My Taylor is rich

FEANICSES workshop

Xavier Thirioux
IRIT/INPT

may 2018

Outline

Taylor Approximations

Context: static analysis of plant-controller systems

- Provide approximations of non-polynomial functions

Goal: flexible framework for an algebra of Taylor expansions

- Multivariate case (any dimension)
- Certified errors
- Independent value and error domain
- On-demand refinable approximations (any order)
- Support integral/differential operators (physical-level invariants)
- Solve differential equations (fixed points)

Some Relevant Works

- J. Karczmarczuk: "Functional Differentiation of computer Programs". 1D Taylor expansions, refinable, without errors.
- E. Martin-Dorel: "Certified, Efficient and Sharp Univariate Taylor Models in COQ". 1D Taylor expansions, not refinable, with errors.
- K. Makino & M. Berz: "Rigourous analysis of nonlinear motion in particle accelerators": 1D Taylor expansions, not refinable, with errors and ODEs.
- *FLOW**: `https://flowstar.org/`: "A verification tool for cyber-physical systems".
- Automatic differentiation (<u>forward</u>/backward modes).
- Many (mostly C++) libraries for (symmetric) tensors of arbitrary rank and dimension.

Design Choices

Concerns

- Emphasis on correction: mostly functional implementation, strong properties statically enforced
- Highly modular design with clear algebraic signatures

Some Choices

- Symmetric tensors algebra as building block
- Expansions specialized at point $\mathbf{0}$ with $\mathbf{0}$-centered errors
- Convolution for fast product
- Error **functions** to reuse same expansion anywhere around $\mathbf{0}$
- Laziness to compute Taylor expansions on demand
- Specialized Taylor expansion and error for exact polynomial functions

Implementation Language: Ocaml

Features

- Correct usage of data-structures, as regards size, dimension, order, etc
- Various algebras through module system: value/error, symmetric tensor, Taylor expansion (1D, standard, refined)
- Intimate blending of proofs and programs

[1] Generalized Algebraic Data Types

Implementation Language: Ocaml

Features

- Correct usage of data-structures, as regards size, dimension, order, etc
- Various algebras through module system: value/error, symmetric tensor, Taylor expansion (1D, standard, refined)
- Intimate blending of proofs and programs

Helps focusing on numerical concerns: correctness of approximation, precision, convergence

[1] Generalized Algebraic Data Types

Implementation Language: Ocaml

Features

- Correct usage of data-structures, as regards size, dimension, order, etc
- Various algebras through module system: value/error, symmetric tensor, Taylor expansion (1D, standard, refined)
- Intimate blending of proofs and programs

Helps focusing on numerical concerns: correctness of approximation, precision, convergence

Why not use a proof-assistant ?

- With GADT[1], (arithmetical) proofs can be embedded in Ocaml
- Sparse imperative features, for the sake of efficiency
- Designed as a library, not an end-user product
- Doesn't cope so well with laziness

[1] Generalized Algebraic Data Types

A Glimpse of Type-level Arithmetics

From a client's viewpoint (sample properties)

- "Adding two tensors of order R yields a tensor of order R".
- "Multiplying tensors of respective orders R_1 and R_2 yields a tensor of order R such that $R = R_1 + R_2$".

For the developer

- Every size/dimension/order information of data structures is reflected in their types.
- Dimension-related proofs are computed along regular values.
- Negligible computational cost (in our application).
- Useful to guide the design of complex recursive algorithms.
- Removal of useless/impossible cases in pattern-matching.
- In theory, a compiler could remove proofs from generated code.

