Rigorous Numerics and Linear Algebra

Fabian Immler

Logroño, November 2017

Technische Universität München

Overview

A Verified ODE Solver

Linear Algebra

Relativization To HOL-Algebra

Case Study

A Verified, Rigorous Numerical ODE Solver

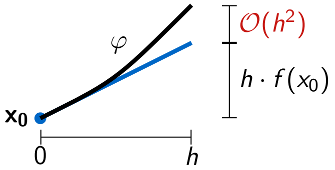
- ODE: $\dot{x}(t) = f(x(t))$
- flow: solution of ODE for initial condition $x(0) = x_0$:

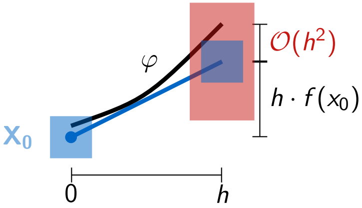
$$\lambda t.\ \phi(x_0, t)$$

- approximation scheme: Runge-Kutta method, e.g.,

$$\phi(x_0, h) = x_0 + h \cdot f(x_0) + \mathcal{O}(h^2)$$

- rigorous numerics: set-based computing





Rigorous Numerics: Real Numbers

- Syntax and Semantics:

$$\begin{aligned} aexp = \;& Add\ aexp\ aexp \\ |\;& Mult\ aexp\ aexp \\ |\;& Minus\ aexp \\ |\;& Inverse\ aexp \\ |\;& Num\ \mathbb{R} \\ |\;& Var\ \mathbb{N} \\ |\;& \ldots \end{aligned}$$

$$\begin{aligned} [\![Add\ a\ b]\!]_{vs} &= [\![a]\!]_{vs} + [\![b]\!]_{vs} \\ [\![Mult\ a\ b]\!]_{vs} &= [\![a]\!]_{vs} \cdot [\![b]\!]_{vs} \\ [\![Minus\ a]\!]_{vs} &= -[\![a]\!]_{vs} \\ [\![Inverse\ a]\!]_{vs} &= 1/[\![a]\!]_{vs} \\ [\![Num\ r]\!]_{vs} &= r \\ [\![Var\ i]\!]_{vs} &= vs\ !\ i \\ &\ldots \end{aligned}$$

- Approximation:

$$approx : aexp \rightarrow interval\ \ list \rightarrow interval$$

$$set\text{-}of\text{-}ivl : interval \rightarrow \mathbb{R}\ set$$

$$(\forall i.\ xs\ !\ i \in XS\ !\ i) \implies [\![e]\!]_{xs} \in set\text{-}of\text{-}ivl\,(approx\ \ e\ \ XS)$$

Rigorous Numerics: Euclidean Space

- Euclidean Space:

$$\textit{eucl-of} : \mathbb{R}\ \textit{list} \to \alpha :: \textit{euclidean-space}$$

$$\textit{es} :: \textit{aexp}\ \ \textit{list}$$

$$[\![\textit{es}]\!]_{\textit{vs}} = \textit{eucl-of}\ (\textit{map}\ (\lambda e.\ [\![e]\!]_{\textit{vs}})\ \textit{es})$$

- Approximation:
  - $\textit{approxs} : \textit{aexp}\ \ \textit{list} \to \textit{interval}\ \ \textit{list} \to \textit{interval}\ \ \textit{list}$
  - $\textit{set-of-ivls} : \textit{interval}\ \ \textit{list} \to \alpha :: \textit{euclidean-space}\ \ \textit{set}$
  - $\textit{approxs}\ \ \textit{es}\ \textit{XS} = \textit{map}\ (\lambda e.\ \textit{approx}\ \ e\ \textit{XS})\ \textit{es}$
  - $(\forall i.\ \textit{xs}\ !\ i \in \textit{XS}\ !\ i) \implies [\![\textit{es}]\!]_{\textit{xs}} \in \textit{set-of-ivls}(\textit{approxs}\ \ \textit{es}\ \textit{XS})$

Rigorous Numerics: "Matrices"

- $A : \mathbb{R}^{n\times m}$, $B : \mathbb{R}^{m\times \ell}$
- $A = \textit{eucl-of as}$, $B = \textit{eucl-of bs}$
- $$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \textit{eucl-of } [a_1, a_2, a_3, a_4]$$
- matrix operations: $A * B = (\textit{eucl-of as}) * (\textit{eucl-of bs})$

  $$\ldots = (\textit{eucl-of } (\textit{mm-mult-lists } m\ n\ \ell\ \textit{as bs}))$$
- *mm-mult-lists* ::

  $$\mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \textit{aexp list} \to \textit{aexp list} \to \textit{aexp list}$$

Therefore

use *approxs* (*mm-mult-lists* ...) ... for rigorous numerical computations of matrices.

