Computable Analysis for Extraction of Certified Programs

Holger Thies

Kyoto University

Computability in Europe 2025
July 14-18, 2025
Universidade de Lisboa, Lisbon, Portugal

京都大学
KYOTO UNIVERSITY

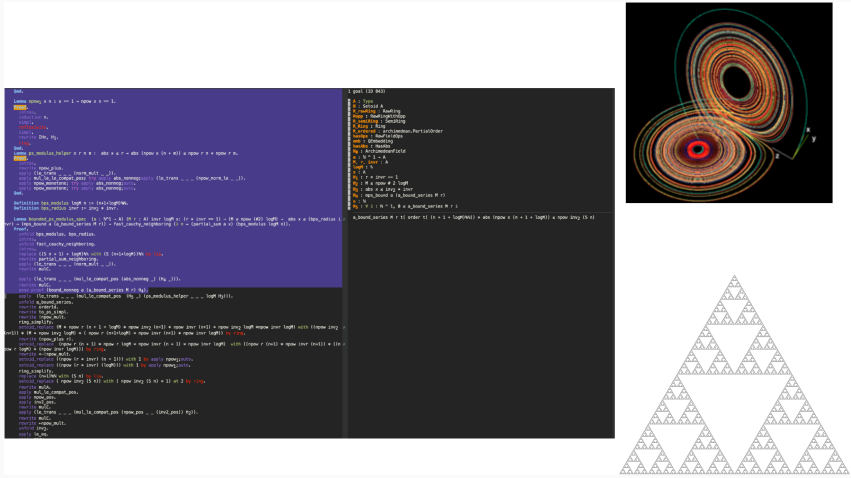
This work is supported by JSPS KAKENHI Grant Numbers JP23K28036, and JP24K20735.

Overview

1. Real numbers and computation in the Rocq proof assistant
2. cAERN
3. Initial Value Problems and the Cauchy-Kovalevskaya theorem

Computation and Formal Proof

A **proof assistant** is an interactive tool for writing and checking formal proofs.



Computation in Rocq

There are several ways to use computation in the Rocq (formerly Coq) proof assistant:

ROCQ

```
Fixpoint div2 (n : nat) :=
  match n with
  | 0 | 1 => 0
  | S (S n') => (div2 n')+1
  end.

Compute (div2 100). (* = 50 : nat *)
```

Computation in Rocq

There are several ways to use computation in the Rocq (formerly Coq) proof assistant:

ROCQ

```
Fixpoint div2 (n : nat) :=
  match n with
  | 0 | 1 => 0
  | S (S n') => (div2 n')+1
  end.

Compute (div2 100). (* = 50 : nat *)
```

Prove specification in additional step:

```
Lemma div2_spec : ∀ n,  ∃ r, r ≤ 1 ∧ 2*div2 n + r = n.
Proof. ... Qed.
```

Computation in Rocq

Extract a witness from a constructive proof:

ROCQ

```
Lemma half_exists : ∀ n, {m | ∃ r, r ≤ 1 ∧ 2*m + r = n}.
Proof.
  apply (well_founded_induction lt_wf).
  intros n H;destruct n.
  exists 0;exists 0;split;auto.        (* n = 0 *)
  destruct n as [| n'].
  exists 0; exists 1;split;auto.       (* n = 1 *)
  destruct (H n') as [m P];[lia|].     (* n  = n'+2*)
  exists (S m).
  destruct P as [r [P1 P2]];exists r; split;auto;lia.
Defined.
Compute (proj1_sig (half_exists 100)). (* = 50 : nat *)
```

Computation in Rocq

Extract a witness from a constructive proof:

ROCQ

```
Lemma half_exists : ∀ n, {m | ∃ r, r ≤ 1 ∧ 2*m + r = n}.
Proof.
  apply (well_f
  intros n H;de
  exists 0;exis
  destruct n as
  exists 0; exi
  destruct (H n') as [m P];[lia|].     (* n  = n'+2*)
  exists (S m).
  destruct P as [r [P1 P2]];exists r; split;auto;lia.
Defined.
Compute (proj1_sig (half_exists 100)). (* = 50 : nat *)
```

Universes **Prop** for logical propositions and **Type** for computational data:

$$\exists x : A,\ P\,x : \text{Prop} \quad \text{(logical existence)}$$

$$\{\, x : A \mid P\,x \,\} : \text{Type} \quad \text{(computational witness)}$$

Computation in Rocq

Extract a witness from a constructive proof:

ROCQ

```
Lemma half_exists : ∀ n, {m | ∃ r, r ≤ 1 ∧ 2*m + r = n}.
Proof.
  apply (well_founded_induction lt_wf).
  intros n H;destruct n.
  exists 0;exists 0;split;auto.        (* n = 0 *)
  destruct n as [| n'].
  exists 0; exists 1;split;auto.       (* n = 1 *)
  destruct (H n') as [m P];[lia|].     (* n  = n'+2*)
  exists (S m).
  destruct P as [r [P1 P2]];exists r; split;auto;lia.
Defined.
Compute (proj1_sig (half_exists 100)). (* = 50 : nat *)
```

Program Extraction

Extraction of Program Code (e.g. Ocaml or Haskell):

ROCQ

```
Lemma half_exists : ∀ n, {m | ∃ r, r ≤ 1 ∧ 2*m + r = n}.
Proof. ... Defined.
Extraction half_exists.
```

Program Extraction

Extraction of Program Code (e.g. Ocaml or Haskell):

ROCQ

```
Lemma half_exists : ∀ n, {m | ∃ r, r ≤ 1 ∧ 2*m + r = n}.
Proof. ... Defined.
Extraction half_exists.
```

OCaml

```
(** val half_exists : nat -> nat **)
let rec half_exists = function
| O -> O
| S n -> (match n with
          | O -> O
          | S n0 -> let s = half_exists n0 in S s)
```

Computation vs. Program Extraction

Computation in Rocq

+ Easy to use
+ Enables checking examples and testing conjectures quickly
+ Can be used directly in formal proofs

− Can be very slow
− Not suited for large computations

Program Extraction

+ Generates stand-alone programs
+ Typically faster
+ Allows integration of extracted code into existing developments

− Needs trust in extraction process
− May still need manual optimization
− Cannot be used directly within proofs
− Additional effort to compile/run

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.
  - Uses sequences of rational numbers (type Q) to encode reals.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.
  - Uses sequences of rational numbers (type Q) to encode reals.
  - Can compute limits of converging sequences with known modulus of convergence and output aribitrarily precise rational approximations.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.
  - Uses sequences of rational numbers (type Q) to encode reals.
  - Can compute limits of converging sequences with known modulus of convergence and output aribitrarily precise rational approximations.
  - Computation is often very inefficient.

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.
  - Uses sequences of rational numbers (type Q) to encode reals.
  - Can compute limits of converging sequences with known modulus of convergence and output aribitrarily precise rational approximations.
  - Computation is often very inefficient.
- Equality defined as equivalence relation (Setoid).