Multivariate Taylor Expansion

- Canonical presentation of a Taylor expansion at order R in dimension N:

$$f(\mathbf{x}) = \sum_{|\alpha|<R} \mathbf{D}_f^{\alpha}(\mathbf{0}) \odot \frac{\mathbf{x}^{\alpha}}{\alpha!} + \sum_{|\alpha|=R} \mathbf{D}_f^{\alpha}(\epsilon * \mathbf{x}) \odot \frac{\mathbf{x}^{\alpha}}{\alpha!}$$

$$\mathbf{x} \in \mathbb{R}^N \qquad \alpha \in \mathbb{N}^N \qquad \epsilon \in [0,1]$$

- Not the standard "tensorized" version, where N-dimensional expansions are 1D expansions which coefficients are $N-1$-dimensional expansions, etc
- For practicality and efficiency

Multivariate Taylor Model

- Taylor model derived from Taylor remainder:

$$| \, f(\mathbf{x}) - \sum_{|\alpha| \leqslant R} \mathbf{D}_f^{\alpha}(\mathbf{0}) \odot \frac{\mathbf{x}^{\alpha}}{\alpha!} \, | \leqslant \sum_{|\alpha| = R} \boldsymbol{\epsilon}_f^{\alpha}(\mathbf{x}) \odot \frac{|\mathbf{x}^{\alpha}|}{\alpha!}$$

- With a bounding error (tensor) such that, for any $\epsilon \in [0, 1]$:

$$| \, \mathbf{D}_f^{\alpha}(\epsilon * \mathbf{x}) - \mathbf{D}_f^{\alpha}(\mathbf{0}) \, | \leqslant \boldsymbol{\epsilon}_f^{\alpha}(\mathbf{x})$$

$\rightarrow$ Symmetric tensors of $(\mathit{value}, \mathit{error\ function})$ couples: $(\mathbf{D}_f^{\alpha}(\mathbf{0}), \boldsymbol{\epsilon}_f^{\alpha})$

Values & Errors

Values

- Could be any numerical domain, **but** operations we intend to support on Taylor expansions should be reflected.
- FP numbers, complex numbers, certified FP numbers (intervals), etc.

Errors

- The same error function may occur many times
  $\rightarrow$ memoization.
- Specialized zero errors simplify computations
  $\rightarrow$ special case in data-structure.

Values & Errors

Domain atoms

$$\begin{array}{ll} \text{constants}: & (k, (x_0, \ldots, x_{N-1}) \mapsto 0) \\ \text{variables}: & (0, (x_0, \ldots, x_{N-1}) \mapsto |x_i|) \end{array}$$

Domain operators

$$\begin{aligned}
&(v_1, \epsilon_1) + (v_2, \epsilon_2) \triangleq (v_1 + v_2, \epsilon_1 + \epsilon_2) \\
&(v_1, \epsilon_1) - (v_2, \epsilon_2) \triangleq (v_1 - v_2, \epsilon_1 + \epsilon_2) \\
&\alpha \times (v, \epsilon) \triangleq (\alpha \times v, \alpha \times \epsilon) \\
&(v_1, \epsilon_1) \times (v_2, \epsilon_2) \triangleq (v_1 \times v_2, v_1 \times \epsilon_2 + v_2 \times \epsilon_1 + \epsilon_1 \times \epsilon_2) \\
&e^{(v,\epsilon)} \triangleq (e^v, e^v \times (e^\epsilon - 1)) \\
&\log(v, \epsilon) \triangleq (\log v, \log(1 + \tfrac{\epsilon}{v})) \\
&\sin(v, \epsilon) \triangleq (\sin v, |\sin v| \times (1 - \cos \epsilon) + |\cos v| \times |\sin \epsilon|) \\
&\cos(v, \epsilon) \triangleq (\cos v, |\cos v| \times (1 - \cos \epsilon) + |\sin v| \times |\sin \epsilon|) \\
&\ldots
\end{aligned}$$

Symmetric Tensors

Generalities

- Symmetric tensor = homogeneous polynomial.
- Occurrences vs indices representation → ordered.indices

  $\mathbf{S}_{i_1,\ldots,i_R} = \mathbf{S}_{o_0,\ldots,o_{N-1}}$, such that $o_k = \#\{j \mid i_j = k\}$.

