formal proofs of properties is usually the need for a high guarantee of correctness in safety critical software, like the software used in avionics, robotics, medicine, cryptography, and so on. In order to make such formal proofs, we need to have formalizations for the notions involved. For example, for formal proofs about a numerical method we need to use results from real analysis, for formal proofs about a cryptographic protocol we need notions from group theory or probability theory, for formal proofs about a programming language we need to describe its semantics. So, it is essential to have a formalization of theories, whether purely mathematical or mixed with computer science notions, in our proof assistants.

Formalization of mathematics is interesting in itself and not only for its application to formal proofs of programs. The formal verification of mathematical theorems has an interest when we think of "big and complex" theorems. By this we mean theorems that have proofs of hundreds of pages or theorem that have proofs that contain computations made by machines, as such computations might be inaccurate and thus the proof might not be valid. In these cases, a machine checked proof can remove all doubts in the proof of the theorem. At present, there is a complete formal proof for the Four Color Theorem [35] whose initial mathematical proof relied heavily on computations done on machines. There are also two ongoing projects, one for the formal proof of Kepler's conjecture [41] because the initial proof relies on computations and one for the formal proof for the classification of finite groups [4] because the initial proof is very long and combines several mathematical theories.

Some of the more widely used proof assistans are ACL2 [55], Agda [1], Coq [8; 20], HOL [37], HOL Light [38], Isabelle[70], Matita [5], Mizar [6], Nuprl [40], PVS [74]. This list is not exhaustive. The interested reader can consult some surveys on existing proof assistants like [7] or [76].

Why these formalizations? Numerical methods are widely used in safety critical software and this justifies their formal study. The problems we will discuss in this manuscript come from algorithms used in robotics: solving systems of equations in order to determine the next safe position of the robot.

We can of course mention the classical examples of software and hardware bugs that were very costly, like the explosion of the first Ariane 5 flight [57], the problem on an oil platform [19], the Pentium division errors [18], the system failure of the USS Yorktown [47], and so on. Formal verification of critical software and hardware can be a solution. There are several formal studies that have been conducted successfully on floating point verification [13; 46], interval analysis [26; 60], algorithms used in computation [10; 44] or numerical methods [15; 59]. These developments show that applying formal methods to computations and numerical methods can be a success.

1.2 How we work in a proof assistant

Our formalizations are carried out in the proof assistant Coq. In what follows, we want to illustrate a few features of our proof assistant.

In Coq we program and do proofs by using the same language. In this language all the objects we manipulate have a type. These types respect a certain discipline that says how we build new types and how we give types to objects. The types together with their discipline form the type system. Coq's type system is called the Calculus of Inductive Constructions and the interested reader can find many details in [21; 69]. The theoretical properties of Coq's type system ensure that the proofs done in Coq are valid.

In Coq's type system, the type of an object also has a type. The types of types are called sorts. In Coq there are two sorts: the sort Type and the sort Prop. The sort Type is appropriate when we define a type of data, while the sort Prop is used for defining a logical proposition. The objects of a certain type T are called terms of type T.

We take some examples. We start by defining the type nat of natural numbers as Peano integers:

```
Inductive nat: Type := O : nat | S : nat → nat.
```

We say that a natural number is either O or the successor S of another natural number. We use the keyword Inductive to give this definition. An **inductive definition** is a definition by cases that gives all the ways a term can be constructed, by potentially using previously constructed terms of the same type (like in the case of the successor S for nat). Each way of constructing a term is called a **constructor**. For nat we have two constructors: O and S.

An inductive type like nat will contain all the terms generated by applying its constructors a finite number of times. In particular, nat will contain as expected O, S O, S (S O), S (S (S O)) and so on. Just to clarify the jargon, in our example, nat is a type while O, S O, S (S O) are terms of type nat.

We can use other constructs to define new types in Coq like Definition and Structure. To illustrate the use of Structure we define the type posRat of positive rational numbers. We choose to formalize a positive rational number as a pair of natural numbers: num corresponding to the numerator and den corresponding to the denominator, together with a proof that the denominator is not zero. This formalization is for didactic purposes only: it is not convenient to work with rationals in this way since $\frac{1}{2}$ and $\frac{2}{4}$ will not be equal. Here is our toy example:

```
Structure posRat: Type := PosRat{ num: nat; den: nat; den_pos: den <> O }.
```

Note that the type of the field den_pos of the structure is den <> O and thus depends on the term den of type nat. This is possible because the type system of Coq allows dependent types. We have dependent types when a type can depend on a term of some other type. Dependent types give a greater expressive power to the type system, but not all type systems support dependent types.

Now we can write functions on our data types. Coq allows us to write recursive functions. We can write a recursive function if at least one of its arguments, called the principal argument, is a term of an inductive type. The recursive calls will be made on this principal argument, but always on a structurally smaller term (the "bigger" term is some constructor applied to the "smaller" term). This kind of recursion is called structural recursion and it ensures that the function terminates: a term of an inductive type is obtained by applying its constructors a finite number of times and each recursive call "removes" one such application of constructor, so inevitably this process will terminate in a finite number of steps. We give the example of addition of two natural numbers.

```
Fixpoint plus (n m: nat) {struct n} : nat := match n with
  | O ⇒ m | S n' ⇒ S (plus n' m) end.
```

The principal argument in this definition is n and this is indicated by {struct n}. The function is then defined by analyzing the argument n. If it is O, we just return m, without making a recursive call. If n is S n' (the successor of some nat n'), then we make a recursive call to the function plus on n' and m. We notice that the recursive call is indeed made on n' which is a structurally smaller term than n.

It is not very agreeable to always write plus for addition of two natural numbers. We would like to use the infix notation + instead. Coq allows us to do this, by using the following command:

```
Notation "n + m" := (plus n m).
```

The notation + is only used in the interaction with the human user. For all internal purposes, the proof assistant uses the actual name of the function plus.

Until this point we presented Coq as a programming language in which we can define data types and write functions on them. Coq also offers the possibility to describe properties of our data types and functions and to prove that they hold. To state properties we use the Coq keyword Lemma or Theorem.

We take an example: we want to state and prove that addition on natural numbers is commutative. First, we assume that we have already proved the following two lemmas:

```
Lemma plus_n_O : forall n: nat, n = n + 0.
Lemma plus_n_Sm : forall n m: nat, S (n + m) = n + S m
```

This will help us illustrate how existing proofs are used for providing new ones.

Next, we state the result of interest to us:

```
Lemma plus_comm : forall m n: nat, n + m = m + n.
```

We now need to provide a proof for this lemma. Since we are in an interactive theorem prover, we need to tell the system how this lemma is proved and it will check that our reasoning steps are correct. Once we typed in the statement of our lemma the system responds by giving the current context and the current goal. The context is the information we have at a certain stage in the proof and the goal is what we need to prove. The context is printed above the horizontal line and the goal under it. In our case the context is empty.

```
1 subgoal
___________________________________(1/1)
forall m n : nat, n + m = m + n
```

We are now in proof mode. To clearly delimit the proof we can (optionally) use the keyword

```
Proof.
```

To start the proof, on paper we would say " Let m, n be two given natural numbers." To do this in Coq we use the tactic

```
intros m n.
```

The context and the goal become:

```
1 subgoal
m : nat
n : nat
___________________________________(1/1)
n + m = m + n
```

In paper mathematics we would at this point be tempted to reason by induction on n. Coq has a tactic for doing exactly this, so we can tell the system

```
induction n.
```

As expected in an induction on natural numbers, we have two things to prove: first, that the property holds when n = 0 and second that if the property holds for an arbitrary rank n then it holds for rank S n, the successor of n. Coq generates two goals corresponding to these two cases.

```
2 subgoals
m :  nat
___________________________________(1/2)
0 + m = m + 0

___________________________________(2/2)
S n + m = m + S n
```

However, we cannot see the context for the second subgoal, and the proof commands that the user edits now are for the first subgoal. But, before continuing with the proof, we explain for the curious reader how does Coq know how to do induction on natural numbers, just by looking at the definition of the type nat.

We use induction when we want to show that a property P holds for all terms of an inductive type. An inductive principle roughly says that to show P for all terms we just need to show P for all the ways in which we can build a term (for every constructor in the definition). This means:

- in the case of a simple constructor (a constructor which does not use terms of the same type in order to build a term), we just need to show the properly holds for the term produced by the constructor. This is the case for the constructor O of the type nat. We need to show P O.

- in the case of a constructor that uses some terms of the same type in order to produce a new term, we have to show that if the property holds for the initial terms then it will hold for the term obtain from the constructor. This is the case for the constructor S of the type nat. We need to show forall n, P n→P (S n).

The induction principle for nat is called nat_ind, it is generated automatically by Coq and it looks like this:

```
nat_ind :
  forall P : nat → Prop, (* the property P *)
    P 0 → (* the case of constructor O *)
    (forall n : nat, P n → P (S n)) → (* the case of constructor S *)
  forall n : nat, P n (* conclusion *)
```

To see the whole capability of Coq's inductive definitions and how induction principles look like in other cases the reader should consult [8].

Getting back to our proof, our first goal is:

```
m :  nat
___________________________________(1/2)
0 + m = m + 0
```

From the definition of the plus function we know that if the first argument is O, then the sum is just the second argument. We tell the proof assistant to make this change:

change (O + m) with m.

```
2 subgoals
m :  nat
___________________________________(1/2)
m = m + 0

___________________________________(2/2)
S n + m = m + S n
```

The tactic change works in this situation because the two expressions are *convertible*. This means both expressions can be changed according to some precisely defined rules (called reduction rules) into a same expression. These rules consist of:

- applying a function to its arguments (β-reduction);
- replacing a definition by its body (δ-reduction);
- replacing a local definition by its body (ζ-reduction);
- replacing a case analysis on a given term of an inductive type by the appropriate branch (ι-reduction).

In our case, the system made three reductions: a δ-reduction to expand the definition of plus, a β-reduction to apply the function to the arguments O and m and a ι-reduction to select the O branch in the definition.

We notice that our current goal m $=$ m $+$ 0 is just an instance of the lemma plus_n_O that we allowed ourselves to use. We apply this lemma:

apply plus_n_O.

This completely solves our first subgoal, so we are left with just one goal, corresponding to the second step of our induction.

```
1 subgoal
m :  nat
n :  nat
IHn :  n + m = m + n
____________________________________________(1/1)
S n + m = m + S n
```

We notice we can put our goal in a more convenient shape by using the equality given by the lemma plus_n_Sm.

rewrite ← plus_n_Sm.

The arrow ← in the tactic means that the system looks for a term in the goal that matches the right-hand side of the equality in the lemma plus_n_Sm and replaces it with the left-hand side of the same equality.

```
1 subgoal
m :  nat
n :  nat
IHn :  n + m = m + n
____________________________________________(1/1)
S n + m = S (m + n)
```

Again, according to the definition of plus the term S n + m reduces to S (n + m).

change (S n + m) with (S (n + m)).

```
1 subgoal
m :  nat
n :  nat
IHn :  n + m = m + n
____________________________________________(1/1)
S (n + m) = S (m + n)
```

We can now use the induction hypothesis IHn.

rewrite IHn.

```
1 subgoal
m :  nat
n :  nat
IHn :  n + m = m + n
____________________________________(1/1)
S (m + n) = S (m + n)
```

Since we did not put an arrow, the system looks by default for terms that match the left-hand side of the equality IHn and replaces them by the term on the right-hand side. Our goal now becomes trivial, as we just have to show that a term is equal to itself.

trivial.

The tactic trivial tries a series of very simple steps, like checking if the goal is a syntactic equality (like in our case) or if the goal is identical to some hypothesis. Now we have proved all our goals and the system responds

```
Proof completed.
```

The user says

Qed.

and the systems says.

```
plus_comm is defined
```

This indicates that the plus_comm lemma can now be used to prove other properties.

This proof was done with didactic purposes, to illustrate how the interaction between the user and the proof environment takes place. The same proof can be done in one line:

Proof.
intros; omega.
Qed.

The tactic omega is an automatic procedure integrated in the Coq system that deals with statements on Peano natural numbers. For more details on the kinds of problems that Coq can solve automatically and on how the user can define new automatic procedures we refer the reader again to [8; 20].

Another interesting thing about the proofs of logical propositions is that they are done by default in intuitionistic logic. This means in Coq the law of excluded middle is not valid by default. The Coq system, however, allows the user to add new axioms to the logic, whether they are already provided by the system in special files or directly given by the user. This is interesting as it gives more power to the systems and allows the user to prove more results. But this can also be dangerous as carelessly adding axioms can lead to an inconsistent system. We will discuss some of these axioms later on in the manuscript.

In the beginning of the section we claimed that in Coq we write programs and proofs in the same typed language. But so far, we presented separately Coq's programming and proving capabilities. We saw at the beginning of the section that Prop is a sort, which means the elements of Prop are themselves types. But the elements of Prop are the logical propositions. So a logical proposition is a type. Giving a proof of the logical proposition is simply building a term of the type that is the logical proposition. Let's look again at our example:

```
forall m n, n + m = m + n
```

is a logical proposition and thus it has sort Prop. Proving the lemma plus_comm consisted in defining a term of type forall m n, n + m = m + n. At the end of the proof Coq said `plus_comm is defined`. This means we defined plus_comm as a term of type forall m n, n + m = m + n. This is what is called the Curry-Howard isomorphism which says that logical propositions are types and the proofs of logical propositions are terms of the corresponding types.

Until this point we made a clear distinction between types and sorts, in order to allow the novice reader to understand more easily what is going on. But sorts are just types (of other types), and in the type system there are no distinctions between them, so we will not make any distinctions in the rest of the manuscript.

In this section we tried to provide an introduction to some of Coq's main features. More details of these features and other features will be presented later on in the manuscript, usually when they play a role in the development we are describing. These technical features are described in special sections, entitled **Technical details**. We chose to separate the Coq specific details of the implementation, so a reader not interested in them could easily skip them. To the same end we only refer to technical sections in other technical sections. This section can be seen as the first technical section of the manuscript.

1.3 Soft Introduction to Concepts to Be Formalized

1.3.1 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 an initial approximation 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:

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

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 1.1.

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:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \tag{1.1}$$

From equation 1.1 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

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

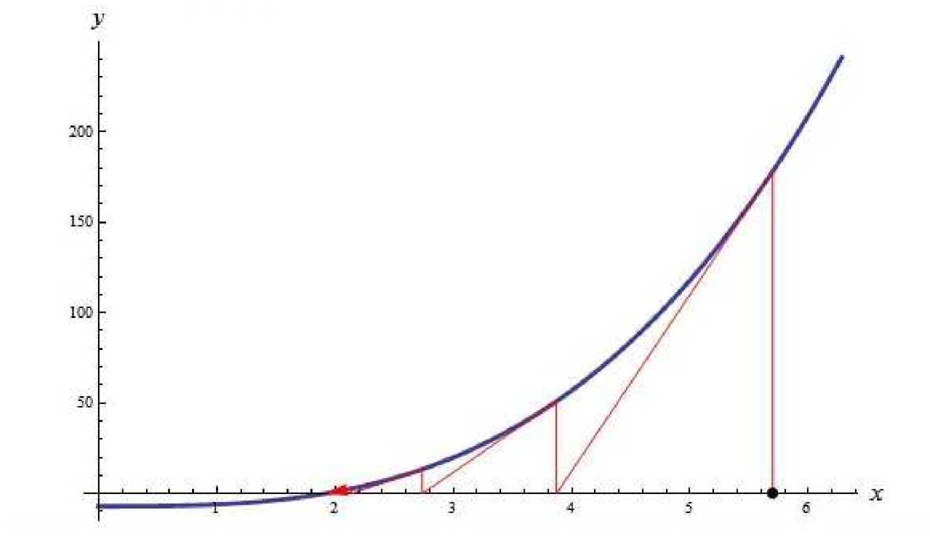

Figure 1.1: Newton's method for $f(x) = x^3 - 7$

this is equivalent to

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

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:

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

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

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

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

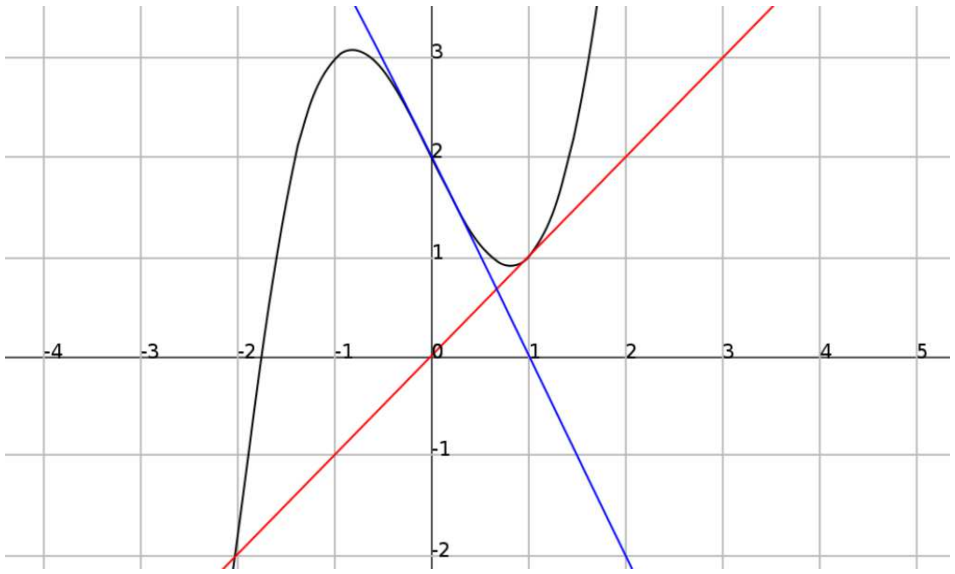

Figure 1.2: Newton's method oscillates for $f(x) = x^3 - 2x + 2$ and $x_0 = 0$

We get an oscillating sequence of 0 and 1 without converging to the root, as illustrated in Figure 1.2.

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 detail later on.

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

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

For $p = 2$ we have

$$f(X) = (f_1(X), f_2(X)),\ X = (x_1, x_2)$$

and determining a root means finding a solution for the following system of equations:

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

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

$$J_f(X) = J_f(x_1, x_2) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1}(x_1, x_2) & \dfrac{\partial f_1}{\partial x_2}(x_1, x_2) \\ \\ \dfrac{\partial f_2}{\partial x_1}(x_1, x_2) & \dfrac{\partial f_2}{\partial x_2}(x_1, x_2) \end{pmatrix}$$

Then Newton's method becomes:

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

It is straightforward to see how this works in dimension p. We have

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

and the system of equations

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

The Jacobian matrix is given by

$$J_f(X) = J_f(x_1, x_2, \ldots, x_p) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1}(X) & \dfrac{\partial f_1}{\partial x_2}(X) & \ldots & \dfrac{\partial f_1}{\partial x_p}(X) \\ \\ \dfrac{\partial f_2}{\partial x_1}(X) & \dfrac{\partial f_2}{\partial x_2}(X) & \ldots & \dfrac{\partial f_2}{\partial x_p}(X) \\ \\ \ldots & \ldots & \ldots & \ldots \\ \\ \dfrac{\partial f_p}{\partial x_1}(X) & \dfrac{\partial f_p}{\partial x_2}(X) & \ldots & \dfrac{\partial f_p}{\partial x_p}(X) \end{pmatrix} \tag{1.2}$$

Newton's method is the same as in the two dimensional case.

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

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, Sténine, Ostrowski, Kantorovitch and others are concerned with establishing sufficient conditions for the convergence of Newton's method. According to [29] Kantorovitch gives the following sufficient conditions for the convergence of Newton's method.

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 $J_f(X) = [\frac{\partial f_i(X)}{\partial x_j}]$ has an inverse for $X = X_0$, $\Gamma_0 = J_f^{-1}(X_0)$ with $\|\Gamma_0\| \leq A_0$;*
2. *$\|\Gamma_0 f(X_0)\| \leq B_0 \leq \frac{\varepsilon}{2}$;*
3. *$\sum\limits_{k=1}^{p} |\frac{\partial^2 f_i(X)}{\partial x_j \partial x_k}| \leq C$ 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

$$X_{n+1} = X_n - J_f^{-1}(X_n) f(X_n) \tag{1.3}$$

$(n = 1, 2, ...)$ converges and the limit vector $X^* = \lim\limits_{n\to\infty} X_n$ is a solution of the initial system, so that $\|X^* - X_0\| \leq 2B_0 \leq \varepsilon$.

It is not important for the reader to understand all the details right away. We give further explanations in chapter 3 where we describe the formalization of this theorem inside the proof assistant Coq.

By theorem 1 we have the precise conditions for the function f and for the initial point X_0 under which Newton's method converges to the root of the function. We can show that in a certain domaine this root is unique. We can also precisely establish at what speed the method converges. This means that for each n we can determine a Δ_n such that:

$$\|X^* - X_n\| \leq \Delta_n$$

So, at each iteration we know how far we are from the root X^* we are approximating.

Newton's method is also locally stable. This means that there is a neighborhood of X_0 in which we can choose an initial point and the method will still converge.

1.3.2 Newton's method with rounding

In our description of Newton's method up till here we assumed that the computations are made with "true" real numbers. By this we mean that no rounding is performed during this 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. We detail these conditions and we formally prove the convergence of the altered method in section 3.2 of chapter 3. This result will be useful when proving formal correctness of computations with Newton's method in a library of exact real arithmetic.

1.3.3 Exact real arithmetic

When talking about exact real arithmetic we usually mean computation in arbitrary precision. One way to implement such an arithmetic is to represent real numbers as a potentially infinite list of digits where the digits can be computed one at a time. The operations on real numbers are implemented as lazy algorithms that work in the following manner: they produce a digit of the real number we want to compute and they gather enough information to be able to produce the next digit, if required. This way we can get the result at the precision we desire. Such a library is implemented in the proof assistant Coq and described in [52].

Newton's method seems particularly adequate in such a framework. At each iteration we get an approximation of the root at a given precision. Also we have the information necessary to increase this precision by doing new iterations from where we left off. In section 3.3 of chapter 3 we present the details of the library on exact real arithmetic and the way Newton's method is adapted for this setting. We also show that using rounding at a certain number of digits improves the performance of the computations. As an application, we implement an algorithm for computing the square root of a number by using Newton's method. The square root of a positive real number a is the root of the function

$$f_{sqrt}(x) = x^2 - a$$

The corresponding Newton's sequence is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

Treating Newton's method in the context of exact real number computations is joint work with Nicolas Julien.

1.3.4 Interval analysis

An important part of working with numerical methods is dealing with the errors introduced by rounding or by the method itself. So we mostly manipulate approximations of some ideal value at a certain precision. Put differently, the ideal value is in an interval of possible values given by the approximation and the precision. The mathematics branch corresponding to this description is interval analysis and it constitutes a tool for dealing with errors in a uniform and robust way.

Let's take a simple example. If we want to multiply $-\pi$ and $\sqrt{2}$, we usually say $-\pi$ is approximately -3.14, $\sqrt{2}$ is approximately 1.41 and we give an approximate result:

$$-\pi * \sqrt{2} \approx -3.14 * 1.41 = -4.4274$$

In interval analysis, instead of approximating, we know for sure that $-\pi \in [-3.15, -3.14]$ and $\sqrt{2} \in [1.41, 1.42]$. We also know that by multiplying a value in $[-3.15, -3.14]$ and a value in $[1.41, 1.42]$ we get a value in $[-4.473, -4.4274]$.

$$[-3.15, -3.14] * [1.41, 1.42] = [-4.473, -4.4274]$$

Like with Newton's method before, we are interested in solving systems of equations, but this time in the context of interval analysis. In particular, we are interested in systems of linear equations with interval coefficients. Here is an example of such a system in the case of two equations and two unknowns:

$$\begin{cases} [1,2]x_1 & + & [2,4]x_2 & = [-1,1] \\ [2,4]x_1 & + & [1,2]x_2 & = [1,2] \end{cases} \tag{1.4}$$

Solving such a system means determining all pairs $(x_1, x_2) \in \mathbb{R}^2$ that satisfy the equations for some choice of coefficients in their corresponding intervals. The set of all these pairs forms the solution set of the system of linear interval equations 1.4.

