

Certified, Efficient and Sharp Univariate Taylor Models in Coq

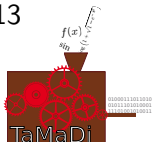
Érik Martin-Dorel

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

Postdoc in project-team Marelle, Inria Sophia Antipolis – Méditerranée, France

Joint work with Micaela Mayero, Ioana Paşca, Laurence Rideau, Laurent Théry

SYNASC'2013, Timișoara, Romania
26th September 2013



Outline

- 1 Context and preliminary remarks
- 2 Taylor models for basic functions
- 3 Benchmarks
- 4 Conclusion

Context and Motivations

What? Compute polynomial approximations of univariate functions with **certified error bounds**: for a given function f over an interval¹ \mathbf{I} , compute P, ϵ and formally prove that $\forall x \in \mathbf{I}, |f(x) - P(x)| \leq \epsilon$

¹Intervals (and polynomials with interval coefficients) will be printed in bold.

Context and Motivations

What? Compute polynomial approximations of univariate functions with **certified error bounds**: for a given function f over an interval¹ \mathbf{I} , compute P, ϵ and formally prove that $\forall x \in \mathbf{I}, |f(x) - P(x)| \leq \epsilon$

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

¹Intervals (and polynomials with interval coefficients) will be printed in bold.

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 $\mathbf{X} = [0, 1]$, a naive use of IA gives $\text{eval}(f, \mathbf{X}) = [0, 1]$ while the image of \mathbf{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, Δ) , where $P(x) = \sum_{i=0}^n P_i \cdot (x - x_0)^i$ and Δ is an interval.

A Taylor Model (P, Δ) over I approximates a whole set of functions:

$$\llbracket (P, \Delta) \rrbracket_I = \{f : I \rightarrow \mathbb{R} \mid \forall x \in I, f(x) - P(x) \in \Delta\}.$$

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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

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

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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

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

Naive algorithm to compute an interval-based Taylor model

- Input: function f , intervals \mathbf{I} and \mathbf{x}_0 (containing x_0), integer $n \geq 0$
- Output: (P, Δ) , where polynomial P has interval coefficients $\frac{f^{(i)}(\mathbf{x}_0)}{i!}$ and Δ is an enclosure of $\Delta_n(x_0, x, \xi)$ for $x, \xi \in \mathbf{I}$ and $x_0 \in \mathbf{x}_0$

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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

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

Naive algorithm to compute an interval-based Taylor model

- Input: function f , intervals I and \mathbf{x}_0 (containing x_0), integer $n \geq 0$
- Output: (P, Δ) , where polynomial P has **interval coefficients** $\frac{f^{(i)}(\mathbf{x}_0)}{i!}$ and Δ is an enclosure of $\Delta_n(x_0, x, \xi)$ for $x, \xi \in I$ and $x_0 \in \mathbf{x}_0$

→ **Rounding errors are easily handled by interval arithmetic**

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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

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

Naive algorithm to compute an interval-based Taylor model

- Input: function f , intervals I and \mathbf{x}_0 (containing x_0), integer $n \geq 0$
- Output: (P, Δ) , where polynomial P has interval coefficients $\frac{f^{(i)}(x_0)}{i!}$ and Δ is an enclosure of $\Delta_n(x_0, x, \xi)$ for $x, \xi \in I$ and $x_0 \in \mathbf{x}_0$

→ Rounding errors are easily handled by interval arithmetic

→ **Uniform computation of (P, Δ)**

Methodology of Taylor models

Define arithmetic operations on Taylor models TM_{add} , TM_{mul} , TM_{comp} , TM_{div} :

- Addition: $(P_1, \Delta_1) \oplus (P_2, \Delta_2) = (P_1 + P_2, \Delta_1 + \Delta_2)$
- Similar rule for multiplication, composition, and division (see paper)

A two-step strategy:

- 1 Apply these operations recursively on the structure of the function
- 2 For basic functions: compute Δ using the Taylor-Lagrange formula

Why using this 2-step strategy for composite functions?

Interval enclosures Δ for $\Delta_n(x_0, x, \xi)$ can be **largely overestimated**.

Example

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

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

Why using this 2-step strategy for composite functions?

Interval enclosures Δ for $\Delta_n(x_0, x, \xi)$ can be **largely overestimated**.

