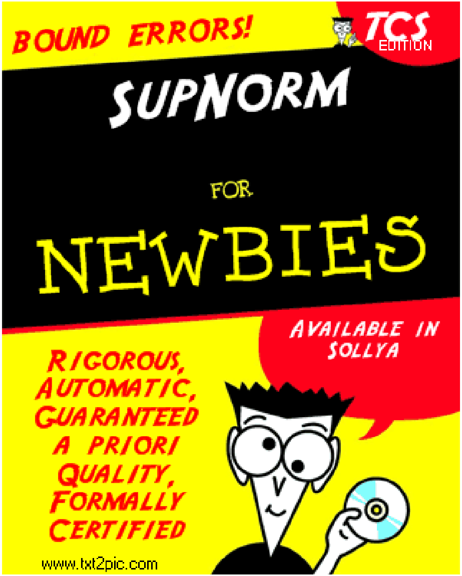


Mioara Joldes

Journées TAMADI,
27-28 Oct. 2010

Supremum Norms and Rigorous Polynomial Approximations

Based on:

- S. Chevillard, J. Harrison, M.J., Ch. Lauter, *Efficient and accurate computation of upper bounds of approximation errors*, accepted for TCS.
- N. Brisebarre, M.J., *Chebyshev interpolation polynomial-based tools for rigorous computing*, ISSAC 2010.

Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  $\rightsquigarrow$ Previous techniques

Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  $\rightsquigarrow$ Previous techniques
- Taylor Models and Enhancements

Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  $\rightsquigarrow$ Previous techniques
- Taylor Models and Enhancements
- Chebyshev Models

Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  $\rightsquigarrow$ Previous techniques
- Taylor Models and Enhancements
- Chebyshev Models
- Comparison, Experimental Results

Origins

- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.

Origins

- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.
- General scheme:
  - Argument reduction

Origins

- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.
- General scheme:
  - Argument reduction
  - Approximation: approximate f by a polynomial p on $[a, b]$.

Origins

- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.
- General scheme:
  - Argument reduction
  - Approximation: approximate f by a polynomial p on $[a, b]$.
  - Implementation: write a program that evaluates p.

Origins

- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.
- General scheme:
  - Argument reduction
  - Approximation: approximate f by a polynomial p on $[a, b]$.
  - Implementation: write a program that evaluates p.
- From a validation point of view:
  - Bound the errors occurring during the evaluation of p.

Origins

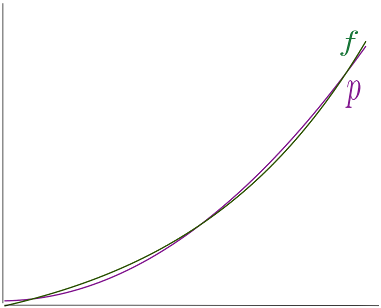
- Development of `libms`.
- Implementation of functions f such as $f = \exp$, $f = \sin$, etc.
- General scheme:
  - Argument reduction
  - Approximation: approximate f by a polynomial p on $[a, b]$.
  - Implementation: write a program that evaluates p.
- From a validation point of view:
  - Bound the errors occurring during the evaluation of p.
  - Bound the error between p and f: compute

$$\|\varepsilon\|_\infty = \sup_{x \in [a,\, b]} \{|\varepsilon(x)|\}$$

where $\varepsilon(x) = f(x) - p(x)$ or $\varepsilon(x) = p(x)/f(x) - 1$.

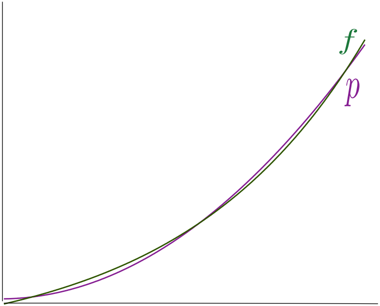
Example:

$f(x) = e^{1/\cos(x)},\ x \in [0,1],\ \ p(x) = \sum_{i=0}^{10} c_i x^i,$



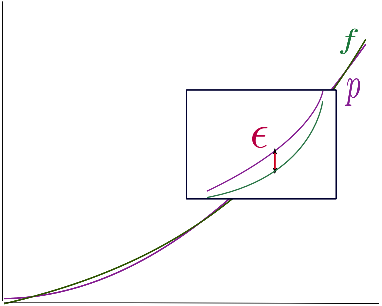
Example:

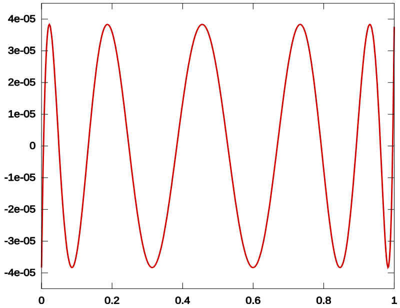
$f(x) = e^{1/\cos(x)},\ x \in [0,1],\ \ p(x) = \sum_{i=0}^{10} c_i x^i,$
$\varepsilon(x) = f(x) - p(x)$



Example:

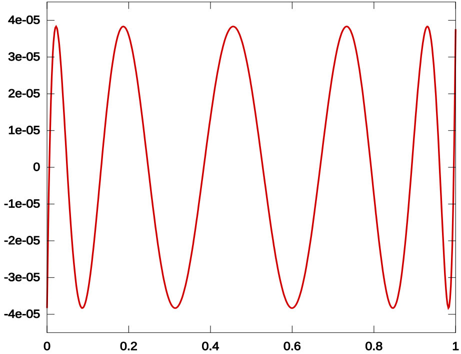
$f(x) = e^{1/\cos(x)}$, $x \in [0, 1]$, $p(x) = \sum_{i=0}^{10} c_i x^i$,
$\varepsilon(x) = f(x) - p(x)$ s.t. $\|\varepsilon\|_\infty = \sup_{x\in[a,b]}\{|\varepsilon(x)|\}$ is as small as possible (Remez algorithm)





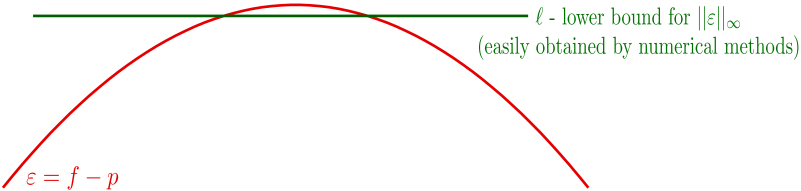
Example:

$f(x) = e^{1/\cos(x)}, \;\; x \in [0, 1], \;\; p(x) = \sum_{i=0}^{10} c_i x^i,$
$\varepsilon(x) = f(x) - p(x)$ s.t. $\|\varepsilon\|_\infty = \sup_{x \in [a,\, b]}\{|\varepsilon(x)|\}$ is as small as possible (Remez algorithm)

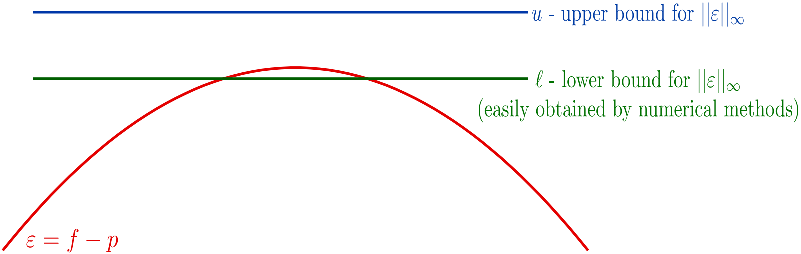


Our problem - How to find ℓ and u such that $\|\varepsilon\|_\infty \in [\ell, u]$?

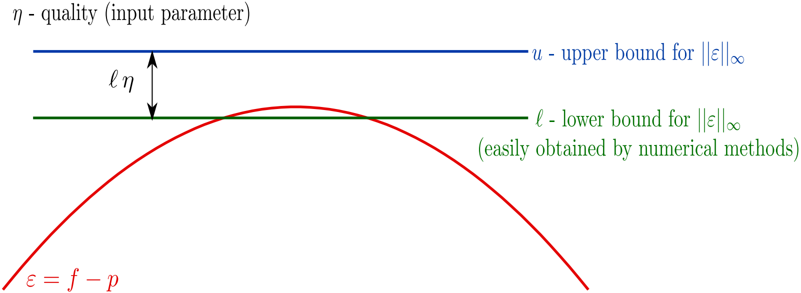
Our problem - How to find ℓ and u such that $||\varepsilon||_\infty \in [\ell, u]$?



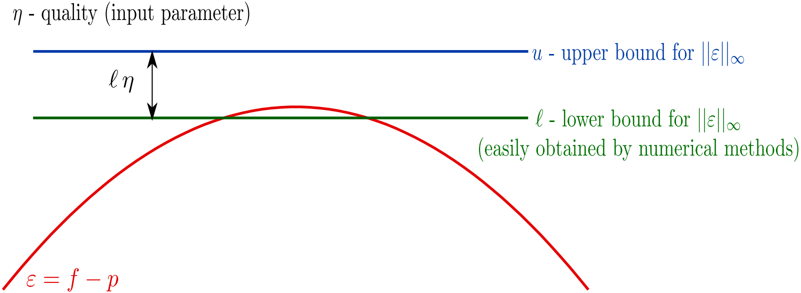
Our problem - How to find ℓ and u such that $||\varepsilon||_\infty \in [\ell, u]$?