Real Numbers in Rocq

The Rocq standard library:

- Classical real numbers have been included in the standard library for a long time.
- Since Coq 8.11 (2020), the standard library also contains constructive reals as abstract interfaces for constructive and computable real numbers.
- Only one instantiation (ConstructiveCauchyReals) in the standard library.
  - Uses sequences of rational numbers (type Q) to encode reals.
  - Can compute limits of converging sequences with known modulus of convergence and output aribitrarily precise rational approximations.
  - Computation is often very inefficient.
- Equality defined as equivalence relation (Setoid).
  - Requires more complex reasoning: Standard tactics can not be used directly, instances must be shown to respect equality.

Real Numbers in Rocq

CoRN (Constructive Coq Repository at Nijmegen) [1]

- Large library for constructive analysis in Rocq.

[1] Cruz-Filipe, Geuvers & Wiedijk, *C-CoRN, the Constructive Coq Repository at Nijmegen*, Math. Knowledge Manag., 2004.

Real Numbers in Rocq

CoRN (Constructive Coq Repository at Nijmegen) [1]

- Large library for constructive analysis in Rocq.
- Based on metric spaces and the completion monad.

[1] Cruz-Filipe, Geuvers & Wiedijk, *C-CoRN, the Constructive Coq Repository at Nijmegen*, Math. Knowledge Manag., 2004.

Real Numbers in Rocq

CoRN (Constructive Coq Repository at Nijmegen) [1]

- Large library for constructive analysis in Rocq.
- Based on metric spaces and the completion monad.
- Similar abstract formalization using type classes as in the standard library.

[1] Cruz-Filipe, Geuvers & Wiedijk, *C-CoRN, the Constructive Coq Repository at Nijmegen*, Math. Knowledge Manag., 2004.

Real Numbers in Rocq

CoRN (Constructive Coq Repository at Nijmegen) [1]

- Large library for constructive analysis in Rocq.
- Based on metric spaces and the completion monad.
- Similar abstract formalization using type classes as in the standard library.
- However, provides a more fine-grained hierarchy of algebraic structures and more efficient implementations of real numbers.

[1] Cruz-Filipe, Geuvers & Wiedijk, *C-CoRN, the Constructive Coq Repository at Nijmegen*, Math. Knowledge Manag., 2004.

Real Numbers in Rocq

CoRN (Constructive Coq Repository at Nijmegen) [1]

- Large library for constructive analysis in Rocq.
- Based on metric spaces and the completion monad.
- Similar abstract formalization using type classes as in the standard library.
- However, provides a more fine-grained hierarchy of algebraic structures and more efficient implementations of real numbers.
- Steeper learning curve.

[1] Cruz-Filipe, Geuvers & Wiedijk, *C-CoRN, the Constructive Coq Repository at Nijmegen*, Math. Knowledge Manag., 2004.

Real Numbers in Rocq

Some other libraries support verification of approximate or floating-point computations:

- **Flocq** [2]
  - Formalizes IEEE-style floating-point arithmetic.
  - Offers customizable precision, rounding modes, and proofs of rounding error bounds.
  - Useful for verifying numerical algorithms where machine-level floats are employed.

[2] Boldo & Melquiond, *Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq*, Proc. of ARITH, 2011.
[3] Martin-Dorel & Melquiond, *Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq*, J. of AR, 2016.

Real Numbers in Rocq

Some other libraries support verification of approximate or floating-point computations:

- **Flocq** [2]
  - Formalizes IEEE-style floating-point arithmetic.
  - Offers customizable precision, rounding modes, and proofs of rounding error bounds.
  - Useful for verifying numerical algorithms where machine-level floats are employed.
- **Coq-Interval** [3]
  - An efficient implementation of interval arithmetic for intervals with floating-point end points.
  - Automatically translates expressions over classical reals into intervals, and provides tactics to solve inequalities.
  - Works well for common proof obligations.

[2] Boldo & Melquiond, *Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq*, Proc. of ARITH, 2011.
[3] Martin-Dorel & Melquiond, *Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq*, J. of AR, 2016.

cAERN

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.
- Similar operations as in ERC libraries:
  - Exact arithmetic
  - Partial (semi-decidable) comparisons
  - Limit for fast fast Cauchy sequences
  - Usual equality

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.
- Similar operations as in ERC libraries:
  - Exact arithmetic
  - Partial (semi-decidable) comparisons
  - Limit for fast fast Cauchy sequences
  - Usual equality
- Classical reasoning in `Prop` while statements in `Type` remain constructive.

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.
- Similar operations as in ERC libraries:
  - Exact arithmetic
  - Partial (semi-decidable) comparisons
  - Limit for fast fast Cauchy sequences
  - Usual equality
- Classical reasoning in `Prop` while statements in `Type` remain constructive.
- Soundness of axiomatization via a realizability interpretation.

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.
- Similar operations as in ERC libraries:
  - Exact arithmetic
  - Partial (semi-decidable) comparisons
  - Limit for fast fast Cauchy sequences
  - Usual equality
- Classical reasoning in Prop while statements in Type remain constructive.
- Soundness of axiomatization via a realizability interpretation.
- Supports Nondeterminism/Multivalued Computation.

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

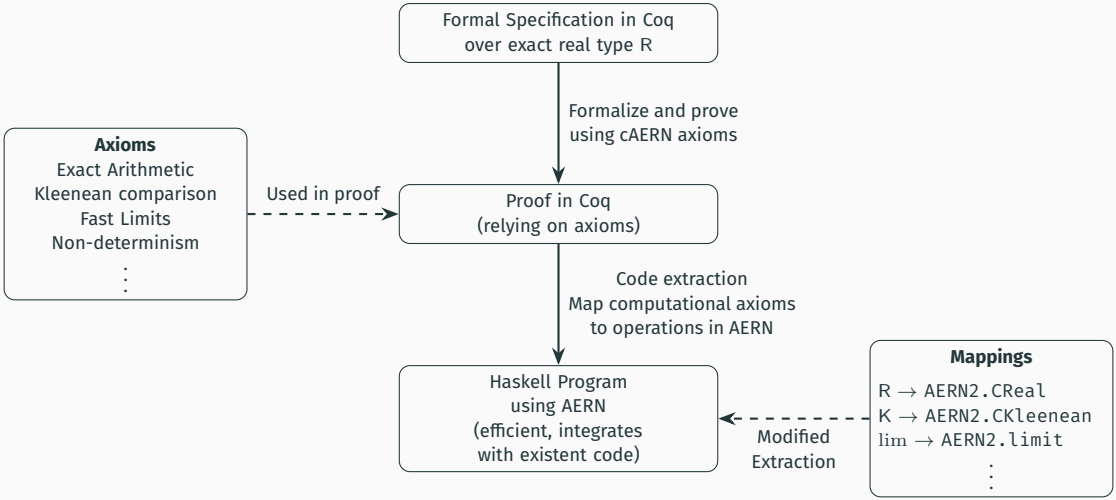
cAERN

cAERN [4]

- Axiomatically defines a real number type, modeling exact real computation.
- Similar operations as in ERC libraries:
  - Exact arithmetic
  - Partial (semi-decidable) comparisons
  - Limit for fast fast Cauchy sequences
  - Usual equality
- Classical reasoning in Prop while statements in Type remain constructive.
- Soundness of axiomatization via a realizability interpretation.
- Supports Nondeterminism/Multivalued Computation.
- Extraction to Haskell, **on top of the AERN library** for efficient exact real computation.