Example

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

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

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: F : interval evaluator for function f ; $\mathbf{x}_0 \subset I$ and $n \in \mathbb{N}$

Input: $T(\mathbf{y}_0, n)$: order- n Taylor polynomial of f around \mathbf{y}_0

Output: (P, Δ)

- 1: $P \leftarrow T(\mathbf{x}_0, n)$
- 2: $\Gamma \leftarrow [X^{n+1}] T(I, n + 1)$
- 3: **if** $(\sup \Gamma \leq 0$ **or** $\inf \Gamma \geq 0)$ **and** I is bounded **then**
- 4: $\mathbf{a} \leftarrow [\inf I]$
- 5: $\mathbf{b} \leftarrow [\sup I]$
- 6: $\Delta_{\mathbf{a}} \leftarrow F(\mathbf{a}) - P(\mathbf{a} - \mathbf{x}_0)$
- 7: $\Delta_{\mathbf{b}} \leftarrow F(\mathbf{b}) - P(\mathbf{b} - \mathbf{x}_0)$
- 8: $\Delta_{\mathbf{x}_0} \leftarrow F(\mathbf{x}_0) - P(\mathbf{x}_0 - \mathbf{x}_0)$
- 9: $\Delta \leftarrow \Delta_{\mathbf{a}} \vee \Delta_{\mathbf{b}} \vee \Delta_{\mathbf{x}_0}$
- 10: **else**
- 11: $\Delta \leftarrow \Gamma \times (I - \mathbf{x}_0)^{n+1}$
- 12: **end if**

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: F : interval evaluator for function f ; $x_0 \in I$ and $n \in \mathbb{N}$

Input: $T(y_0, n)$: order- n Taylor polynomial of f around y_0

Output: (P, Δ)

- 1: $P \leftarrow T(x_0, n)$
- 2: $\Gamma \leftarrow [X^{n+1}] T(I, n + 1)$ Compute an enclosure of $f^{(n+1)}(\xi)/(n+1)!$, $\xi \in I$
- 3: **if** $(\sup \Gamma \leq 0$ **or** $\inf \Gamma \geq 0)$ **and** I is bounded **then**
- 4: $a \leftarrow [\inf I]$
- 5: $b \leftarrow [\sup I]$
- 6: $\Delta_a \leftarrow F(a) - P(a - x_0)$
- 7: $\Delta_b \leftarrow F(b) - P(b - x_0)$
- 8: $\Delta_{x_0} \leftarrow F(x_0) - P(x_0 - x_0)$
- 9: $\Delta \leftarrow \Delta_a \vee \Delta_b \vee \Delta_{x_0}$
- 10: **else**
- 11: $\Delta \leftarrow \Gamma \times (I - x_0)^{n+1}$ Naive enclosure of the Taylor-Lagrange formula
- 12: **end if**

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: F : interval evaluator for function f ; $\mathbf{x}_0 \subset I$ and $n \in \mathbb{N}$

Input: $T(\mathbf{y}_0, n)$: order- n Taylor polynomial of f around \mathbf{y}_0

Output: (P, Δ)

- 1: $P \leftarrow T(\mathbf{x}_0, n)$
- 2: $\Gamma \leftarrow [X^{n+1}] T(I, n+1)$ Compute an enclosure of $f^{(n+1)}(\xi)/(n+1)!$, $\xi \in I$
- 3: **if** $(\sup \Gamma \leq 0$ **or** $\inf \Gamma \geq 0)$ **and** I is bounded **then**
- 4: $a \leftarrow [\inf I]$
- 5: $b \leftarrow [\sup I]$
- 6: $\Delta_a \leftarrow F(a) - P(a - \mathbf{x}_0)$
- 7: $\Delta_b \leftarrow F(b) - P(b - \mathbf{x}_0)$
- 8: $\Delta_{x_0} \leftarrow F(\mathbf{x}_0) - P(\mathbf{x}_0 - \mathbf{x}_0)$ Evaluate the Taylor-Lagrange remainder of f at three point-intervals of I
- 9: $\Delta \leftarrow \Delta_a \vee \Delta_b \vee \Delta_{x_0}$
- 10: **else**
- 11: $\Delta \leftarrow \Gamma \times (I - \mathbf{x}_0)^{n+1}$
- 12: **end if**

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: F : interval evaluator for function f ; $\mathbf{x}_0 \subset I$ and $n \in \mathbb{N}$

Input: $T(\mathbf{y}_0, n)$: order- n Taylor polynomial of f around \mathbf{y}_0 \Leftarrow to provide

Output: (P, Δ)

- 1: $P \leftarrow T(\mathbf{x}_0, n)$
- 2: $\Gamma \leftarrow [X^{n+1}] T(I, n + 1)$
- 3: **if** $(\sup \Gamma \leq 0$ **or** $\inf \Gamma \geq 0)$ **and** I is bounded **then**
- 4: $\mathbf{a} \leftarrow [\inf I]$
- 5: $\mathbf{b} \leftarrow [\sup I]$
- 6: $\Delta_{\mathbf{a}} \leftarrow F(\mathbf{a}) - P(\mathbf{a} - \mathbf{x}_0)$
- 7: $\Delta_{\mathbf{b}} \leftarrow F(\mathbf{b}) - P(\mathbf{b} - \mathbf{x}_0)$
- 8: $\Delta_{\mathbf{x}_0} \leftarrow F(\mathbf{x}_0) - P(\mathbf{x}_0 - \mathbf{x}_0)$
- 9: $\Delta \leftarrow \Delta_{\mathbf{a}} \vee \Delta_{\mathbf{b}} \vee \Delta_{\mathbf{x}_0}$
- 10: **else**
- 11: $\Delta \leftarrow \Gamma \times (I - \mathbf{x}_0)^{n+1}$
- 12: **end if**

