Advances in the Formalisation of Univariate Taylor Models in COQ

Érik Martin-Dorel

http://erik.martin-dorel.org/

Post-doctorant, équipe-projet Marelle
Inria Sophia Antipolis – Méditerranée

joint work with members of CoqApprox

CoqApprox meeting
16 Juillet 2013
LIP – ENS de Lyon

Outline

Context and Motivations

What? Compute polynomial approximations of univariate functions with certified error bounds: $\forall x \in \boldsymbol{I},\ |f(x) - P(x)| \leqslant \epsilon$

Why? The correctness of such bounds is a key part of the reliability of numerical software implementing mathematical functions

How? Rely on Taylor Models, and interval arithmetic (cf. next slide)

Overview of Interval Arithmetic (IA)

- Interval = pair of real numbers (or floating-point numbers)
- E.g., $[3.1415, 3.1416] \ni \pi$
- Operations on intervals, e.g., $[2,4] - [0,1] := [2-1, 4-0] = [1,4]$, with the enclosure property: $\forall x \in [2,4],\ \forall y \in [0,1],\ x - y \in [1,4]$.
- Tool for bounding the range of functions

- Dependency problem: for $f(x) = x \cdot (1-x)$ and $\boldsymbol{X} = [0,1]$, a naive use of IA gives $\mathrm{eval}(f, \boldsymbol{X}) = [0,1]$ while the image of $\boldsymbol{X}$ by f is $[0, \frac{1}{4}]$
- IA is not directly applicable to bound approximation errors $e := f - P$ which notably raise some cancellation issues

Overview of Taylor Models (TMs)

The function f is replaced with $(P, \boldsymbol{\Delta})$, where $P = \sum_{i=0}^{n} P_i \cdot (x - x_0)^i$ and $\boldsymbol{\Delta}$ is an interval.

A Taylor Model $(P, \boldsymbol{\Delta})$ over $\boldsymbol{I}$ approximates a whole set of functions:

$$[\![(P, \boldsymbol{\Delta})]\!]_{\boldsymbol{I}} = \{f : \boldsymbol{I} \to \mathbb{R} \mid \forall x \in \boldsymbol{I},\ f(x) - P(x) \in \boldsymbol{\Delta}\}\,.$$

Taylor-Lagrange remainder and validity predicate

Theorem (Taylor-Lagrange)

If f is $n+1$ times derivable on $\boldsymbol{I}$, then $\forall x \in \boldsymbol{I}$, $\exists \xi$ between x_0 and x s.t.:

$$f(x) = \underbrace{\left(\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i \right)}_{\textit{Taylor expansion}} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0)^{n+1}}_{\Delta_n(x_0, x, \xi)}.$$

Definition (`i_validTM`)

$(\boldsymbol{P}, \boldsymbol{\Delta})$ is a valid Taylor Model for f over $\boldsymbol{I}$ around $\boldsymbol{x_0}$ if $\boldsymbol{x_0} \subset \boldsymbol{I}$, $0 \in \boldsymbol{\Delta}$, and

$$\forall x_0 \in \boldsymbol{x_0},\ \exists \alpha_0 \in \boldsymbol{P}_0, \ldots, \alpha_n \in \boldsymbol{P}_n,\ \forall x \in \boldsymbol{I},\ \ f(x) - \sum_{i=0}^{n} \alpha_i (x - x_0)^i \in \boldsymbol{\Delta}.$$

Methodology of Taylor Models

Define arithmetic operations on Taylor Models:

- $\mathtt{TM}_{\mathtt{add}}$, $\mathtt{TM}_{\mathtt{mul}}$, $\mathtt{TM}_{\mathtt{comp}}$, and $\mathtt{TM}_{\mathtt{div}}$
- E.g., $\mathtt{TM}_{\mathtt{add}} : \Big((P_1, \mathbf{\Delta}_1),\ (P_2, \mathbf{\Delta}_2)\Big) \mapsto (P_1 + P_2, \mathbf{\Delta}_1 + \mathbf{\Delta}_2)$.

A two-step strategy:

- Apply these operations recursively on the structure of the function
- Use Taylor-Lagrange remainder for atoms (basic functions)

Why using this 2-step strategy for composite functions?

Interval enclosures $\boldsymbol{\Delta}$ for $\Delta_n(x_0, x, \xi)$ can be largely overestimated.

Example

$f(x) = e^{1/\cos x}$ over $\boldsymbol{I} = [0, 1]$ around $x_0 = \frac{1}{2}$, with $n = 13$.