  $$\mathbf{S}(X_0,\ldots,X_{N-1}) = \sum_{\sum_{k=0}^{N-1} o_k = R} \mathbf{S}_{o_0,\ldots,o_{N-1}} \times X_0^{o_0} \times \ldots \times X_{N-1}^{o_{N-1}}$$

- (N,R) tensors form a $\binom{N+R-1}{R}$ vector space.
- (N,R) tensors form a R-graded N-dimensional algebra.
- *Binary Decision Diagram* -like recursive scheme:

$$\mathbf{S}(X_0,\ldots,X_{N-1}) \;=\; \mathbf{S}_0(X_0,\ldots,X_{N-2}) + X_{N-1}.\mathbf{S}_1(X_0,\ldots,X_{N-1})$$

Symmetric Tensors

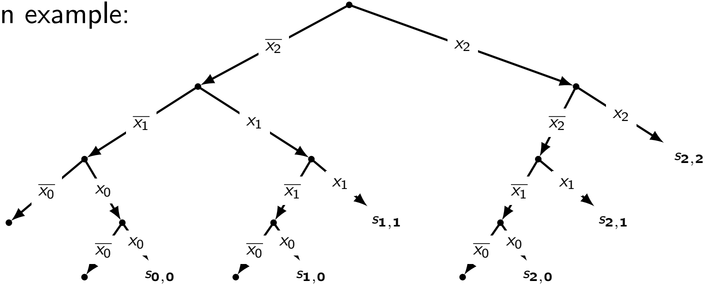
Data-structure

- Recursive scheme:

$$
\begin{aligned}
ST(0, R) &= \varnothing \\
ST(N+1, 0) &= \mathbb{V} \times \mathbb{E} \\
ST(N+1, R+1) &= ST(N, R+1) \\
&\quad \times ST(N+1, R)
\end{aligned}
$$

```
type ('a, _, _) st =
 | Nil:     ('a, Nat.zero, 'r) st
 | Leaf:    'a
         -> ('a, 'n Nat.succ, Nat.zero) st
 | Node:    ('a, 'n, 'r Nat.succ) st
         *  ('a, 'n Nat.succ, 'r) st
         -> ('a, 'n Nat.succ, 'r Nat.succ) st
```

- An example:



$$X_2(X_2 s_{2,2} + X_1 s_{2,1} + X_0 s_{2,0}) + X_1(X_1 s_{1,1} + X_0 s_{1,0}) + X_0 X_0 s_{0,0}$$

Operations on Symmetric Tensors

Simple structural operations

- Functorial operations.
- Linear operations.

```
let rec map : type n r. ('a -> 'b) -> ('a, n, r) st -> ('b, n, r) st =
  fun f st ->
  match st with
  | Nil             -> Nil
  | Leaf v          -> Leaf (f v)
  | Node (stl, str) -> Node (map f stl, map f str)

let rec apply : type n r. ('a -> 'b, n, r) st -> ('a, n, r) st -> ('b, n, r) st =
  fun stf sta ->
  match stf, sta with
  | Nil              , Nil               -> Nil
  | Leaf f           , Leaf a            -> Leaf (f a)
  | Node (stfl, stfr), Node (stal, star) -> Node (apply stfl stal,
                                                  apply stfr star)

let sum st1 st2 = apply (map R.( + ) st1) st2

let hadamard st1 st2 = apply (map R.( * ) st1) st2
```

Operations on Symmetric Tensors

Tensor product

- Implemented with side-effects for efficiency.
- Optimal complexity: $\theta((R_1 \times R_2)^N)$.
- Two functions ($\mathbf{S} = \mathbf{S}_0 + X_{N-1}.\mathbf{S}_1$):
$$\begin{cases} \mathbf{S} \times \mathbf{T} \triangleq \mathbf{S}_0 \times' \mathbf{T} + X_{N-1}.(\mathbf{S}_1 \times \mathbf{T}) \\ \mathbf{S} \times' \mathbf{T} \triangleq \mathbf{S} \times \mathbf{T}_0 + X_{N-1}.(\mathbf{S} \times' \mathbf{T}_1) \end{cases}$$