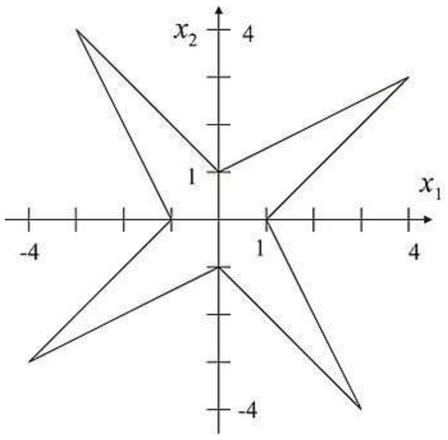

Figure 1.3: Solution set for system 1.4

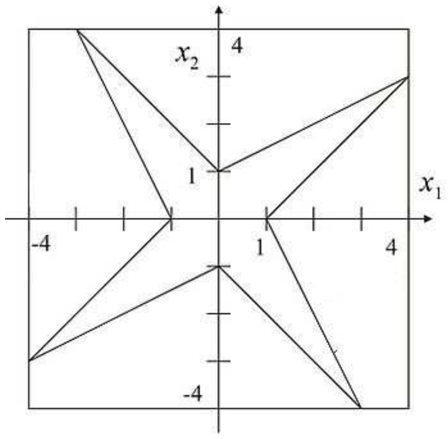

Figure 1.4: Bounds for the solution set for system 1.4

There are two steps in solving a systems of linear interval equations. The first step is to analyze the interval matrix associated to the system, in our example, the matrix:

$$\begin{pmatrix} [1,2] & [2,4] \\ [2,4] & [1,2] \end{pmatrix}$$

We have to establish if this interval matrix is regular, that is, if all real matrices that we can build by choosing values in the corresponding intervals have non-null determinant.

The second step consists in determining the bounds of the solution set. This step can be performed only if the interval matrix is indeed regular. We do not try to determine the solution set exactly because in general it has complicated shapes. For example the solution set for system 1.4 is represented in Figure 1.3 and what we want is the box represented in Figure 1.4 that bounds the solution set.

The formalization conducted on the topic only treats step one: checking regularity of an interval matrix. This work is detailed in chapter 4.

1.4 Formalizing a Numerical Method

Now that we saw what kind of concepts we formalized, let's explain what does a formalization process entail for a numerical method. We want to express the properties of our method. We saw in the case of Newton's method that there are theorems like Kantorovitch's that describe the method. In order to formalize such theorems we need to be capable to handle in our proof assistant all the concepts that appear in the theorem. Proof assistants come equipped with certain libraries on mathematical theories, but it is sometimes the case that not all concepts we need are formalized inside the proof assistant. For example, in the proof assistant Coq in order to treat Newton's method we have support for real analysis, but we do not have support for multivariate analysis. So, before starting the proof of Kantorovitch's theorem we need to formalize all multivariate analysis concepts needed in the proof.

Thus, the first step in a formalization is to give all the background theories. For our work we needed multivariate analysis for Newton's method, some results on real matrices both for Newton's method and for interval analysis and a basic description of intervals and interval arithmetic. All these are theories of general interest and are organized in reusable libraries. They are described in chapter 2 of this document. We also provide a brief survey on how these theories and related results are treated in other proof assistants.

Once all the basic theories are in place, we can proceed with the formalization of the desired theorem which is the second step of the formalization process.

Since we are talking about numerical methods we want to be able to describe the computation performed with the method. There is sometimes a big difference between the method described in the literature and the method implemented in practice, so these optimizations and adaptations need to be taken in to account and formally verified also. In the case of Newton's method, we need to handle rounding and provide a proof for the properties of the new method. Treating the optimizations and verifying computations performed with the method can be consider steps three and four of the formalization process. For Newton's method we treated all four steps. For solving systems of linear interval equations we treated step one, by providing a formalizations for basic interval arithmetic and partially step two by formally verifying criteria of regularity for interval matrices.

Chapter 2

Formalized Mathematical Theories for Numerical Methods

Proof assistants are becoming more and more mature and equipped with libraries on mathematical theories that ease the verification of numerical algorithms. Some of the main theories we are concerned with are real analysis and linear algebra. In what follows we present the formalizations available on these topics. However, not all we need is available in existing libraries. In particular we need formalizations on specific concepts on real matrices as well as on multivariate analysis and interval analysis.

2.1 Existing formalizations

2.1.1 Real analysis

Concepts on real analysis are currently treated in several proof assistants: HOL Light, PVS, ACL2, Isabelle, Coq.

An important issue is representing the real numbers. There are several choices: real numbers can be defined axiomatically as a complete ordered field satisfying the least upper bound principle or real numbers can be constructed and the corresponding properties can be proved on the model. There are several constructions for the reals, the most famous being the Dedekind model, based on the notion of cut, the Cantor model, which uses Cauchy sequences of rational numbers and the Weierstrass model which uses decimal fractions.

Another important issue is to see how real analysis concepts can be efficiently formalized based on a given representation. There are two main approaches. One approach is to follow classical analysis where concepts are expressed using the usual $\varepsilon - \delta$ definitions. As an example, here is the definition of limit for a function f at a point

a

$$\lim_{x \to a} f(x) = l \Leftrightarrow \forall \epsilon \in \mathbb{R}, \exists \delta \in \mathbb{R}, x \neq a, |x - a| < \delta \to |f(x) - l| < \epsilon$$

The other approach is to use non-standard analysis as first introduced by Robinson [72]. In non-standard analysis we deal with the system of hyperreal numbers or non-standard real numbers $*\mathbb{R}$, which is an extension of the standard real numbers that treats in a systematic way infinite and infinitesimal quantities. An infinite numbers is a number larger than any number of the form $(1 + 1 + \ldots + 1)$. The inverse of an infinite is an infinitesimal. An infinitesimal is a nonzero quantity, but smaller in absolute value than any positive standard real. Two real numbers are infinitely close if their difference is infinitesimal, and we note the infinitely close relation by $\approx$. Here is the example for the non-standard definition of limit for a function f at a point a

$$\lim_{x \to a} f(x) = l \Leftrightarrow \forall x \in *\mathbb{R}, x \approx a \wedge x \neq a \to *f(x) \approx l$$

where $*f$ is the extension of the standard real function f to the hyperreals.

When using the infinitely close relation $\approx$, the manipulation of objects like derivatives or limits of functions becomes algebraic and therefore theorems are easier to automate. This is the big argument in favor of using non-standard analysis inside proof assistants.

In what follows we present what approaches have been used for the formalization of real analysis in proof assistants and what kind of results these libraries cover.

For HOL Light, the main work can be found in [43]. The real numbers are constructed using an adaptation of Cantor's method. Real analysis is dealt with in a classical way. One interesting fact is the way limits are implemented. As opposed to other systems, which treat independently the limit of a sequence and that of a function, HOL describes them using one concept by relying on the theory of nets. Results are achieved around continuity, differentiation, integrability and transcendental functions. We mention that a quantifier elimination procedure has also been implemented for this theory. Some applications of the developed theory involve verifications for floating point algorithms [42; 44; 46; 51].

In Isabelle one can actually find most of the concepts formalized in both classical and non-standard analysis. What is interesting is the proof of equivalence between the concepts in the two approaches[30].

The ACL2 proof assistant has a non-standard approach to real analysis. [31] offers an introduction to non-standard analysis techniques and shows how they can be used to reason mechanically about concepts like transcendental functions. The formalization of continuity, differentiability etc. allows proving theorems such as the intermediate value theorem and Rolle's theorem.

The PVS proof assistant has an implementation of basic real analysis built on an axiomatic definition of the reals [28]. This implementation includes definitions of convergence, continuity, differentiability of real-valued functions and proofs for theorems around these concepts (for example, the mean value theorem).

In Coq, two approaches have been explored. The Coq Standard Library called `Reals` provides an axiomatic definition of the real numbers and classical $\varepsilon - \delta$ concepts for real analysis. The library contains a bunch of results for real analysis: sequences and series, transcendental functions, concepts of limit, continuity, differentiation, integration, calculus theorems like the mean value theorem,the fundamental theorem of calculus etc.

There also exists a constructive formalization of real analysis in Coq [24]. In C-CoRN (Coq Constructive Repository at Nijmegen [25]) the reals are build as a Cauchy completion of the rationals. This library is based on the constructive approach to mathematics initiated by [12] which forbids the use of the excluded middle and the axiom of choice in reasoning steps. In constructive mathematics, whenever we say something exists we must have a way to produce that something. Also, all functions must be terminating algorithms. In particular we cannot assume the existence of test functions that compare two Cauchy sequences and verify that they are equivalent, in other words that two real numbers are equal. Working with constructive mathematics forces us to avoid some of the reasoning steps that are possible in classical mathematics reasoning.

The C-CoRN library contains a lot of results proved in this setting of constructive mathematics. We can mention the constructive formalization of the fundamental theorem of algebra [33].

2.1.2 Matrices

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 [64] implements matrices in order to deal with linear programs and treats the special case of sparse matrices. In the development presented in [22] conducted in ACL2 matrices are implemented in a way that insures computation efficiency. In Coq there are several formalizations for matrices and linear algebra. We cite [58] and [75] as standard Coq contributions, and [11] as an implementation of matrices using the SSReflect [36] extension of Coq. HOL Light has a development on matrices described in [45].

2.2 Mixing COQ and SSReflect

Our formalizations are made in the proof assistant Coq with the SSReflect extension. One of the reasons for this choice is the number of formalized concepts already available in the libraries of the proof assistant. We used extensively the Coq standard library on real numbers and real analysis and the SSReflect library on matrices. We present both of these libraries and show how they work together.

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 $x > y$.

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 [36]. The proof of the Four Color Theorem [35] and the on-going effort to formally verify Feit-Thompson theorem illustrate the power of SSReflect. For example, the Feit-Thompson theorem is of major importance in group theory. 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 rely on 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; 32].

The Mix

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 real 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.

.... Technical Details 1.

`Implementation:` SSReflect `basic libraries,` Coq `real numbers`
Coq`: reflection, coercions, canonical structures, equality, the type` Prop

Reflection. 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, the predicate reflect links the proposition to the boolean true when the decision procedure says the proposition is true, and to the boolean 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 in this case 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 the use of *coercions* and *canonical structures*.

Equality. Equality in Coq is a syntactic equality, also called Leibniz equality. With this definition a term can only be equal to itself. Equality in Coq has type Prop. This means for T of type Type and a b of type T, the term a = b is of type Prop.

A decidable equality is a binary boolean function equivalent to the Leibniz equality. In SSReflect, a type with decidable equality is implemented as a type sort together with a function eq: sort→sort→bool that reflects the standard Coq equality on that type. This means 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 form having a very large development and explained in detail in [32].

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

Coercions. The coercion mechanism implemented in Coq allows us to view a certain type as 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.

Canonical Structures. 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 (see section 1.2). The type of natural numbers is called nat. Based on the inductive definition of nat we can build a boolean equality predicate eqn : nat→nat→bool. Using the reflect predicate we can say that the Leibniz equality x = y is equivalent to the boolean equality eqn x y.

```
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 representant 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 representant. 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 representant of the set. The representant 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 is a type for which a finite enumeration of all its elements can be provided. Thus, a finType is formalized as a structure that contains the type, the list of all elements of the type 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 can be 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 represented as a list. 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 because 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.

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:

$$\texttt{\\big[op/nil]\_(i} \leftarrow \texttt{r | P i) F}$$

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:

$$\texttt{\\big[+/0]\_(i < n) x i}$$

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

$$\texttt{\\big[*/1]\_(i} \leftarrow \texttt{I) v i}$$

and the third:

$$\texttt{\\big[Rmax/0]\\\_(i} \leftarrow \texttt{r | v i != w) (norm (v i) - w)}$$

Notation conventions are added so that indexed sums and products of natural numbers or of 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 [9].

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'll go even further in our technical details and explain how this fits in Coq's formalism.

Prop and Type. In Coq the type of logical propositions is Prop (see section 1.2) and it is a type with special features. As we saw in section 1.2, 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 $P \vee Q$ 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 Coq standard library on real numbers library, the trichotomy axiom is stated using this disjunction under Type.

```
Axiom total_order_T : forall r1 r2:R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
```

Having this axiom in the standard library makes real number comparison and equality "testable", thus bluring the distinction between decidable and non-decidable properties in the Coq practice. In a constructive setting like the CoRN library on real numbers, we would not have been able to test equlity of two real numbers (as this is intrinsically undecidable). This axiom on standard library real numbers makes reasoning on such numbers compatible with classical mathemtics proofs.

Using the trichotomy axiom we can define a function eqr: 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.

```
(* 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 endow to 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.

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.

.. End technical details.

2.3 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. Our development on 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 talk about special kinds of matrices, so we define what it means for a matrix to be:

- symmetric : $\forall ij, A_{ij} = A_{ji}$
- positive definite (for square matrices) : $\forall x \in \mathbb{R}^n, x \neq 0 \Rightarrow x^T A x > 0$

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 [29] 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 $\|A\| \leq \|B\|$

For the proofs described in chapter 3, 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

$$\|A\| < 1 \Rightarrow \det(E_p - A) \neq 0 \tag{2.1}$$

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 $(A_k)_{k\in\mathbb{N}}$

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

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

$$\sum_{k=0}^{\infty} A_k = \lim_{N\to\infty} \sum_{k=0}^{N} A_k \tag{2.2}$$

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

.... Technical Details 2.

`Implementation: 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 section 1 that SSReflect has a library for dealing with indexed operations and finite sums are a special case of such operations (in the code below denoted by \sum). 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 (in the code below denoted 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. We give two definitions of convergence of matrix series, that we prove equivalent.

```
(* convergence of a matrix series according to the definition 2.2 *)
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 *)
```

We note that in the first definition the sum is a sum of matrices, so this definition will help us when we need to manipulate indexed sums of matrices. 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.

... End technical details.

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:

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

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:

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

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 relation 2.1.

```
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.

$$\|A\| = \max_i \sum_j |a_{ij}| \tag{2.3}$$

For the proofs described in chapter 4 we need a formalization for the notion of eigenvalue of a square matrix. An eigenvalue can be defined as a root of the characteristic polynomial associated to the matrix. The characteristic polynomial has real coefficients and can thus have complex values. This is an issue as Coq does not have a library for complex numbers. Luckily, in our study we only need to talk about the eigenvalues of symmetric matrices, and it is well know (though we do not prove this formally) that symmetric matrices have all real eigenvalues. For such a symmetric square matrix we can define the eigenvalues by using the notions of characteristic polynomial and root that are available in the SSReflect libraries and described in [11].

```
Variable A: 'M[R]_n.
Hypothesis Hsym: symmetric A.
Definition eigenv := root (char_poly A).
```

As tradition wants it, we will use λ to denote eigenvalues. However, manipulating eigenvalues by using the above definition raises some technical issues described in what follows.

.... Technical Details 3.

`Implementation: indexed operations and polynomials`

In the SSReflect library on polynomials, root is a predicate. This means that given a polynomial P with coefficients of type T and a value x of the same type T, (root P x) returns true if x is a root of P and false otherwise. This implementation raises two problems in our case.

First, the type of the roots of a polynomial is the same as the type of coefficients. In this implementation we cannot consider complex roots of real polynomials. This issue contributed to our choice of restricting the implementation of eigenvalues to symmetric matrices, and therefore real polynomials with all roots real.

The second problem with the implementation of root as a predicate is that there are situations in which this implementation is not very convenient, for example, in defining the smallest eigenvalue. The natural choice would be to say it is the minimum of all eigenvalues, by using SSReflect's indexed operations library, in a definition like:

```
Definition lambda_min A := \big[Rmin / 0 ]_(lambda ← r | eigenv A lambda ) lambda.
```

Translated to standard mathematics this reads:

$$\lambda_{min} = \min_{\substack{\lambda \in r \\ \lambda \text{ eigenvalue of } A}} \lambda$$

The question is, who is this mysterious r ? If we remember the syntax for indexed operations presented in section 1, we realize that the indexes over which our operation ranges are gathered in a list. So in order to define the minimum of eigenvalues, we need to have a list which contains the eigenvalues. For our purposes of formalization we will assume that such a list exists. This will allow us to give the appropriate definition.

```
Variable eigens: 'M[R]_n → seq R.
(* seq is the type of lists in SSReflect *)

Hypothesis Heigens: forall x, eigenv A x = eigens A x.
```

```
(* x satisfies the predicate (eigenv A) is equivalent to x is in the list
    (eigens A) *)
```

```
Definition lambda_min A := \big[Rmin / 0 ]_(lambda <- (eigens A) ) lambda.
```

We notice that in this second definition of lambda_min no filter is needed on the list of values, whereas in the first definition we required that lambda be an eigenvalue by using the filter (eigenv A lambda). The reason is the hypothesis Heigens which ensures that the list (eigens A) contains exactly the eigenvalues of A.

There is still an issue with this second definition: the value by default is 0 (zero). This means that when looking for the minimum in the list we compare the smallest element so far with the current element in the list until we reach the end, that is until we get the empty list. For the empty list we return the value by default. This works well when we have the neutral element of the operation as value by default. For example, for addition, we add all elements in the list, and for the empty list we add zero. But in the case of minimum, the operation does not have a neutral element on the set of real numbers. To patch this we use our knowledge of the theory (but that we do not formalize): we use the fact that the list of values is not empty, so we give as value by default the last value in the list. We know the list is not empty as it is the list of roots of a polynomial of degree at least one and for which we know it has all roots real. The definition becomes:

```
Definition lambda_min A :=
  \big[Rmin / (last 0 (eigens A)) ]_(lambda <- (eigens A) ) lambda.
```

Other fixes for issues with having the minimum and maximum as indexed operations will be discussed in technical section 5.

.. End technical details.

In the special case of symmetric matrices we are treating, we define the spectral radius $\rho(A)$ as the maximum of the absolute values of the eigenvalues. A collection of basic results are established for eigenvalues and the spectral radius of a symmetric matrix:

- for each eigenvalue λ there is an associated eigenvector (a vector $x \neq 0$ such that $Ax = \lambda x$),
- the absolute value of an eigenvalue is smaller than the norm of the matrix,
- the spectral radius is smaller than the norm of the matrix,

- all eigenvalues of a positive definite matrix are positive.

However, there are some results that are not completely formalized around properties of the Rayleigh quotient associated to the symmetric matrix A and the nonnull vector x. The Rayleigh quotient is defined as follows:

$$R(A,x) = \frac{x^T A x}{x^T x}$$

The result we need is

$$\lambda_{\min} \leq R(A,x) \leq \lambda_{\max}$$

This proof is not complicated. It requires some concepts of multivariate analysis which are partially available thanks to the work described in section 2.4.

Note

We agree that this small study of eigenvalues in the special case of symmetric real matrices is not satisfactory. At present a more comprehensive formalization is being undertaken by Guillaume Cano. This work aims at formalizing complex numbers, eigenvalues of real matrices in general, as well as important results such as the Perron-Frobenius theorem.

2.4 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 ourselves the notions from multivariate analysis. We start by formalizing real vectors in Coq.

2.4.1 Vectors in $\mathbb{R}^p$

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 (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 $v = (v_0, v_1, \ldots, v_{p-1})$.

At the moment of our first implementation of vectors, the SSReflect libraries did not have an implementation of matrices. So, considering possible implementations of vectors revealed some interesting issues, which we discuss in the following technical section.

.... Technical Details 4.

```
Implementation:  vectors as lists and functions
Coq:  dependent types
```

We want to compare the possible solutions for implementing in Coq real vectors, or elements of $\mathbb{R}^p$. We want an implementation that allows us to easily define functions and operations on our vectors and that allows us to have proofs close to the mathematical intuition. We can view a vector $v = (v_0, v_1, \ldots, v_{p-1})$ in two ways: as a list of length p of real numbers or as a function from $\{0, 1, \ldots, p-1\}$ to $\mathbb{R}$.

A list of length p is traditionally defined as a dependent type. Here is the definition given in [58] for a list of elements of type A and of given length.

```
Inductive vect (A: type): nat → Type :=
  | vnil : vect 0
  | vcons : forall n : nat, A → vect n → vect (S n).
```

Then, implementing basic operations like addition requires skilfull handling of dependent types. Here is the definition for addition from [58].

```
Fixpoint addvect (n : nat) (v : vect A n) {struct v} :
vect A n → vect A n :=
 match v in (vect _ k) return (vect A k → vect A k) with
  | vnil ⇒ fun v' ⇒ vnil A
  | vcons n1 x1 v1 ⇒
      fun v' : vect A (S n1) ⇒
      match v' in (vect _ k) return (k = S n1 → vect A k) with
      | vnil ⇒ fun h ⇒ vnil A
      | vcons n2 x2 v2 ⇒
          fun h ⇒
          vcons A n2 (Aplus x1 x2) (* Aplus is addition on the type A *)
            (addvect n2
               (eq_rec n1 (fun n : nat ⇒ vect A n) v1 n2
                  (eq_add_S_tr n1 n2 (sym_eq h))) v2)
      end (refl_equal (S n1))
  end.
```

But lists are useful, for example, in having indexed operations on the elements of the vector.

Defining vectors as functions makes implementing operations like addition straightforward.

```
Definition addvect_fun v1 v2 := fun i ⇒ v1 i + v2 i.
```

So both possible representations have their benefits, and we want to take advantage of all of them by having two views for vectors. This is well supported by the libraries of Coq's SSReflect extension. A vector is implemented as a finfun from 'I_p to R. As explained in technical section 1, this means a vector is a list of p elements, but it is coerced to a function 'I_p→R. We thus have the two views on vectors: as functions and as lists.

.... **End technical details.**

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 for building a vector.

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

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:

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

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

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

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

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

.... Technical Details 5.

`Implementation: maximum as an indexed operation`

Defining the maximum norm is straightforward using the library on indexed operations:

Definition norm_max (v: Rvec 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 easily be 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, $0 \le x$) which is

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

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

Nevertheless, the use of the maximum as an indexed operation posed some difficulties. As stated before, 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:

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

Basic lemmas on the norm are proved by moving to this operator (equivalent to the initial one on the desired subset) and using the indexed operation library.

.. End technical details.

Other norms can be instantiated without difficulty. This maximum norm is compatible to the matrix norm defined in section 2.3, 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.

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

To express the fact that $\mathbb{R}^p$ is a metric space, we use the definition in the standard library of Coq. This will help us consider the properties of sequences and functions in a metric space and in the special case of $\mathbb{R}^p$ Some technical issues around the definition of metric space in Coq are detailed in the following section.

.... Technical Details 6.

`Implementation: metric spaces`

In Coq's standard library the definition for a metric space is

```
Structure Metric_Space: Type := Build_Metric_Space {
  Base: Type;
  dist: Base → Base → R;
  dist_pos: forall x y: Base, dist x y ≥ 0;
  dist_sym: forall x y: Base, dist x y = dist y x;
  dist_refl: forall x y: Base, dist x y = 0 ↔ x = y;
  dist_tri: forall x y z: Base, dist x y ≤ dist x z + dist z y}.
```

However, there are no properties proved on a general metric space. This structure is only used to define the limit in a point of a function between two metric spaces. The definition is then instantiated for the real numbers and all results on limits are established in the special case of a real function. Also, convergence of sequences and Cauchy criterion are defined just for sequences of real numbers, without using the Metric_Space structure.

To have a more homogeneous formalization and to avoid duplication of proofs we define all these concepts and prove the corresponding properties in a general metric space. For a more comfortable use of the structure, we first declare a coercion from a Metric_Space to its Base type.