- Automatic differentiation and Taylor-Lagrange formula:
  $\boldsymbol{\Delta} = [-1.94 \cdot 10^2,\ 1.35 \cdot 10^3]$

Why using this 2-step strategy for composite functions?

Interval enclosures $\boldsymbol{\Delta}$ for $\Delta_n(x_0, x, \xi)$ can be largely overestimated.

Example

$f(x) = e^{1/\cos x}$ over $\boldsymbol{I} = [0, 1]$ around $x_0 = \frac{1}{2}$, with $n = 13$.

- Automatic differentiation and Taylor-Lagrange formula:
  $\boldsymbol{\Delta} = [-1.94 \cdot 10^2,\ 1.35 \cdot 10^3]$
- Taylor Models:
  $\boldsymbol{\Delta} = [-8.74 \cdot 10^{-4}, 4.63 \cdot 10^{-3}]$

The COQ formal proof assistant

We use COQ for

- programming
  - richly typed functional language
  - specify algorithms and theorems
- proving
  - use higher-order logic
  - build proofs interactively
  - develop automatic tactics
  - check proofs

The COQ formal proof assistant

We use COQ for

- programming
  - richly typed functional language
  - specify algorithms and theorems
  - **perform computations**
- proving
  - use higher-order logic
  - build proofs interactively
  - develop automatic tactics
  - check proofs

COQ libraries involved in the formalisation

- SSReflect
  - libraries on arithmetic, lists, and "big" operators
  - tactic facilities
- CoqInterval
  - Abstract interface for intervals
  - Instantiation to intervals with floating-point bounds
  - Formal verification with respect to the Reals library

  for $x, y : \mathbb{R}$

  and $\boldsymbol{X}, \boldsymbol{Y} : \mathbb{IR}$

$$x \in \boldsymbol{X} \ \wedge \ y \in \boldsymbol{Y} \implies x + y \in \boldsymbol{X} + \boldsymbol{Y}$$

$$x \in \boldsymbol{X} \implies \exp(x) \in \mathbf{exp}(\boldsymbol{X})$$

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: $\boldsymbol{F}$: interval evaluator for function f; $\boldsymbol{x_0} \subset \boldsymbol{I}$ and $n \in \mathbb{N}$
Input: $\boldsymbol{T}(\boldsymbol{y_0}, n)$: order-$n$ Taylor polynomial of f around $\boldsymbol{y_0}$
Output: $(\boldsymbol{P}, \boldsymbol{\Delta})$

1: $\boldsymbol{P} \longleftarrow \boldsymbol{T}(\boldsymbol{x_0}, n)$
2: $\boldsymbol{\Gamma} \longleftarrow [X^{n+1}]\, \boldsymbol{T}(\boldsymbol{I}, n+1)$
3: **if** $(\sup \boldsymbol{\Gamma} \leqslant 0$ **or** $\inf \boldsymbol{\Gamma} \geqslant 0)$ **and** $\boldsymbol{I}$ is bounded **then**
4: $\quad \boldsymbol{a} \longleftarrow [\inf \boldsymbol{I}, \inf \boldsymbol{I}]$
5: $\quad \boldsymbol{b} \longleftarrow [\sup \boldsymbol{I}, \sup \boldsymbol{I}]$
6: $\quad \boldsymbol{\Delta_a} \longleftarrow \boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}\,(\boldsymbol{a} - \boldsymbol{x_0})$
7: $\quad \boldsymbol{\Delta_b} \longleftarrow \boldsymbol{F}(\boldsymbol{b}) - \boldsymbol{P}\,(\boldsymbol{b} - \boldsymbol{x_0})$
8: $\quad \boldsymbol{\Delta_{x_0}} \longleftarrow \boldsymbol{F}(\boldsymbol{x_0}) - \boldsymbol{P}\,(\boldsymbol{x_0} - \boldsymbol{x_0})$
9: $\quad \boldsymbol{\Delta} \longleftarrow \boldsymbol{\Delta_a} \vee \boldsymbol{\Delta_b} \vee \boldsymbol{\Delta_{x_0}}$
10: **else**
11: $\quad \boldsymbol{\Delta} \longleftarrow \boldsymbol{\Gamma} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}$
12: **end if**

A generic formalization

When formalizing Taylor Models $(\boldsymbol{P}, \boldsymbol{\Delta})$, focus on being generic w.r.t.

- the type of coefficients of polynomial $\boldsymbol{P}$,
- the type of $\boldsymbol{P}$ and the implementation of related operations
- the type of interval $\boldsymbol{\Delta}$
- the correctnesss proofs, relying on COQ's standard library Reals

D-finite functions (a.k.a. holonomic functions)

Definition

A D-finite function is a solution of a homogeneous linear ordinary differential equation with polynomial coefficients:
$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0$, for given $a_k \in \mathbb{K}[X]$.