Our problem - How to find ℓ and u such that $||\varepsilon||_\infty \in [\ell, u]$?



Our problem - How to find ℓ and u such that $||\varepsilon||_\infty \in [\ell, u]$?



Actual difficulty: certified upper bound u

Our approach

- Automatic: no parameter requires to be manually adjusted.

Our approach

- Automatic: no parameter requires to be manually adjusted.
- Accept any function f defined by an expression.

Our approach

- Automatic: no parameter requires to be manually adjusted.
- Accept any function f defined by an expression.
- Guaranteed a priori quality η of the result.

Our approach

- Automatic: no parameter requires to be manually adjusted.
- Accept any function f defined by an expression.
- Guaranteed a priori quality η of the result.
- Correctly handles removable discontinuities
  e.g. $f(x) = \sin(x)$, $p(x) = x\,q(x)$ and $\varepsilon(x) = p(x)/f(x) - 1$.

Our approach

- Automatic: no parameter requires to be manually adjusted.
- Accept any function f defined by an expression.
- Guaranteed a priori quality η of the result.
- Correctly handles removable discontinuities
  e.g. $f(x) = \sin(x)$, $p(x) = x\,q(x)$ and $\varepsilon(x) = p(x)/f(x) - 1$.
- Could generate a complete formal proof without much effort.

State of the art

It is a univariate global optimization problem.

[1] Hansen '92, Kearfott '96, Messine '97

State of the art

It is a univariate global optimization problem.

- Purely numerical algorithms: find the zeros of ε' (e.g., Newton's algorithm).

[1] Hansen '92, Kearfott '96, Messine '97

State of the art

It is a univariate global optimization problem.

- Purely numerical algorithms: find the zeros of ε' (e.g., Newton's algorithm).
  $\leadsto$ Not rigorous: we might miss some of the zeros.

[1] Hansen '92, Kearfott '96, Messine '97

State of the art

It is a univariate global optimization problem.

- Purely numerical algorithms: find the zeros of ε' (e.g., Newton's algorithm).
  $\rightsquigarrow$ Not rigorous: we might miss some of the zeros.
- General-purpose rigorous global optimization algorithms[1]
  - use Interval Arithmetic (IA)
  - Branch and bound: bisection, Interval Newton's algorithm.

[1] Hansen '92, Kearfott '96, Messine '97

Interval Arithmetic (IA)

- Each interval = pair of floating-point numbers
  (multiple precision IA libraries exist, e.g. MPFI[2])

[2] http://gforge.inria.fr/projects/mpfi/

Interval Arithmetic (IA)

- Each interval = pair of floating-point numbers (multiple precision IA libraries exist, e.g. MPFI[2])
- $\pi \in [3.1415, 3.1416]$

[2] http://gforge.inria.fr/projects/mpfi/

Interval Arithmetic (IA)

- Each interval = pair of floating-point numbers (multiple precision IA libraries exist, e.g. MPFI[2])
- $\pi \in [3.1415, 3.1416]$
- Interval Arithmetic Operations
  Eg. $[1, 2] + [-3, 2] = [-2, 4]$

[2] http://gforge.inria.fr/projects/mpfi/

Interval Arithmetic (IA)

- Each interval = pair of floating-point numbers (multiple precision IA libraries exist, e.g. MPFI[2])
- $\pi \in [3.1415, 3.1416]$
- Interval Arithmetic Operations
  Eg. $[1, 2] + [-3, 2] = [-2, 4]$
- Range bounding for functions
  Eg. $x \in [-1, 2], f(x) = x^2 - x + 1$
  $F(X) = X^2 - X + 1$
  $F([-1, 2]) = [-1, 2]^2 - [-1, 2] + [1, 1]$
  $F([-1, 2]) = [0, 4] - [-1, 2] + [1, 1]$
  $F([-1, 2]) = [-1, 6]$

[2] http://gforge.inria.fr/projects/mpfi/

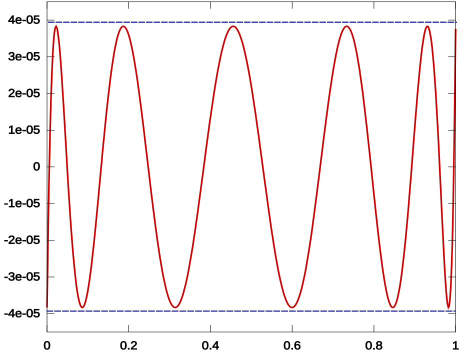
Interval Arithmetic (IA)