```
Coercion Base: Metric_Space ↣ Sortclass.
```

Then we declare the corresponding Canonical Structures for the metric spaces R with distance $|x - y|$ (or, in Coq syntax Rabs (x−y)) and vec R p with distance dist_Rp defined above.

```
Canonical Structure metricSpace_R := Build_Metric_Space R Rabs ...
Canonical Structure metricSpace_Rp p := Build_Metric_Space (vec R p) (@dist_Rp p) ...
```

These constructs (coercions, implicit arguments and canonical structures) will help us automatically infer the Metric_Space structure from the context and make the script more readable.

We prove general properties for the distance operator and we prove that all metric spaces are separated spaces (or Hausdorff spaces), that is for every pair of distinct points x, y we can find a pair of disjoint neighborhoods U_x, U_y for the points.

.. End technical details.

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:

$$\forall \varepsilon \in \mathbb{R}, 0 < \varepsilon \Rightarrow \exists N \in \mathbb{N} \text{ such that } \forall m, n \in \mathbb{N}, N \leq m, N \leq n \Rightarrow \text{dist}\ (X_m, X_n) < \varepsilon) \tag{2.4}$$

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:

$$\forall \varepsilon > 0, \exists \alpha > 0, \forall x \in M, 0 < \text{dist}_M(x, x_0) < \alpha \Rightarrow \text{dist}_{M'}(f(x), l) < \varepsilon \tag{2.5}$$

We note that Coq's standard library already contains a generic definition for limit that takes into account a set $D \subseteq M$ meant to model the definition domain of the function f:

$$\forall \varepsilon > 0, \exists \alpha > 0, \forall x \in M, x \in D \wedge \text{dist}_M(x, x_0) < \alpha \Rightarrow \text{dist}_{M'}(f(x), l) < \varepsilon \tag{2.6}$$

There are two main differences between the two definitions for the limit:

- the constraint on the point x_0 at which we compute the limit,
- the domain taken into account when computing the limit.

The Coq standard library definition 2.6 imposes that the point x_0 at which we compute the limit has to also satisfy the conditions in the definition: if $x_0 \in D$ and since $\mathrm{dist}_M(x_0, x_0) < \alpha$ we need to have $\mathrm{dist}_{M'}(f(x_0), l) < \varepsilon$, and this for all ε. This implies that if the function f is defined at a point x_0, then the limit at that point will be the value of the function. Otherwise put, any function defined at a point x_0 and with a limit at x_0 is necessarily continuous in x_0.

In our version we do not take into account the point x_0, that is we only ask the value of the function to be close to l on points in the neighborhood of x_0, but not in x_0. In particular, we will allow the value of the function f in x_0 to be different from its limit at that same point. This corresponds to a kind of discontinuity of the function f in x_0. As an example, take the following discontinuous function $f : \mathbb{R} \to \mathbb{R}$

$$f(x) = \begin{cases} 1 & \text{if } x \neq 0; \\ 0 & \text{if } x = 0 \end{cases}$$

This function is definable in the Coq library of real numbers, since we can do a case analysis of two reals being equal or not. We should therefore be able to talk about the fact that it has a certain type of discontinuity.

We note that our definition of limit 2.5 is only valid if x_0 is an accumulation point for the definition domain of the function (that is in all neighborhood of the point x_0 there are other points of the definition domain). In our definition we consider only total functions, and for total functions on $\mathbb{R}$ or $\mathbb{R}^p$ any point is an accumulation point.

The Coq standard library version considers a domain D in which the limit of the function has to be considered, but it does not impose any properties on this domain. Since the definition of limit is also different there are no conditions imposed on the domain, or on the point where we compute the limit (in particular, the point does not have to be an accumulation point).

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.

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

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.

2.4.3 Derivatives

We begin our study of derivatives by implementing partial derivatives for functions from $\mathbb{R}^p$ to $\mathbb{R}$. 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 then follows the implementation of derivatives in the Coq standard library.

.... Technical Details 7.

```
Implementation:  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 as before.

.. End technical details.

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:

$$\begin{aligned}
f(a) - f(b) &= f(a_1, \ldots, a_p) - f(b_1, \ldots, b_p) = f(a_1, \ldots, a_p) - f(b_1, a_2, \ldots, a_p) + \\
&+ f(b_1, a_2, \ldots, a_p) - f(b_1, b_2, a_3, \ldots, a_p) + \ldots + \\
&+ f(b_1, \ldots, b_{p-1}, a_p) - f(b_1, \ldots, b_p) = \\
&= \sum_{i=1}^{p} (a_i - b_i) \frac{\partial f(b_1, \ldots, b_{i-1}, c_i, a_{i+1}, \ldots, a_p)}{\partial a_i}
\end{aligned}$$

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 $f(a_1, \ldots, a_p) - f(b_1, a_2, \ldots, a_p)$ of a vector function varying in the first argument as a real function. All these are nontrivial reasoning steps for a mechanized system. We give some details in the following technical section.

.... Technical Details 8.

`Implementation:  implicit steps on paper`

The purpose of this technical section is just to illustrate that reasoning steps that are considered trivial enough to be overlooked on paper sometimes require a lot of formalization effort.

There are several steps we need to take in order to formalize the above formula. We first define a function modif_v that modifies the i-th component of a vector to some given value r.

```
Definition modif_v (x: vec R p) (i: 'I_p) (r:R): vec R p :=
  \vec_(j < p) if i == j then r else (x j).
```

We then define a function R→R by modifying the i-th component of a vector function.

```
Definition modif (f: vec R p → R)(x: vec R p)(i: 'I_p )(r: R): R := f (modif_v x i r).
```

We show how partial derivability of the initial function relates to the derivability of the real function.

```
Lemma dp_impl_der_pt: forall f x i l,
  part_deriv_pt_1 f x i l ↔ derivable_pt_lim (modif f x i) (x i) l.
Lemma dbl_impl_derb: forall f x i, dbl_pt_1 f i x → derivable_pt (modif f x i) (x i).
Lemma eq_dp_der: forall f x i (prp:dbl_pt_1 f i x),
  derive_pt (modif f x i) (x i) (dbl_impl_derb prp) = dp_1 prp.
```

We define the vector that has the elements of a given vector a up the i-th component and the elements of a vector b for the rest.

```
Definition g (a b : vec R p) i :=
  \vec_(j < p) if (j < i) then b j else a j.
```

We give some definitions and technical lemmas that help us manipulate the concepts we introduced so far.

```
Lemma modif_id: forall f x i, modif f x i (x i) = f x.
Lemma eq_ga: forall a b (i: 'I_p), g a b i i = a i.
Lemma eq_gb: forall a b (i: 'I_p), modif_v (g a b i) i (b i) i = b i.
Definition hf f a b i := f (g a b i) − f (g a b (S i)).
Lemma modif_to_p: forall a b, g a b p = b.
Lemma fun_one_var: forall a b (i: 'I_p), g a b (S i) = modif_v (g a b i) i (b i).
Lemma sum_h: forall f a b i, \sum_(j< (S i)) hf f a b j = f a − f (g a b (S i)).
Lemma write_to_sum: forall f a b, \sum_(j< p) hf f a b j = f a − f b.
```

We are now able to prove the first equality in the statement.

```
Lemma write_to_sum2: forall f a b,
  \sum_(j < p) (f (g a b j) − f (modif_v (g a b j) j (b j))) = f a − f b.
```

We formalize a general sum such that the sum whichappears in the last equality of our formula will be an instance of this general sum.

```
Definition sum_dp_vi f c v (pr:forall i x, dbl_pt_1 f i x) := \sum_i (dp_1 (pr i (c i))) ∗ v i.
```

We are now able to give the proof of our whole statement.

```
Lemma pag9: forall f a b (pr: forall i x, dbl_pt_1 f i x),
  (forall i, cont (fun x ⇒ dp_1 (pr i x)) a) →
    exists c, (forall i, dist (c i) a ≤ dist b a) ∧ (f b − f a = sum_dp_vi c (b −^ a) pr).
```

.. End technical details.

The most relevant result we needed for Kantorovitch's theorem is Taylor's formula for functions of class $C^{(2)}$. The statement and the proof are as follows:

Lemma 2 (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 $c \in (a, a+v)$ so 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

$$g'(t) = \sum_{i=1}^{p} \frac{\partial f(a+tv)}{\partial x_i} v_i \tag{2.7}$$

$$g''(t) = \sum_{i,j=1}^{p} \frac{\partial^2 f(a+tv)}{\partial x_i \partial x_j} v_i v_j \tag{2.8}$$

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 (2.7) and (2.8).

We define the equivalent in several dimensions of the derivative of a function, that is the Jacobian matrix.

We have $f : \mathbb{R}^p \to \mathbb{R}^p$, that is

$$f(x) = (f_1(x), f_2(x), \ldots, f_p(x)),\ \text{with } x = (x_1, x_2, \ldots, x_p)$$

The Jacobian matrix is given by

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

The Jacobian matrix of a function f at a point x_0 is just a real matrix and we thus have the results from section 2.3 that can be used.

2.4.4 Related formalizations

The only proof assistant that already has a formalization of multivariate analysis is HOL Light [45]. 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.

2.5 Interval Analysis

Interval analysis is a branch of mathematics motivated by its practical applications. It is of use when dealing with inequalities, approximate numbers or error bounds in computations, so it is closely related to numerical analysis.

We use an interval x as the formalization of the intuitive notion of an unknown number $\tilde{x}$ known to lie in x. In interval analysis we do not say that the value of a variable is a certain number, but we say that a value of a variable is in an interval of possible values. For example, when dealing with numbers that cannot be represented exactly like π or $\sqrt{2}$ we usually say that π is approximately equal to 3.14. Instead, in interval analysis we say that π is exactly in the interval $[3.14, 3.15]$. For an operation where we have errors in the inputs we can give an approximate result

$$-\pi * \sqrt{2} \approx -3.14 * 1.41 = -4.4274$$

On the other hand, when we do an operation in interval analysis we no longer use approximations. For example for the above multiplication, whenever we have 2 values in the input intervals then their product is a value in the result interval.

$$[-3.15, -3.14] * [1.41, 1.42] = [-4.473, -4.4274]$$

So regardless of the imprecision in the input data, we can always be sure that the result will be in the computed bounds. In interval analysis we cannot be wrong because of rounding errors or method errors, we can only be imprecise by giving a very big interval for the expected value.

2.5.1 Description

To talk about real intervals more formally we use the presentation in [62]. We call a real interval (by an "abus de langage") a set of the form

$$x = [\underline{x}, \overline{x}] := \{\tilde{x} \in \mathbb{R} \mid \underline{x} \le \tilde{x} \le \overline{x}\}$$

where $\underline{x}, \overline{x}$ are elements of $\mathbb{R}$ with $\underline{x} \le \overline{x}$. In particular intervals are closed, bounded, connected and nonempty subsets of $\mathbb{R}$ and we can use the standard set theoretic notation. The set of all real intervals is denoted by $\mathbb{IR}$. We use x as notation for a generic interval, $\underline{x}$ or $\inf(x)$ for the lower bound of x and $\overline{x}$ or $\sup(x)$ as the upper bound of x.

An interval is called

- thin, if $\underline{x} = \overline{x}$,
- thick, if $\underline{x} < \overline{x}$.

Thin intervals contain only one real number and we can identify a thin interval with the unique number contained in it. In particular, real numbers need not be distinguished notationally from intervals.

To each real interval we can associate the midpoint and the radius of the interval:

$$x_c = \mathrm{mid}(x) := \frac{\overline{x} + \underline{x}}{2} \;; \quad \Delta_x = \mathrm{rad}(x) := \frac{\overline{x} - \underline{x}}{2}$$

An alternative characterization of an interval uses these two quantities

$$x = [\underline{x}, \overline{x}] = [x_c - \Delta_x, x_c - \Delta_x]$$

We define operations on intervals. We keep in mind that what we expect from, for example addition, is that whenever we have two numbers in some intervals we want their sum to be in the sum interval. So, addition will be described as follows (for simplicity, we use the same symbol + for both addition of intervals and of real numbers):

$$x + z := \square\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\}$$

If S is a set of real numbers, the symbol $\Box S$ denotes the smallest interval that contains the set S. In the case of addition we have

$$x + z := \Box\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} = \{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} = [\underline{x} + \underline{z}, \overline{x} + \overline{z}] \tag{2.9}$$

We define opposite and multiplication in the same manner

$$-x := \Box\{-\tilde{x} \mid \tilde{x} \in x\} = \{-\tilde{x} \mid \tilde{x} \in x\} = [-\overline{x}, -\underline{x}]$$

$$xz := \Box\{\tilde{x}\tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} = \{\tilde{x}\tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} = [\min(\underline{x}\underline{z}, \underline{x}\overline{z}, \overline{x}\underline{z}, \overline{x}\overline{z}), \max(\underline{x}\underline{z}, \underline{x}\overline{z}, \overline{x}\underline{z}, \overline{x}\overline{z})]$$

2.5.2 Rounded interval arithmetic

Up to here we discussed intervals with real bounds, that is intervals $[\underline{x}, \overline{x}]$ with $\underline{x}, \overline{x} \in \mathbb{R}$. However, in practice we usually have intervals $[\underline{x}, \overline{x}]$ with $\underline{x}, \overline{x} \in M$, where M is a machine representable subset of real numbers, for example, floating point numbers. To work in such a setting we need to use rounding, but in the spirit of interval analysis. We need to make sure that all the values we are interested in are in the rounded interval. We will denote rounding an interval with $\Diamond$

$$\Diamond x := [\nabla\underline{x}, \Delta\overline{x}]$$

∇ means downward rounding, that is

$$\nabla a = \max_{\substack{m \in M \\ m \le a}} m$$

and Δ means upward rounding

$$\Delta a = \min_{\substack{m \in M \\ m \ge a}} m$$

The operation $\Diamond x := [\nabla\underline{x}, \Delta\overline{x}]$ is called outward rounding of the interval x and it satisfies the property

$$x \subseteq \Diamond x$$

With this rounding operation we can define a rounded arithmetic. For example, rounded addition of two intervals is defined as the outward rounding of the ideal addition of the intervals

$$x +^{\Diamond} z = \Diamond(x + z)$$

Opposite and multiplication are treated similarly

$$-^{\Diamond} x = \Diamond(-x)$$

$$x *^{\Diamond} z = \Diamond(x * z)$$

In our example from the beginning of the section

$$[-3.15, -3.14] * [1.41, 1.42] = [-4.473, -4.4274]$$

by rounding to two decimal digits we get

$$[-3.15, -3.14] *^{\Diamond} [1.41, 1.42] = \Diamond[-4.473, -4.4274] = [\nabla -4.473, \Delta -4.4274] = [-4.48, -4.42]$$

The results in the two arithmetics are different. This shows that some properties of an ideal operation will no longer hold for the rounded operation. This is true for addition, opposite and multiplication. If for ideal addition we had, according to relation 2.9

$$\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} = x + z$$

for rounded addition we have only an inclusion

$$\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} \subseteq x +^{\Diamond} z$$

However, the proofs that we intend to formalize and that we will discuss in chapter 4 are based on relations like 2.9, which in their turn depend on having an ideal arithmetic. This is why, in our formalization in Coq, we use intervals with real bounds.

2.5.3 Implementation

Now we need to find a good way to formalize real intervals in Coq. We want to be as close as possible to the "pen and paper" description in [62]

$$x := [\underline{x}, \overline{x}], \quad \underline{x}, \overline{x} \in \mathbb{R}, \quad \underline{x} \leq \overline{x}$$

We need to capture two aspects:

- we have a dual view of intervals: on one hand an interval can be seen as a pair of real numbers representing its lower and upper bounds and on the other hand an interval is the set of real numbers comprised between the two bounds;
- a real number can be seen as an interval.

To achieve this we define an interval as a structure that contains two real numbers `inf` and `sup` representing the lower and upper bounds and a proof that the lower bound is smaller than the upper bound.

```
Structure IR : Type := ClosedInt { inf : R ; sup : R ; leq_proof : inf ≤b sup }.
```

The next technical section explains the mechanisms that make our definition work as we would expect.

.... Technical Details 9.

`Implementation:  real intervals`
`Coq:  proof irrelevance and the type Prop`

We need to talk again about the type of logical propositions Prop. Two proofs of the same proposition are not necessarily equal. This can produce undesired effects when we have terms that depend on proofs. Let's consider as an example a modified version of our intervals

```
Structure IR' : Type := ClosedInt { inf : R ; sup : R ; leq_proof_prop : inf ≤ sup }.
```

leq_proof_prop is a term of type inf≤sup, otherwise said, a proof of the proposition inf≤sup, as inf≤sup is indeed of type Prop, according to the standard library definition.

So, an interval is a triplet (inf, sup, leq_proof_prop). Now, take the intervals $x = (1, 2, \mathsf{leqx})$ and $z = (1, 2, \mathsf{leqz})$. To show that $x = z$ we not only have to show that $1 = 1$ and $2 = 2$ but also that $\mathsf{leqx} = \mathsf{leqz}$. This latter equality is not provable in general for an arbitrary Prop and in the basic logic of Coq. It is only provable in general if additional axioms are used.

However, in the basic logic, there are particular propositions for which we can show they have a unique proof. It is the case of a proposition expressing equality of two booleans or, more generally, of two terms of a type equipped with a decidable equality. For all the theoretical details see [48].

Since we want equality of two intervals to be just the equality of the two bounds we need to make our definition fit in the case above. We achieve this by defining a boolean function Rleb that is true when inf≤sup and false otherwise. This is possible thanks to the trichotomy axiom (see section 1).

```
(* lemma derived from the trichotomy axiom *)
Lemma Rle_dec: forall r1 r2, {r1 <= r2} + {~ r1 <= r2}.

Definition Rleb r1 r2 := match (Rle_dec r1 r2) with
    | left _ => true | right _ => false end.
```

In the SSReflect framework a boolean value b can be interpreted as a Prop by using the coercion is_true which maps the boolean b to the Prop b = true.

```
Coercion is_true (b: bool) := b = true.
```

So, for the type of the proof that inf is smaller than sup we can use the boolean Rleb inf sup that will be coerced to the Prop Rleb inf sup = true, thus giving a well typed expression. This gives our actual implementation.

To summarize, inf $\leq_b$ sup is a notation for Rleb inf sup. The type of the field leq_proof, in our actual definition of intervals, is a proposition obtained by coercing the boolean Rleb inf sup to the proposition is_true (Rleb inf sup). This reduces to Rleb inf sup = true, according to the definition above. This latter expression is an equality between booleans and therefore has a unique proof. So, two intervals will be equal if their bounds are equal, as in this case the proofs will be equal thanks to this unicity.

.. End technical details.

Thanks to our choice of implementation we can prove that equality of two intervals is equivalent to the equality of the respective bounds.

Lemma eq_intervalP : forall x z : IR, x = z ↔ inf x = inf z ∧ sup x = sup z.

So our intervals can be viewed as pairs of real numbers. We can also view an interval as the set of real numbers between the lower and the upper bound of the interval. This is done by using Coq's coercions (see technical section 1). This coercion allows us to transparently use our intervals as sets of real numbers. We also define a coercion from a real number to the corresponding thin interval, so we can directly use real numbers as intervals.

We define the midpoint mid and the radius rad of an interval. The membership relation can also be expressed as

$$\tilde{x} \in x \Leftrightarrow |\tilde{x} - x_c| \leq \Delta_x$$

The corresponding Coq lemma is

Lemma in_mid_rad: forall (x: IR) (tx: R), tx \in x ↔ Rabs (tx − mid x) ≤ rad x.

In the above statement, \in is an infix notation for belonging to a set. This lemma illustrates how the coercion mechanism lets us directly use an interval as a set.

We define the elementary operations on intervals that we described in section 2.5.1 by giving the explicit formulas to compute their bounds. We take the example of addition.

$$x + z := [\underline{x} + \underline{z}, \overline{x} + \overline{z}]$$

In implementing our operations we take into account that our definition of intervals contains a proof that the lower bound is smaller than the upper bound. We need to provide these proofs before actually defining the operations. For the operations we are considering this is not a big effort as they are straightforward. We give the proof and define addition according to the formula above:

```
(* proof that addition is well defined *)
Lemma addI_wd : forall x z, inf x + inf z ≤_b sup x + sup z.

Definition addI x z :=
  @ClosedInt (inf x + inf z) (sup x + sup z) (add_i_wd x z).
```

Opposite and multiplication are defined in the same manner. However, we define separately the multiplication of an interval by a scalar, even though this is equivalent to the multiplication by a thin interval:

$$ax := [\min(a\underline{x}, a\overline{x}), \max(a\underline{x}, a\overline{x})]$$

The reason for this choice is that multiplication of an interval by a scalar enjoys more algebraic properties than interval multiplication in general. For example, distributivity holds in the following expression because x is thin

$$x(y+z) = xy + xz, \quad x \text{ is thin}$$

For an arbitrary interval x, we only have an inclusion

$$x(y+z) \subseteq xy + xz$$

When using these properties it is more convenient if they are attached to a specific operation than if we have to provide a proof that the interval is thin each time we use them.

Now we can prove the desired properties on our intervals. We saw in section 2.5.1 that we are interested in equalities like

$$x + z = \{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} \tag{2.10}$$

We show this equality by proving both inclusions of the corresponding sets. We have one inclusion that is straightforward:

$$\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} \subseteq [\underline{x} + \underline{z}, \overline{x} + \overline{z}] \tag{2.11}$$

as everytime we have two real numbers $\tilde{x}, \tilde{z}$ with $\tilde{x} \in x$ (which means $\underline{x} \leq \tilde{x} \leq \overline{x}$) and $\tilde{z} \in z$ (which means $\underline{z} \leq \tilde{z} \leq \overline{z}$) then their sum $\tilde{x} + \tilde{z} \in [\underline{x} + \underline{z}, \overline{x} + \overline{z}]$ which is by definition $x + z$.
The other inclusion is less straightforward:

$$[\underline{x} + \underline{z}, \overline{x} + \overline{z}] \subseteq \{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\}$$

We need to show that each time a number belongs to the sum of two intervals x and z, then there exists $\tilde{x} \in x$ and $\tilde{z} \in z$ such that our number is written as $\tilde{x} + \tilde{z}$. The difficulty comes form the fact that the decomposition of a number in a sum is not unique. To give a decomposition of a real $s \in x + z$ in an appropriate sum we consider the following cases:

$$s = \begin{cases} \underline{x} + (s - \underline{x}) & \text{, if } s \in [\underline{x} + \underline{z}, \underline{x} + \overline{z}] \text{ with } \underline{x} \in x, (s - \underline{x}) \in z \\ (s - \overline{z}) + \overline{z} & \text{, if } s \in [\underline{x} + \overline{z}, \overline{x} + \overline{z}] \text{ with } (s - \overline{z}) \in x, \overline{z} \in z \end{cases}$$

The proof of equality (2.10) does not appear in standard books of interval analysis, as it is clear for the trained mathematician that the equality is trivially true. However, in a formal system we needed to go into some detail to show this equality. We remark also that equality (2.10) does not hold for an interval arithmetic that uses outward rounding of the interval bounds as we saw in section 2.5.2. In this case only the first inclusion holds (relation (2.11)).