Property

The Taylor coefficients of these functions satisfy a *linear recurrence with polynomial coefficients*

D-finite functions (a.k.a. holonomic functions)

Definition

A D-finite function is a solution of a homogeneous linear ordinary differential equation with polynomial coefficients:
$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0$, for given $a_k \in \mathbb{K}[X]$.

Property

The Taylor coefficients of these functions satisfy a *linear recurrence with polynomial coefficients* $\rightarrow$ fast numerical computation of the coefficients

Example (the exponential function)

The Taylor coefficients of $\exp$ at x_0 satisfy the recurrence
$\forall n \in \mathbb{N},\ (n+1)u_{n+1} = u_n$, with $u_0 = \exp(x_0)$ as an initial condition.

$\ln$, $\sin$, $\arcsin$, $\sinh$, $\operatorname{arcsinh}$, $\arctan$, $\operatorname{arctanh}$... are D-finite; $\tan$ is not

Proofs for composite functions

Proof of the algorithm for each algebraic rule

- $\mathtt{TM}_{\mathtt{add}}$: straightforward
- $\mathtt{TM}_{\mathtt{mul}}$: rely on truncated multiplication of polynomials
- $\mathtt{TM}_{\mathtt{comp}}$: rely on $\mathtt{TM}_{\mathtt{mul}}$, $\mathtt{TM}_{\mathtt{add}}$ and TMs for constant functions
- $\mathtt{TM}_{\mathtt{div}}$: it's a TM for $f \times \left(\left(x \mapsto \frac{1}{x}\right) \circ g\right)$

Comparison with a dedicated tool implemented in C

Sollya [S.Chevillard, M.Joldeș, C.Lauter]

- written in C
- based on libraries MPFR, MPFI
- contains an implementation of univariate Taylor Models
- in an imperative-programming framework
- polynomials as arrays of coefficients

CoqApprox

- formalized in COQ
- based on the CoqInterval library
- implements Taylor Models using a similar algorithm
- in a functional-programming framework
- polynomials as lists of coefficients (linear access time)

Validity predicate for floating-point Taylor Models

From any pair $(\boldsymbol{P}, \boldsymbol{\Delta})$ that satisfies `i_validTM`, we can easily produce a pair $(P, \boldsymbol{\Delta'})$ where P is polynomial with floating-point coefficients s.t.:

Definition (`f_validTM`)

$(P, \boldsymbol{\Delta'})$ is a valid TM for f over $\boldsymbol{I}$ around $\boldsymbol{x_0}$ if we have $\boldsymbol{x_0} \subset \boldsymbol{I}$, $0 \in \boldsymbol{\Delta'}$, and

$$\forall \xi_0 \in \boldsymbol{x_0}, \quad \forall x \in \boldsymbol{I}, \quad f(x) - \sum\nolimits_i P_i \cdot (x - \xi_0)^i \in \boldsymbol{\Delta'}.$$

In the code, the function `i2f_tm` transforms $(\boldsymbol{P}, \boldsymbol{\Delta})$ into $(P, \boldsymbol{\Delta'})$.

Some benchmarks for basic functions

	Execution time			Approximation error		
	COQ	Sollya	ratio	naive COQ	COQ	Sollya
$f(x) = \sin x$ $\boldsymbol{I} = [-1, 1]$ order=80 prec=500	0.146s	**0.092s**	1.6	$1.79 \cdot 2^{-402}$	$1.79 \cdot 2^{-402}$	$1.79 \cdot 2^{-402}$
$f(x) = \frac{1}{x}$ $\boldsymbol{I} = [1, 3]$ order=100 prec=125	**0.022s**	0.165s	0.13	$1 \cdot 2^{0}$	$1 \cdot 2^{-101}$	$1 \cdot 2^{-101}$
$f(x) = \sqrt{x}$ $\boldsymbol{I} = [1, 3]$ order=100 prec=125	**0.037s**	0.169s	0.22	$1.98 \cdot 2^{-12}$	$1.60 \cdot 2^{-112}$	$1.60 \cdot 2^{-112}$
$f(x) = \frac{1}{\sqrt{x}}$ (as a basic function) $\boldsymbol{I} = [1, 3]$ order=100 prec=125	**0.029s**	0.424s	0.068	$1.80 \cdot 2^{-5}$	$1.27 \cdot 2^{-105}$	$1.27 \cdot 2^{-105}$