Operations on Symmetric Tensors

Order-changing operations

- Instantiation (fixing an index to k):

$$\blacksquare\,[\,\blacksquare\,] : (N+1, R+1)\,ST \to k \leqslant N \to (N+1, R)\,ST$$
$$\mathbf{S}[N-1] \triangleq \mathbf{S}_1$$
$$\mathbf{S}[k] \triangleq \mathbf{S}_0[k] + X_{N-1}.\mathbf{S}_1[k], \quad \text{for } k < N-1$$

- Generalization (multiplication by X_k):

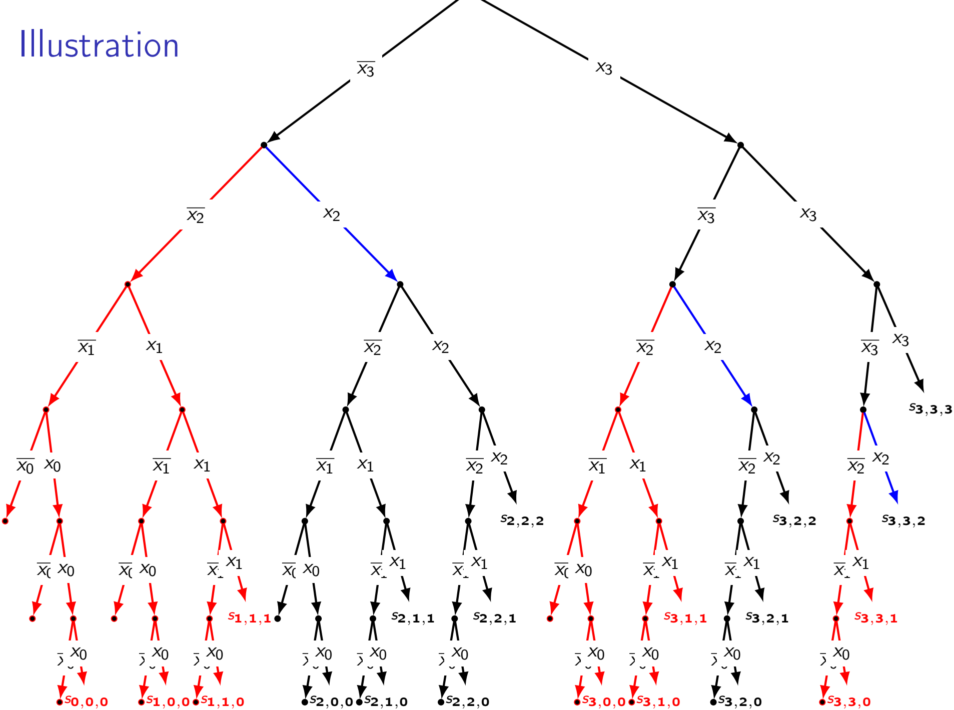
$$\blacksquare \uparrow \blacksquare : (N+1, R)\,ST \to k \leqslant N \to (N+1, R+1)\,ST$$
$$\mathbf{S}\uparrow(N-1) \triangleq \mathbf{0} + X_{N-1}.\mathbf{S}$$
$$\mathbf{S}\uparrow k \triangleq \mathbf{S}_0\uparrow k + X_{N-1}.(\mathbf{S}_1\uparrow k), \quad \text{for } k < N-1$$