We show that addition on intervals enjoys nice properties: it is associative, commutative, accepts the thin interval 0 as a neutral element. This means that the set of real intervals with addition has a commutative monoid structure. This will ease our work, as general theorems concerning the commutative monoid structure are directly available from the SSReflect libraries, in particular we will be able to use lemmas concerning indexed operations when defining operations on interval matrices as we shall see in section 2.5.4. We remark that other properties are not satisfied. For example, an interval that is not thin does not have an opposite with respect to the neutral element, which means addition on intervals is not a group operation. This also means that intervals with addition and multiplication do not form a ring. This fact will play a role in our manipulation of interval matrices (see section 2.5.4).

Properties relating the bounds of an interval, the center, the radius and operations on intervals usually simplify to straightforward properties of real numbers. Such proofs can often be discarded by automatic procedures, like `ring` [39] or `field` [27] for dealing with equalities and `fourier` for dealing with inequalities over the real numbers in Coq.

2.5.4 Interval matrices

An interval $m \times n$ matrix is a $m \times n$ matrix with interval elements

$$A = [A_{ij}]_{m\times n},\ A_{ij} \in \mathbb{IR}.$$

Interval vectors are not treated separately: a vector is a special kind of matrix. We have column vectors, they are therefore $n \times 1$ matrices.
An interval matrix is interpreted as a set of real matrices by the convention

$$A = \{\tilde{A} \in M(\mathbb{R})_{m\times n} \mid \tilde{A}_{ij} \in A_{ij}, i = 1, \ldots, m, j = 1, \ldots, n\}.$$

The concepts we described for intervals generalize to interval matrices, usually componentwise. This allows us to relate certain real matrices to each interval matrix: the lower and upper bound matrix, the midpoint matrix and the radius matrix.

$$\underline{A} = \inf(A) := [\underline{A}_{ij}] \qquad \overline{A} = \sup(A) := [\overline{A}_{ij}]$$

$$A_c = \mathrm{mid}(A) := [\mathrm{mid}(A_{ij})] \qquad \Delta_A = \mathrm{rad}(A) := [\mathrm{rad}(A_{ij})]$$

An example of Coq definition:

```
Definition minf (A : 'M[IR]_(m, n)) := \matrix_(i, j) inf (A i j).
```

Operations on matrices are defined in the same way as operations on intervals. Here the $\square$ denotes the smallest interval matrix that contains the set.

$$A + B := \square\{\tilde{A} + \tilde{B} \mid \tilde{A} \in A, \tilde{B} \in B\} = \{\tilde{A} + \tilde{B} \mid \tilde{A} \in A, \tilde{B} \in B\}$$

$$-A := \square\{-\tilde{A} \mid \tilde{A} \in A\} = \{-\tilde{A} \mid \tilde{A} \in A\}$$

$$AB := \square\{\tilde{A}\tilde{B} \mid \tilde{A} \in A, \tilde{B} \in B\} \neq \{\tilde{A}\tilde{B} \mid \tilde{A} \in A, \tilde{B} \in B\} \tag{2.12}$$

We stress that for multiplication the set of matrix products $\{\tilde{A}\tilde{B} \mid \tilde{A} \in A, \tilde{B} \in B\}$ need not be an interval matrix (in the above definition, the last two sets need not be equal). We illustrate this inequality by an example.

Example 1. $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $x = \begin{pmatrix} [-1, 0] \\ [1, 2] \end{pmatrix}$ then we have $\begin{pmatrix} 0 \\ 2 \end{pmatrix} \in Ax = \begin{pmatrix} [0, 2] \\ [1, 2] \end{pmatrix}$

but $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$ not of the form $\tilde{A}\tilde{x}$ with $\tilde{A} \in A, \tilde{x} \in x$ because A is a thin matrix, therefore $\forall \tilde{A} \in A, \tilde{A} = A$ and solving $A\tilde{x} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$ gives $\tilde{x} = \begin{pmatrix} 2 \\ -2 \end{pmatrix} \notin x$.

We note that operations on interval matrices can be alternatively defined by using the operations on intervals.

$$(A+B)_{ij} = A_{ij} + B_{ij} \tag{2.13}$$

$$(-A)_{ij} = -A_{ij} \tag{2.14}$$

$$(AB)_{ij} = \sum_k A_{ik}B_{kj} \tag{2.15}$$

In our implementation we used of course the SSReflect library on matrices. We recall that there is no constraint on the type of elements in order to define a matrix, but to use the generic operations from the SSReflect library we need to have a ring structure on the type. We saw in the previous section that real intervals with addition and multiplication do not form a ring. So we need to define specific operations for interval matrices. We chose to implement operations by using the definitions 2.13 - 2.15.

.... Technical Details 10.

`Implementation:  operations on interval matrices`

We give the example of multiplication, to illustrate work with indexed operations in SSReflect.

```
Definition mmulI (A : 'M[IR]_(m, n)) (B : 'M[IR]_(n, p)) :=
  \matrix_(i, j) \big[addI / 0 ]_k mulI (A i k) (B k j).
```

This definition translates exactly the relation 2.15: \big announces a indexed operation with the operator addI, the addition of intervals. The expression to be added is mulI, the interval multiplication of A i k and B k j, the respective terms of matrices A and B.

Here the fact that interval addition has a commutative monoid structure comes in handy, as we can use straightforwardly many theorems on indexed operations in order to get the desired properties for the matrix multiplication.

.. End technical details.

We establish all the necessary properties for our interval matrix operations. For example, similar to the characterization for addition of two intervals 2.10, we show the characterization for the multiplication of an interval matrix by a real vector.

$$A\tilde{x} = \{\tilde{A}\tilde{x} \mid \tilde{A} \in A\} \tag{2.16}$$

Here, the same issues arise as for the proof of relation 2.10. We note that this result is not true in general, for the multiplication of an interval matrix by an interval vector, as shown in example 1.

We also treat special properties for square matrices. The interval matrix A is called regular if each scalar matrix $\tilde{A} \in A$ is nonsingular (which means $\det \tilde{A} \neq 0$), and it is said to be singular otherwise.

```
Definition regular (A : 'M[IR]_n) := forall tA, inSetm A tA → \det tA <> 0.
Definition singular (A : 'M[IR]_n) := exists tA, inSetm A tA ∧ \det tA = 0.
```

2.5.5 Related formalizations

Several formalization for interval arithmetic are already available in proof assistants. We cite [26] as a formalization in PVS, [60] as one of the existing formalizations in Coq and [49] as a Isabelle/HOL formalization. All these formalizations are concerned with using interval analysis for doing formally correct and accurate computations. They usually cover:

- basic operations on interval
- interval enclosures for elementary functions
- techniques to increase accuracy in computation
- rounded interval arithmetic
- automated procedures to compute and prove bounds for expressions
- links between the formalization and external tools for interval computations

The formalization [77] in Coq uses interval arithmetic and Taylor models to do global optimization of a given expression and in a given interval.

Our main purpose however is to formalize more theoretical results from interval analysis, in particular methods for solving systems of equations. We will cover this topic in detail in chapter 4. In order to treat such methods, we need an implementation of intervals that ensures the necessary properties, for example relations 2.10 and 2.16. Such equalities only hold in an ideal arithmetic, so implementing rounded arithmetic was not an option.

2.6 Conclusion

We presented in this chapter the formalization for elements of mathematical theories needed in the study of numerical methods. Though we do not provide complete libraries that cover all the aspects of the theories, we have sufficient results for the studies we conducted on Newton's method (in chapter 3) and on systems of linear interval equations (in chapter 4). More precisely we provided complements to the matrix library in SSReflect. These complements treat specific notions on real matrices like norms and eigenvalues. The formalization on eigenvalues is not completed yet. We also provided a basic library of interval analysis, containing operations on interval and interval matrices. We formalized multivariate analysis concepts covering norms of vectors, functions and sequences of vectors and notions related to them in metric spaces: limit, convergence, continuity, derivability.

The formalized theories are reusable in other formal studies and easily extensible with new notions.

In the formalization of mathematical theories we have as important tasks: choosing a formalization of the basic concepts that is adapted to the infrastructure, and providing the technical lemmas on the basic concepts that make them easy to use in high-level proofs.

We tried to present our formalizations of mathematical theories by separating the more general issues from the Coq-specific ones (which we isolated in technical sections). However, when describing a mathematical theory in a proof assistant it is essential to take into account the technical features of the system. A good use of these features will ease the task of formalization of proofs. In most cases the "good way" to formalize some concept will be the same in different proof assistants, but the technical details of their formalization will differ. For example, (real) vectors are formalized in both Coq and HOL Light as functions from a finite domain of size n to $\mathbb{R}$. But Coq has dependent types while HOL Light does not. So, in Coq the finite domain is formalized as the dependent type of natural numbers smaller then n, while in HOL Light the finite domain is implemented as a type with the right cardinal, n. The use of other Coq mechanism like coercions and Canonical Structures also eased our work. Versions of such mechanisms exist in other proof assistants.

Once we choose for a concept a formalization that both corresponds to the mathematical object and takes advantage of the features of the proof assistant, we still need to provide a bunch of *technical lemmas*. These technical lemmas fall in two categories.

The first category contains results in relation to the specific features of the proof assistant and not related to the mathematical proofs. Examples of such lemmas are: the equivalence of the two implementations for sums (see technical section 2) and the fact that equality of two intervals is equivalent to equality of their bounds (see section 2.5.3). The second category contains results that are usually omitted in a mathematical proof as they are considered obvious, but that need to be made explicit in a proof assistant. An example of such proof is equality 2.10 for the addition of two intervals. Having the appropriate technical lemmas is very important for the successful formalization of high-level proofs.

Though not the original purpose of our work, providing formalizations for background theories used in the verification of numerical methods took a large amount of time in the whole formalization process. The files corresponding to these formalization represent more than half of the entire development presented in this manuscript.

The results presented in this chapter were published in [9; 66; 67; 68]. The Coq *development is available on-line at:*
http://www-sop.inria.fr/marelle/Ioana.Pasca/phd

Chapter 3

Solving Equations and Systems of Equations with Newton's Method

Some uses and versions of Newton's method

Newton's method is commonly used to determine the root of a given function when we have an initial approximation of this root. We recall the definition of this method in the case of a function $f : A \subseteq \mathbb{R} \to \mathbb{R}$ and initial approximation x_0

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.1}$$

This root finding process can be generalized to a function $f : A \subseteq \mathbb{R}^p \to \mathbb{R}^p$ with initial approximation X_0

$$X_{n+1} = X_n - J_f(X_n)^{-1} f(X_n) \tag{3.2}$$

where J_f denotes the jacobian matrix (see section 1.3, definition 1.2). In fact there are variants of Newton's method suited for different contexts. For example, we can generalize Newton's method for finding the roots of a complex function:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

or of a function defined in a Banach space:

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

where f'_{X_n} is the Fréchet derivative applied a the point X_n.

We can also generalize the derivative we use, that is we can use the higher order derivatives instead of just the first derivative. For a function derivable $d + 1$ times we have Householder's method:

$$x_{n+1} = x_n + d\frac{(1/f)^{(d-1)}(x_n)}{(1/f)^{(d)}(x_n)}$$

For $d = 1$ we get Newton's method and for $d = 2$ we get Halley's method:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2(f'(x_n))^2 - f(x_n)f''(x_n)}$$

We can also avoid the use of the derivative and approximate it by finite differences. This yields the secant method For a function $f : A \subseteq \mathbb{R} \to \mathbb{R}$ two initial approximations x_0 and x_1 we get

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

Properties of Newton's method

In what follows we will only study Newton's method for determining the root of a function in one or several real variables, in other terms, we will study the sequences 3.1 and 3.2 above.

The main reasons for the wide usage of Newton's method are the fast convergence of the method and the local stability with respect to the initial approximation. Under good conditions Newton's method has quadratic convergence. In practice this means the number of precise decimal digits doubles at each iteration. The local stability means that there is not just one "good" initial approximation, but there is an entire neighborhood of initial approximation that will yield a convergent sequence. However, we saw in the examples of the introductory section 1.3.1, that in the absence of some "good" conditions, Newton's method will not converge.

There are several studies that provide sufficient conditions for the convergence of Newton's method to the root of the given function. We use the results of convergence presented in [29] and attributed to Kantorovitch. In [29] we also have proofs for the speed of convergence, for the local stability of newton's method, as well as for the unicity of the root of the function in a certain neighborhood of the initial approximation.

In the following section we will study these results in the case of a real function with one variable.

3.1 Proofs for properties of Newton's method

3.1.1 Statements

We can consider Newton's method for a real function with one variable as a simplified version of the method in several variables. We give the statement of the theorems and the idea of the proofs in this simpler version. The structure of the proofs is the same in one or several dimensions. We chose to discuss this simplified case because the concepts are easier to understand by the reader. This simplified model of the problem 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. We make the appropriate comments for the generalization of the proof to several dimensions in section 3.1.3.

Theorem 3 (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 $\overline{U_\varepsilon}(x_0) = \{x : |x - x_0| \leq \varepsilon\} \subset (a, b)$. If the following conditions hold:*

1. $f'(x_0) \neq 0$ *and* $\left|\dfrac{1}{f'(x_0)}\right| \leq A_0$*;*

2. $\left|\dfrac{f(x_0)}{f'(x_0)}\right| \leq B_0 \leq \dfrac{\varepsilon}{2}$*;*

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

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1, 2, \dots \tag{3.3}$$

converges and $\lim\limits_{n\to\infty} x_n = x^*$ is a solution of the initial system, so that

$$|x^* - x_0| \leq 2B_0 \leq \varepsilon.$$

The theorem of convergence of Newton's method shows that this method is indeed appropriate for determining the root of the function. The sufficient conditions for convergence say that there has to be some sort of balance between: the value of the derivative, the adjustment we make at each iteration in Newton's method, and the possible variation of the value of the derivative.

We give the statement here with a slight generalization in comparison to the statement in [29]. Hypothesis 3 is weaker in the reference, as it states

$$3'.\ \ \forall x \in (a, b), |f''(x)| \leq C. \tag{3.4}$$

Kantorovitch' theorem is valid under the stronger hypothesis we give, but the reference [29] chooses to give it under the weaker form in order to have a better manipulation of the constants involved. Since in one dimension, the proof involves only minor changes, we give the stronger version.

Other properties of newton's method are given by the following theorems.

Theorem 4 (Uniqueness)**.** *Under the conditions of Theorem 3 the root x^* of the function f is unique in the interval $[x_0 - 2B_0, x_0 + 2B_0]$.*

The unicity of the solution in a certain domain is used in practice for isolating the roots of the function.

Theorem 5 (Speed of convergence)**.** *Under the conditions of Theorem 3 the speed of the convergence of Newton's method is given by*

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

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 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.

Theorem 6 (Local stability)**.** *If the conditions of Theorem 3 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 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 [29]. The outline of the proof for theorem 3 as presented in [29] 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.

We notice that for the initial approximation x_0 we have the constants A_0, B_0, μ_0 that are associated to this approximation and that characterize the behavior of the function f in x_0. The proof of the theorem introduces such constants associated to every element of Newton's sequence. These associated constants at rank n are given by

$$A_n = 2A_{n-1} \tag{3.5}$$

$$B_n = A_{n-1}B_{n-1}^2C = \frac{1}{2}\mu_{n-1}B_{n-1} \tag{3.6}$$

$$\mu_n := 2A_nB_nC = \mu_{n-1}^2 \tag{3.7}$$

With the help of these associated constants we are able to characterize the behavior of the function f at each approximation x_n given by Newton's method. Reasoning by induction we manage to get for each x_n properties that are analogous to those of x_0 More precisely, we get:

$$f'(x_n) \neq 0 \text{ and } \left|\frac{1}{f'(x_n)}\right| \leq A_n \text{ (analogous to hypothesis 1)} \tag{3.8}$$

$$\left|\frac{f(x_n)}{f'(x_n)}\right| \leq B_n \leq \frac{\varepsilon}{2^{n+1}} \text{ (analogous to hypothesis 2)} \tag{3.9}$$

$$\mu_n \leq 1 \text{ (analogous to hypothesis 4)} \tag{3.10}$$

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

These imply

$$\overline{U}_\varepsilon(x_0) \supset \overline{U}_{\frac{\varepsilon}{2}}(x_1) \supset \ldots \supset \overline{U}_{\frac{\varepsilon}{2^n}}(x_n) \supset \ldots \tag{3.11}$$

From (3.11) 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 arbitrarily 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 (3.3) 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 6 (local stability) we prove that the new initial approximation x_0' satisfies similar hypotheses as those for x_0. The constants associated to x_0' are $A' = \dfrac{4}{3+\mu_0}A_0$ and $B' = \dfrac{3+\mu_0}{4\mu_0}B_0$. This entails $\mu' = 2A'B'C = 1$ 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 3 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 (3.7)) and we would not be able to do an approximation again, because Theorem 6 requires $\mu'' < 1$. To correct this, we impose a finer approximation $|x_0 - x_0'| \leq \dfrac{(1-\mu_0)}{4\mu_0}B_0$. This new approximation yields the following formulas for the associated constants:

$$A' = \frac{8}{7+\mu_0}A_0 \tag{3.12}$$

$$B' = \frac{\mu_0^2 + 46\mu_0 + 17}{8(7+\mu_0)\mu_0}B_0 \tag{3.13}$$

this implies

$$\mu' = \frac{\mu_0^2 + 46\mu_0 + 17}{(7+\mu_0)^2} < 1 \tag{3.14}$$

We summarize these results in:

Corollary 1. *If the conditions of Theorem 3 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* $|x_0' - x_0| \leq \dfrac{1-\mu_0}{4\mu_0}B_0$ *the associated Newton's process converges to the root* x^*.

3.1.2 Formalization issues

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 $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$, 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. 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.

Paper proofs vs. formal proofs. The formal proofs basically consist in 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 the ones presented in section 3.2. For the statements of the theorems in Coq we refer the reader to appendix B.1.

3.1.3 Moving to several dimensions

The formalization of multivariate analysis concepts presented in section 2.4, chapter 2 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. The formalization done for the real case is a good guide in our development as the hypothesis and lemmas needed for the multidimensional case are 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

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

generalizes to

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

where E_p is the identity square matrix of size p.
This adaptation is sometimes far from trivial as we saw for the latter example which has been detailed in section 2.3 of chapter 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 2.3, 2.4 of chapter 2. The statement of the theorem is generalized for a function

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

This means f is of the form

$$f(x) = (f_1(x), f_2(x), \ldots, f_p(x))$$

Starting for an initial approximation $x_0 \in \mathbb{R}^p$, we search the root of the function f. Finding the root is equivalent to solving the following system of equations:

$$\begin{cases} f_1(x) = 0 \\ f_2(x) = 0 \\ \ldots \\ f_p(x) = 0 \end{cases}$$

The notion of derivative is generalized to the notion of jacobian matrix (see section 2.4.3, chapter 2).
The notion of C^2 in $\mathbb{R}^p$ means twice partially derivable, with continuous second order partial derivatives.
Newton's method becomes

$$x_{n+1} = x_n - J_f(x_n)^{-1} f(x_n)$$

The hypotheses of theorem 3 are then generalized as follows:

1. the jacobian matrix $J_f(x) = \left[\dfrac{\partial f_i(x)}{\partial x_j}\right]$ has an inverse for $x = x_0$, $\Gamma_0 = J_f^{-1}(x_0)$ with $\|\Gamma_0\| \leq A_0$;
2. $\|\Gamma_0 f(x_0)\| \leq B_0 \leq \frac{\varepsilon}{2}$;
3. $\displaystyle\sum_{k=1}^{p} \left|\frac{\partial^2 f_i(x)}{\partial x_j \partial x_k}\right| \leq C$ 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 $\mu_0 = 2pA_0B_0C \leq 1$

The proof of Kantorovitch's theorem in several dimensions has the same structure as the proof in one dimension. We associate the same sequences to Newton's sequence

$$A_n = 2A_{n-1}$$
$$B_n = \frac{1}{2}\mu_{n-1}B_{n-1}$$
$$\mu_n = \mu_{n-1}^2$$

We use them in the same way as in the proof in one dimension in order to characterize each element of Newton's sequence and deduce the desired properties.

The complete statement and proof of Kantorovitch's theorem and related results are available in appendix A. The formal statements are available in appendix B.3.

3.2 Newton's method with rounding

In our description of Newton's method up till 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:

$$t_0 = \operatorname{rnd}_0(x_0)$$
$$t_{n+1} = \operatorname{rnd}_{n+1}(t_n - \frac{f(t_n)}{f'(t_n)})$$

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 3 - 6 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 7 (Convergence with rounding)**.** *We consider a function $f : (a, b) \to \mathbb{R}$ and an initial approximation x_0 satisfying the conditions in theorem 3.*
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:

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

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 $\lim_{k\to\infty} t_k = x^*$ where x^* is the root of the function f given by theorem 3*

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 3 - 6 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 $t_1 := \text{rnd}(x_1)$

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 3. This means that Newton's method from this initial point converges to the root x^* (see theorem 3).
- 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 3) we can associate the constants A_1, B_1 to the initial iteration of this sequence and get the corresponding hypotheses from theorem 3.
- 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 3. 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 3 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

$Y_{k+1}^0 = \text{rnd}_{k+1}(Y_k^1)$;

$Y_{k+1}^{n+1} = Y_{k+1}^{n} - \frac{f(Y_{k+1}^{n})}{f'(Y_{k+1}^{n})}$.

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 3 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 3 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 3 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 3.5, 3.6 and 3.7 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 $Y_1^0 = \text{rnd}_1(\overline{Y}_0^0)$. 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:

$$0 < \overline{\mu}_0 < 1$$

$$\left[\overline{Y}_0^0 - \frac{2}{\overline{\mu}_0}\overline{B}_0 \ , \ \overline{Y}_0^0 + \frac{2}{\overline{\mu}_0}\overline{B}_0\right] \subset (a,b)$$

$$\left|\text{rnd}_1(\overline{Y}_0^0) - \overline{Y}_0^0\right| \leq \frac{1-\overline{\mu}_0}{4\overline{\mu}_0}\overline{B}_0$$

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 (3.12) and (3.13) in section 3.1

$$A'_1 = \frac{8}{7+\overline{\mu}_0}\overline{A}_0 = \frac{8}{7+\overline{\mu}_0}(2A'_0)$$

$$B'_1 = \frac{\overline{\mu}_0{}^2 + 46\overline{\mu}_0 + 17}{8(7+\overline{\mu}_0)\overline{\mu}_0}\overline{B}_0 = \frac{\overline{\mu}_0{}^2 + 46\overline{\mu}_0 + 17}{8(7+\overline{\mu}_0)\overline{\mu}_0}(A'_0 B'_0{}^2 C)$$

$$\mu'_1 = 2A'_1 B'_1 C = \frac{\overline{\mu}_0^2 + 46\overline{\mu}_0 + 17}{(7+\overline{\mu}_0)^2} = \frac{\mu'^{2\,2}_0 + 46\mu'^2_0 + 17}{(7+\mu'^2_0)^2}$$

We find ourselves again in the conditions of theorem 3 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:

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

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

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

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:

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

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.

□

3.3 Newton's method and exact real computations

Since Newton's method is a numerical method, we would like to be able compute with this method. In our formalization so far we used the real numbers from Coq's standard library. As explained in section 2.2, these real numbers are defined by axioms. This definition has two consequences. The first is that the real numbers have all the desired theoretical properties, which makes theorem proving with these numbers more agreeable and close to "pencil and paper" proofs [59]. The second consequence, however, is that the standard library real numbers have no (or little) computational meaning. For example, addition is an operator is an operator with the good properties (commutative, associative) given by axioms. There is no actual algorithm behind addition, so we cannot add two real numbers and compute the result. The closest thing we can do in some cases is assert that the result is a certain value and prove this is correct by using the properties of real numbers.

We note that Coq is not a special case and proof assistants in general provide libraries with results from real analysis [23; 30; 31; 43], but with formalizations for real numbers that are not well suited for computations.

