The Picard Algorithm for Ordinary Differential Equations in Coq

Evgeny Makarov and Bas Spitters

Radboud University Nijmegen[*]

Abstract. Ordinary Differential Equations (ODEs) are ubiquitous in physical applications of mathematics. The Picard-Lindelöf theorem is the first fundamental theorem in the theory of ODEs. It allows one to solve differential equations numerically. We provide a constructive development of the Picard-Lindelöf theorem which includes a program together with sufficient conditions for its correctness. The proof/program is written in the Coq proof assistant and uses the implementation of efficient real numbers from the CoRN library and the MathClasses library. Our proof makes heavy use of operators and functionals, functions on spaces of functions. This is faithful to the usual mathematical description, but a novel level of abstraction for certified exact real computation.

Key words: Coq; Exact real computation; Ordinary Differential Equations; Constructive mathematics; Type classes

1 The Picard-Lindelöf Theorem

We present the mathematical ideas behind our formalization. Let $v : [-a, a] \times [-K, K] \to \mathbf{R}$ be continuous such that $v(x, 0) = 0$. Assume that $L > 0$ is such that $aL < 1$ and[1]

$$|v(x, y) - v(x, y')| \le L|y - y'| \tag{1}$$

for all $x \in [-a, a]$ and $y, y' \in [-K, K]$. Consider the initial value problem

$$f'(x) = v(x, f(x)), \quad f(0) = 0.$$

To solve this equation we define the Picard operator $Pf(t) := \int_0^t v(x, fx)dx$ and observe that a fixed point $f = Pf$ is a solution to the differential equation, which can be seen by differentiating both sides. To find such a fixed point, we first show that P is a contraction.

[*] The research leading to these results has received funding from the European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

[1] If v is differentiable in the second argument, then we can choose

$$L := \sup_x \sup_{y \in [-K,K]} \frac{d}{dy} v(x, y).$$

Lemma 1. *The Picard operator is a contraction on the metric space $C([-a,a],\mathbf{R})$ with constant $aL < 1$. If $f \subset [-a,a] \times [-K,K]$, then so is Pf.*

Proof.

$$\begin{aligned}\sup_{t\in[-a,a]} \left|\int_0^t v(x,fx)dx - \int_0^t v(x,gx)dx\right| &\leq \sup_{t\in[-a,a]} \int_0^t |v(x,fx) - v(x,gx)|dx \\ &\overset{(1)}{\leq} aL\|f-g\|_\infty\end{aligned}$$

Here $\|h\|_\infty$ is $\sup_{[-a,a]} |h(x)|$. Since $v(x,0) = 0$,

$$\begin{aligned}|Pf(t)| = \left|\int_0^t v(x,fx)dx\right| &\leq t \sup_x |v(x,fx)| \\ &\leq a \sup_x |v(x,fx) - v(x,0)| \leq aLK \leq K.\end{aligned}$$

We can now apply the Banach fixed point theorem to the Picard operator on the complete metric space $C[-a,a]$ and obtain a fixed point.

2 A computational library for analysis

We depend a huge code base, the CoRN library [1] combined with the recent MathClasses library [2, 3]. Part of this work[2] is adapting code from the old library to the new coding style.

2.1 Metric spaces using type classes

We provide a type class based presentation of metric spaces, roughly following [4]. This definition of metric spaces uses a closed ball relation `ball e x y` which intuitively means that $d(x,y) \leq e$. We define the completion monad on metric spaces and we define *complete* metric spaces as the existence of a retract of the embedding of X into its completion. The completion consists of regular functions, a refinement of the notion of a Cauchy sequence.

```
Class Limit := lim : RegularFunction → X.
Class CompleteMetricSpaceClass `{Limit} := cmspc :> Surjective reg_unit (inv := lim).
Definition tends_to (f : RegularFunction) (l : X) :=
  forall e : Q, 0 < e → ball e (f e) l.
Lemma limit_def `{CompleteMetricSpaceClass} (f : RegularFunction) :
  forall e : Q, 0 < e → ball e (f e) (lim f).
```

To be able to reuse some of the old results, we prove that each old complete metric space — using only records, but no type class automation — defines one based on type classes. We want to consider the complete metric space $C[-a,a]$, so we define closed submetric spaces determined by a ball.