Operations on Symmetric Tensors

Order-changing operations

- With an auxiliary coefficient tensor:
  $(\Delta_k)_{o_0,\ldots,o_{N-1}} \triangleq 1 + o_k$, for $\sum_i o_i = R$
- Integration: $\int_0^{X_k} \mathbf{S}(X_0, \ldots, x_k, \ldots, X_{N-1}) dx_k \triangleq (\mathbf{S} \odot \Delta_k^{-1}) \uparrow k$
- Differentiation: $\frac{d\mathbf{S}(X_0,\ldots,X_{N-1})}{dX_k} \triangleq \mathbf{S}[k] \odot \Delta_k$

Illustration



Order-Changing Operations

```
let rec set : type n d k r. (d, k, n) Nat.add ->
  (R.t, n Nat.succ, r Nat.succ) st -> (R.t, n Nat.succ, r) st =
  fun pr st ->
    match pr, st with
    | Nat.Zadd    , Node (stl, str) -> str
    | Nat.Sadd pr', Node (stl, str) ->
      match str with
      | Node _ -> Node (set pr' stl, set pr str)
      | Leaf _ -> match set pr' stl with | Leaf v -> Leaf v

let rec lift : type n d k r. r Nat.isnat -> k Nat.isnat -> (d, k, n) Nat.add ->
  (R.t, n Nat.succ, r) st -> (R.t, n Nat.succ, r Nat.succ) st =
  fun r k pr st ->
    match pr, st with
    | Nat.Zadd    , _               -> Node (make k (Nat.S r) R.zero, st)
    | Nat.Sadd pr', Leaf v          -> Node (lift r k pr' (Leaf v), Leaf R.zero)
    | Nat.Sadd pr', Node (stl, str) -> Node (lift r k pr' stl,
                                             lift (Nat.pred r) k pr str)
```

Operations on Symmetric Tensors

Error refinement

- Basis rotation ($\mathbf{S}(X_0, X_1, \ldots, X_{N-1}) \mapsto \mathbf{S}(X_1, X_2, \ldots, X_0)$):

$$\circlearrowleft \mathbf{S} \triangleq \circlearrowleft (\mathbf{S}_0 + X_{N-1}.\mathbf{S}_1) = \mathbf{S}_0(X_1, \ldots, X_{N-1}) + X_0. \circlearrowleft \mathbf{S}_1$$

$\rightarrow$ Useful in case of $\neq$ error magnitudes along $\neq$ dimensions. Helps balancing these differences each other out

- Reduction and partial reduction ($\sum_i r_i = k$, $\sum_i o_i = R$):

$$(\Sigma^k \mathbf{S})_{(r_0,\ldots,r_{N-1})} \triangleq \sum_{(o_0,\ldots,o_{N-1}) \geqslant (r_0,\ldots,r_{N-1})} \mathbf{S}_{(o_0,\ldots,o_{N-1})}$$

where $(o_0, \ldots, o_{N-1}) \geqslant (r_0, \ldots, r_{N-1})$ is the sub-tree ordering

$\rightarrow$ Cuts off higher-order terms by turning them into pure errors (zero-valued tensors).

Taylor Expansions

Generalities

- Power series as streams of tensors $f_Z(Z) \triangleq \sum_{r \in \mathbb{N}} \mathbf{T}_r.Z^r$:

  where $\mathbf{T}_r \simeq \sum_{|\alpha|=r} (\frac{\mathbf{D}_f^{\alpha}(\mathbf{0})}{\alpha!}, \frac{\mathbf{D}_f^{\alpha}(\lambda * \mathbf{x})}{\alpha!}).\mathbf{x}^{\alpha} \simeq \mathbf{T}_r^v, \mathbf{T}_r^{\epsilon}$
- Taylor models $\mathcal{TM}(f, R, \epsilon)$:

$$\forall \mathbf{x} \in \mathbb{R}^N. |\mathbf{x}| \leqslant \boldsymbol{\epsilon} \Rightarrow |f(\mathbf{x}) - \sum_{r=0}^{R} \mathbf{T}_r^v \odot \mathbf{x}^r| \leqslant \Sigma^0(\mathbf{T}_R^{\epsilon}(\boldsymbol{\epsilon}) \odot |\mathbf{x}|^R)$$

Operations on Taylor Expansions

Causality requirement

- Every operator has to be causal, i.e. its n-th order Taylor expansion depends at most on n-th order parts of its arguments
- Natural for derivative part (wrt typical derivation formulas).
- Not so natural for error part (incentive for accurate errors).
- Mandatory for: reliability (cost prediction), solving PDEs, etc.
- Errors can be refined afterwards.