Linear Algebra?

(Total) derivatives are linear

- Derivative of $f : \mathbb{R}^n \to \mathbb{R}^m$ at x:

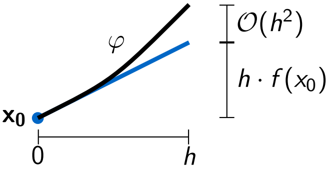
$$\mathrm{D}f|_x : \mathbb{R}^n \to \mathbb{R}^m, \text{ linear}$$

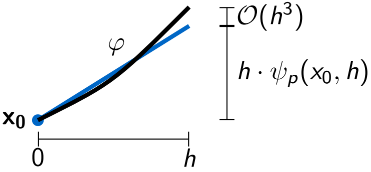
$$f(x+v) \approx f(x) + \mathrm{D}f|_x \cdot v$$

Required for:

- multivariate Taylor series expansion (Runge-Kutta)
- derivative of flow (sensitivity analysis)

Multivariate Taylor Series Expansion





- two-stage Runge-Kutta method:

$$\psi_p(x_0, h) = (1 - \frac{1}{2p})f(x_0) + \frac{1}{2p}f(x_0 + hpf(x_0))$$

- *approx* remainder term $s_1 \in [0, 1]$,$s_2 \in [0, 1]$

$$\mathcal{O}(h^3) = \frac{h^3}{2} \cdot \Big(\frac{1}{3}\big(f''(x(hs_1 + t)) \cdot (f(x(hs_1 + t))) \cdot (f(x(hs_1 + t))) + f'(x(hs_1 + t)) \cdot (f'(x(hs_1 + t)) \cdot (f(x(hs_1 + t))))\big) - \frac{p}{2}f''(x(t) + hps_2 f(x(t))) \cdot (f(x(t))) \cdot (f(x(t)))\Big)$$

Derivative of The Flow

$$\phi(x+v,t) \approx \phi(x,t) + \mathrm{D}\phi_t|_x \cdot v$$

$$\mathrm{D}\phi_t|_x : \mathbb{R}^n \to \mathbb{R}^n \cong \mathbb{R}^{n\times n}$$

Theorem (Variational Equation)

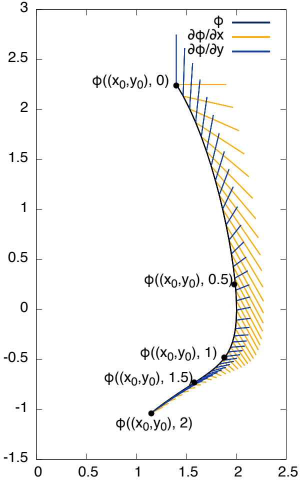
$\mathrm{D}\phi_t|_{x_0} = W_{x_0}(t)$ *for the var.eq.:*

$$\begin{cases} \dot{W}_{x_0}(t) = \mathrm{D}f|_{\phi(x_0,t)} * W_{x_0}(t) \\ W_{x_0}(0) = Id \end{cases}$$

Theorem (Concrete Example)

$$\phi((1.4, 2.25), 2) \in ([0.1; 0.2], [-1.1; -1])$$

$$\mathrm{D}\phi_2|_{(1.4,2.25)} \in \begin{pmatrix} [0.2; 0.3] & [0.4; 0.5] \\ [-0.2; -0.1] & [0.2; 0.3] \end{pmatrix}$$



Representation of Linear Functions

- *typedef* $(\alpha \to_{bl} \beta) = \{f :: \alpha \to \beta \mid \textit{bounded-linear}\ f\}$ for α, β :: *real-normed-vector*
- with $\|f\| = \textit{onorm}\ f = \max_{\|x\| \leq 1} \|f\ x\|$: $(\alpha \to_{bl} \beta)$ :: *real-normed-vector*
- unfortunately for α, β :: *euclidean-space*, $(\alpha \to_{bl} \beta) \not{::}$ *euclidean-space*

Cannot use $(\alpha \to_{bl} \beta)$ for solving the variational equation!

Transfer *euclidean-space*:

For γ, δ :: *euclidean-space* with $\textit{DIM}(\gamma) = \textit{DIM}(\delta)$: $\gamma \cong \delta$

Transfer: $(\alpha \to_{bl} \alpha) \cong (\mathbb{R}^n \to_{bl} \mathbb{R}^n)$ for $\textit{DIM}(\alpha) = n$
And: $(\mathbb{R}^n \to_{bl} \mathbb{R}^n) \cong (\mathbb{R}^{n \times n})$

Solve variational equation for $\mathbb{R}^{n \times n}$ :: *euclidean-space*.

