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 Pașca, 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 interval[1] $\boldsymbol{I}$, compute P, ϵ and formally prove that $\forall x \in \boldsymbol{I},\ |f(x) - P(x)| \leqslant \epsilon$

[1] Intervals (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 interval[1] $\boldsymbol{I}$, compute P, ϵ and formally prove that $\forall x \in \boldsymbol{I},\ |f(x) - P(x)| \leqslant \epsilon$

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

[1] Intervals (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 $\boldsymbol{X} = [0,1]$, 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 $(P, \mathbf{\Delta})$, where $P(x) = \sum_{i=0}^{n} P_i \cdot (x - x_0)^i$ and $\mathbf{\Delta}$ is an interval.

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

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

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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.:

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

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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

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

Naive algorithm to compute an interval-based Taylor model

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

Taylor-Lagrange formula and interval-based Taylor models

Theorem (Taylor-Lagrange formula)

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.:

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

Naive algorithm to compute an interval-based Taylor model

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

$\rightarrow$ 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 $\boldsymbol{I}$, then $\forall x \in \boldsymbol{I}$, $\exists \xi$ between x_0 and x s.t.:

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

Naive algorithm to compute an interval-based Taylor model

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

$\rightarrow$ Rounding errors are easily handled by interval arithmetic

$\rightarrow$ Uniform computation of $(\boldsymbol{P}, \boldsymbol{\Delta})$

Methodology of Taylor models

Define arithmetic operations on Taylor models $\mathtt{TM}_{\mathtt{add}}, \mathtt{TM}_{\mathtt{mul}}, \mathtt{TM}_{\mathtt{comp}}, \mathtt{TM}_{\mathtt{div}}$:

- Addition: $(\boldsymbol{P}_1, \boldsymbol{\Delta}_1) \oplus (\boldsymbol{P}_2, \boldsymbol{\Delta}_2) = (\boldsymbol{P}_1 + \boldsymbol{P}_2, \boldsymbol{\Delta}_1 + \boldsymbol{\Delta}_2)$
- 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 $\boldsymbol{\Delta}$ using the Taylor-Lagrange formula

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:
  $\boldsymbol{\Delta} = [-8.74 \cdot 10^{-4}, 4.63 \cdot 10^{-3}]$

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: $(\boldsymbol{P}, \boldsymbol{\Delta})$

$\boldsymbol{P} \leftarrow \boldsymbol{T}(\boldsymbol{x_0}, n)$
$\boldsymbol{\Gamma} \leftarrow [X^{n+1}]\ \boldsymbol{T}(\boldsymbol{I}, n+1)$
if $(\sup \boldsymbol{\Gamma} \leqslant 0$ **or** $\inf \boldsymbol{\Gamma} \geqslant 0)$ **and** $\boldsymbol{I}$ is bounded **then**
  $\boldsymbol{a} \leftarrow [\inf \boldsymbol{I}]$
  $\boldsymbol{b} \leftarrow [\sup \boldsymbol{I}]$
  $\boldsymbol{\Delta}_{\boldsymbol{a}} \leftarrow \boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}\,(\boldsymbol{a} - \boldsymbol{x_0})$
  $\boldsymbol{\Delta}_{\boldsymbol{b}} \leftarrow \boldsymbol{F}(\boldsymbol{b}) - \boldsymbol{P}\,(\boldsymbol{b} - \boldsymbol{x_0})$
  $\boldsymbol{\Delta}_{\boldsymbol{x_0}} \leftarrow \boldsymbol{F}(\boldsymbol{x_0}) - \boldsymbol{P}\,(\boldsymbol{x_0} - \boldsymbol{x_0})$
  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}_{\boldsymbol{a}} \vee \boldsymbol{\Delta}_{\boldsymbol{b}} \vee \boldsymbol{\Delta}_{\boldsymbol{x_0}}$
else
  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Gamma} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}$
end if

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

Generic computation of sharp remainders for basic functions

Algorithm (Zumkeller's technique)

Input: $\boldsymbol{F}$: 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: $(\boldsymbol{P}, \boldsymbol{\Delta})$