Principal operations

- Linear operations.
- Product, division ($\rightarrow$ convolution).
- Composition with elementary functions.
- Differentiation, integration.

Taylor Expansions

Composition

- $(f \circ g)$, where f is $1D$ and $g(\mathbf{0}) = 0$:

$$f \circ g \triangleq \sum_{r \in \mathbb{N}} f_r . g^r = \sum_{r \in \mathbb{N}} \mathbf{T}_r . Z^r$$

- g is (at least) of order 1 and then g^r is of order r. So:

$$\mathbf{T}_r^v = \sum_{k=0}^{r} f_k^v . [Z^r] g^k$$

  where $[Z^r]g^k$ is the r-th order tensor of g^k.
- Many ways to build an error term $\mathbf{T}_r^\epsilon$ ...
- For instance, $\mathbf{T}_r^\epsilon = \mathbf{f}_r^\epsilon \circ g_k$, where

  $g_k(\boldsymbol{\epsilon}) \triangleq |\sum_{r=0}^{k} \mathbf{T}_r^v \odot \boldsymbol{\epsilon}^r| + \Sigma^0(\mathbf{T}_k^\epsilon(\boldsymbol{\epsilon}) \odot |\boldsymbol{\epsilon}|^k)$

Taylor Expansions

1D series of elementary functions

- Factorize out constant part (evaluation at point **0**).
- Compute the n-th derivative in the Value-Error domain, which yields:
  - a value at **0**.
  - an error function.
- Examples:

- $\exp(x_0 + x') = \exp(x_0)\exp(x')$
  $\mathbf{D}^{r}_{\exp}(\epsilon) = \exp(\epsilon)$
- $\log(x_0 + x') = \log(x_0) + \log(1 + \frac{x'}{x_\mathbf{0}})$
  $\mathbf{D}^{r+1}_{\log}(\epsilon) = -(r+1)! * (\frac{-1}{1+\epsilon})^{r+1}$
- $\sin(x_0 + x') = \sin(x_0)\cos(x') + \cos(x_0)\sin(x')$
  $\mathbf{D}^{2r}_{\sin}(\epsilon) = (-1)^r \sin(\epsilon)$
- $\mathrm{atan}(x_0 + x') = \mathrm{atan}(x_0) + \mathrm{atan}(\frac{x'}{1+x_\mathbf{0}x'})$
  $\mathbf{D}^{r+2}_{\mathrm{atan}}(\epsilon) = (-2(r+1)\epsilon\mathbf{D}^{r+1}_{\mathrm{atan}}(\epsilon) - r * (r+1) * \mathbf{D}^{r}_{\mathrm{atan}}(\epsilon))/(1+\epsilon^2)$

Taylor Expansions

Disciplined convolution

- Efficient product:

$$\Big(\sum_{r\in\mathbb{N}} \mathbf{T}_r.Z^r\Big) \times \Big(\sum_{r\in\mathbb{N}} \mathbf{S}_r.Z^r\Big) = \sum_{r\in\mathbb{N}} \Big(\sum_{i\in\mathbb{N}} \mathbf{T}_i \times \mathbf{S}_{r-i}\Big) Z^r$$

- Convolution structure $(l_1 + l_2 = \ldots = r_1 + r_2 = r)$:

$\mathbf{T}_{l_1}$	$\mathbf{T}_{l_1+1}$	$\ldots$	$\mathbf{T}_{r_1}$
$\mathbf{S}_{l_2}$	$\mathbf{S}_{l_2-1}$	$\ldots$	$\mathbf{S}_{r_2}$

- Column-wise tensor products give all additive contributions to order r tensor.
- Adding elements $(r \;\rightarrow\; r+1)$:

	$\mathbf{T}_{l_1}$	$\mathbf{T}_{l_1+1}$	$\ldots$	$\mathbf{T}_{r_1}$	$\mathbf{T}_{r_1+1}$
$\mathbf{S}_{l_2+1}$	$\mathbf{S}_{l_2}$	$\mathbf{S}_{l_2-1}$	$\ldots$	$\mathbf{S}_{r_2}$	

Solving PDE with Picard-Lindelöf theorem

- For an equation in solved-form:

$$f(x) = f(0) + \int_0^x expr(h, f(h))dh$$

The following sequence of iterates:

$$\phi_0(x) = 0, \phi_{n+1}(x) = F(\phi_n)(x) = f(0) + \int_0^x expr(h, \phi_n(h))dh$$

converges to a solution of the equation.

Taylor Expansions

Solving PDE with Picard-Lindelöf theorem

- How to compute (a Taylor expansion of) $\lim_{n \to \infty} \phi_n$?
  $\rightarrow$ We just interpret functional F in our Taylor algebra.
- Even simpler[2], the limit $\phi_\infty = F(\phi_\infty)$ is an ordinary recursive (lazy) value.
- The polynomial expansion comes for free, no solving or iteration is needed (just write down the solved-form equation).
- Mutual and multi-dimensional (causal) equations are also supported.

[2] in Ocaml

Taylor Expansions

Solving PDE with Picard-Lindelöf theorem

- How to compute (a Taylor expansion of) $\lim_{n\to\infty} \phi_n$?
  $\rightarrow$ We just interpret functional F in our Taylor algebra.
- Even simpler[2], the limit $\phi_\infty = F(\phi_\infty)$ is an ordinary recursive (lazy) value.
- The polynomial expansion comes for free, no solving or iteration is needed (just write down the solved-form equation).
- Mutual and multi-dimensional (causal) equations are also supported.

- What for the error part ?

[2] in Ocaml

Taylor Expansions

Solving PDE with Picard-Lindelöf theorem

- For the error part, extra fixed point computation is needed.
- Suppose one wants to compute an error tensor part at order k.
- Assume $\phi_k(\mathbf{x}) = \sum_{r=0}^{k} \mathbf{T}_r^v \odot \mathbf{x}^r \pm \Sigma^0(\mathbf{T}_k^\epsilon(\mathbf{x}) \odot \mathbf{x}^k)$
- Then $\phi_{k+1} = F(\phi_k)$, by virtue of integration, has an order $k+1$ error term, $\mathbf{T}_{k+1}^\epsilon$.
- So we have a fixed point whenever, for a given $\mathbf{x}$: $\Sigma^0(\mathbf{T}_k^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^k) \leqslant \Sigma^0 |\mathbf{T}_k^v \odot |\mathbf{x}|^k \pm \mathbf{T}_{k+1}^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^{k+1}|$