[4] Konečný, Park & T., *Extracting Efficient Exact Real Computation from Proofs*, J. Log. Comput., 2024.

cAERN Overview



Code extraction example

```
Lemma real_max_prop :
  forall x y, {z | (x >= y → z = x)
                 ∧ (x < y → z = y)}.
  Proof.
    intros.
    apply real_mslimit_P_lt.
    + (* max is single-valued *)
      ...
    + (* construct limit *)
      intros.
      apply (mjoin (x>y − prec n)
                   (y>x − prec n)).
      ++ intros [c1|c2].
         +++ (* when x > y − 2^−n *)
         exists x.
         ...
         +++ (* when x < y − 2^−n *)
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

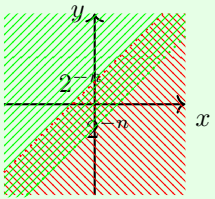
```
real_max_prop ::
     AERN2.CReal ->
     AERN2.CReal ->
     AERN2.CReal
real_max_prop x y =
  AERN2.limit (\n ->
    Prelude.id (\h -> case h of {
                        P.True -> x;
                        P.False -> y})
     (m_split x y (prec n)))
```

Code extraction example

```
Lemma real_max_prop
  forall x y, {z |  (x
                   ∧ (
  Proof.
    intros.
    apply real_mslim
    + (* max is sing
      ...
    + (* construct l
      intros.
      apply (mjoin (x
                   (y
      ++ intros [c1|c
         +++ (* when
         exists x.
         ...
         +++ (* when
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

How to define $\max(x, y)$ without using the representation?

- Apart from primitive operations on reals, cAERN provides a multivalued choice operation. Can be used to defined M-split:



- Use M-split to define a sequence $(z_n)_{n\in\mathbb{N}}$ such that $z_n = x$ if $x - y > 2^{-n}$, $z_n = y$ if $y - x > 2^{-n}$, and any of the two otherwise.
- $(z_n)_{n\in\mathbb{N}}$ is a fast Cauchy sequence converging to $\max(x, y)$.

Code extraction example

```
Lemma real_max_prop :
  forall x y, {z | (x >= y → z = x)
                 ∧ (x < y → z = y)}.
  Proof.
    intros.
    apply real_mslimit_P_lt.
    + (* max is single-valued *)
      ...
    + (* construct limit *)
      intros.
      apply (mjoin (x>y − prec n)
                   (y>x − prec n)).
      ++ intros [c1|c2].
         +++ (* when x > y − 2^−n *)
         exists x.
         ...
         +++ (* when x < y − 2^−n *)
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

```
real_max_prop ::
     AERN2.CReal ->
     AERN2.CReal ->
     AERN2.CReal
real_max_prop x y =
  AERN2.limit (\n ->
    Prelude.id (\h -> case h of {
                        P.True -> x;
                        P.False -> y})
     (m_split x y (prec n)))
```

Code extraction example

```
Lemma real_max_prop :
  forall x y, {z | (x >= y → z = x)
                 ∧ (x < y → z = y)}.
  Proof.
    intros.
    apply real_mslimit_P_lt.
    + (* max is single-valued *)
      ...
    + (* construct limit *)
      intros.
      apply (mjoin (x>y − prec n)
                   (y>x − prec n)).
      ++ intros [c1|c2].
         +++ (* when x > y − 2^−n *)
         exists x.
         ...
         +++ (* when x < y − 2^−n *)
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

```
real_max_prop ::
    AERN2.CReal ->
    AERN2.CReal ->
    AERN2.CReal
real_max_prop x y =
  AERN2.limit (\n ->
    Prelude.id (\h -> case h of {
                        P.True -> x;
                        P.False -> y})
     (m_split x y (prec n)))
```

Code extraction example

```
Lemma real_max_prop :
  forall x y, {z | (x >= y → z = x)
                 ∧ (x < y → z = y)}.
  Proof.
    intros.
    apply real_mslimit_P_lt.
    + (* max is single-valued *)
      ...
    + (* construct limit *)
      intros.
      apply (mjoin (x>y − prec n)
                   (y>x − prec n)).
      ++ intros [c1|c2].
         +++ (* when x > y − 2^−n *)
         exists x.
         ...
         +++ (* when x < y − 2^−n *)
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

```
real_max_prop ::
    AERN2.CReal ->
    AERN2.CReal ->
    AERN2.CReal
real_max_prop x y =
  AERN2.limit (\n ->
    Prelude.id (\h -> case h of {
                        P.True -> x;
                        P.False -> y})
      (m_split x y (prec n)))
```

Code extraction example

```
Lemma real_max_prop :
  forall x y, {z | (x >= y → z = x)
                 ∧ (x < y → z = y)}.
  Proof.
    intros.
    apply real_mslimit_P_lt.
    + (* max is single-valued *)
      ...
    + (* construct limit *)
      intros.
      apply (mjoin (x>y − prec n)
                   (y>x − prec n)).
      ++ intros [c1|c2].
         +++ (* when x > y − 2^−n *)
         exists x.
         ...
         +++ (* when x < y − 2^−n *)
         exists y.
         ...
      ++ apply M_split.
         apply prec_pos.
  Defined.
```