Alternative Representation?

- *typedef* $(\alpha \to_\ell \beta) = \{f :: \alpha \to \beta \mid \mathit{linear}\ f\}$ for α, β :: *euclidean-space*
- with Frobenius norm $\|f\| = \sqrt{\sum_{i \in Basis} \sum_{j \in Basis} .\ (f\ i) \bullet j}$: $(\alpha \to_\ell \beta)$ :: *euclidean-space*

Relativization To HOL-Algebra

Idea/Vision(Johannes): combining the best of two extreme approaches:

- HOL-Algebra/records: explicit structures
- HOL-Analysis/type-classes,locales:

HOL-Algebra / sets / records

- explicit carrier sets
- records for structures

```
record ('a, 'b) module = "'b ring" +
  smult :: "['a, 'b] => 'b" (infixl "⊙ı" 70)

locale module = R?: cring + M?: abelian_group M for M (structure) +
  assumes smult_closed [simp, intro]:
      "[| a ∈ carrier R; x ∈ carrier M |] ==> a ⊙⇘M⇙ x ∈ carrier M"
    and smult_l_distr:
      "[| a ∈ carrier R; b ∈ carrier R; x ∈ carrier M |] ==>
      (a ⊕ b) ⊙⇘M⇙ x = a ⊙⇘M⇙ x ⊕⇘M⇙ b ⊙⇘M⇙ x"
    and smult_r_distr:
      "[| a ∈ carrier R; x ∈ carrier M; y ∈ carrier M |] ==>
      a ⊙⇘M⇙ (x ⊕⇘M⇙ y) = a ⊙⇘M⇙ x ⊕⇘M⇙ a ⊙⇘M⇙ y"
    and smult_assoc1:
      "[| a ∈ carrier R; b ∈ carrier R; x ∈ carrier M |] ==>
      (a ⊗ b) ⊙⇘M⇙ x = a ⊙⇘M⇙ (b ⊙⇘M⇙ x)"
    and smult_one [simp]:
      "x ∈ carrier M ==> 1 ⊙⇘M⇙ x = x"
```

HOL-Analysis / types / type classes / locales

- types for carrier sets
- type classes (whenever possible) and locales for structures

```
locale module =
  fixes scale :: "'a::ring_1 ⇒ 'b::ab_group_add ⇒ 'b"
  assumes scale_right_distrib: "scale a (x + y) = scale a x + scale a y"
    and scale_left_distrib: "scale (a + b) x = scale a x + scale b x"
    and scale_scale: "scale a (scale b x) = scale (a * b) x"
    and scale_one: "scale 1 x = x"
begin
```

- ## Transfer Rules (one per structure):

```
lemma (in monoid) monoid_add_transfer[transfer_rule]:—‹TODO: do this in the monoid-locale?! would a
  includes lifting_syntax
  assumes [transfer_rule]: "right_total R" "bi_unique R"
  shows "((R ===> R ===> R) ===> R ===> op=)
      (λpls zro. monoid ⦇carrier=Collect (Domainp R), mult=pls, one=zro, … = b⦈)
      class.monoid_add"
lemma comm_monoid_add_transfer[transfer_rule]:
  includes lifting_syntax
  assumes [transfer_rule]: "right_total R" "bi_unique R"
  shows "((R ===> R ===> R) ===> R ===> op=)
      (λpls zro. comm_monoid ⦇carrier=Collect (Domainp R), mult=pls, one=zro, … = b⦈)
      class.comm_monoid_add"
lemma group_add_transfer[transfer_rule]:
  includes lifting_syntax
  assumes [transfer_rule]: "right_total R" "bi_unique R"
  shows "((R ===> R ===> R) ===> (R ===> R ===> R) ===> R ===> (R ===> R) ===> op=)
      (λminus pls zro uminus. group ⦇carrier=Collect (Domainp R), mult=pls, one=zro, … = b⦈ ∧
         (∀x∈Collect (Domainp R). uminus x = inv⇘⦇carrier=Collect (Domainp R), mult=pls, one=zro, …
         (∀x∈Collect (Domainp R). ∀y∈Collect (Domainp R). minus x y = pls x (uminus y)))
      class.group_add"
```

- ## implicit theorem:

```
lemma inv_unique_class: "y = y'" if "y + x = 0" "x + y' = 0" for x y y'::"'a::monoid_add"
```

- ## explicit theorem:

```
lemma (in monoid) inv_unique_easy:
  assumes eq: "y ⊗ x = 1" "x ⊗ y' = 1"
    and G: "x ∈ carrier G" "y ∈ carrier G" "y' ∈ carrier G"
  shows "y = y'"
```