$\boldsymbol{P} \leftarrow \boldsymbol{T}(\boldsymbol{x_0}, n)$
$\boxed{\boldsymbol{\Gamma} \leftarrow [X^{n+1}]\, \boldsymbol{T}(\boldsymbol{I}, n+1)}$ — Compute an enclosure of $f^{(n+1)}(\xi)/(n+1)!,\ \xi \in \boldsymbol{I}$
if $(\sup \boldsymbol{\Gamma} \leqslant 0$ **or** $\inf \boldsymbol{\Gamma} \geqslant 0)$ **and** $\boldsymbol{I}$ is bounded **then**
    $\boldsymbol{a} \leftarrow [\inf \boldsymbol{I}]$
    $\boldsymbol{b} \leftarrow [\sup \boldsymbol{I}]$
    $\boldsymbol{\Delta}_{\boldsymbol{a}} \leftarrow \boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}(\boldsymbol{a} - \boldsymbol{x_0})$
    $\boldsymbol{\Delta}_{\boldsymbol{b}} \leftarrow \boldsymbol{F}(\boldsymbol{b}) - \boldsymbol{P}(\boldsymbol{b} - \boldsymbol{x_0})$
    $\boldsymbol{\Delta}_{\boldsymbol{x_0}} \leftarrow \boldsymbol{F}(\boldsymbol{x_0}) - \boldsymbol{P}(\boldsymbol{x_0} - \boldsymbol{x_0})$ — Evaluate the Taylor-Lagrange remainder of f at three point-intervals of $\boldsymbol{I}$
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}_{\boldsymbol{a}} \vee \boldsymbol{\Delta}_{\boldsymbol{b}} \vee \boldsymbol{\Delta}_{\boldsymbol{x_0}}$
else
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Gamma} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}$
end if

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 $\boldsymbol{y_0}$ $\Longleftarrow$ to provide

Output: $(\boldsymbol{P}, \boldsymbol{\Delta})$

$\boldsymbol{P} \leftarrow \boldsymbol{T}(\boldsymbol{x_0}, n)$

$\boldsymbol{\Gamma} \leftarrow [X^{n+1}]\ \boldsymbol{T}(\boldsymbol{I}, n+1)$

if $(\sup \boldsymbol{\Gamma} \leqslant 0$ **or** $\inf \boldsymbol{\Gamma} \geqslant 0)$ **and** $\boldsymbol{I}$ is bounded **then**

  $\boldsymbol{a} \leftarrow [\inf \boldsymbol{I}]$

  $\boldsymbol{b} \leftarrow [\sup \boldsymbol{I}]$

  $\boldsymbol{\Delta}_{\boldsymbol{a}} \leftarrow \boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}\,(\boldsymbol{a} - \boldsymbol{x_0})$

  $\boldsymbol{\Delta}_{\boldsymbol{b}} \leftarrow \boldsymbol{F}(\boldsymbol{b}) - \boldsymbol{P}\,(\boldsymbol{b} - \boldsymbol{x_0})$

  $\boldsymbol{\Delta}_{\boldsymbol{x_0}} \leftarrow \boldsymbol{F}(\boldsymbol{x_0}) - \boldsymbol{P}\,(\boldsymbol{x_0} - \boldsymbol{x_0})$

  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}_{\boldsymbol{a}} \vee \boldsymbol{\Delta}_{\boldsymbol{b}} \vee \boldsymbol{\Delta}_{\boldsymbol{x_0}}$

else

  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Gamma} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}$

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:
$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0$, 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

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

Property

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

Example (the exponential function)

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

$\ln$, $\sin$, $\arcsin$, $\sinh$, $\operatorname{arcsinh}$, $\arctan$, $\operatorname{arctanh}\ldots$ 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 $\boldsymbol{X}, \boldsymbol{Y} : \mathbb{IR}$

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

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

Comparison with a dedicated tool implemented in C

Sollya [S.Chevillard, M.Joldeș, 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
$f(x) = 1/x$ $\boldsymbol{I} = [1,3]$ deg=100 prec=125	**0.022s**	0.165s	7.6x faster	$1{\cdot}2^{0}$	$\mathbf{1{\cdot}2^{-101}}$	
$f(x) = \sqrt{x}$ $\boldsymbol{I} = [1,3]$ deg=100 prec=125	**0.037s**	0.169s	4.5x faster	$1.98{\cdot}2^{-12}$	$\mathbf{1.60{\cdot}2^{-112}}$	
$f(x) = \sin x$ $\boldsymbol{I} = [-1,1]$ 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
$f(x) = e^x \sin x$ $\boldsymbol{I} = [-\frac{3}{2}, \frac{3}{2}]$ deg=100 prec=500	1.010s	**0.306s**	3.3x slower	$1.63 \cdot 2^{-423}$		
$f(x) = e^{1/\cos x}$ $\boldsymbol{I} = [0, 1]$ deg=100 prec=100	52.92s	**0.653**	81x slower	$1.97 \cdot 2^{-49}$	$\mathbf{1.99 \cdot 2^{-89}}$	$\mathbf{1.98 \cdot 2^{-89}}$
$f(x) = \frac{\sin x}{\cos x}$ $\boldsymbol{I} = [-1, 1]$ deg=100 prec=100	11.15s	**0.570s**	20x slower	$1.45 \cdot 2^{26}$	$1.12 \cdot 2^{-64}$	$\mathbf{1.82 \cdot 2^{-96}}$

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

$\rightarrow$ we can thus formally prove that $|f(x) - TM_f(x)| \leqslant \epsilon_1$ for $x \in \boldsymbol{I}$.

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 $x \in \boldsymbol{I}$.

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

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 Joldeș and Jean-Michel Muller for their help.