```
real_max_prop ::
    AERN2.CReal ->
    AERN2.CReal ->
    AERN2.CReal
real_max_prop x y =
  AERN2.limit (\n ->
    Prelude.id (\h -> case h of {
                        P.True -> x;
                        P.False -> y})
      (m_split x y (prec n)))
```

cAERN

+ Extracted programs are relatively efficient (comparable to hand-written AERN code).
+ Often more readable than low-level extraction.
+ Extracted code can be combined with non-verified parts of the AERN library.
+ Simple axiomatization, no setoids, proofs similar to using textbook reals.

− No direct computation in the proof assistant.
− Need to trust not only extraction, but also the implementation of ERC operations in AERN.
− (Minor) manual pre-processing of the extracted code is required before running.
− Direct implementation on names can sometimes be easier/more efficient than over generic abstract type.

Ordinary Differential Equations (ODEs)

As a practical application, we consider initial value problems (IVPs) for ordinary differential equations (ODEs). An IVP is an ODE together with an initial condition at

$$\dot{\vec{y}}(t) = \vec{F}(t, \vec{y}(t)), \quad \vec{y}(t_0) = \vec{y}_0, \quad \vec{F} : D \subseteq \mathbb{R}^{d+1} \to \mathbb{R}^d.$$

A solution is a differentiable function $\vec{y} : I \to \mathbb{R}^d$, where $I \subseteq \mathbb{R}$ is an open interval containing t_0, that satisfies the IVP.

Ordinary Differential Equations (ODEs)

As a practical application, we consider initial value problems (IVPs) for ordinary differential equations (ODEs). An IVP is an ODE together with an initial condition at

$$\dot{\vec{y}}(t) = \vec{F}(t, \vec{y}(t)), \quad \vec{y}(t_0) = \vec{y}_0, \quad \vec{F} : D \subseteq \mathbb{R}^{d+1} \to \mathbb{R}^d.$$

A solution is a differentiable function $\vec{y} : I \to \mathbb{R}^d$, where $I \subseteq \mathbb{R}$ is an open interval containing t_0, that satisfies the IVP.

Remarks:

- Higher-order ODEs $\vec{y}^{(k)} = \vec{G}(t, \vec{y}, \ldots, \vec{y}^{(k-1)})$ can be rewritten as a first-order ODEs by introducing additional variables for the higher derivatives $\vec{y}^{(2)}, \ldots, \vec{y}^{(k)}$.
- Any ODE can be turned into an equivalent autonomous ODE by adding t as additional variable.
- By translation, one can assume $t_0 = 0$ and $\vec{y}_0 = \vec{0}$ without loss of generality.

Computability and Complexity of ODEs

Assume $f : [0,1] \times [-1,1] \to \mathbb{R}$ and consider the IVP $\dot{y}(t) = f(t, y(t)); y(0) = 0$

Assumptions on f	**Computability/Complexity of Solution**
(Polynomial-time) computable	All solutions can be non-computable [5]
+ Unique solution	Computable, but complexity can be arbitrarily high [6]
+ Lipschitz continuous	PSPACE complete [7]
+ Analytic	Solution is polynomial-time computable [8]

[5] Aberth, *The failure in computable analysis of a classical existence theorem for differential equations*, 1971.
[6] Ko, *On the computational complexity of ordinary differential equations*, Journal of Computer and System Sciences, 1983.
[7] Kawamura, *Lipschitz Continuous Ordinary Differential Equations are Polynomial-Space Complete*, Comp. Complexity, 2010.
[8] Müller & Moiske, *Solving initial value problems in polynomial time*, Proc. of JAIIO — Panel '93, 1993.

ODE solving in Proof Assistants

The problem seems quite natural for formal verification in a proof assistant:

- It is a classical topic in analysis with many applications in science and engineering.
- Most ODEs do not have a closed-form solution, therefore numerical methods are required to approximate solutions.
- Guaranteeing correctness of these approximations is important for safety-critical applications.

ODE solving in Proof Assistants

The problem seems quite natural for formal verification in a proof assistant:

- It is a classical topic in analysis with many applications in science and engineering.
- Most ODEs do not have a closed-form solution, therefore numerical methods are required to approximate solutions.
- Guaranteeing correctness of these approximations is important for safety-critical applications.

Although other proof assistants (e.g. Isabelle/HOL) include substantial formalizations of ODE theory and numerical methods, very little is available in Rocq so far.

ODE solving in Proof Assistants

The problem seems quite natural for formal verification in a proof assistant:

- It is a classical topic in analysis with many applications in science and engineering.
- Most ODEs do not have a closed-form solution, therefore numerical methods are required to approximate solutions.
- Guaranteeing correctness of these approximations is important for safety-critical applications.

Although other proof assistants (e.g. Isabelle/HOL) include substantial formalizations of ODE theory and numerical methods, very little is available in Rocq so far.

Reasons include:

- Traditionally more focus on discrete math, logic, programming languages, and finite structures.
- Less developed theory and libraries for real analysis (but growing).
- Constructive reasoning natural in Rocq and can be more challenging.

ODE solving in Rocq

Some prior works in Rocq:

- The CoRN library includes an implementation of the Picard Iteration method. [9]

[9] Makarov & Spitters. *The Picard algorithm for ordinary differential equations in Coq*, Proc. of ITP 2013.
[10] Park & T. *A Coq Formalization of Taylor Models and Power Series for Solving ODEs*, Proc. of ITP 2024.

ODE solving in Rocq

Some prior works in Rocq:

- The CoRN library includes an implementation of the Picard Iteration method. [9]
- cAERN has a solver for one-dimensional polynomial ODEs. [10]

[9] Makarov & Spitters. *The Picard algorithm for ordinary differential equations in Coq*, Proc. of ITP 2013.
[10] Park & T. *A Coq Formalization of Taylor Models and Power Series for Solving ODEs*, Proc. of ITP 2024.

ODE solving in Rocq

Some prior works in Rocq:

- The CoRN library includes an implementation of the Picard Iteration method. [9]
- cAERN has a solver for one-dimensional polynomial ODEs. [10]
- A few other works deal with specific systems or algorithms, but do not formalize a general theory.

[9] Makarov & Spitters. *The Picard algorithm for ordinary differential equations in Coq*, Proc. of ITP 2013.
[10] Park & T. *A Coq Formalization of Taylor Models and Power Series for Solving ODEs*, Proc. of ITP 2024.

ODE solving in Rocq

Some prior works in Rocq:

- The CoRN library includes an implementation of the Picard Iteration method. [9]
- cAERN has a solver for one-dimensional polynomial ODEs. [10]
- A few other works deal with specific systems or algorithms, but do not formalize a general theory.

[9] Makarov & Spitters. *The Picard algorithm for ordinary differential equations in Coq*, Proc. of ITP 2013.
[10] Park & T. *A Coq Formalization of Taylor Models and Power Series for Solving ODEs*, Proc. of ITP 2024.

ODE solving in Rocq

Some prior works in Rocq:

- The CoRN library includes an implementation of the Picard Iteration method. [9]
- cAERN has a solver for one-dimensional polynomial ODEs. [10]
- A few other works deal with specific systems or algorithms, but do not formalize a general theory.

New implementation:

- Formalized in the Rocq proof assistant (approx. 10000 lines of code).
- Supports arbitrary dimension.
- Uses type-classes to support different implementations of real numbers.
- Usable both for program extraction and computation inside Rocq.
- More efficient implementation.