Boilerplate Code...

```
lemma inv_unique_class: "y = y'" if "y + x = 0" "x + y' = 0" for x y y'::"'a::monoid_add"
  by (metis add.left_neutral add.right_neutral add.semigroup_axioms semigroup.assoc that(1) that(2))

lemmas internalized_sort = inv_unique_class[internalize_sort "'a::monoid_add"]
lemmas dictionary_second_step = internalized_sort[unoverload plus, unoverload Groups.zero]

lemma (in monoid) inv_unique_easy:
  assumes eq: "y ⊗ x = 1" "x ⊗ y' = 1"
    and G: "x ∈ carrier G" "y ∈ carrier G" "y' ∈ carrier G"
  shows "y = y'"
proof -
  from one_closed have ne: "carrier G ≠ {}" by blast
  {
    assume T: "∃(Rep :: 'boo ⇒ 'a) Abs. type_definition Rep Abs (carrier G)"
    from T obtain rep :: "'boo ⇒ 'a" and abs :: "'a ⇒ 'boo" where t: "type_definition rep abs (carrier G)"
      by auto

    text‹Setup for the Transfer tool.›
    define cr_b where "cr_b == λr a. r = rep a"
    note type_definition_Domainp[OF t cr_b_def, transfer_domain_rule]
    note typedef_right_total[OF t cr_b_def, transfer_rule]
    note typedef_bi_unique[OF t cr_b_def, transfer_rule]
    note typedef_right_unique[OF t cr_b_def, transfer_rule]
    note typedef_left_unique[OF t cr_b_def, transfer_rule]

    have G_eq: "⦇carrier = carrier G, monoid.mult = op ⊗, one = 1, … = monoid.more G⦈ = G"
      by auto
    have ?thesis
      text‹Relativization by the Transfer tool.›
      using dictionary_second_step[where 'a = 'boo, untransferred, where plus="mult G" and zero="one G"
          and b = "monoid.more G", of y x y']
        G monoid_axioms eq
      by (auto simp: G_eq)
  } note this[cancel_type_definition, OF ne]
  then show ?thesis .
qed
```

- implicit theorem:

```
lemma diff_add_eq_diff_diff_swap: "a - (b + c) = a - c - b"
```

- explicit theorem:

```
lemma (in group) diff_add_eq_diff_diff_swap:
  assumes G: "a ∈ carrier G"  "b ∈ carrier G"  "c ∈ carrier G"
  shows "a ⊗ inv (b ⊗ c) = (a ⊗ inv c) ⊗ inv b"
```

Boilerplate Code...

```
lemmas diff_diff_eq_is = diff_add_eq_diff_diff_swap[internalize_sort "'a::group_add"]
lemmas diff_diff_eq_uo = diff_diff_eq_is[unoverload plus, unoverload Groups.zero, unoverload uminus, unoverload minus]

lemma (in group) diff_add_eq_diff_diff_swap:
  assumes G: "a ∈ carrier G"  "b ∈ carrier G"  "c ∈ carrier G"
  shows "a ⊗ inv (b ⊗ c) = (a ⊗ inv c) ⊗ inv b"
proof -
  from one_closed have ne: "carrier G ≠ {}" by blast
  {
    assume T: "∃(Rep :: 'boo ⇒ 'a) Abs. type_definition Rep Abs (carrier G)"
    from T obtain rep :: "'boo ⇒ 'a" and abs :: "'a ⇒ 'boo" where t: "type_definition rep abs (carrier G)"
      by auto

    text‹Setup for the Transfer tool.›
    define cr_b where "cr_b == λr a. r = rep a"
    note type_definition_Domainp[OF t cr_b_def, transfer_domain_rule]
    note typedef_right_total[OF t cr_b_def, transfer_rule]
    note typedef_bi_unique[OF t cr_b_def, transfer_rule]
    note typedef_right_unique[OF t cr_b_def, transfer_rule]
    note typedef_left_unique[OF t cr_b_def, transfer_rule]

    have G_eq: "⦇carrier = {x. x ∈ carrier G}, monoid.mult = op ⊗, one = 1, … = monoid.more G⦈ = G"
      by auto
    have ?thesis
      text‹Relativization by the Transfer tool.›
      using diff_diff_eq_uo[where 'a = 'boo, untransferred, where plus="mult G" and zero="one G"
          and b = "monoid.more G", where minus ="λx y. x ⊗ inv y" and uminus = "λx. inv x",
          of a b c]
        G group_axioms
      unfolding G_eq
      by (auto simp:)
  } note this[cancel_type_definition, OF ne]
  then show ?thesis .
qed
```

More automation for locale $\rightsquigarrow$ record?

Thank you!