Some benchmarks for composite functions

	Execution time			Approximation error		
	COQ	Sollya	ratio	naive COQ	COQ	Sollya
$f(x) = e^x \sin x$ $\boldsymbol{I} = [-\frac{3}{2}, \frac{3}{2}]$ order=100 prec=500	1.010s	**0.306s**	3.3	$1.63 \cdot 2^{-423}$	$1.63 \cdot 2^{-423}$	$1.63 \cdot 2^{-423}$
$f(x) = e^{1/\cos x}$ $\boldsymbol{I} = [0, 1]$ order=100 prec=100	52.92s	**0.653**	81	$1.97 \cdot 2^{-49}$	$1.99 \cdot 2^{-89}$	$\mathbf{1.98 \cdot 2^{-89}}$
$f(x) = \frac{\sin x}{\cos x}$ $\boldsymbol{I} = [-1, 1]$ order=100 prec=100	11.15s	**0.570s**	20	$1.45 \cdot 2^{26}$	$1.12 \cdot 2^{-64}$	$\mathbf{1.82 \cdot 2^{-96}}$
$f(x) = \frac{1}{\sqrt{x}}$ (as a composite function) $\boldsymbol{I} = [1, 3]$ order=100 prec=125	37.683s	**0.424s**	89	$1.98 \cdot 2^{-12}$	$1.27 \cdot 2^{-105}$	$1.27 \cdot 2^{-105}$

Brief conclusion

CoqApprox: a modular formalization of rigorous polynomial approximation using Taylor Models in the COQ proof assistant

- with a generic approach involving D-finite functions
- taking advantage of the CoqInterval library for interval arithmetic

$\rightarrow$ ability to efficiently compute some formally verified TMs in COQ

Proof status

Fun/Op	Reals	CoqInterval	CoqApprox
cst	☒	☒	☒
id	☒	☒	☒
inv	☒	☒	☒
$\sqrt{\cdot}$	☒	☒	☒
$\frac{1}{\sqrt{\cdot}}$	☒	☒	☒
exp	☒	☒	☒
sin	☒	☒	☒
cos	☒	☒	☒
arctan	☒	☒	☐
ln	☒	☐	☐
arcsin	☐	☐	☐
arccos	☐	☐	☐
$\mathtt{TM}_{\mathtt{add}}$			☒
$\mathtt{TM}_{\mathtt{mul}}$			☒
$\mathtt{TM}_{\mathtt{comp}}$			☒
$\mathtt{TM}_{\mathtt{div}}$			☒

Functions missing from support libraries

Functions missing from the Reals library

- cannot provide a proof for the Taylor Model
- adding them is so far done in a case-by-case manner

→ find a generic way of adding a new function to Reals

→ e.g. by using a differential equation or a recurrence relation as definition

- need to provide the n-th derivatives of the function

Functions missing from CoqInterval

- cannot provide an initial value for the Taylor Model

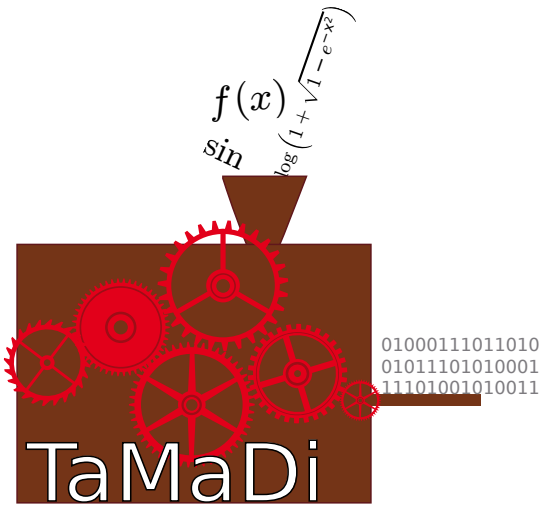
→ just implement the missing functions in CoqInterval

→ may use some extra techniques, such as fixed point theorems

Perspectives for CoqApprox

- Add more functions ($\cosh$, $\tan$, etc.) and prove them
- Optimize the multiplication algorithm for Taylor Models
- Implement Chebyshev Models $\rightsquigarrow$ tighter remainders
- Taylor Models with Relative Remainder?
- Combine TMs with some polynomial global optimization technique
  - E.g., Bernstein polynomials
- Consider the possible generalization to the multivariate case
- Investigate ways to ease the definition of RPAs from the ODE
- Investigate the possible use of an "affine arithmetic" module
- Investigate RPAs based on other families of orthogonal polynomials
- Consider alternative techniques for verifying error bounds
  - fixed-point theorems?
  - majorant series?

End of the Talk



Thank you for your attention!

The TaMaDi project homepage:
http://tamadi.gforge.inria.fr/