[9] Makarov & Spitters. *The Picard algorithm for ordinary differential equations in Coq*, Proc. of ITP 2013.
[10] Park & T. *A Coq Formalization of Taylor Models and Power Series for Solving ODEs*, Proc. of ITP 2024.

The Cauchy-Kovalevskaya Theorem

Theorem (ODE version of the Cauchy-Kovalevskaya Theorem)
Let $U \subseteq \mathbb{R}^d$ be open and $f : U \to \mathbb{R}^d$ analytic. Then the initial value problem

$$\dot{\vec{y}}(t) = \vec{F}(\vec{y}(t)), \quad \vec{y}(0) = \vec{y}_0$$

has a unique solution which is analytic on an open interval $I \subseteq \mathbb{R}$ containing 0.

The Cauchy-Kovalevskaya Theorem

Theorem (ODE version of the Cauchy-Kovalevskaya Theorem)
Let $U \subseteq \mathbb{R}^d$ be open and $f : U \to \mathbb{R}^d$ analytic. Then the initial value problem

$$\dot{\vec{y}}(t) = \vec{F}(\vec{y}(t)), \quad \vec{y}(0) = \vec{y}_0$$

has a unique solution which is analytic on an open interval $I \subseteq \mathbb{R}$ containing 0.

Main idea of the classical proof:

- Expand $\vec{F}(\vec{y})$ in a formal power series around $\vec{y_0}$.
- Derive a recurrence for the Taylor coefficients of $\vec{y}(t)$ from the ODE $\dot{\vec{y}} = \vec{F}(\vec{y})$.
- Prove that the formal series has a positive radius of convergence by dominating it by a geometric majorant sequence.

Formal Power Series Solution

Let

$$\dot{\vec{y}}(t) = \vec{F}(\vec{y}(t)), \quad \vec{y}(0) = \vec{y}_0.$$

We inductively define a sequence of functions $\vec{F}^{[n]} : \mathbb{R}^d \to \mathbb{R}^d$ of multivariate functions by

$$\vec{F}^{[0]}(\vec{x}) = \vec{x}, \text{ and}$$
$$\vec{F}^{[n+1]}(\vec{x}) = J_{\vec{F}^{[n]}}(\vec{x}) \cdot \vec{f}(\vec{x})$$

Define a function by the formal power series

$$\vec{y}(t) = \sum_{i=0}^{\infty} \frac{1}{n!} \vec{F}^{[n]}(\vec{y_0}) t^n$$

By deriving $\vec{y}$ (as formal power series) and comparing coefficients, we see that $\vec{y}$ satisfies the IVP.

Formal Power Series Solution

Let

$$\dot{\vec{y}}(t) = \vec{F}(\vec{y}(t)), \quad \vec{y}(0$$

$$\dot{y}(t) = F(y(t))$$
$$\ddot{y}(t) = \dot{y}(t) \cdot F'(y(t))$$
$$= F(y(t)) \cdot F'(y(t))$$
$$\vdots$$

We inductively define a sequence of functions $\vec{F}^{[n]}$

$$\vec{F}^{[0]}(\vec{x}) = \vec{x}, \text{ and}$$
$$\vec{F}^{[n+1]}(\vec{x}) = J_{\vec{F}^{[n]}}(\vec{x})$$

Define a function by the formal power series

$$\vec{y}(t) = \sum_{i=0}^{\infty} \frac{1}{n!} \vec{F}^{[n]}(\vec{y_0}) t^n$$

By deriving $\vec{y}$ (as formal power series) and comparing coefficients, we see that $\vec{y}$ satisfies the IVP.

The Method of Majorants

- A (single-variate) power series $M(x) = \sum_{k=0}^{\infty} A_k x^k$ is said to majorize a d-variate power series $f(x) = \sum_{\alpha \in \mathbb{N}^d} a_\alpha x^\alpha$ if

$$\forall k \in \mathbb{N}, \quad \sum_{|\alpha|=k} \left|a_\alpha\right| \ \leq \ A_k, \quad \text{where } |\alpha| = \sum_{i=1}^{d} \alpha_i.$$

The Method of Majorants

- A (single-variate) power series $M(x) = \sum_{k=0}^{\infty} A_k x^k$ is said to majorize a d-variate power series $f(x) = \sum_{\alpha \in \mathbb{N}^d} a_\alpha x^\alpha$ if

$$\forall k \in \mathbb{N}, \quad \sum_{|\alpha|=k} \left|a_\alpha\right| \ \leq \ A_k, \quad \text{where } |\alpha| = \sum_{i=1}^{d} \alpha_i.$$

- If R is the radius of convergence of M then f converges absolutely on the polydisc $\left\{\mathbf{x} \in \mathbb{R}^d : \max_{1 \leq i \leq d} |x_i| < R\right\}$.

The Method of Majorants

- A (single-variate) power series $M(x) = \sum_{k=0}^{\infty} A_k x^k$ is said to majorize a d-variate power series $f(x) = \sum_{\alpha \in \mathbb{N}^d} a_\alpha x^\alpha$ if

$$\forall k \in \mathbb{N}, \quad \sum_{|\alpha|=k} \left|a_\alpha\right| \ \leq \ A_k, \quad \text{where } |\alpha| = \sum_{i=1}^{d} \alpha_i.$$

- If R is the radius of convergence of M then f converges absolutely on the polydisc $\left\{\mathbf{x} \in \mathbb{R}^d : \max_{1 \leq i \leq d} |x_i| < R\right\}$.
- We are mostly interested in simple (geometric) majorants of the form $\sum_{k=0}^{\infty} MR^k$ for constant $M, R \in \mathbb{R}$.

The Method of Majorants

- A (single-variate) power series $M(x) = \sum_{k=0}^{\infty} A_k x^k$ is said to majorize a d-variate power series $f(x) = \sum_{\alpha \in \mathbb{N}^d} a_\alpha x^\alpha$ if

$$\forall k \in \mathbb{N}, \quad \sum_{|\alpha|=k} \left|a_\alpha\right| \;\leq\; A_k, \quad \text{where } |\alpha| = \sum_{i=1}^{d} \alpha_i.$$

- If R is the radius of convergence of M then f converges absolutely on the polydisc $\left\{\mathbf{x} \in \mathbb{R}^d : \max_{1 \leq i \leq d} |x_i| < R\right\}$.
- We are mostly interested in simple (geometric) majorants of the form $\sum_{k=0}^{\infty} MR^k$ for constant $M, R \in \mathbb{R}$.
- In that case, the tail of the series satisfies

$$\Big|\sum_{|\alpha|>k} a_\alpha\, x^\alpha\Big| \;\leq\; M \sum_{n=k+1}^{\infty} (R\|x\|)^n \;=\; M\,\frac{(R\|x\|)^{k+1}}{1-R\|x\|}, \quad (R\|x\| < 1).$$

Solution Majorants

- We can show that if the right-hand side function $f : \mathbb{R}^d \to \mathbb{R}^d$ has a geometric majorant (M, R) then $(1, 2dMR)$ works as a geometric majorant for the solution y, guaranteeing convergence of $y(t)$ for all t with $|t| < \frac{1}{2dMR}$.

Solution Majorants

- We can show that if the right-hand side function $f : \mathbb{R}^d \to \mathbb{R}^d$ has a geometric majorant (M, R) then $(1, 2dMR)$ works as a geometric majorant for the solution y, guaranteeing convergence of $y(t)$ for all t with $|t| < \frac{1}{2dMR}$.
- Example:

$$\dot{y}(t) = y(t)^2 \; ; \; y(0) = 1.$$

Solution Majorants

- We can show that if the right-hand side function $f : \mathbb{R}^d \to \mathbb{R}^d$ has a geometric majorant (M, R) then $(1, 2dMR)$ works as a geometric majorant for the solution y, guaranteeing convergence of $y(t)$ for all t with $|t| < \frac{1}{2dMR}$.
- Example:

$$\dot{y}(t) = y(t)^2 \; ; \; y(0) = 1.$$

- The Taylor series around 1 has coefficients $-1, 2, 1$, thus $R = 1, M = 2$ can be used for the bounds.

Solution Majorants

- We can show that if the right-hand side function $f : \mathbb{R}^d \to \mathbb{R}^d$ has a geometric majorant (M, R) then $(1, 2dMR)$ works as a geometric majorant for the solution y, guaranteeing convergence of $y(t)$ for all t with $|t| < \frac{1}{2dMR}$.
- Example:

$$\dot{y}(t) = y(t)^2 \; ; \; y(0) = 1.$$

- The Taylor series around 1 has coefficients $-1, 2, 1$, thus $R = 1, M = 2$ can be used for the bounds.
- We get radius $\frac{1}{4}$ for the solution, while the actual solution $y(t) = \frac{1}{1-t}$ has radius 1.

Solution Majorants

- We can show that if the right-hand side function $f : \mathbb{R}^d \to \mathbb{R}^d$ has a geometric majorant (M, R) then $(1, 2dMR)$ works as a geometric majorant for the solution y, guaranteeing convergence of $y(t)$ for all t with $|t| < \frac{1}{2dMR}$.
- Example:

$$\dot{y}(t) = y(t)^2 \; ; \; y(0) = 1.$$

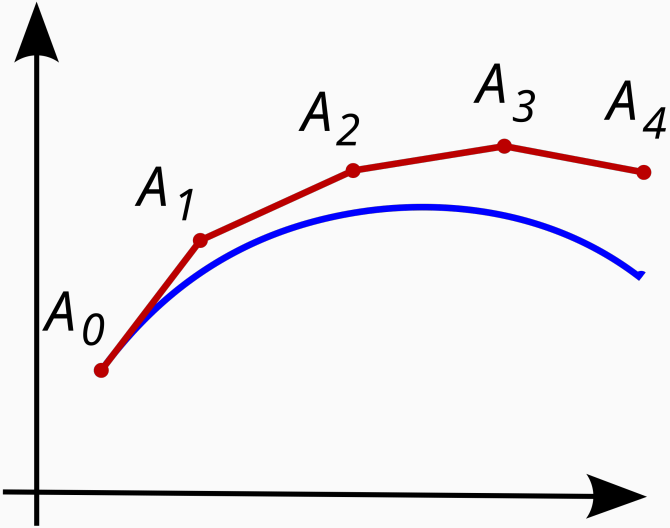
- The Taylor series around 1 has coefficients $-1, 2, 1$, thus $R = 1, M = 2$ can be used for the bounds.
- We get radius $\frac{1}{4}$ for the solution, while the actual solution $y(t) = \frac{1}{1-t}$ has radius 1.
- On the other hand for

$$\dot{y}(t) = \frac{1}{1 - y(t)} = \sum_{k=0}^{\infty} y(t)^k \; ; \; y(0) = 0$$

we get radius $\frac{1}{2}$ which is identical to the actual radius of the solution

$$y(t) = 1 - \sqrt{1 - 2t}.$$

Extending the solution



Experiments

- Usable for for small radius for Rocq's CauchyReals, but quite slow:

ROCQ