Taylor Expansions

Solving PDE with Picard-Lindelöf theorem

- For the error part, extra fixed point computation is needed.
- Suppose one wants to compute an error tensor part at order k.
- Assume $\phi_k(\mathbf{x}) = \sum_{r=0}^{k} \mathbf{T}_r^v \odot \mathbf{x}^r \pm \Sigma^0(\mathbf{T}_k^\epsilon(\mathbf{x}) \odot \mathbf{x}^k)$
- Then $\phi_{k+1} = F(\phi_k)$, by virtue of integration, has an order $k+1$ error term, $\mathbf{T}_{k+1}^\epsilon$.
- So we have a fixed point whenever, for a given $\mathbf{x}$: $\Sigma^0(\mathbf{T}_k^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^k) \leqslant \Sigma^0|\mathbf{T}_k^v \odot |\mathbf{x}|^k \pm \mathbf{T}_{k+1}^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^{k+1}|$
- Stronger but more tractable component-wise constraint: $\mathbf{T}_k^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^k \leqslant |\mathbf{T}_k^v \odot |\mathbf{x}|^k \pm \mathbf{T}_{k+1}^\epsilon(\mathbf{x}) \odot |\mathbf{x}|^{k+1}|$
- The least such $\mathbf{T}_k^\epsilon$ may be computed component-wise, by dichotomy for instance.

Conclusion

- GADT and type-level arithmetics are a great help.
- Only numerical bugs unveiled (∼6.5kLoc).
- How to get rid of proof terms ?
- Unfamiliar co-inductive error formulation, but more flexible.
- Hard work to tame complexity blow-ups.

Perspectives

- Complete PDE solving (error terms).
- More efforts toward efficiency (memory allocations).
- Certified floating-point errors.

Perspectives

- Complete PDE solving (error terms).
- More efforts toward efficiency (memory allocations).
- Certified floating-point errors.
- Better error models such as intervals, zonotopes:

$$v \pm \epsilon(\mathbf{A}) \text{ becomes } v + \sum_{i \in [0, N-1]} \mathbf{X}_i \kappa_i(\mathbf{A}) + \rho(\mathbf{A})$$

- Less space-wasting tensor scheme with co-tensors.
- Disciplined composition with Faa Di Bruno's formula:

$$\frac{d^n}{dx^n} f(g(x)) = \sum \frac{n!}{m_1!\, m_2!\, \cdots\, m_n!} f^{(m_1 + \cdots + m_n)}(g(x)) \prod_{j=1}^{n} \left(\frac{g^{(j)}(x)}{j!} \right)^{m_j}$$

where $\sum_{i=1}^{n} i * m_i = n$

- Beyond monomial basis: Poisson basis, Hermite basis.
- Beyond natural exponents: Laurent series, Puiseux series.

Perspectives

- Complete PDE solving (error terms).
- More efforts toward efficiency (memory allocations).
- Certified floating-point errors.
- Better error models such as intervals, zonotopes:

$$v \pm \epsilon(\mathbf{A}) \text{ becomes } v + \sum_{i \in [0, N-1]} \mathbf{X}_i \kappa_i(\mathbf{A}) + \rho(\mathbf{A})$$

- Less space-wasting tensor scheme with co-tensors.
- Disciplined composition with Faa Di Bruno's formula:

$$\frac{d^n}{dx^n} f(g(x)) = \sum \frac{n!}{m_1!\, m_2! \cdots m_n!} f^{(m_1 + \cdots + m_n)}(g(x)) \prod_{j=1}^{n} \left(\frac{g^{(j)}(x)}{j!} \right)^{m_j}$$

where $\sum_{i=1}^{n} i * m_i = n$

- Beyond monomial basis: Poisson basis, Hermite basis.
- Beyond natural exponents: Laurent series, Puiseux series.
- Full correction proofs (algebraic and approximation properties).

A small demo

Goddard's rocket equations:

$$\left\{ \begin{array}{ll} \dot{r} & = v \\ \dot{v} & = -\frac{D(r,v)}{m}\frac{v}{||v||} - g(r) + C\frac{u}{m} \\ \dot{m} & = -b||u|| \end{array} \right.$$

where:

$$\begin{array}{rl} D(r,v) & = K_D||v||^2 e^{-k_r(||r||-1)} \quad \text{is the drag} \\ g(r) & = G\frac{m}{||r||^2} \quad \text{is the gravity} \\ C\frac{u}{m} & \text{is the thrust} \\ b||u|| & \text{is the fuel consumption} \end{array}$$

Thank you
for your attention !

Any questions ?