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

Definition

A D -finite function is a solution of a homogeneous linear ordinary differential equation (LODE) with polynomial coefficients:

$$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0, \text{ for given } a_k \in \mathbb{K}[X].$$

Property

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

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

Definition

A D -finite function is a solution of a homogeneous linear ordinary differential equation (LODE) with polynomial coefficients:

$$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0, \text{ for given } a_k \in \mathbb{K}[X].$$

Property

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

Example (the exponential function)

The Taylor coefficients of \exp at x_0 satisfy the recurrence

$$\forall n \geq 1, u_n = u_{n-1}/n, \text{ with } u_0 = \exp(x_0) \text{ as an initial condition.}$$

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

Coq libraries involved in the formalisation

- SSReflect/MathComponents
 - tactic facilities
 - libraries on arithmetic, lists, and big operators such as \sum and \prod
- Coq.Interval
 - Abstract interface for intervals (`IntervalOps`)
 - Instantiation to intervals with floating-point bounds
 - Formal verification with respect to the Reals standard library:

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

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

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

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

Comparison with a dedicated tool implemented in C

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

- written in C
- based on the C libraries GMP, MPFR and MPFI
- contains an implementation of univariate Taylor models
- in an imperative programming framework
- polynomials as arrays of coefficients
- not formally proved

CoqApprox

- formalised in Coq
- based on the internals of the library Coq.Interval
- implements Taylor models using a similar algorithm
- in a purely-functional programming framework
- polynomials as lists of coefficients (linear access time)
- formally proved in Coq



Some benchmarks for basic functions (more in the paper)

	Execution time			Approximation error		
	Coq	Sollya	Coq/Sollya	naive Coq	Coq	Sollya
$f(x) = 1/x$ $I = [1, 3]$ deg=100 prec=125	0.022s	0.165s	7.6x faster	$1 \cdot 2^0$	$1 \cdot 2^{-101}$	
$f(x) = \sqrt{x}$ $I = [1, 3]$ deg=100 prec=125	0.037s	0.169s	4.5x faster	$1.98 \cdot 2^{-12}$	$1.60 \cdot 2^{-112}$	
$f(x) = \sin x$ $I = [-1, 1]$ deg=80 prec=500	0.146s	0.092s	1.6x slower	$1.79 \cdot 2^{-402}$		

Column “naive Coq” \rightsquigarrow naive use of the Taylor-Lagrange formula

Column “Coq” \rightsquigarrow rely on Zumkeller’s technique

Some benchmarks for composite functions

	Execution time			Approximation error		
	Coq	Sollya	Coq/Sollya	naive Coq	Coq	Sollya
$f(x) = e^x \sin x$ $I = [-\frac{3}{2}, \frac{3}{2}]$ deg=100 prec=500	1.010s	0.306s	3.3x slower	1.63·2 ⁻⁴²³		
$f(x) = e^{1/\cos x}$ $I = [0, 1]$ deg=100 prec=100	52.92s	0.653	81x slower	1.97·2 ⁻⁴⁹	1.99·2⁻⁸⁹	1.98·2⁻⁸⁹
$f(x) = \frac{\sin x}{\cos x}$ $I = [-1, 1]$ deg=100 prec=100	11.15s	0.570s	20x slower	1.45·2 ²⁶	1.12·2 ⁻⁶⁴	1.82·2⁻⁹⁶

Column “naive Coq” \rightsquigarrow naive use of the Taylor-Lagrange formula

Column “Coq” \rightsquigarrow rely on Zumkeller’s technique

Conclusion and Perspectives

CoqApprox: a Coq library of Rigorous Polynomial Approximation

- **Efficient computation** of Taylor models with **sharp remainders**
- **Machine-checked proofs** of correctness based on **generic data-types**

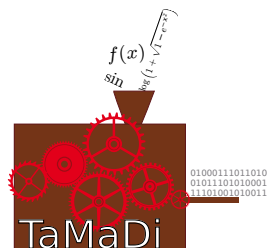
→ we can thus formally prove that $|f(x) - TM_f(x)| \leq \epsilon_1$ for $x \in I$.

To do: combine CoqApprox with a polynomial global optimisation method

- E.g., rely on Bernstein polynomials or Sums of Squares in Coq
- **Devise a tactic** to formally prove $|TM_f(x) - P(x)| < \epsilon_2$ for $x \in I$.

→ will be able to automatically prove $|f(x) - P(x)| < \epsilon$ for $x \in I$.

End of the talk



Thank you for your attention!

The CoqApprox homepage:

<http://tamadi.gforge.inria.fr/CoqApprox/>

We acknowledge our colleagues Nicolas Brisebarre, Mioara Joldeş and Jean-Michel Muller for their help.