```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-10)).
```

Experiments

- Usable for for small radi

```
     = (0, 1 :: nil)
       :: (1 # 8,
           43101704795103564092 9
           # 38037120977794498560 0 :: nil) :: nil
     : list (Q * list Q)
Finished transaction in 0.024 secs (0.023u,0.s) (successful)
```

ROCQ

```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-10)).
```

Experiments

- Usable for for small radius for Rocq's CauchyReals, but quite slow:

ROCQ

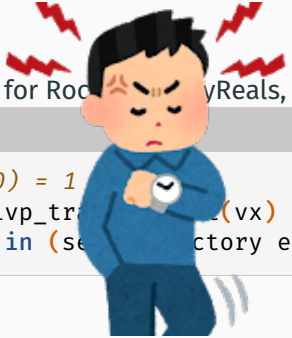
```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-26)).
```

Experiments

- Usable for for small radius for Roc yReals, but quite slow:

ROCQ

```
(* y'(t) = y(t) ; y(0) = 1
Definition exp := (pivp_tr (vx) 0 t(1) 1).
Time Eval vm_compute in (se ctory exp (-26)).
```



Experiments

- Usable for for small radi

ROCQ

```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-26)).
```

```
     = (0, 1 :: nil)
       :: (1 # 8,
           155036963271846663967011156406809236762930435868607765399 3
           # 136819639873526360772777062976511488211263700126924800000 0
           :: nil) :: nil
     : list (Q * list Q)
Finished transaction in 95.896 secs (88.74u,6.45s) (successful)
```

Experiments

- Usable for for small radius for Rocq's CauchyReals, but quite slow:

ROCQ

```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-26)).
```

- A bit faster using CoRN, but still only usable for very small interval.

Experiments

- Usable for for small radius for Rocq's CauchyReals, but quite slow:

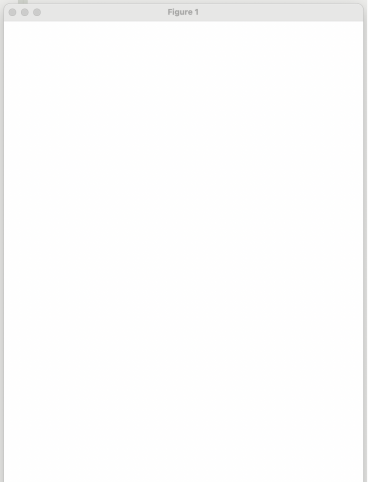
ROCQ

```
(* y'(t) = y(t) ; y(0) = 1 *)
Definition exp := (pivp_trajectory t(vx) 0 t(1) 1).
Time Eval vm_compute in (seq_trajectory exp (-26)).
```

- A bit faster using CoRN, but still only usable for very small interval.
- Much faster using interval arithmetic (without explicit convergence).

Demo

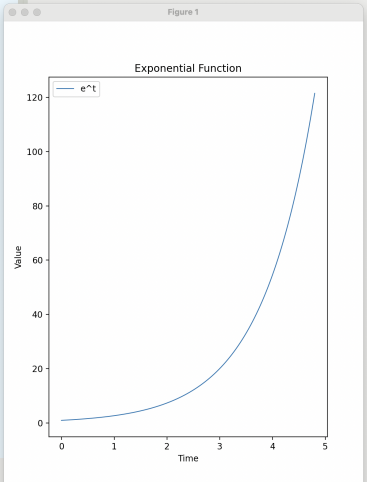
```
(* interval solvers with different parameters *)
Module IIVP3_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 3%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP3_params.