To fulfill this need of computing with real numbers, a considerable effort has been invested in implementing libraries for exact real computations inside proof assistants [52; 56; 65]. These libraries provide verified computations for a set of operations and elementary functions on real numbers. We shall refer to numbers from such implementations as "exact reals", as opposed to the "theoretical" real numbers which we shall call "axiomatic reals". The recipe for all these exact real libraries is to have an implementation of real numbers that is on one hand suitable for computation and on the other hand easy to validate by linking it to the reference implementation of the proof assistant. Very little proofs are actually performed on the exact reals, most high level proofs are done on the axiomatic real and "inherited" to the exact reals thanks to the relation between the two types of real numbers.

Concerning Newton's method, we already established, in the previous sections, a number of properties by using the axiomatic reals of Coq. We would like to transfer these "theoretical" properties to the computations done with exact reals. In what follows we present a Coq library of exact real arithmetic and show how Newton's method can be adapted to this setting and how the proofs previously formalized for Newton's method can be used to verify computations.

3.3.1 A Coq library for exact real arithmetic

We present in this section the exact real arithmetic library developed in Coq by N. Julien and described in [52]. In this library a real number in the interval $[-1, 1]$ is represented by its infinite sequence of digits. This corresponds to the intuitive way of thinking about real numbers, for example $\frac{1}{3}$ is written as $0.33333\ldots$, as we are used to considering positive digits in base 10 . However, the Coq implementation allows us to consider an arbitrary base β and signed digits, that is both negative and positive digits. 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 an infinite sequence of digits and for which we know the first digit can be written as:

$$r = [\![s_1 :: s]\!]_\beta = \frac{s_1 + [\![s]\!]_\beta}{\beta} \tag{3.15}$$

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: $\left[\frac{k-1}{\beta}, \frac{k+1}{\beta}\right]$. 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 is sufficient to know two digits of the input to decide the first digit of the output. 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. Also, since our real numbers are infinite sequences, the algorithms need to be designed in such a way that we are always able to provide an extra digit of the result. In Coq we have a way of defining potentially infinite data types, called co-inductive types and a way of writing functions that output such infinite data, in a lazy manner, called co-recursive functions. The details of these features of Coq can be found in [34]. The following technical section provides a brief overview of the Coq features and of the implementation of real numbers as infinite sequences of digits.

.... Technical Details 11.

`Implementation: real numbers as infinite sequences of digits`
`Coq: co-inductive types and co-recursive functions`

In Coq we can define potentially infinite data types by using the keyword CoInductive. This way, for each type A we can define the type of infinite sequences of elements of type A. Such an infinite sequence is formed from an element of type A and an infinite sequence. We call this data type stream and we define it as follows.

```
CoInductive stream (A : Type) : Type :=
  | 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.

With this definition, real numbers are implemented as streams of signed digits of a certain base β. A digit is just an integer and the implementation of digits provides an interface that allows the user to chose both the base and the implementation of integers. Here, we will denote their type by digit.

A new term of a co-inductive type can be defined by using a co-recursive function. As for recursive function, there are certain restriction that need to be respected when we write a co-recursive function. For a recursive function one of the arguments had to be an inductive type and no restrictions were imposed on the return type. For a co-recursive function, there are no restrictions imposed on the arguments, but the return type has to be a co-inductive type. A recursive call can be made on a structurally smaller term, while a co-recursive call can only be made under a constructor of the co-inductive type. Also, at each step, the co-recursive function has to produce some finite piece of data.

Let's look at an example. We can define the stream of all zeros, by using a co-recursive function.

```
CoFixpoint zero : stream digit := Cons 0 zero.
```

The function zero produces a stream of digits, which is a co-inductive type. The co-recursive call to the function zero is made under the constructor Cons and at each step the function produces a 0. Thus, the function is correctly defined as it respects all the constraints. Moreover, we notice that the function has no arguments.

Since co-recursive functions produce potentially infinite data, they enjoy lazy evaluation. In the case of streams of digits, a co-recursive function computes only the digits demanded by the user, one by one.

.... **End technical details.**

Once we defined the exact reals as infinite sequences of signed digits, we need to show that they behave as real numbers. We do this by linking their behavior to the behavior of the axiomatic real numbers from Coq's standard library. The relation between the two types of real numbers is called represents and it is defined according to relation 3.15: if an infinite sequence s of signed digits in base β represents the axiomatic real $r \in [-1, 1]$, and if k is a signed digit of the base β, then the infinite sequence starting with k and continuing with s represents the axiomatic real $\dfrac{k+r}{\beta} \in [-1, 1]$.

Using this link between the two kinds of reals, we show that: the infinite sequence of zeros represents the axiomatic real 0, the multiplication defined on infinite sequences represents the multiplication defined on axiomatic reals, and so on. This guarantees that the computations with exact real numbers respect the properties of axiomatic real numbers. We give in the following technical section more details on the validation of the exact real arithmetic library.

.... **Technical Details 12.**

```
Implementation:  validation of exact real computations
Coq:  proofs by co-induction
```

We formalize the represents relation between exact reals and axiomatic reals by using a co-inductive definition.

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

The statement that the multiplication is correct is formulated in the following manner:

Theorem mult_correct :
  forall x y vx vy, represents x vx $\rightarrow$ represents y vy $\rightarrow$ represents (x $\times$ y) (vx $*$ vy).

It 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 then the multiplication of streams x and y (here denoted $\times$) will represent the multiplication of axiomatic reals vx and vy (here denoted $*$).

We note that when we want to prove such a statement our goal is a co-inductive type. The proof will be a term of the co-inductive type. We saw in technical section 11 that in order to build a term of a co-inductive type we need a co-recursive function. There is a Coq tactic called cofix that helps build the co-inductive term by decomposing the task of writting the co-recursive function.

.... **End technical details.**

3.3.2 Correctness of Newton's method

We want to prove correctness of computation with Newton's method on exact reals. We do this by using the relation represents between the exact reals and axiomatic reals. We already implemented Newton's on axiomatic reals.

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.

We implement the method for exact 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 $\ominus$ g exn end.

Now we can write a represents relation between the elements of the same rank in the two sequences.

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

Showing that the relation holds is not complicated. It requires as hypotheses a represents relation between the initial approximations of the two versions of Newton's method as well as a represents relation between the function g and the function f / f'. The proof then follows from the correction of the subtraction on streams with respect to the subtraction on axiomatic reals.

The represents relation between the two versions of Newton's method 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 3 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 5 (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 co-recursive functions on infinite sequences of digits: 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.

3.3.3 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 the function make_digit available in the library of exact real arithmetic described in 3.3.1. The function make_digit requires knowing a number with a precision δ in order to produce the appropriate first digit. This precision δ is equal to $\frac{\beta-2}{2\beta^2}$ due to technical reasons explained in [52]. The important property of make_digit is: if we use make_digit to produce the first digit of a number r_1, then any number r_2 in a radius of δ from r_1 will admit a representation starting with the same first digit. This is possible because of the redundant representation due to the use signed digits and because of the choice of δ.

We use this property to obtain the first digit of the root x^* of f. We first determine the number of Newton iterations n need to obtain a precision δ for the root. This means we determine $n \in \mathbb{N}$ such that $|x_n - x^*| \leq \delta$ by using Theorem 5 (speed of convergence). By the properties of function make_digit, we can 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

$$f_1(x) := f\left(\frac{d_1+x}{\beta}\right),$$

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

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

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 ex0 g n)) with
   | d1 :: xs1 ⇒ d1 :: exact_newton (fun x ⇒ (β ⊙ g (d1 :: x))) xs1 n
  end.
```

We note that $\beta\odot$ is a function provided by the library described in section 3.3 that computes efficiently the multiplication of a stream by the base. It is possible to determine a value for n that is sufficient to allow the production of a digit at each co-recursive call. This simplifies the algorithm but reduces the quadratic convergence to linear convergence.

The formal verification 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 3 - 5 (see section 3.1) on axiomatic reals and the theorem that links Newton's method on exact reals to Newton's method on streams (section 3.3.2). We need to show that if the initial function f satisfies the hypotheses of Theorem 3, 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 3 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 $\left|\dfrac{1}{f'(x_0)}\right| \leq A_0$;
4. $\left|\dfrac{f(x_0)}{f'(x_0)}\right| \leq B_0 \leq \dfrac{\varepsilon}{2}$;
5. $\mu_0 = 2A_0B_0C \leq 1$.

We analyze $f_1(x) := f\left(\dfrac{d_1+x}{\beta}\right)$ for which we have $f_1'(x) = \dfrac{1}{\beta}f'\left(\dfrac{d_1+x}{\beta}\right)$ 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. $|f_1'(x) - f_1'(y)| = \left|\dfrac{1}{\beta}f'\left(\dfrac{d_1+x}{\beta}\right) - \dfrac{1}{\beta}f'\left(\dfrac{d_1+y}{\beta}\right)\right| \leq$
   $\leq \dfrac{1}{\beta}C\left|\dfrac{d_1+x}{\beta} - \dfrac{d_1+y}{\beta}\right| = \dfrac{1}{\beta^2}C|x-y|$
3. $f_1'(\overline{x}_n) = f'(x_n) \neq 0$ and $\left|\dfrac{1}{f_1'(\overline{x}_n)}\right| = \left|\beta\dfrac{1}{f'(x_n)}\right| \leq \beta A_n$;
4. $\left|\dfrac{f_1(\overline{x}_n)}{f_1'(\overline{x}_n)}\right| = \left|\beta\dfrac{f(x_n)}{f'(x_n)}\right| \leq \beta B_n$;
5. $\overline{\mu}_n = 2\beta A_n \beta B_n \dfrac{1}{\beta^2}C = 2A_nB_nC \leq 1$.

Relations 3. - 5. are given by the proof of Theorem 3. We are now able to prove that represents (exact_newton g ex0 n)x^*.