We have various classes of functions — uniformly continuous, Lipschitz, ... In order to be able to treat them all at once, we define a type class. In this way we can define e.g. the supremum metric once for all relevant spaces of functions.

[2] https://github.com/EvgenyMakarov/corn/tree/master/ode

```
Class Func (T X Y : Type) := func : T → X → Y.
```

We need to be careful, since this introduces an equality on function spaces, determined by the metric space of functions, but we already have the extensional equality. We want Coq to automatically find the latter for us. Moreover, we want to prevent Coq from looping. This could happen in the following way. Suppose we define the equality on a metric space using the ball, then one way to find an equality is to find a ball relation. Consider:

```
Global Instance Linf_func_metric_space_ball : MetricSpaceBall T :=
  λ e f g, forall x, ball e (func f x) (func g x).
```

This has two class arguments: Func T X Y and MetricSpaceBall Y. If we put MetricSpaceBall Y first, then the equality on some type T may require a ball on a fresh Y (since Y is not in the conclusion of Linf_func_metric_space_ball), and this would call Linf_func_metric_space_ball again and require a ball on a fresh Y1, etc.

We can prevent this by using the fact that instances of the leftmost type class arguments are searched first. So we made Func T X Y the first Class argument in Linf_func_metric_space_ball. We have few instances of Func, and the first argument of Func in those instances is of the form, say, UniformlyContinuous X Y, or Lipschitz X Y. So, if T does not have this form, then the search for an instance of Func T X Y fails immediately. As a result, if T is, e.g., A → B, then the equality on T found by Coq is extensional equality, not the one is obtained not through MetricSpaceBall. For more on this kind of logic programming see [5, 2].

The uniformly continuous functions between two complete metric spaces form a complete metric space. The distance between two functions may be infinite.

2.2 An axiomatic treatment of integration.

There are at least two Coq formalizations of the integral. The CoRN formalization closely follows Bishop's treatment of the Riemann integral. As argued by Dieudonné, it seems better to treat the Cauchy integral (only continuous functions) and develop the full theory of Lebesgue integration when we need to go further. This is roughly the approach taken by Spitters and O'Connor [6] who developed Cauchy integration theory for $C[0, 1]$. Here we take a different approach. We define the integral for locally uniformly continuous functions $\mathbf{Q} \to \mathbf{R}$ with an abstract specification similar to the one by Bridger [7, Ch5.].

```
Class Integral (f: Q → CR) := integrate: forall (from: Q) (w: QnonNeg), CR.
Class Integrable `{!Integral f}: Prop :=
  { integral_additive:
      forall (a: Q) b c, ∫f a b + ∫f (a+' b) c == ∫f a (b+c)%Qnn
  ; integral_bounded_prim: forall (from: Q) (width: Qpos) (mid: Q) (r: Qpos),
      (forall x, from ≤ x ≤ from+width → ball r (f x) ('mid)) →
      ball (width * r) ∫( f from width) (' (width * mid))
  ; integral_wd:> Proper (Qeq ==> QnonNeg.eq ==> @st_eq CRasCSetoid) ∫( f) }.
```

Here CR is the completion of the rationals, i.e. the reals. The types Qpos and Qnn are the positive and nonnegative rational numbers respectively. The last line says that the integral respects the various setoid equalities. This specification is complete in that it uniquely characterizes the integral. The class thus expresses that a function $\int$ is an implementation of the integral. We provide a reference implementation using the technology of type classes. It is less abstract, but twice as fast as the development in [6].

2.3 Picard iteration

We define the Picard operator from uniformly continuous functions to uniformly continuous functions. We define a function extend which extends a function from an interval to to real line in a constant way. This allows us to define:

```
Definition picard' (f : sx → sy) `{!IsUniformlyContinuous f mu} : Q → CR :=
  λ x, y0 + int (extend x0 rx (v ∘ (together Datatypes.id f) ∘ diag)) x0 x.
```

This automatically finds the proof that (extend x0 rx (v ∘ (together Datatypes .id f)∘ diag)) is integrable. This requires some care as all assumptions, such as $0 \leq$ rx, need to be in the context. We then prove that the Picard operator is a contraction.