Module IIVP10_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 10%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP10_params.


Module IIVP15_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 15%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP15_params.

Module IIVP3  := IIVP IIVP3_params.
Module IIVP10  := IIVP IIVP10_params.
Module IIVP15  := IIVP IIVP15_params.

(* exponential function y' = y *)
Definition exp_t := (IIVP10.itrajectory t(vx) 1 0 5.0).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).
```



Demo

```
(* interval solvers with different parameters *)
Module IIVP3_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 3%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP3_params.

Module IIVP10_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 10%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP10_params.


Module IIVP15_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 15%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP15_params.

Module IIVP3  := IIVP IIVP3_params.
Module IIVP10  := IIVP IIVP10_params.
Module IIVP15  := IIVP IIVP15_params.

(* exponential function y' = y *)
Definition exp_t := (IIVP10.itrajectory t(vx) 1 0 5.0).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).
```

```
Warning: Declaring a scope implicitly is deprecated; use in advance an
explicit "Declare Scope bigQ_scope.".
[undeclared-scope,deprecated-since-8.10,deprecated,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 4.001 secs (3.611u,0.31s) (successful)
```

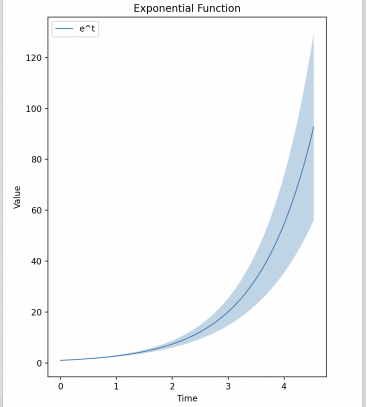


Demo

```
(* interval solvers with different parameters *)
Module IIVP3_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 3%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP3_params.

Module IIVP10_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 10%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP10_params.


Module IIVP15_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 15%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP15_params.

Module IIVP3  := IIVP IIVP3_params.
Module IIVP10  := IIVP IIVP10_params.
Module IIVP15  := IIVP IIVP15_params.

(* exponential function y' = y *)
Definition exp_t := (IIVP3.itrajectory t(vx) 1 0 5.0).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).
```

```
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 2.954 secs (2.604u,0.309s) (successful)
```

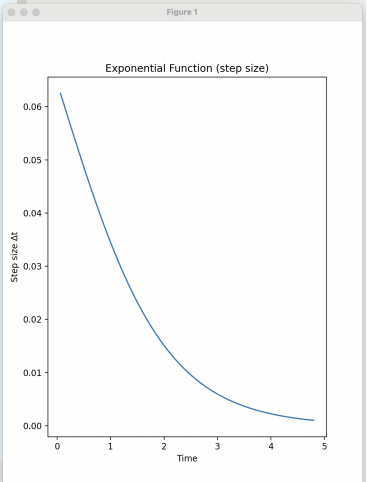
Figure 1



Demo

```
(* interval solvers with different parameters *)
Module IIVP3_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 3%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP3_params.

Module IIVP10_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 10%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP10_params.


Module IIVP15_params  <: IIVP_PARAMS.
  Definition prec := 30%positive. (* interval precision *)
  Definition order := 15%N. (* taylor expansion order *)
  Definition max_steps := 1000%N. (* max number of iterations *)
  Definition step_factor := (Q2Fa 0.25) (* factor of max step size for each step *).
End IIVP15_params.

Module IIVP3  := IIVP IIVP3_params.
Module IIVP10  := IIVP IIVP10_params.
Module IIVP15  := IIVP IIVP15_params.

(* exponential function y' = y *)
Definition exp_t := (IIVP10.itrajectory t(vx) 1 0 5.0).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "step_size"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).
```

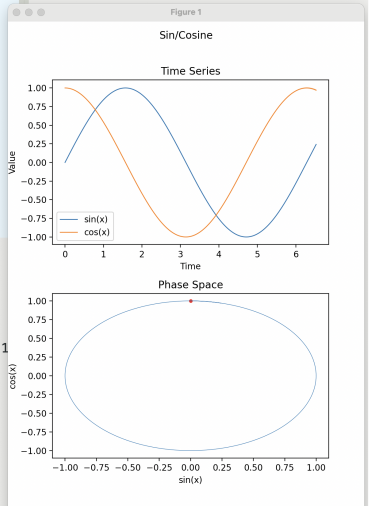
```
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 3.742 secs (3.427u,0.289s) (successful)
```



Demo

```
Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).

(* plot types :
 "time_series",
"phase_space",
"both", "scatter",
"step_size",
"width" *)

(*sin/cosine y1'= y2; y2'=-y1 *)
Definition sin_cos_t := (IIVP10.itrajectory t(vy;-vx) t(0;1) 0 6.5).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "both"
     "Sin/Cosine"
     (("sin(x)" : string) :: ("cos(x)" : string) :: nil)
     sin_cos_t).

(* y1' = y2;
   y2' = mu * (1 - y1^2)y_2 - y_1;
   y0 = (0,0.1) *)
Definition vdp_example := (vdp_ivp  0.5).

Definition vdp_t := (IIVP15.itrajectory vdp_example.(ivp_rhs) vdp_example.(ivp_y0) 0 1
*).


Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "both"
     "Van der Pol Oscillator"
     (("$x$" : string) :: ("$\dot x$" : string) :: nil)
     vdp_t).
```

```
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 1.336 secs (1.19u,0.125s) (successful)
```

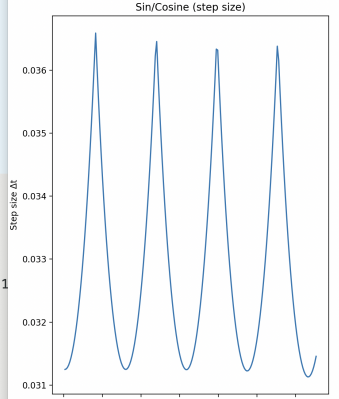


Demo

```
Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Exponential Function"
     (cons ("e^t" : string) nil)
     exp_t).

(* plot types :
 "time_series",
"phase_space",
"both", "scatter",
"step_size",
"width" *)

(*sin/cosine y1'= y2; y2'=-y1 *)
Definition sin_cos_t := (IIVP10.itrajectory t(vy;-vx) t(0;1) 0 6.5).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "step_size"
     "Sin/Cosine"
     (("sin(x)" : string) :: ("cos(x)" : string) :: nil)
     sin_cos_t).

(* y1' = y2;
   y2' = mu * (1 - y1^2)y_2 - y_1;
   y0 = (0,0.1) *)
Definition vdp_example := (vdp_ivp  0.5).