3.3.4 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 behavior 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 $f_{sqrt}(x) = \beta^2x^2 - a$ 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. 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 : forall (a : stream digit) (va : R),
      represents a va → represents (Ssqrt a) ((sqrt va)/β).
```

3.4 Related work

Two other libraries of formally verified exact real arithmetic are available: one in PVS, described in [56] and one in Coq described in [65]. The computations in these two libraries are verified with respect to the reference implementation of real numbers in PVS and C-CoRN, respectively. We also mention the work of in [63] concerned both with exact real arithmetic and with co-inductive aspects This works aims to obtain all field operations on real numbers via the Edalat-Potts algorithm for lazy exact arithmetic.

Newton's method has been studied on a representation of exact real numbers in [50], but not in a formally verified setting.

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 [17]. Of course, in these cases the speed of convergence is better than in the general case.

3.5 Conclusion and future work

We presented in this chapter a formal study for Newton's method. This study contains the formalization of well known theoretical properties of Newton's method: convergence, speed of convergence, unicity of the root in a certain domain, local stability. These properties are established for Newton's method both in one dimension and in several dimensions. The formal proofs are done on the axiomatic representation of real numbers from Coq's standard library. For Newton's method in one dimension we push the study further and provide a new proof (based on the above properties) that ensures convergence of Newton's method when we introduce rounding in the computation. This original result is presented in Theorem 7. We also study Newton's method in the context of certified exact real number computations in a joint work with Nicolas Julien.

The proof of theorem 7 has an interest from a proof engineering point of view. We were able to come up with the proof because we had formalized theorems 3 - 6 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 apart from the proof, allowing the user to better understand their importance and use similar sequences in a 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.

Theorem 7 ensures convergence in the presence of rounding and the rounding performed is modeled by a generic function. This means the theorem can be applied in different circumstances, for different ways of rounding a real number: using one of the rounding modes accepted by floating point standards, rounding to a given number of digits, rounding to a certain rational number and so on. Thus, theorem 7 opens the way to the formal study of Newton's method on floating point numbers. This should be achievable in the short term as there are formalizations of floating point arithmetic in Coq, like the work in [13].

The study of Newton's method on exact real numbers is done in Coq by using the implementation of exact reals described in [52]. But there is another library of exact real arithmetic in Coq described in [65]. The implementation of exact reals in this latter library is validated by an isomorphism between the exact reals and the "theoretical" reals in the C-CoRN library (see section 2.1.1 of chapter 2 and paper [25]). The C-CoRN reals are linked to the axiomatic reals by another isomorphism (see [54]). We should be able to validate computation with Newton's method done in the exact real library presented in [65] by using the two isomorphisms and the proofs done on the axiomatic reals and presented in section 3.1.

The algorithm presented in section 3.3.3 is not efficient. An optimized version of this algorithm will be included in the library [52] by Nicolas Julien. This version makes two optimizations to the basic algorithm: it takes into account rounding to a given number of digits which is possible thanks to theorem 7 and it builds the function under the co-recursive call in a more efficient manner. The proof of theorem 7 and algorithm 3.3.3 serve as a basis for these optimizations.

The results presented in this chapter were published in [9; 53; 66; 67]. The Coq *development is available on-line at:*
http://www-sop.inria.fr/marelle/Ioana.Pasca/phd

Chapter 4

Regularity of Interval Matrices

We recall from chapter 2, section 2.5 that an interval matrix $A = (A_{ij})$ is a matrix whose coefficients are intervals. Such a matrix ca be seen as a set of real matrices. A real matrix $\tilde{A}$ belongs to an interval matrix A, if for all indexes i, j, the coefficient $\tilde{A}_{ij}$ in the real matrix belongs to the interval A_{ij}. A square interval matrix A is called regular if for all real matrices $\tilde{A}$ belonging A the determinant $\det \tilde{A}$ is not zero.

Establishing whether an interval matrix is regular is an important issue that intervenes in several kinds of problems. We already saw in chapter 3 that in order to establish the convergence of Newton's method a necessary conditions is to start in a point where the jacobian matrix is invertible, or otherwise put, the determinant of this matrix is not zero. In real world applications we usually do not have an exact matrix, so we can replace it by an interval matrix that is sure to contain the real matrix. If we establish that this interval matrix is regular, then we can conclude that our jacobian matrix has non null determinant.

Another example where the regularity of interval matrices appears is the study of systems of linear interval equations. The problem of solving a system of linear interval equations can be formulated as follows: assume that the coefficients and right-hand side of a system of n linear equations in n variables are not determined exactly, but are only known to lie in some real intervals (this could be due to data errors, rounding errors, method errors). Such a system of linear interval equations represents a family of ordinary linear systems obtained by fixing values for the coefficients and the right-hand side in the corresponding intervals. Each of these ordinary systems has a unique solution provided that the real matrix associated to it is regular. These solutions constitute the solution set of the initial system of linear interval equations. Here is an example of such system where $n = 2$.

Example 2.

$$\begin{cases} [2,4]x \; + \; [-1,1]y \; = [-3,3] \\ [-1,1]x \; + \; [2,4]y \; = [0,0] \end{cases} \tag{4.1}$$

There are two main issues involved in solving these systems:

1. checking regularity of the interval matrix associated to the system

2. computing an enclosure of the solution set, as this set cannot usually be determined exactly.

The second step will only be performed if the interval matrix is indeed regular, which means that the solution set is bounded. With our study of regularity of interval matrices in this chapter, we cover the first of the two steps.

A formal study of methods for solving systems of linear interval equations in a proof assistant will guarantee the correctness of these methods. This is desirable because systems of linear interval equations appear frequently in safety-critical software. The reference [71] on which our study is based, was pointed out to us by researchers interested in robotics [61]. They use the results in [71] to establish that a certain interval matrix is regular and then solve the associated system to get the set of valid coordinates for the next position of the robot. This particular robot is designed for providing help in crisis situations by lifting a stretcher with an injured person from an accident scene. This is one example of many such safety critical applications that motivate our formal study of interval analysis.

There are a bunch of criteria for testing regularity of interval matrices. Forty of them are listed in a recent paper [73]. Among them there are criteria of theoretical interest only (which we shall alternatively call basic criteria) and efficient criteria used in practice. The basic criteria are usually (but not always) easier to understand and to prove correct. They are often not of practical interest but they are used as a basis to obtain more efficient criteria. It is the case of the sufficient conditions for regularity and singularity of interval matrices discussed in [71].

We decided to treat regularity of interval matrices from the point of view of solving systems of linear interval equations, so we start by giving some basic notions, such as the solutions set of a system of linear interval equations.

4.1 The solution set of a system of linear interval equations

Let us give a more formal description of the concepts mentioned above. A system of linear interval equations with coefficient matrix $A \in M(\mathbb{IR})_{m\times n}$ and right-hand side $b \in \mathbb{IR}^m$ is defined as the family of linear systems of equations

$$\tilde{A}\tilde{x} = \tilde{b} \text{ with } \tilde{A} \in A, \tilde{b} \in b$$

Example 3. In the example we took in the introductory paragraph, system 4.1, we have the coefficient interval matrix

$$A = \begin{pmatrix} [2,4] & [-1,1] \\ [-1,1] & [2,4] \end{pmatrix}$$

and the right-hand side interval vector

$$b = \begin{pmatrix} [-3,3] \\ [0,0] \end{pmatrix}$$

The *solutions set* of such a system is given by:

$$\Sigma(A,b) := \{\tilde{x} \in \mathbb{R}^n \mid \exists \tilde{A} \in A, \exists \tilde{b} \in b \text{ such that } \tilde{A}\tilde{x} = \tilde{b}\}$$

We begin by giving some alternative characterizations of the solution set:

$$\Sigma(A,b) = \{\tilde{x} \in \mathbb{R}^n \mid A\tilde{x} \cap b \neq \emptyset\} = \{\tilde{x} \in \mathbb{R}^n \mid 0 \in A\tilde{x} - b\}$$

We show part of this proof in order to motivate our choice of formalization for intervals discussed in section 2.5 of chapter 2.

Proof excerpt.
We show: $\{\tilde{x} \in \mathbb{R}^n \mid A\tilde{x} \cap b \neq \emptyset\} \subseteq \Sigma(A,b)$.
Consider $\tilde{x}$ such that $A\tilde{x} \cap b \neq \emptyset$.
Then $A\tilde{x} \cap b$ contains some $\tilde{b} \in \mathbb{R}^m$.
Clearly $\tilde{b} \in b$.
Also, $\tilde{b} \in A\tilde{x}$.
We have $A\tilde{x} = \{\tilde{A}\tilde{x} \mid \tilde{A} \in A\}$ (relation 2.16, section 2.5, chapter 2).
This implies, $\tilde{b} = \tilde{A}\tilde{x}$ for some $\tilde{A} \in A$.
Therefore $\tilde{x} \in \Sigma(A,b)$. □

We explained in section 2.5 of chapter 2 that properties like relation 2.16 that we used above only hold if we work in an ideal arithmetic. Such proofs motivate our choice of formalizing intervals by using an axiomatic description of real numbers, which ensures the desired behavior.

Another characterization of the solution set is given by:

$$\tilde{x} \in \Sigma(A,b) \Leftrightarrow |A_c\tilde{x} - b_c| \leq \Delta_A|\tilde{x}| + \Delta_b.$$

In what follows we will only be interested in square matrices $A \in M(\mathbb{IR})_{n\times n}$. In the study of $\Sigma(A,b)$ the regularity of the interval matrix A plays an important role, for example in establishing that $\Sigma(A,b)$ is non-empty and bounded.

We give several criteria of regularity for an interval matrix and we illustrate them each time the interval matrix associated to the system of linear interval equations 4.1. The coefficient matrix of this system is

$$A = \begin{pmatrix} [2,4] & [-1,1] \\ [-1,1] & [2,4] \end{pmatrix} \tag{4.2}$$

The midpoint and radius matrices associated to this interval matrix are

$$A_c = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} \qquad \Delta_A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \tag{4.3}$$

The interval matrix A is regular. To show this, we take a look at the value of the determinant of a given matrix $\tilde{A} \in A$.

$$\tilde{A} = \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{12} \\ \tilde{a}_{21} & \tilde{a}_{22} \end{pmatrix} \text{ with } \tilde{a}_{11} \in [2,4], \tilde{a}_{12} \in [-1,1], \tilde{a}_{21} \in [-1,1], \tilde{a}_{22} \in [2,4]$$

$$\det \tilde{A} = \tilde{a}_{11}\cdot\tilde{a}_{22} - \tilde{a}_{21}\cdot\tilde{a}_{12} \in [2,4]\cdot[2,4] - [-1,1]\cdot[-1,1] = [4,8] - [-1,1] = [3,9]$$

This means, $\forall\tilde{A} \in A, \det\tilde{A} \in [3,9]$. In particular, all matrices contained in A have non-null determinant. Therefore, the interval matrix A is regular, according to the definition.

But using the definition to verify the regularity of an interval matrix is not efficient in practice. This is why we need other criteria for showing regularity of an interval matrix.

4.2 Basic regularity criteria

We first recall a characterization of regularity for real matrices that is available in the SSReflect matrix library:

$$\forall\tilde{A} \in M(\mathbb{R})_{m\times n}, \det\tilde{A} \neq 0 \Leftrightarrow \forall\tilde{x} \in \mathbb{R}^n, \tilde{A}\tilde{x} = 0 \Rightarrow \tilde{x} = 0. \tag{4.4}$$

Based on the previous proofs we can give criteria for checking regularity of interval matrices.

Criterion 1. *A is regular if and only if* $\forall \tilde{x} \in \mathbb{R}^n, 0 \in A\tilde{x} \Rightarrow \tilde{x} = 0$.

We show how this works on our example matrix 4.2.

Example 4. We have to verify the following

$$\forall \tilde{x} \in \mathbb{R}^2, 0 \in A\tilde{x} \Rightarrow \tilde{x} = 0$$

Let us consider $\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \in \mathbb{R}^2$. We assume

$$0 \in A\tilde{x} \Leftrightarrow 0 \in \begin{pmatrix} [2,4] & [-1,1] \\ [-1,1] & [2,4] \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \Leftrightarrow 0 \in \begin{pmatrix} \tilde{x}_1[2,4] + \tilde{x}_2[-1,1] \\ \tilde{x}_1[-1,1] + \tilde{x}_2[2,4] \end{pmatrix}$$

We distinguish 4 cases.

1. $\tilde{x}_1 \geq 0$ and $\tilde{x}_2 \geq 0$. Then we have

$$0 \in \begin{pmatrix} [2\tilde{x}_1 - \tilde{x}_2, 4\tilde{x}_1 + \tilde{x}_2] \\ [-\tilde{x}_1 + 2\tilde{x}_2, \tilde{x}_1 + 4\tilde{x}_2] \end{pmatrix} \Rightarrow \begin{cases} 2\tilde{x}_1 - \tilde{x}_2 \leq 0 \\ -\tilde{x}_1 + 2\tilde{x}_2 \leq 0 \end{cases}$$

   By multiplying the first inequality by 2 and adding the second we get $3\tilde{x}_1 \leq 0$ which implies $\tilde{x}_1 \leq 0$. Since by hypothesis we have $\tilde{x}_1 \geq 0$, so we deduce that $\tilde{x}_1 = 0$. We do the same for $\tilde{x}_2$ and deduce $\tilde{x}_2 = 0$.

We treat the remaining cases in an analogous manner.

Criterion 2. *A is regular if and only if* $\forall \tilde{x} \in \mathbb{R}^n, |A_c\tilde{x}| \leq \Delta_A|\tilde{x}| \Rightarrow \tilde{x} = 0$.

We show how this works on our example matrix 4.2.

Example 5. The midpoint matrix A_c and radius matrix Δ_A are given by relation 4.3. We need to show

$$\forall \tilde{x} \in \mathbb{R}^2, |A_c\tilde{x}| \leq \Delta_A|\tilde{x}| \Rightarrow \tilde{x} = 0$$

Let us take an arbitrary $\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \in \mathbb{R}^2$ and replace the actual values

$$\left| \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \right| \leq \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \left| \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \right|$$

By computing we get

$$\begin{pmatrix} 3|\tilde{x}_1| \\ 3|\tilde{x}_2| \end{pmatrix} \leq \begin{pmatrix} |\tilde{x}_1| + |\tilde{x}_2| \\ |\tilde{x}_1| + |\tilde{x}_2| \end{pmatrix}$$

We have a system of inequalities

$$\begin{cases} 3|\tilde{x}_1| \leq |\tilde{x}_1| + |\tilde{x}_2| \\ 3|\tilde{x}_2| \leq |\tilde{x}_1| + |\tilde{x}_2| \end{cases} \Leftrightarrow \begin{cases} 2|\tilde{x}_1| \leq |\tilde{x}_2| \\ 2|\tilde{x}_2| \leq |\tilde{x}_1| \end{cases} \Leftrightarrow \begin{cases} 4|\tilde{x}_1| \leq |\tilde{x}_1| \\ 4|\tilde{x}_2 \leq |\tilde{x}_2| \end{cases}$$

We deduce that $\tilde{x}_1 = 0$ and $\tilde{x}_2 = 0$. We therefore verified the criterion 2, and we can conclude that the matrix A is regular.

We can express singularity of an interval matrix A in the same terms.

Criterion 3. *A is singular if and only if $\exists \tilde{x} \in \mathbb{R}^n, \tilde{x} \neq 0$ such that*

$$|A_c\tilde{x}| \leq \Delta_A|\tilde{x}|. \tag{4.5}$$

Moreover, we can build a singular matrix from a solution of the inequality 4.5. Let $\tilde{x} \neq 0$ be such a solution.
We can consider the vectors $y, z \in \mathbb{R}^n$ defined by

$$y_i = \begin{cases} (A_c\tilde{x})_i/(\Delta_A|\tilde{x}|)_i & \text{, if } (\Delta_A|\tilde{x}|)_i \neq 0, \\ 1 & \text{, if } (\Delta_A|\tilde{x}|)_i = 0 \end{cases} \qquad z_j = \begin{cases} 1 & \text{, if } \tilde{x}_j \geq 0, \\ -1 & \text{, if } \tilde{x}_j < 0 \end{cases}$$

Then for the matrix $\tilde{A}$ given by

$$\tilde{A}_{ij} = (A_c)_{ij} - y_i z_j (\Delta_A)_{ij}$$

we have $\tilde{A} \in A$ and $\tilde{A}\tilde{x} = 0$ for $\tilde{x} \neq 0$, then from 4.4 we get $\det \tilde{A} = 0$.

The criteria 1 and 2 of regularity of an interval matrix are not used in practice as they require a number of arithmetic operations which is exponential in the matrix size n (cf. [71]).

4.3 Efficient regularity criteria

We present efficient criteria for checking regularity that are based on checking positive definiteness of a matrix and computing eigenvalues.

Generally, the proofs for these criteria follow rather naturally from the proofs presented in the previous sections on real matrices (section 2.3 of chapter 2) and on basic regularity criteria. To illustrate this we give a criterion that establishes regularity by a positive definiteness check and we detail its proof.

Criterion 4. *If the matrix*

$$A_c^T A_c - \|\Delta_A^T \Delta_A\| E_n, \ \textit{(where } E_n \textit{ is the identity matrix)}$$

is positive definite for some consistent matrix norm $\|\cdot\|$, then A is regular.

Proof. We do a proof by contradiction. We suppose that A is singular, so by Criterion 3 we get that there exists an $x \neq 0$ such that $|A_c x| \leq \Delta_A |x|$. We may normalize x to achieve $\|x\|_2 = 1$.
Then we have

$x^T A_c^T A_c x \leq |A_c x|^T |A_c x|$ - by properties of transpose and absolute value

$|A_c x|^T |A_c x| \leq (\Delta_A |x|)^T \Delta_A |x|$ - by hypothesis

$(\Delta_A |x|)^T \Delta_A |x| = |x|^T \Delta_A^T \Delta_A x$ - by properties of the transpose

$|x|^T \Delta_A^T \Delta_A x \leq \lambda_{\max}(\Delta_A^T \Delta_A)$ - by properties of Rayleigh's quotient

$\lambda_{\max}(\Delta_A^T \Delta_A) \leq \rho(\Delta_A^T \Delta_A)$ - by definition of the spectral radius

$\rho(\Delta_A^T \Delta_A) \leq \|\Delta_A^T \Delta_A\|$ - by properties relating spectral radius to norm

$\|\Delta_A^T \Delta_A\| = \|\Delta_A^T \Delta_A\|(x^T x)$ - by hypothesis that $1 = (\|x\|_2)^2 = x^T x$.

Reading the beginning and the end we get

$$x^T A_c^T A_c x \leq \|\Delta_A^T \Delta_A\|(x^T x)$$

This is equivalent to

$$x^T (A_c^T A_c - \|\Delta_A^T \Delta_A\| E_n) x \leq 0$$

which means that the matrix $(A_c^T A_c - \|\Delta_A^T \Delta_A\| E_n)$ is not positive definite, a contradiction to the hypothesis.

□

The above proof gives an idea of how we prove such criteria based on concepts we previously described.

We show how this criterion is used on our example matrix 4.2.

Example 6. The midpoint matrix A_c and radius matrix Δ_A are given by relation 4.3. We need to show

$$\forall \tilde{x} \in \mathbb{R}^2, \tilde{x} \neq 0, \tilde{x}^T \cdot (A_c^T A_c - \|\Delta_A^T \Delta_A\| E_2) \cdot \tilde{x} > 0$$

Let $\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} \in \mathbb{R}^2, \tilde{x} \neq 0$. We start by computing the matrix

$$A_c^T A_c - \|\Delta_A^T \Delta_A\| E_2 = \begin{pmatrix} 9 & 0 \\ 0 & 9 \end{pmatrix} - \left\| \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \right\| \cdot E_2 = 9E_2 - 2E_2 = 7E_2$$

We substitute the result

$$\tilde{x}^T \cdot 7E_2 \cdot \tilde{x} = 7(\tilde{x}_1^2 + \tilde{x}_2^2) > 0 \text{ as } x \neq 0$$

This means the matrix $(A_c^T A_c - \|\Delta_A^T \Delta_A\| E_2)$ is positive definite and therefore the interval matrix A is regular.

Here, we checked positive definiteness by using the definition. But there are more efficient algorithms for doing this check, hence the practical interest of this criterion (cf. [71]).

We give a criterion that ensures regularity at the cost of evaluating eigenvalues for symmetric matrices.

Criterion 5. *If the following inequality holds*

$$\lambda_{max}(\Delta_A^T \Delta_A) < \lambda_{min}(A_c^T A_c)$$

then A is regular.

We show how this criterion is used on our example matrix 4.2.

Example 7. The midpoint matrix A_c and radius matrix Δ_A are given by relation 4.3. We need to show

$$\lambda_{max}(\Delta_A^T \Delta_A) < \lambda_{min}(A_c^T A_c)$$

$$\lambda_{max}(\Delta_A^T \Delta_A) = \lambda_{max} \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} = \max\{0, 4\} = 4$$

$$\lambda_{min}(A_c^T A_c) = \lambda_{min} \begin{pmatrix} 9 & 0 \\ 0 & 9 \end{pmatrix} = \min\{9, 9\} = 9$$

Since $4 < 9$, we have indeed that $\lambda_{max}(\Delta_A^T \Delta_A) < \lambda_{min}(A_c^T A_c)$. We can therefore conclude that the interval matrix A is regular.

As a dual problem, we can also formulate criteria for checking singularity of an interval matrix. We give a criterion in terms of the approximate midpoint inverse R. This is very convenient as in practice we generally have the inverse computed in finite precision arithmetic which may affect validity of criteria given in terms of the exact midpoint inverse A_c^{-1}. However, we present such criteria also, as Corollary 2.

Criterion 6. *If there exist a matrix R such that*

$$(I + |I - A_c R|)_j \leq (\Delta_A |R|)_j$$

for some $j \in \{1, \ldots, n\}$, then A is singular.

Corollary 2. *If A_c is regular and $\max_j (\Delta_A |A_c^{-1}|)_{jj} \geq 1$, then A is singular.*

The Coq statements corresponding to the above criteria are listed in appendix B.4

Note

We formalized all but one regularity criteria described by [71]. This last criterion is based on the Perron-Frobenius theorem. A formalization of this theorem is being undertaken in an independent study by Guillaume Cano. Once this study completed, we will be able to fully formalize this criterion also.

4.4 Conclusion and future work

The study of regularity of interval matrices was motivated by needs of researchers interested in robotics who use such criteria in their daily work. We managed to formally verify criteria like 4, 5 which correspond to conditions used in practice. As pointed out in the introduction of this chapter, checking regularity of the interval matrix is just the first step in the resolution of systems of linear interval equations. To complete the study we still need to tackle the second step and provide a formal verification for an algorithm that computes the enclosure of the solution set.

For now all proofs are done for intervals with real bounds. These bounds are represented by the axiomatic reals in Coq's standard library so we cannot compute with these real numbers. In the long run, however, this work should serve as a theoretical basis to verify properties of actual computation. We note that it is a commonly used approach to have properties verified on a abstract model of real numbers or intervals and use them to validate a concrete implementation of real or interval arithmetic. For example the interval arithmetic tool Gappa does computations on machine floating point numbers and uses a Coq library on abstract reals to validate these computations [16]. In chapter 3 we saw that a description on axiomatic reals of properties for Newton's method can be used to verify computations with Newton's method on computable reals.

We can also envision generalizing this work to intervals with rational bounds or with floating point bounds. Since floats and rationals are themselves real numbers and intervals are computed by outward rounding, the results presented here should apply.

For example, let us suppose we have F an interval matrix where the ends of the intervals are floating point numbers and we managed to verify a certain regularity criterion for F. This criterion says that all real matrices included in F are regular. In particular all floating point matrices included in F are regular. However, this is still an issue to investigate: to which degree criteria proved for ideal arithmetic are still suitable in the floating point world.

The results presented in this chapter were published in [68]. The Coq *development is available on-line at:*
`http://www-sop.inria.fr/marelle/Ioana.Pasca/phd`

Chapter 5

Conclusions and Perspectives

Overview. We presented in chapter 2 the formalization of mathematical theories covering notions for real matrices, for multivariate analysis and for interval analysis. This formalization is carried out in the Coq proof assistant with the SSReflect extension. The formalization for real matrices is an extension to the SSReflect matrix library and it deals with matrix norms, eigenvalues and properties of special kinds of matrices (symmetric, positive definite). The formalization for multivariate analysis covers the notions related to vectors and their norms, sequences and their convergence, functions, continuity and partial derivability of these functions. The formalization for interval analysis covers the definition of real intervals, operations and other notions related to intervals and interval matrices.

These formalizations were done for the formal studies presented in chapters 3 and 4, but they are of interest in themselves as they cover general notions and results that can be reused in other studies. This is why we decided to present them in a separate chapter.

In chapter 3 we presented a formal study on Newton's method commonly used for solving equations and systems of equations. The study has three parts: we formalized well known results from the literature, we gave new results on Newton's method when rounding is performend during the computation and we studied Newton's method in the context of exact real number computations.

In chapter 4 we presented a formal study for criteria of regularity and singularity of interval matrices. This study can be seen as a continuation of the study on Newton's method, by looking closer at how we can show that the jacobian matrix is invertible. We can also consider this study as part of work on systems of linear interval equations.

Formalizing a numerical method. We presented in this manuscript the formal study of several elements involved in the numerical resolution of systems of equations. We studied two problems: Newton's method and regularity of interval matrices. These studies allow us to identify several phases in a formal development concerning a numerical method:

1. develop libraries for the background theories needed in the development;
2. formalize results from the literature concerning the properties of the method;
3. study computation using the method in a formal setting;
4. provide proofs of correctness for new optimizations for the method.

The first phase (formalization of mathematical theories) may be regarded as a formalization that is independent from the formal study of a numerical method. However, the existing libraries in proof assistants are currently not comprehensive enough to cover all the notions one may need, so the time required to develop these libraries has to be taken into account in the overall effort. Nevertheless, proof assistants are becoming more mature and, in particular, are more equipped with suitable libraries for various formal studies. This is very desirable if we want proof assistants to be more easily accessible to non-expert users. We saw in chapter 2 that the user has to know many technical details of the system in order to provide proper definitions for the notions he or she needs. But once these definitions are provided together with the technical lemmas that make their use close to paper mathematics, the user can forget (most of) the technical details and do high level proofs without worrying about the implementation details.

The second phase (formalization of results from the literature concerning the properties of the method) is usually what developers have in mind when talking about the formal study of the method. During this phase of the development, we perform the verification of well known results by reproducing the reasoning steps made on paper to make sure that these steps are valid. This gives the opportunity to discover details in the proof that have been overlooked or that are wrong, superfluous hypothesis or even improvements of the results.