The Picard operator maps $[-a, a] \times [-K, K]$ to itself. Hence, we can iterate it and apply the Banach Fixed Point theorem.

```
Context `{MetricSpaceClass X}{Xlim : Limit X}{Xcms : CompleteMetricSpaceClass X}.
Context (f : X → X) `{!IsContraction f q} (x0 : X).
Let x n := nat_iter n f x0.
Let a := lim (reg_fun x _ cauchy_x).
Lemma banach_fixpoint : f a = a.
```

Applying this to the Picard operator on the metric space $C[-a, a]$, we find that there is an f such that $Pf = f$. This is the required solution to the differential equation.

2.4 Timings

We have made very initial experiments with computation inside Coq. For e^x at $x = 1/2$, we obtain two correct decimal digits, i.e. one after the decimal point instantaneously, however three digits takes too long. The timings show that our implementation performs reasonably well, but there are a number of possible improvements:

- Use reals based on dyadic rationals, as in [3].
- Use Newton iteration instead of Picard iteration. A variant of the work in [8] might be useful here.
- Use an improved algorithm for the integral such as Simpson integration. A constructive proof of Simpson integration can be found in [9].
- Use Coq's experimental native_compute; see [10].

Conclusion

We are working towards a verified implementation of a simple ODE-solver in Coq. We mention related work by Immler and Hölzl in Isabelle [11]. They go further in that they also implement the Euler method. At times, our implementation seems more natural, in that we can use dependent types to express for instance the type of continuous functions on a given interval. Their code can be extracted from Isabelle to SML and produces approximately two verified decimal digits. We obtain a bit less, but we compute inside the COQ proof assistant. Hence, all our computations are actually verified. Finally, we would like to mention the work on verifying a C-program for the wave equation [12, 13].

Acknowledgements We would like to thank Jelle Herold and Eelis van der Weegen who were involved in earlier initiatives to formalize the Picard Theorem.

References

1. Cruz-Filipe, L., Geuvers, H., Wiedijk, F.: C-CoRN, the Constructive Coq Repository at Nijmegen. In: MKM. (2004) 88–103
2. Spitters, B., van der Weegen, E.: Type classes for mathematics in type theory. MSCS, special issue on "Interactive theorem proving and the formalization of mathematics" **21** (2011) 1–31
3. Krebbers, R., Spitters, B.: Type classes for efficient exact real arithmetic in coq. LMCS (2013)
4. O'Connor, R.: Certified Exact Transcendental Real Number Computation in Coq. In: TPHOLs 2008. Volume 5170 of LNCS. (2008) 246–261
5. Gonthier, G., Ziliani, B., Nanevski, A., Dreyer, D.: How to make ad hoc proof automation less ad hoc. In: ICFP. (2011) 163–175
6. O'Connor, R., Spitters, B.: A computer verified, monadic, functional implementation of the integral. TCS **411**(37) (2010) 3386–3402
7. Bridger, M.: Real analysis, a constructive approach. Pure and Applied Mathematics (New York). Wiley (2007)
8. Julien, N., Pasca, I.: Formal Verification of Exact Computations Using Newton's Method. In: TPHOLs 2009. Volume 5674 of LNCS. (2009) 408–423
9. Coquand, T., Spitters, B.: A constructive proof of Simpson's rule. Logic and Analysis **4**(15) (2012) 1–8
10. Boespflug, M., Dénès, M., Grégoire, B.: Full reduction at full throttle. In: CPP. Volume 7086 of LNCS. (2011) 362–377
11. Immler, F., Hölzl, J.: Numerical analysis of ordinary differential equations in isabelle/hol. Interactive Theorem Proving (2012) 377–392
12. Boldo, S., Clément, F., Filliâtre, J., Mayero, M., Melquiond, G., Weis, P.: Formal proof of a wave equation resolution scheme: the method error. Interactive Theorem Proving (2010) 147–162
13. Boldo, S., Clement, F., Filliâtre, J., Mayero, M., Melquiond, G., Weis, P.: Wave equation numerical resolution: a comprehensive mechanized proof of a C program. Journal of Automated Reasoning (2011) 1–34