Definition vdp_t := (IIVP15.itrajectory vdp_example.(ivp_rhs) vdp_example.(ivp_y0) 0 1
*).


Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "both"
     "Van der Pol Oscillator"
     (("$x$" : string) :: ("$\dot x$" : string) :: nil)
     vdp_t).
```

```
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 0.768 secs (0.678u,0.089s) (successful)
```

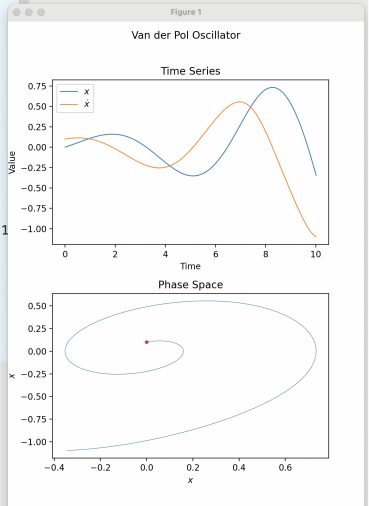
Figure 1



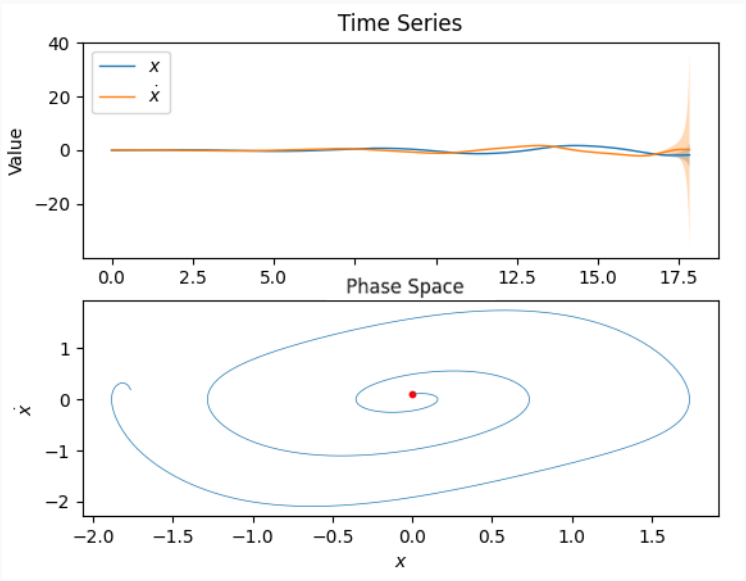
Demo

```
 "time_series",
"phase_space",
"both", "scatter",
"step_size",
"width" *)

(*sin/cosine y1'= y2; y2'=-y1 *)
Definition sin_cos_t := (IIVP10.itrajectory t(vy;-vx) t(0;1) 0 6.5).

Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "time_series"
     "Sin/Cosine"
     (("sin(x)" : string) :: ("cos(x)" : string) :: nil)
     sin_cos_t).

(* y1' = y2;
   y2' = mu * (1 - y1^2)y_2 - y_1;
   y0 = (0,0.1) *)
Definition vdp_example := (vdp_ivp  0.5).

Definition vdp_t := (IIVP15.itrajectory vdp_example.(ivp_rhs) vdp_example.(ivp_y0) 0 1
).


Time Redirect "data" Eval vm_compute in
  (print_trajectory
     "both"
     "Van der Pol Oscillator"
     (("$x$" : string) :: ("$\dot x$" : string) :: nil)
     vdp_t).

(* exact reals *)


Require Import Coq.Reals.Abstract.ConstructiveReals.
From Coq.Reals Require Import ConstructiveCauchyReals.
From Coq.Reals.Cauchy Require Import ConstructiveRcomplete.
Require Import coqreals.
```

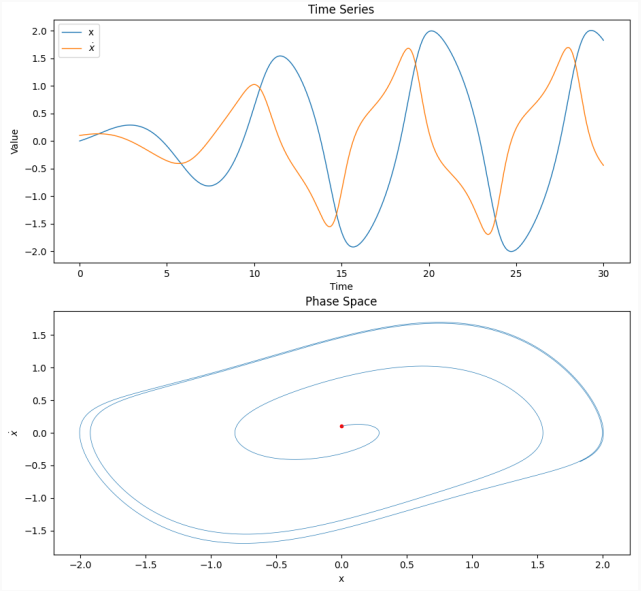
```
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]
Warning: Output directory is unset, using
"/Users/holgerthies/work/taylor_rocqs". Use command line option
"-output-directory to set a default directory.
[default-output-directory,filesystem,default]

Finished transaction in 17.335 secs (17.019u,0.282s) (successful)
```



Demo


Time Series
Value
40
20
0
−20
x
$\dot{x}$
0.0
2.5
5.0
12.5
15.0
17.5
Phase Space
$\dot{x}$
1
0
−1
−2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

Demo



Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.
- Interval version might be better suited for most applications; exact reals might still be useful for computing mathematical constants, etc.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.
- Interval version might be better suited for most applications; exact reals might still be useful for computing mathematical constants, etc.
- In general, efficient computation inside the Rocq proof assistant is challenging.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.
- Interval version might be better suited for most applications; exact reals might still be useful for computing mathematical constants, etc.
- In general, efficient computation inside the Rocq proof assistant is challenging.
- Possible improvements: Better error bounds e.g. by using Taylor's theorem, Taylor models, efficient program extraction,etc.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.
- Interval version might be better suited for most applications; exact reals might still be useful for computing mathematical constants, etc.
- In general, efficient computation inside the Rocq proof assistant is challenging.
- Possible improvements: Better error bounds e.g. by using Taylor's theorem, Taylor models, efficient program extraction,etc.

Summary and Future Work

- Formalized a constructive version of the Cauchy-Kovalevskiya theorem in Rocq, allowing to compute solution to analytic IVPs on small time intervals.
- Additionally, implemented a faster version based on interval computation.
- Interval version might be better suited for most applications; exact reals might still be useful for computing mathematical constants, etc.
- In general, efficient computation inside the Rocq proof assistant is challenging.
- Possible improvements: Better error bounds e.g. by using Taylor's theorem, Taylor models, efficient program extraction,etc.