The third phase (formal study of computations) is more specific to numerical methods, as their main purpose is to solve a problem through computations. But dealing with computation is not simple. On machines, real numbers are always approximated to some finite precision format, so we need to deal with the rounding errors properly.

The fourth phase (study of new properties) corresponds to what a proof assistant should ideally be for its users: an environment that allows the study of well known properties, but also the discovery and verification of new properties for a notion or an algorithm.

The study of Newton's method goes through all these phases. We needed to provide background mathematics for multivariate analysis. We then formalized results from the literature concerning the properties of Newton's method. We studied computations with the method in a library of exact real arithmetic and we proved a new result concerning a modified Newton's method that takes into account rounding at each step.

The study of regularity criteria for interval matrices covers the first two phases. We provided a formalization of interval arithmetic and results from the literature on sufficient criteria for regularity of interval matrices. The next step will be to have computation with intervals, but also, thanks to our formalization, to guarantee the results of the computation.

As similar study we have [15] and [14], where the authors also discuss the formal study of a numerical method by splitting the work in two phases (corresponding to the two papers). They tackle a numerical resolution scheme for a partial derivative equation and they separate the study of the method error and the rounding error. For the study of the method error they deal with the fact that they have a numerical method, and thus the result of the method is an approximation of the "true" solution. For the study of the rounding error they deal with the error introduced by using floating point numbers during the computations. These two steps roughly correspond to phases 2 and 3 of our study. However, in their work the study of the method error is separate from the study of the rounding error so it is not very clear how the two kinds of errors participate in getting the overall error. In our study on Newton's method with rounding, the result in theorem 7, section 3.2 gives the overall error, by combining the two contributing kinds of errors.

Formalizing a method. From a more general point of view, this work can be seen as dealing with formal verification of algorithms that require non-trivial mathematical knowledge for their proof of correctness. We generally think of formal verification developments as part of one of two categories: formal verification of software or hardware systems and formal verification of mathematical theories. In our work the two aspects are combined. We deal with algorithms and in order to prove them correct we use several mathematical theories.

When dealing with such methods there is always a certain gap between the "theoretical" method and the actual implementation. The "theoretical" method is what is proved correct on paper, but this does not necessarily correspond to the implementation. There are two main reasons for this. The first reason is that results on paper usually talk about some abstract, ideal concepts (for example, real numbers), while on machines we only benefit from an approximation of these concepts (for example, floating point numbers). The second reason is that proofs of correctness of some method are valid under certain hypotheses. In practice, it is often the case that the method is used in cases where the hypotheses are not verified. For the first reason, it is the job of the mathematician to find a proof that is convenient for the objects available on a machine. For the second reason, it is the job of the computer scientist to apply the method only under appropriate hypotheses. In this second case, a proof assistant can definitely help, as it will prevent the use of a theorem without the appropriate hypotheses. But in the first case a proof assistant can also help, it can serve as an environment for discovering new proofs, as in our case for the proof of theorem 7, section 3.2.

A recipe that seems successful is doing the proofs on some abstract model of the object, doing the "work" on a machine representable model and linking the two models. It is our approach for Newton's method: the proofs are done on axiomatic real numbers, the computations are done inside a library of exact real arithmetic but there is a link between the two models for real numbers that ensures correctness of our results. This is the case of other studies, whether the two models are inside the proof assistant, like [65], or the machine model is external to the proof assistant like [16].

Impact. The work presented here had an impact on two other formal developments. The study of Newton's method provided the necessary theoretical results for the design and implementation of an efficient algorithm for computing roots in the library of exact real arithmetic [52]. This algorithm is an optimized version of the algorithm described and proved correct in section 3.3.3. This optimized algorithm uses rounding in Newton's sequence by stopping the computation as soon as an appropriate number of digits has been obtained. This is allowed thanks to theorem 7 in section 3.2. The optimized algorithm also builds the function under the co-recursive call in a more efficient way. The optimized algorithm will be included in the library [52].

The study on interval matrices has revealed the need to have a comprehensive formalization for eigenvalues of real matrices. This implies formalizing results on complex numbers, complex analysis, linear algebra and topology. This study has been undertaken by Guillaume Cano in his master's thesis and will be continued during his PhD thesis.

Directions. Possible future work around our developments was discussed at the end of chapter 3 for Newton's method and at the end of chapter 4 for methods of solving linear systems of interval equations. One important step for this work is to go from the study of our methods on an axiomatic formalization of real numbers to their formal study on machine representable real numbers. For Newton's method this was achieved with the study of the method in a library of exact real number arithmetic. But we can push this even further (both for Newton's method and for regularity criteria) with a study on floating point numbers. This study can be done in several ways. We can use a formalization of floating point numbers like [13] and do a verification completely inside the proof assistant. We can implement the method in another programming language (for example C) and use a tool like Frama-C [2] to prove correctness inside the proof assistant. We can also link the formalization to a specialized software for the numerical methods. This is the approach taken, for example, with the Gappa tool [3] for interval arithmetic.

We can of course continue this work by investigating other numerical methods. We intend to finish the formal study on systems of linear interval equations. We treated regularity of interval matrices. We still need to treat methods for bounding the solution set of a system of linear interval equations in order to have a complete study. We can also envision other types of numerical methods. For example, the formalization on multivariate analysis can serve as a basis for treating issues like numerically solving ordinary differential equations or partial differential equations. Numerical algorithms are also often used for integration. Dealing with integration would require adding concepts on integrals in our formalization for multivariate analysis.

Appendix A

Mathematical Proofs for Newton's Method

This appendix contains the most important results concerning Newton's method as they appear in [29]. We decided to include this appendix because the above reference is not easy to find.

Let $x = (x_1, \ldots, x_p) \in \mathbb{R}^p$ and $f(x) = (f_1(x), \ldots, f_p(x))$, where $f_i \in C^{(1)} (i = 1, \ldots p)$.

Definition 1. By the derivative $f'(x)$ we understand the jacobian matrix of the system of functions $f_i, (i = 1, \ldots, p)$ with respect to the variables $x_1, \ldots, x_p$

$$f'(x) = \left[\frac{\partial f_i}{\partial x_j}\right] \tag{A.1}$$

The matrix function

$$F(x) = \begin{bmatrix} f_{11}(x) & \ldots & f_{1r}(x) \\ \ldots & \ldots & \ldots \\ f_{p1}(x) & \ldots & f_{pr}(x) \end{bmatrix}$$

can be considered like the set of r functions, corresponding to the columns in the above matrix

$$F_1(x) = (f_{11}(x), \ldots, f_{p1}(x)), \ldots, F_r(x) = (f_{1r1}(x), \ldots, f_{pr}(x))$$

It is natrual to understand by derivative $F'(x)$, the set

$$F'(x) = [F_1'(x) \ldots F_r'(x)],$$

where

$$F_k'(x) = \begin{bmatrix} \frac{\partial f_{1k}}{\partial x_1} & \dots & \frac{\partial f_{1k}}{\partial x_p} \\ \dots & \dots & \dots \\ \frac{\partial f_{pk}}{\partial x_1} & \dots & \frac{\partial f_{pk}}{\partial x_p} \end{bmatrix}$$

are the jacobian matrices ($k = 1, \dots r$).

Definition 2. If $F(x) = [f_{ij}(x)]$ is a functional matrix of size $p \times r$ and $f_{ij} \in C^{(1)}$ we set

$$f'(x) = [F_k'(x)],$$

where

$$[F_k'(x)] = \left[\frac{\partial f_{ik}}{\partial x_j}\right] \quad (i, j = 1, \dots, p;\ k = 1, \dots, r).$$

If the vector function $f(x) = [f_i(x)]$ is such that $f_i \in C^{(2)}$, then

$$f''(x) = [W_1(x) \dots W_p(x)],$$

with

$$W_k(x) = \left[\frac{\partial^2 f_i}{\partial x_k \partial x_j}\right] \ (k = 1, \dots, p).$$

We use in what followes the following norm:

$$\|f(x)\| = \max_i |f_i(x)|;$$

$$\|f'(x)\| = \max_i \sum_{j=1}^{p} \left|\frac{\partial f_i(x)}{\partial x_j}\right|;$$

$$\|f''(x)\| = \max_k \|W_k(x)\| = \max_k \left\{\max_i \sum_{j=1}^{p} \left|\frac{\partial^2 f_i(x)}{\partial x_k \partial x_j}\right|\right\}, etc.$$

In an analogous manner

$$\|F(x)\| = \max_i \sum_{j=1}^{p} |f_{ij}(x)|;$$

$$\|F'(x)\| = \max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial f_{ij}(x)}{\partial x_k}\right|.$$

Lemma 8. *If*

$$F(x) = [f_{ij}(x) \quad (p \times r),$$

where $f_{ij}(x)$ are continuous with their first partial derivatives in a convex domaine which contains the points x and $x + \Delta x$, then

$$\|F(x + \Delta x) - F(x)\| \le r\|\Delta x\| \cdot \|F'(\xi)\|,$$

where $\xi = x + \theta\Delta x, 0 < \theta < 1$.

Proof. By applying Taylor's formula we get:

$$F(x+\Delta x) - F(x) = [f_{ij}(x+\Delta x) - f_{ij}(x)] = \left[\sum_{k=1}^{p} \frac{\partial f_{ij}(\xi_{ij})}{\partial x_k} \Delta x_k\right]$$

with $\xi_{ij} = x + \theta_{ij}\Delta x,\ 0 < \theta_{ij} < 1;\ i = 1, \dots, p;\ j = 1, \dots, r.$
For a given x and $x + \Delta x$, we have:

$$\|F(x+\Delta x) - F(x)\| = \max_i \sum_{j=1}^{r}\sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ij})}{\partial x_k}\Delta x_k\right| \leq \max_i \sum_{j=1}^{r}\sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ij})}{\partial x_k}\right| |\Delta x_k| \leq$$

$$\leq \max_k |\Delta x_k| \cdot \sum_{j-1}^{r} \max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ij})}{\partial x_k}\right| = r\|\Delta x\| \max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ij})}{\partial x_k}\right|.$$

Since the number of couples (i, j) is finite, there exists a couple (a, b) such that

$$\max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ij})}{\partial x_k}\right| = \sum_{k=1}^{p} \left|\frac{\partial f_{ab}(\xi_{ab})}{\partial x_k}\right| \leq \max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial f_{ij}(\xi_{ab})}{\partial x_k}\right| = \|F'(\xi)\|,$$

where $\xi = \xi_{ab}$.

Therefore

$$\|F(x+\Delta x) - F(x)\| \leq r\|\Delta x\| \cdot \|F'(\xi)\|,$$

which concludes the proof.

□

Corollary 3. *If $f(x) = [f_1(x) \dots f_p(x)]$ then we have*

$$\|f(x+\Delta x) - f(x)\| \leq \|\Delta x\| \cdot \|f'(\xi)\|$$

where $\xi = x + \theta\Delta x, 0 < \theta < 1$.

Corollary 4. *With $f(x) \in C^{(2)}$ we have:*

$$\|f'(x+\Delta x) - f'(x)\| \leq p\|\Delta x\| \cdot \|f''(\xi)\|$$

where$\xi = x + \theta\Delta x, 0 < \theta < 1$.

Lemma 9. *If $f(x) = [f_1(x) \dots f_p(x)] \in C^{(2)}$ in a convex domain which contains the points x and $x + \Delta x$, then*

$$\|f(x+\Delta x) - f(x) - f'(x)\Delta x\| \leq \frac{1}{2}p\|\Delta x\|^2 \cdot \|f''(\xi)\| \tag{A.2}$$

where $\xi = x + \theta\Delta x, 0 < \theta < 1$.

Proof. By using Taylor's formula, we get:

$$\|f(x+\Delta x) - f(x) - f'(x)\Delta x\| = \|[f_i(x+\Delta x) - f_i(x) - df_i(x_i)]\Delta x\| =$$

$$\leq \frac{1}{2}\left\|\left[\sum_{j,k}\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\Delta x_j \Delta x_k\right]\right\| \leq \frac{1}{2}\left\|\left[\sum_j |\Delta x_j| \sum_k \left|\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\right| |\Delta x_k|\right]\right\| \leq$$

$$\leq \frac{1}{2}\max_j |\Delta x_j| \cdot \max_k |\Delta x_k| \cdot \left\|\left[\sum_j \sum_k \left|\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\right|\right]\right\| =$$

$$= \frac{1}{2}\|\Delta x\|^2 \left\|\left[\sum_j \sum_k \left|\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\right|\right]\right\|$$

where $\xi_i = x + \theta_i \Delta x, 0 < \theta_i < 1$.

Since

$$\sum_k \left|\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\right| \leq \max_{i,j} \sum_k \left|\frac{\partial^2 f_i(\xi_i)}{\partial x_j \partial x_k}\right| = \sum_k \left|\frac{\partial^2 f_a(\xi_a)}{\partial x_b \partial x_k}\right| \leq \max_{i,j} \sum_k \left|\frac{\partial^2 f_i(\xi_a)}{\partial x_j \partial x_k}\right| = \|f''(\xi_a)\|$$

We therefore get:

$$\|f(x+\Delta x) - f(x) - f'(x)\Delta x\| \leq \frac{1}{2}\|\Delta x\|^2 \cdot [\|f''(\xi)\|] = \frac{1}{2}p\|\Delta x\|^2 \cdot \|f''(\xi)\|$$

where $\xi = \xi_a = x + \theta \Delta x, 0 < \theta < 1$.

□

The theorem asserting the existence of solutions of a system of equations and the convergence of Newton's method is as follows:

Theorem 10 (Kantorovitch)**.** *Consider a system of non-linear algebraic or transcendent equations*

$$f(x) = 0 \tag{A.3}$$

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 $J_f(x) = \left[\frac{\partial f_i(x)}{\partial x_j}\right]$ has an inverse for $x = x_0$, $\Gamma_0 = J_f^{-1}(x_0)$ 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

$$\mu_0 = 2pA_0B_0C \leq 1 \tag{A.4}$$

Then, for the initial approximation x_0, the Newton process

$$x_{n+1} = x_n - J_f^{-1}(x_n)f(x_n) \tag{A.5}$$

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

Proof. Let us introduce the notations

$$h_n = \|x_{n+1} - x_n\| = \max_k |(x_{n+1})_k - (x_n)_k|,$$

$$\Gamma_n = W^{-1}(x_n), \quad (n = 0, 1, 2, \ldots)$$

The formula A.5 implies

$$h_n = \|\Gamma_n f(x_n)\|$$

Conditions 1 - 4 give estimations of the quantities Γ_n and $\Gamma_n f(x_n)$. Let us first examine the case $p = 1$. By using condition 2, we have:

$$h_0 = \|x_1 - x_0\| = \|W^{-1}(x_0)f(x_0)\| \leq B_0 \leq \frac{\varepsilon}{2}$$

therefore

$$h_0 \leq B_0$$

and

$$\overline{U}_{\frac{\varepsilon}{2}}(x_1) \subset \overline{U}_{\varepsilon}(x_0)$$

In order to evaluate $\Gamma_1 = W^{-1}(x_1)$, we apply the relation $(AB)^{-1} = B^{-1}A^{-1}$ and get the expression in the following form

$$\Gamma_1 = [W(x_0)\Gamma_0 W(x_1)]^{-1} = [\Gamma_0 W(x_1)]^{-1}\Gamma_0 \tag{A.6}$$

Taking into account condition 1 of the theorem we have:

$$\|E - \Gamma_0 W(x_1)\| = \|\Gamma_0[W(x_0 - W(x_1)]\| \leq$$

$$\leq \|\Gamma_0\|\|W(x_0) - W(x_1)\| \leq A_0\|W(x_0) - W(x_1)\|$$

Since the condition 3 implies

$$\|f''(x)\| = \max_{i,j} \sum_{k=1}^{p} \left|\frac{\partial^2 f_i(x)}{\partial x_j \partial x_k}\right| \leq C$$

and according to corollary 4 of lemma 8 we get:

$$\|W(x_1) - W(x_0)\| = \|f'(x_1) - f'(x_0)\| \leq p\|x_1 - x_0\|C \leq pB_0C$$

and therefore

$$\|E - \Gamma_0 W(x_1)\| \leq pA_0B_0C = \frac{\mu_0}{2} \leq \frac{1}{2}$$

There exists the inverse matrix (see section 2.3, relation 2.1)

$$[\Gamma_0 W(x_1)]^{-1} = \{E - (E - \Gamma_0 W(x_1))\}^{-1}$$

and since $\|E\| = 1$,

$$\|[\Gamma_0 W(x_1)]^{-1}\| \leq \frac{1}{1 - \frac{\mu_0}{2}} \leq 2 \tag{A.7}$$

We deduce from formula A.6:

$$\|\Gamma_1\| \leq \|[\Gamma_0 W(x_1)]^{-1}\| \|\Gamma_0\| \leq 2A_0 = A_1 \tag{A.8}$$

The formula A.5 implies

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

from which we deduce using lemma 9

$$\|f(x_1)\| = \|f(x_1) - f(x_0) - f'(x_0)(x_1 - x_0)\| \leq \frac{1}{2}p\|x_1 - x_0\|^2 \|f''(\xi)\| \leq \frac{1}{2}pB_0^2C$$

with

$$\xi = x_0 + \theta(x_1 - x_0) \text{ and } 0 < \theta < 1$$

Taking inequality A.8 into account we get:

$$\|\Gamma_1 f(x_1)\| \leq \|\Gamma_1\| \|f(x_1)\| \leq 2A_0 \frac{1}{2}pB_0^2C = pA_0B_0^2C = \frac{1}{2}\mu_0 B_0 = B1 \tag{A.9}$$

Thus, for the point x_1 we have

$$\overline{U}_{\frac{\varepsilon}{2}}(x_1) \subset \overline{U}_{\varepsilon}(x_0) \subset \omega$$

and, in addition,

$$\|\Gamma_1\| \leq A_1, \quad h_1\|\Gamma_1 f(x_1)\| \leq B_1$$

where

$$A_1 = 2A_0$$
$$B_1 = \frac{1}{2}\mu_0 B_0 \leq \frac{\varepsilon}{4}$$

This implies

$$\mu_1 = 2pA_1B_1C = 2p2A_0\frac{1}{2}\mu_0B_0C = \mu_0 2pA_0B_0C = \mu_0^2 \leq 1 \tag{A.10}$$

We find ourselves again in the conditions of the theorem, with the difference that now, instead of the neighborhood $\overline{U}_{\varepsilon}(x_0)$ we have the neighborhood $\overline{U}_{\frac{\varepsilon}{2}}(x_1)$ included in the first neighborhood.

By making the same reasoning steps we can establish that the successive approximations $x_n(n = 1, 2, \ldots)$ are well defined and they are such that

$$\overline{U}_{\varepsilon}(x_0) \supset \overline{U}_{\frac{\varepsilon}{2}}(x_1) \supset \ldots \supset \overline{U}_{\frac{\varepsilon}{2^n}}(x_n) \supset \ldots$$

in addition

$$\|\Gamma_n\| = \|W^{-1}(x_n)\| \leq A_n,$$

$$\|\Gamma_n f(x_n)\| = \|x_{n+1} - x_n\| \leq B_n$$

where the constants A_n and B_n are given by the following recurrence relations

$$A_n = 2A_{n-1} \tag{A.11}$$

$$B_n = \frac{1}{2}\mu_{n-1}B_{n-1} \tag{A.12}$$

$$\mu_n = 2pA_nB_nC \tag{A.13}$$

Let us show that the sequence of approximations $x_n(n = 0, 1, \ldots)$ verify Cauchy's criterion (see section 2.4, definition 2.4). Indeed, for $m > 0$ we have:

$$x_{n+m} \in \overline{U}_{\frac{\varepsilon}{2^n}}$$

The, for all given δ

$$\|x_{n+m} - x_n\| \leq \frac{\varepsilon}{2^n} \leq \delta,$$

if $n > N$ and $m > 0$ with N sufficiently large, which is equivalent to Cauchy's criterion. We can deduce the existence of the limit

$$\lim_{n\to\infty} x_n = x^* \in \overline{U}_{\varepsilon}(x_0)$$

Let us now show that x^* is a solution of system A.3. Relation A.5 gives

$$f(x_n) + W(x_n)(x_{n+1} - x_n) = 0$$

Going to the limit in this equality when $n \to \infty$ and taking into account that

$$x_{n+1} - x_n \to 0,$$

as well as the fact that $W(x_n)$ is continuous and bounded in $\overline{U}_{\varepsilon}(x_0)$, we get

$$\lim_{n\to\infty} f(x_n) = 0$$

We get, thanks to the continuity of f:

$$f(\lim_{n\to\infty} x_n) = f(x^*) = 0$$

that is, x^* is a solution of system A.3. In addition,

$$\|x^* - x_0\| = \left\|\sum_{n=0}^{\infty}[x_{n+1} - x_n]\right\| \leq \sum_{n=0}^{\infty}\|x_{n+1} - x_n\| \leq \sum_{n=0}^{\infty} B_n \leq B_0 + \frac{B_0}{2} + \ldots = 2B_0 \leq \varepsilon$$

The theorem is thus completely proved. □

The theorem stating the speed of convergence of Newton's method is as follows:

Theorem 11. *Under the conditions 1 - 4 of theorem 10, the succesive approximations $x_n (n = 0, 1, 2, \ldots)$ verify the inequality:*

$$\|x^* - x_n\| \leq \left(\frac{1}{2}\right)^{n-1} \mu_0^{2^n - 1} B_0,$$

where x^* is a solution of the system and μ_0 is defined by formula A.4.

Proof. By applying relations A.11, A.12 and A.13 we get

$$\mu_n = 2pA_nB_nC = 2p \cdot 2A_{n-1} \cdot \frac{1}{2}\mu_{n-1}B_{n-1} \cdot C = \mu_{n-1} \cdot 2pA_{n-1}B_{n-1}C = \mu_{n-1}^2.$$

This implies

$$\mu_n = \mu_0^{2^n}. \tag{A.14}$$

Then

$$B_n = \frac{1}{2}\mu_{n-1}B_{n-1} = \frac{1}{2}\mu_0^{2^{n-1}} B_{n-1}.$$

Therefore

$$B_n = \frac{1}{2}\mu_0^{2^{n-1}} \cdot \frac{1}{2}\mu_0^{2^{n-2}} \ldots \frac{1}{2}\mu_0^{2^0} B_0 = \left(\frac{1}{2}\right)^n \cdot \mu_0^{2^{n-1}+2^{n-2}+\ldots+1} B_0 = \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0. \tag{A.15}$$

Since

$$\|x_{n+1} - x_n\| \leq B_n,$$

we have for $m > 1$

$$\|x_{n+m} - x_n\| \leq | \, x_{n+1} - x_n\| + | \, x_{n+2} - x_{n+1}\| + \ldots + | \, x_{n+m} - x_{n+m-1}\| \leq$$

$$\leq B_n + B_{n+1} + \ldots + B_{n+m-1} =$$

$$= \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0 + \left(\frac{1}{2}\right)^{n+1} \mu_0^{2^{n+1}-1} B_0 + \ldots \left(\frac{1}{2}\right)^{n+m-1} \mu_0^{2^{n+m-1}-1} B_0 =$$

$$= \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0 \left[1 + \frac{1}{2}\mu_0^{2^n} + \ldots + \left(\frac{1}{2}\right)^{m-1} \mu_0^{2^n(2^{m-1}-1)}\right].$$

Considering that $\mu_0 \leq 1$ we deduce

$$\|x_{n+m} - x_n\| \leq \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0 \left[1 + \frac{1}{2} + \dots \left(\frac{1}{2}\right)^{m-1}\right] \leq \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0.$$

Taking the limit when $m \to \infty$, we get

$$\|x^* - x_n\| \leq \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} B_0 \leq \left(\frac{1}{2}\right)^n \mu_0^{2^n-1} \varepsilon$$

where

$$\mu_0 = 2pA_0B_0C \leq 1.$$

Therefore, for $\mu_0 < 1$ the convergence of Newton's method is superfast.

In particular, for $n = 0$ we get

$$\|x^* - x_0\| \leq 2B_0 \leq \varepsilon.$$

□

The unicity of the solution for the system of equations is given by the following theorem.

Theorem 12. *Under the conditions 1 - 4 of theorem 10, the domaine*

$$\|x - x_0\| \leq 2B_0 \tag{A.16}$$

contains only one solution of the system A.3.

Proof. Let us suppose that besides the solution x^* of the system A.3, defined by Newton's method, there exists another solution x^{**} of the system such that

$$\|x^{**} - x_0\| \leq 2B_0. \tag{A.17}$$

The successive approximations x_n $(n = 0, 1, 2 \dots)$ of Newton's method are contaid in a neighborhood A.16 and respect the condition

$$f(x_n) + W_n(x_{n+1} - x_n) = 0$$

with

$$W_n = W(x_n).$$

Considering that

$$f(x^{**}) = 0$$

we get

$$W_n(x_{n+1} - x^{**}) = f(x^{**}) - f(x_n) - W_n(x^{**} - x_n)$$

and, therefore

$$x_{n+1} - x^{**} = \Gamma_n[f(x^{**}) - f(x_n) - W_n(x^{**} - x_n)],$$

where

$$\Gamma_n = W_n^{-1}.$$

By computing this estimation in norm, we get

$$\|x^{**} - x_{n+1}\| \leq \|\Gamma_n\|\|f(x^{**}) - f(x_n) - W_n(x^{**} - x_n)\|.$$

Using the notations from theorem 10

$$\|\Gamma_n\| \leq A_n.$$

By applying lemma 9 we get the inequality

$$\|f(x^{**}) - f(x_n) - W_n(x^{**} - x_n)\| \leq \frac{1}{2}pC\|x^{**} - x_n\|^2$$

where the constant C is defined according to condition 3 of theorem 10. Then

$$\|x^{**} - x_{n+1}\| \leq \frac{1}{2}pA_nC\|x^{**} - x_n\|^2 \ (n = 0, 1, 2 \ldots). \tag{A.18}$$

By taking in inequality A.18 $p = 0$ and using inequality A.17 we get

$$\|x^{**} - x_1\| \leq \frac{1}{2}pA_0C\|x^{**} - x_0\|^2 \leq 2pA_0B_0^2C,$$

By using the relations A.12 and A.13 we find

$$\|x^{**} - x_1\| \leq \mu_0 B_0 = 2B_1 \tag{A.19}$$

In an analogous manner, for $p = 1$ we deduce from formulas A.18 and A.19:

$$\|x^{**} - x_2\| \leq \frac{1}{2}pA_2C\|x^{**} - x_1\|^2 \leq 2pA_1B_1^2C = \mu_1 B_1 = 2B_2.$$

In general

$$\|x^{**} - x_n\| \leq 2B_n \ (n = 0, 1, 2 \ldots). \tag{A.20}$$

Since relation A.15 implies that $B_n \to 0$ when $n \to \infty$, by taking the limit in inequality A.20, we get

$$x^{**} = \lim_{n\to\infty} x_n = x^*,$$

which means that the solution of system A.3 is unique in the domaine $\|x - x_0\| \leq 2B_0$. □

Remark 1. *If the domaine $\overline{U}_\varepsilon(x_0)$ is such that*

$$\frac{2}{\mu_0}B_0 \leq \varepsilon,$$

then the system A.3 does not have in the extended domaine

$$\|x - x_0\| \leq \frac{2}{\mu_0}B_0 \tag{A.21}$$

other solutions that x^*.

Proof. Let us suppose that the domaine A.21 contains a solution x^{**} of the system A.3 and by the same reasoning as for theorem 10, we get an inequality of the form A.18

$$\|x^{**} - x_{n+1}\| \leq \frac{1}{2}pA_nC\|x^{**} - x_n\|^2 \ (n = 0, 1, 2 \ldots).$$

Since

$$\|x^{**} - x_0\| \leq \frac{2}{\mu_0}B_0$$

we get

$$\|x^{**} - x_1\| \leq \frac{1}{2}pA_0C\frac{4}{\mu_0^2}B_0^2 = 2pA_0B_0C\frac{1}{\mu_0^2}B_0 = \frac{1}{\mu_0}B_0 = \frac{2}{\mu_0^2}B_1 = \frac{2}{\mu_1}B_1,$$

$$\|x^{**} - x_2\| \leq \frac{1}{2}pA_1C\frac{4}{\mu_1^2}B_1^2 = 2pA_1B_1C\frac{1}{2}\mu_1B_1\frac{2}{\mu_1^3} = \mu_1B_2\frac{2}{\mu_1^3} = \frac{2}{\mu_1^2}B_2 = \frac{2}{\mu_2}B_2, \text{ etc.}$$

In general

$$\|x^{**} - x_n\| \leq \frac{2}{\mu_n}B_n \ (n = 0, 1, 2, \ldots).$$

According to relations A.12 and A.13 we get

$$\frac{B_n}{\mu_n} = \frac{1}{2}\frac{B_{n-1}}{\mu_{n-1}} = \left(\frac{1}{2}\right)^n \frac{B_0}{\mu_0} \tag{A.22}$$

Therefore

$$\|x^{**} - x_n\| \leq \left(\frac{1}{2}\right)^{n-1} \frac{B_0}{\mu_0} \ (n = 0, 1, 2, \ldots).$$

This implies

$$x^{**} = \lim_{n\to\infty} x_n = x^*$$

and concludes the proof.

□

The local stability of Newton's method is given by the following theorem.

Theorem 13. *If conditions 1 - 4 of theorem 10 are satisfied and if*

$$\frac{2}{\mu_0}B_0 \leq \varepsilon,$$

with $\mu_0 = 2pA_0B_0C < 1$*, Newton's method converges towards the unique solution* x^* *of system A.3 i the main domaine* $\|x - x_0\| \leq 2B_0$ *whatever the choice of the initial approximation* x'_0 *in the domain*

$$\|x'_0 - x_0\| \leq \frac{1-\mu_0}{2\mu_0}B_0. \tag{A.23}$$

Proof. By analogy with the notations given above

$$W_0 = W(x_0) \text{ and } \Gamma_0 = W_0^{-1}$$

let us introduce

$$W'_0 = W'(x'_0) \text{ and } \Gamma' - 0 = (W'_0)^{-1}.$$

Let us show that the point x'_0 verifies conditions analogous to conditions 1 - 4 of theorem 10.

By using the notations and the proof method from theorem 10, we get

$$\|E - \Gamma_0 W'_0\| = \|\Gamma_0(W_0 - W'_0)\| \leq \|\Gamma_0\|\|W_0 - W'_0\| \leq A_0pC\|x'_0 - x_0\|.$$

By inequality A.23, we get

$$\|E - \Gamma_0 W'_0\| \leq A_0pC\frac{1-\mu_0}{2\mu_0}B_0 = \frac{1-\mu_0}{4} \leq \frac{1}{4}.$$

Then

$$\|(\Gamma_0 W'_0)^{-1} = \|[E - (E - \Gamma_0 W'_0)]^{-1}\| \leq \frac{1}{1 - \|E - \Gamma_0 W'_0\|} \leq \frac{1}{1 - \frac{1-\mu_0}{4}} = \frac{4}{3+\mu_0}. \tag{A.24}$$

Therefore, there exists

$$\Gamma'_0 = (\Gamma_0 W'_0)^{-1}\Gamma_0$$

and

$$\|\Gamma'_0\| \leq \|(\Gamma_0 W'_0)^{-1}\|\|\Gamma_0\| \leq \frac{4A_0}{3+\mu_0} = A'. \tag{A.25}$$

We then deduce

$$\|\Gamma_0 f(x'_0)\|\| \leq \|\Gamma_0\|\|f(x'_0) - f(x_0) - W_0(x'_0 - x_0)\| + \|\Gamma_0 f(x_0)\| + \|x'_0 - x_0\| \leq$$

$$\leq \frac{1}{2}A_0pC\|x'_0 - x_0\|^2 + B_0 + \|x'_0 - x_0\| \leq \frac{1}{4}\mu_0 B_0 \frac{1 - 2\mu_0 + \mu_0^2}{4\mu_0^2} + B_0 + \frac{1-\mu_0}{2\mu_0}B_0 =$$

$$= \frac{1 - 2\mu_0 + \mu_0^2 + 16\mu_0 + 8 - 8\mu_0}{16\mu_0}B_0 = \frac{(3+\mu_0)^2}{16\mu_0}B_0.$$

We deduce, by using inequaity A.24

$$\|\Gamma'_0 f(x'_0)\| = \|(\Gamma_0 W'_0)^{-1}\Gamma_0 f(x'_0)\| \leq$$

$$\leq \|(\Gamma_0 W'_0)^{-1}\| \cdot \|\Gamma_0 f(x'_0)\| \leq \frac{4}{3+\mu_0} \cdot \frac{(3+\mu_0)^2}{16\mu_0} B_0 = \frac{3+\mu_0}{4\mu_0} B_0 = B'. \tag{A.26}$$

By inequalites A.25 and A.26 we get

$$\mu' = 2pA'B'C = 2p\frac{4A_0}{3+\mu_0}\frac{3+\mu_0}{4\mu_0}B_0C = 2pA_0B_0C\frac{1}{\mu_0} = 1.$$

In addition

$$2B' + \|x'_0 - x_0\| \leq \frac{3+\mu_0}{2\mu_0}B_0 + \frac{1-\mu_0}{2\mu_0}B_0 = \frac{2B_0}{\mu_0} \leq \varepsilon$$

and thus

$$2B' \leq \frac{2B_0}{\mu_0} \leq \varepsilon.$$

Therefore, at point x'_0 the conditions of theorem 10 are verified. This means we have

$$\overline{U}_{2B'}(x'_0) \subset \overline{U}_{\frac{2B_0}{\mu_0}}(x_0) \subset \overline{U}_{\varepsilon}(x_0) \tag{A.27}$$

Newton's method

$$x'_{n=1} = x' + n - \Gamma'_n f(x'_n),$$

where

$$\Gamma'_n = W^{-1}(x_n) \ (n = 0, 1, 2 \ldots),$$

converges towards a solution x'^* of system A.3 which is contained in the domain $\overline{U}_{2B'}(x'_0)$. According to formula A.27

$$x'^* \in \overline{U}_{\frac{2B_0}{\mu_0}}(x_0).$$

But according to the remark 1 of theorem 12 in the domain $\overline{U}_{\frac{2B_0}{\mu_0}}(x_0)$ there exists a unique solution x^* of system A.3. Therefore

$$x'^* = x^* \text{ and } x^* = \lim_{n\to\infty} x'_n$$

which concludes the proof.

□

Appendix B

COQ Statements

B.1 Newton's method in one dimension

We place ourselves in the context of the theorem, by expressing our working hypotheses. In the code, the Variable declarations describe objects that are assumed to exist, the Hypothesis declarations describe properties that are assumed to hold and the Fixpoint declaration defines the Newton sequence as a recursive function.

```
Variables a b: R. (*the ends of the interval*)
Variables f f' f'': R → R. (*the function and its derivatives*)
Variables A0 B0 C: R. (*the constants from the theorem*)
Definition mu0 := 2 * A0 * B0 * C.
Variable X0: R. (*the initial approximation*)
Fixpoint Xn (n:nat): R := (*the Newton sequence*)
  match n with | 0 ⇒ X0 |S n ⇒ Xn n − f (Xn n) / f' (Xn n) end.

(*the hypothesis on the function and the constants*)
Hypothesis Hincl: a < X0 < b.
Hypothesis pr: forall t:R, o_I a b t → derivable_pt f t.
Hypothesis Hder_f: forall (t: R) (H: o_I a b t), derive_pt f t (pr t H) = f' t.
Hypothesis pr2: forall t:R, o_I A B t → derivable_pt f' t.
Hypothesis Hder_f': forall (t: R) (H: o_I a b t), derive_pt f' t (pr2 t H) = f'' t.
Hypothesis Hbound_f'': forall t (H: o_I A B t), Rabs (f'' t) ≤ C.
Hypothesis Hdif_f': f' X0 <> 0 .
Hypothesis Habs_a: Rabs (/f' X0) ≤ A0.
Hypothesis Habs_b: Rabs (f X0 / f' X0) ≤ B0 .
Hypothesis Hb_eps: included (c_disc X0 (2*B0)) (o_I a b).
Hypothesis A0_b0_c: 2*A0*B0*C ≤ 1.
```

To clarify the statements above, we mention that o_I a b and c_disc X0 (2∗B0) denote the open interval (a, b) and the closed disc centered in X0 and of radius 2B0, respectively. Hypothesis pr says that function f is derivable on interval (a, b), hypothesis Hder_f states that f' is equal to the derivative of f on the same interval, while hypotheses pr2, Hder_f' and Hbound_f'' give similar conditions for the first and second derivative.

We now start the proof by introducing the sequences $(A_n)_{n\in\mathbb{N}}$ and $(B_n)_{n\in\mathbb{N}}$ (see section 3.1).

```
Fixpoint An n := match n with |0 ⇒ a0 |S n ⇒ 2∗An n end.
Fixpoint Bn n := match n with |0 ⇒ b0 |S n ⇒ An n∗(Bn n)ˆ2∗c end.
```

We also introduce the sequence $(E_n)_{n\in\mathbb{N}}$ for $\varepsilon/2^n$ to enhance readability.

```
Fixpoint En n := match n with |0 ⇒ eps |S n ⇒ (En n)/2 end.
```

We are now able to deduce that $(X_n)_{n\in\mathbb{N}}$ is convergent and the limit of the sequence is the root of f in the desired domain.

```
Theorem newton_exist: {xs:R & Un_cv Xn xs ∧ c_disc X0 (2∗b0) xs ∧ f xs = 0}.
Definition xs:= projT1 kanto_exist.
```

In the domain $\{|x - x^{(0)}| \leq 2B_0\}$, the root xs is unique.

```
Theorem newton_uniq: forall xs2, c_disc X0 (2∗b0) xs2 → f xs2 = 0 → xs = xs2.
```

We can give the speed of convergence:

```
Theorem newton_speed: forall n,
  Rabs (xs − Xn n) ≤ / 2 ˆ (n − 1) ∗ mu0 ˆ (2 ˆ n − 1) ∗ B0 .
```

and establish the local stability of Newton's method:

```
Variable X0':R. (* the new initial approximation *)
Hypothesis Hmu01: 0 < mu0 < 1. (* hypotheses from Theorem 6 *)
Hypothesis Hmudisc: included (c_disc X0 (2 / mu0 ∗ B0)) (c_I a b).
Hypothesis Hdom: Rabs (X0' − X0) ≤ (1 − mu0)/(2 ∗ mu0) ∗ B0.


Theorem newton_stable: Un_cv (Xn X0' f f') xs.
```

For Newton's method with rounding at each step we introduce a new sequence:

```
Fixpoint Tn n := match n with
  |O ⇒ X0 |S n' ⇒ let tn := Tn n' in (rnd (tn − f tn / f' tn) n) end.
```

where rnd is a rounding function with the properties described in Theorem 7. The convergence of Newton with rounding is given by:

```
Lemma newton_rnd_conv : Un_cv Tn xs.
```

B.2 Newton's method with rounding

```
Variables (a0 b0 c: R).
Variable X0: R.
Variable a: R.
Variable b: R.
Variable f: R → R.
Variable f': R → R.
Hypothesis pr: forall t :R, c_I a b t → derivable_pt f t.
Hypothesis Hder_f: forall (t:R)(H:c_I a b t), derive_pt f t (pr t H) = f' t.
Hypothesis Hcont_f': forall y, (c_I a b y) → continuity_pt f' y.
Hypothesis Hlim_f': forall y1 y2: R, c_I a b y1 → c_I a b y2 → Rabs (f' y1 − f' y2) <= c
    ∗ (Rabs (y1−y2)).
Hypothesis Hincl: a < X0 < b.
Hypothesis Hdif_f': f' X0 <> 0 .
Hypothesis Habs_a: Rabs (/(f' X0)) <= a0.
Hypothesis Habs_b: Rabs (f X0/(f' X0)) <= b0 .
```

We model the approximation that come in the computation with machine reals as a function that takes the real to approximate and a natural number which corresponds to the precision at which we are approximating.

```
Variable aprox: R → nat → R.

Fixpoint Yn n {struct n}: nat → R :=
  match n with
    |0%nat ⇒ Xn X0 f f'
    |S n' ⇒ Xn (aprox (Yn n' 1) n) f f'
  end.

Definition Zn n := Yn n 0.

Fixpoint Aseq n {struct n}: R :=
  match n with
    |0%nat ⇒ a0
    |S n' ⇒ a0' (An (Aseq n') 1%nat) (Bn (Aseq n') (Bseq n') c 1%nat) c
end
  with
Bseq n {struct n}: R :=
  match n with
```

```
    |O ⇒ b0
    |S n' ⇒ b0' (An (Aseq n') 1%nat) (Bn (Aseq n') (Bseq n') c 1%nat) c
end.

Fixpoint Tn n {struct n}: R :=
  match n with
    |O ⇒ X0
    |S n' ⇒ let tn := Tn n' in (aprox (tn − f tn / f' tn) n)
  end.

Lemma eq_tn_zn: forall n, Zn n = Tn n.
Hypothesis aprox_in_ab:
forall x n, o_I a b x → o_I a b (aprox x n).
Definition Round n := /3^n ∗ ((1 − (mu a0 b0 c 0)^2)/ (8∗ (mu a0 b0 c 0))∗b0) .
Hypothesis Haprox: forall n,
  Rabs ((aprox (Xn (Yn n 0) f f' 1) (S n)) − Xn (Yn n 0) f f' 1) ⇐ Round n.
Hypothesis mu0_in_bound: 1/2 ⇐ mu a0 b0 c 0 < 1.
Hypothesis Hbla3: included (c_disc X0 (3∗b0)) (c_I a b).

Lemma tn_conv: Un_cv Tn xstar .
Lemma conv_tn_speed: forall n, Rabs (Tn n − xstar) ⇐ / Rpower 2 (INR n − 1) ∗ b0.
```

B.3 Newton's method in several dimensions

```
Variable p: pos_nat.
Variable a0: R.
Variable b0: R.
Variable c: R.
Variable keps: R.
Variable X0: vec R p.
Variable f: vec R p → vec R p.

Hypothesis Hkepos: 0 < keps.
Hypothesis A0_b0_c: 2∗ (INR p) ∗ a0 ∗ b0 ∗ c ⇐ 1.
Hypothesis Hcontf: forall v, cont_Rp f v.

Hypothesis pr: forall (x : vec R p) (x0 : 'I_p), dbl_pt p f x0 x. (* once derivable *)

Hypothesis pr2: forall (i j: 'I_p) (a: vec R p), dbl_pt_2 p f i j a (fun x ⇒ pr x i) .
(* twice derivable *)
```

```
Hypothesis der_cont1: forall a i, cont_Rp (fun x ⇒ der_p p f i x (pr x i)) a.
(* continuous first order derivatives *)

Hypothesis der_cont2: forall a i j, cont_Rp
  (fun v ⇒ der_p_2 p f i j v (fun x ⇒ pr x i) (pr2 i j v)) a .
(* continuous second order derivatives *)

Hypothesis der_bo: forall (v0: vec R p) (i j: 'I_p),
\sum_( k < p) Rabs (der_p_2 p f j k v0 (fun v ⇒ pr v j) (pr2 j k v0) i) <= c.
(* bounded second order derivatives *)

Variable f': vec R p → 'M[R]_p.

Hypothesis Hjac: forall a, f' a = Jac p f a (pr a).
Hypothesis Hinv0: \det (f' X0) <> 0.
Hypothesis Ha: norm_m ((f' X0)^−1m) <= a0.
Hypothesis Hb: norm (mult_mv ( (f' X0)^−1m) (f X0)) <= b0.
Hypothesis Hb0: b0 <= keps*/2.

Fixpoint Xn (n:nat): vec R p:=
  match n with
    |0 ⇒ X0
    |S n ⇒ dif_v (Xn n) (mult_mv ((f' (Xn n))^−1m) (f (Xn n)))
  end .

Theorem kantoroRp_exists:
  exists xs: vec R p, conv Xn xs ∧ norm (dif_v xs X0) <= 2*b0 /\ f xs = vect0.

Theorem kantoro_unic: forall xs2: vec R p,
  norm (dif_v xs2 X0) <= 2*b0 → f xs2 = vect0 → conv Xn xs2.

Theorem conv_speed:
  forall k (xs: vec R p), norm (dif_v xs X0) <= 2*b0 → f xs = vect0 →
  norm (xs −^ Xn k) <= (/ 2^R (INR k − 1)) * mu 0 ^ (Npow 2 k − 1) * b0 .
```

B.4 Criteria for regularity of interval matrices

```
Definition inSetm m n (A: 'M[IR]_(m, n) (vA: 'M[R]_(m, n)) :=
  forall i j, vA i j \in A i j.
```

```
Definition sigma_sol m n (A: 'M[IR]_(m, n)) (b: 'cV[IR]_m) :=
  fun x ⇒ exists vA: 'M_(m, n), inSetm A vA ∧
  exists vb: 'cV_m, inSetm b vb ∧ vA ∗m x = vb.

Variable n': nat.
Notation n := n'.+1.
Variable A: 'M[IR]_n.
Variable b: 'cV[IR]_n.

(* alternative characterization of the solution set *)
Theorem Beeck1:
  forall x, sigma_sol A b x ↔ exists t , setI (inSetm (mmuls_i A x)) (inSetm b) t.

Theorem Beeck2:
 forall x, sigma_sol A b x ↔ inSetm (madd_i (mmuls_i A x) (mopp_i b)) 0 .

Corollary OetteliPrager:
 forall x, sigma_sol A b x ↔
   Mabs (mmid A ∗m x − mmid b) ⇐m: mrad A ∗m Mabs x + mrad b.

Lemma reg_all_sol0:
  regular A ↔ forall (x: 'cV_n) a, inSetm A a → a ∗m x = 0 → x = 0.

(* criterion 1 *)
Lemma reg_int_sol0:
  regular A ↔ (forall x: 'cV_n, inSetm (mmuls_i A x) 0 → x = 0).

(* criterion 2 *)
Lemma reg_ineq_mid_rad:
  regular A ↔
  (forall x: 'cV_n, Mabs (mmid A ∗m x) ⇐m: mrad A ∗m Mabs x → x = 0).

(* criterion 3 *)
Lemma sing_ineq_mid_rad:
  singular A ↔
    (exists x: 'cV_n, Mabs (mmid A ∗m x) ⇐m: (mrad A ∗m Mabs x) ∧ x <> 0).

Lemma oettli_prager_sing:
  {x: 'cV_n | Mabs ( mmid A ∗m x) ⇐m: mrad A ∗m Mabs x ∧ x <> 0}
```

```
    → {vA | inSetm A vA ∧ \det vA == 0}.

(* criterion 6 *)
Theorem thm33: (exists Rm , exists j,
  \col_i (1 + Mabs ( 1 − mmid A ∗ Rm))%Ri i j ⇐m: \col_i (mrad A ∗ (Mabs Rm)) i j)
    → singular A.

Corollary cor34:
  \det (mmid A) != 0 → 1 ⇐ \big[Rmax/0]_(j< n) (mrad A ∗ Mabs (mmid A)^−1) j j
      → singular A.

Variable eigens: matrix R n n → seq R.
Hypothesis HeigAtA: forall A lam, lam \in (eigenv A) ↔ (lam \in (eigens A)).

(* criterion 5 *)
Theorem thm41_lam_minmax:
 lam_max eigens ((mrad A) ^T ∗m mrad A) < lam_min eigens ((mmid A)^T ∗ mmid A)
     → regular A.

Variable mnorm: can_norm.

(* criterion 4 *)
Theorem thm51_posdef:
  pos_def_gen ((mmid A)^T ∗ mmid A − || (mrad A) ^T ∗ mrad A : mnorm| ∗m: 1 )
    → regular A.
```

Bibliography

[1] The Agda Home Page. http://wiki.portal.chalmers.se/agda/pmwiki.php. 3

[2] The Frama-C Home Page. http://frama-c.com/index.html. 99

[3] The Gappa Home Page. http://gappa.gforge.inria.fr/. 99

[4] The Mathematical Componenets Project. http://www.msr-inria.inria.fr/Projects/math-components. 3

[5] The Matita Home Page. http://matita.cs.unibo.it/. 3

[6] The Mizar Home Page. http://www.mizar.org/. 3

[7] Henk Barendregt and Herman Geuvers. Proof-assistants using dependent type systems. pages 1149–1238, 2001. 3

[8] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art:the Calculus of Inductive Constructions.* Springer-Verlag, 2004. 3, 8, 11

[9] Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical Big Operators. In *Theorem Proving in Higher Order Logics, TPHOLs 2008*, volume 5170 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2008. 28, 60, 83

[10] Yves Bertot, Nicolas Magaud, and Paul Zimmermann. A Proof of GMP Square Root. *J. Autom. Reasoning*, 29(3-4):225–252, 2002. 3

[11] Sidi Ould Biha. Formalisation des mathématiques : une preuve du théorème de Cayley-Hamilton. In *Journées Francophones des Langages Applicatifs*, pages 1–14, 2008. 24, 25, 33

[12] E. Bishop and D.S. Bridges. *Constructive analysis.* Springer-Verlag, 1985. 23

[13] Sylvie Boldo. *Preuves formelles en arithmétiques à virgule flottante.* PhD thesis, École Normale Supérieure de Lyon, November 2004. 3, 83, 99

[14] Sylvie Boldo. Floats & Ropes: a case study for formal numerical program verification. In *36th International Colloquium on Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science - ARCoSS*, pages 91–102, Rhodos, Greece, July 2009. Springer. 97

[15] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Formal proof of a wave equation resolution scheme: The method error. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2010. 3, 97

[16] Sylvie Boldo, Jean-Christophe Filliâtre, and Guillaume Melquiond. Combining Coq and Gappa for Certifying Floating-Point Programs. In *16th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning*, volume 5625 of *Lecture Notes in Artificial Intelligence*, pages 59–74, Grand Bend, Canada, July 2009. Springer. 93, 98

[17] R.P. Brent and P. Zimmermann. *Modern Computer Arithmetic.* 2006. In preparation. Available at http://www.loria.fr/zimmerma/mca/pub226.html. 82

[18] Tim Coe. Inside the pentium fdiv bug. *Dr. Dobb's Journal*, 20:129 – 135, 1995. 3

[19] M. Collins, F. Vecchio, R. Selby, and P. Gupta. The failure of an offshore platform. *Concrete International*, 19:159 – 192, 1997. `http://www.concreteinternational.com/pages/featured_article.asp?ID=56`. 3

[20] Coq development team. *The Coq Proof Assistant Reference Manual, version 8.1*, 2006. 3, 11

[21] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988. 4

[22] John Cowles, Ruben Gamboa, and Jeff Van Baalen. Using ACL2 Arrays to Formalize Matrix Algebra. In *ACL2 Workshop*, 2003. 23

[23] 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. 74

[24] Luis Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Phd thesis, University of Nijmegen, April 2004. 23

[25] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-corn, the constructive coq repository at nijmegen. In *MKM*, pages 88–103, 2004. 23, 83

[26] Marc Daumas, Guillaume Melquiond, and César Muñoz. Guaranteed proofs using interval arithmetic. In Paolo Montuschi and Eric Schwarz, editors, *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 188–195, Cape Cod, MA, USA, 2005. 3, 58

[27] David Delahaye and Micaela Mayero. `Field`: une procédure de décision pour les nombres réels en Coq. In *Journées Francophones des Langages Applicatifs, Pontarlier*. INRIA, Janvier 2001. 55

[28] 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. 23

[29] B. Démidovitch et I. Maron. *Éléments de calcul numérique*. Mir - Moscou, 1979. 16, 31, 62, 63, 64, 101

[30] 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. 22, 74

[31] R. Gamboa and M. Kaufmann. Nonstandard Analysis in ACL2. *Journal of automated reasoning*, 27(4):323–428, November 2001. 22, 74

[32] François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging mathematical structures. In *TPHOLs*, pages 327–342, 2009. 25, 26

[33] Herman Geuvers, Freek Wiedijk, and Jan Zwanenburg. A Constructive Proof of the Fundamental Theorem of Algebra without Using the Rationals. In *Types*

for Proofs and Programs, TYPES 2000 International Workshop, Selected Papers, volume 2277 of *LNCS*, pages 96–111, 2002. 23

[34] 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. 76

[35] Georges Gonthier. A computer-checked proof of the four-colour theorem. Available at `http://research.microsoft.com/~gonthier/4colproof.pdf`. 3, 24

[36] 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`. 24

[37] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL : a theorem proving environment for higher-order logic*. Cambridge University Press, 1993. 3

[38] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL : a theorem proving environment for higher-order logic*. Cambridge University Press, 1993. 3

[39] Benjamin Grégoire and Assia Mahboubi. Proving equalities in a commutative ring done right in coq. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005. 55

[40] The PRL Group. *The Nuprl Book*. `http://www.cs.cornell.edu/info/projects/nuprl/book/doc.html`. 3

[41] Thomas C. Hales. Introduction to the flyspeck project. In *Mathematics, Algorithms, Proofs*, 2005. 3

[42] John Harrison. Floating point verification in hol. In E. Thomas Schubert, Phillip J. Windley, and Jim Alves-Foss, editors, *TPHOLs*, volume 971 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1995. 22

[43] John Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998. 22, 74

[44] John Harrison. Formal verification of floating point trigonometric functions. In Jr. and Johnson [51], pages 217–233. 3, 22

[45] 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. 24, 48

[46] John Harrison. Floating-point verification using theorem proving. In Marco Bernardo and Alessandro Cimatti, editors, *SFM*, volume 3965 of *Lecture Notes in Computer Science*, pages 211–242. Springer, 2006. 3, 22

[47] Alden Hayashi. Rough sailing for smart ships. *Scientific American*, 279:26, 1998. 3

[48] Michael Hedberg. A Coherence Theorem for Martin-Löf's Type Theory. *Journal of Functional Programming*, 8(4):413–436, 1998. 52

[49] Johannes Holzl. Proving Inequalities over Reals with Computation in Isabelle/HOL. *International Workshop on Programming Languages for Mechanized Mathematics Systems*, pages 38 – 45, 2009. 58

[50] Namhyun Hur and James H. Davenport. A generic root operation for exact real arithmetic. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *CCA*, volume 2064 of *Lecture Notes in Computer Science*, pages 82–87. Springer, 2000. 82

[51] Warren A. Hunt Jr. and Steven D. Johnson, editors. *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, volume 1954 of *Lecture Notes in Computer Science*. Springer, 2000. 22, 126

[52] Nicolas Julien. Certified exact real arithmetic using co-induction in arbitrary integer base. In *Functional and Logic Programming Symposium (FLOPS)*, LNCS. Springer, 2008. 17, 74, 75, 79, 83, 98

[53] Nicolas Julien and Ioana Pasca. Formal Verification of Exact Computations Using Newton's Method. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *LNCS*, pages 408–423, 2009. 83

[54] Cezary Kaliszyk and Russell O'Connor. Computing with classical real numbers. *Journal of Formalized Reasoning*, 2:27–39, 2009. 83

[55] Matt Kaufmann, Panagiotis Manolios, and J. Strother Moore. *Computer-aided reasoning: an approach*. Kluwer Academic Publishing, 2000. 3

[56] 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. 74, 82

[57] Jacques-Louis Lions and al. Ariane 5 flight 501 failure report by the inquiry board. *Rapport technique, European Space Agency, Paris, France*, 1996. 3

[58] Nicolas Magaud. Ring properties for square matrices. `http://coq.inria.fr/contribs-eng.html`. 23, 37

[59] Micaela Mayero. *Formalisation et automatisation de preuves en analyses reelle et numerique*. PhD thesis, Université de Paris VI, 2001. 3, 74

[60] Guillaume Melquiond. Proving bounds on real-valued functions with computations. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, volume 5195 of *Lectures Notes in Artificial Intelligence*, pages 2–17, Sydney, Australia, 2008. 3, 58

[61] Jean-Pierre Merlet. Interval analysis for certified numerical solution of problems in robotics. *International Journal of Applied Mathematics and Computer Science*, 19:399–412, 2009. 86

[62] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990. 49, 51

[63] Milad Niqui. Coinductive formal reasoning in exact real arithmetic. *Logical Methods in Computer Science*, 4(3:6):1–40, September 2008. 82

[64] Steven Obua. Proving bounds for real linear programs in isabelle/hol. In *Theorem Proving in Higher-Order Logics*, pages 227–244, 2005. 23

[65] 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. 74, 82, 83, 98

[66] Ioana Paşca. A Formal Verification for Kantorovitch's Theorem. In *Journées Francophones des Langages Applicatifs*, pages 15–29, 2008. 60, 83

[67] Ioana Pasca. Formal Proofs for Theoretical Properties of Newton's Method, 2010. INRIA Research Report RR-7228. Available online `http://hal.inria.fr/inria-00463150/en/`. 60, 83

[68] Ioana Pasca. Formally verified conditions for regularity of interval matrices. In *17th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning, Calculemus 2010*, volume 6167 of *Lecture Notes in Artificial Intelligence*, pages 219 – 233. Springer, 2010. 60, 94

[69] Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, Décembre 1996. 4

[70] Lawrence C. Paulson and Tobias Nipkow. *Isabelle : a generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. 3

[71] Georg Rex and Jiri Rohn. Sufficient conditions for regularity and singularity of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 20:437–445, 1998. 86, 90, 92, 93

[72] A. Robinson. *Non-Standard Analysis*. Princeton University Press, 1996. 22

[73] Jiri Rohn. Forty necessary and sufficient conditions for regularity of interval matrices: A survey. *Electronic Journal of Linear Algebra*, 18:500–512, 2009. 86

[74] 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. 3

[75] J. Stein. Documentation for the formalization of Linear Agebra. `http://www.cs.ru.nl/~jasper/`. 23

[76] Freek Wiedijk. Introduction. In Freek Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2006. 3

[77] Roland Zumkeller. Formal Global Optimization with Taylor Models. In *International Joint Conference on Automated Reasoning (IJCAR)*, LNCS (LNAI), pages 408 – 422. Springer, 2006. 58