- Each interval = pair of floating-point numbers (multiple precision IA libraries exist, e.g. MPFI[2])
- $\pi \in [3.1415, 3.1416]$
- Interval Arithmetic Operations
  Eg. $[1, 2] + [-3, 2] = [-2, 4]$
- Range bounding for functions
  Eg. $x \in [-1, 2], f(x) = x^2 - x + 1$
  $F(X) = X^2 - X + 1$
  $F([-1, 2]) = [-1, 2]^2 - [-1, 2] + [1, 1]$
  $F([-1, 2]) = [0, 4] - [-1, 2] + [1, 1]$
  $F([-1, 2]) = [-1, 6]$
  $x \in [-1, 2], f(x) \in [-1, 6]$, but $\mathtt{Im}(f) = [3/4, 3]$

[2] http://gforge.inria.fr/projects/mpfi/

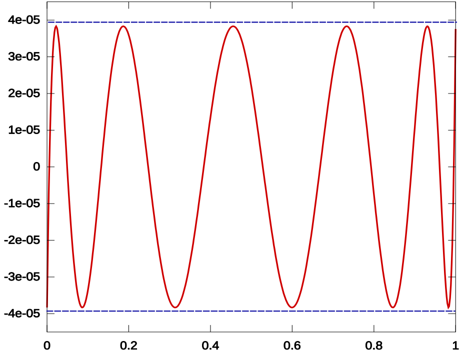
Example (IA -Dependency phenomenon):

$f(x) = e^{1/\cos(x)}$, $x \in [0,1]$, $p(x) = \sum_{i=0}^{10} c_i x^i$,
$\varepsilon(x) = f(x) - p(x)$ s.t. $\|\varepsilon\|_\infty = \sup_{x\in[a,b]}\{|\varepsilon(x)|\}$ is as small as possible (Remez algorithm)



Example (IA -Dependency phenomenon):

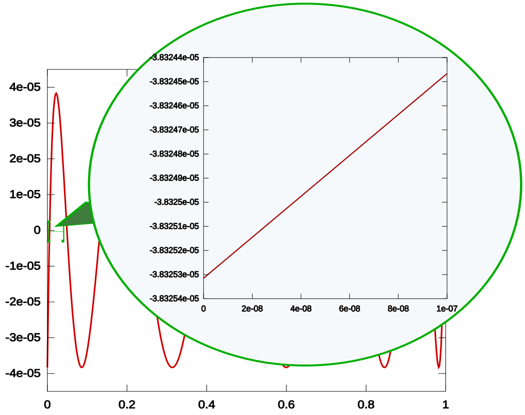
$f(x) = e^{1/\cos(x)}$, $x \in [0, 1]$, $p(x) = \sum_{i=0}^{10} c_i x^i$,
$\varepsilon(x) = f(x) - p(x)$ s.t. $\|\varepsilon\|_\infty = \sup_{x\in[a,b]}\{|\varepsilon(x)|\}$ is as small as possible (Remez algorithm)



Using IA, $\varepsilon(x) \in [-233, 298]$, but $\|\varepsilon(x)\|_\infty \simeq 3.8325 \cdot 10^{-5}$

Example (IA - Dependency phenomenon):

Overestimation can be reduced by using intervals of smaller width.



In this case, over $[0, 1]$ we need 10^7 intervals!

Ad-hoc techniques

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
  - polynomial approximation T
  - interval $\mathbf{\Delta}$ s. t. $f(x) - T(x) \in \mathbf{\Delta}, \forall x \in [a, b]$

Ad-hoc techniques

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
  - polynomial approximation T
  - interval $\mathbf{\Delta}$ s. t. $f(x) - T(x) \in \mathbf{\Delta}, \forall x \in [a, b]$
- Makino and Berz: use Taylor Models[3] for computing $(T, \mathbf{\Delta})$.

[3] Have been formally proved by R. Zumkeller

Ad-hoc techniques

- f replaced with a rigorous polynomial approximation: $(T, \boldsymbol{\Delta})$
  - polynomial approximation T
  - interval $\boldsymbol{\Delta}$ s. t. $f(x) - T(x) \in \boldsymbol{\Delta}, \forall x \in [a, b]$
- Makino and Berz: use Taylor Models[3] for computing $(T, \boldsymbol{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

[3] Have been formally proved by R. Zumkeller

Ad-hoc techniques

- f replaced with a rigorous polynomial approximation: $(T, \boldsymbol{\Delta})$
  - polynomial approximation T
  - interval $\boldsymbol{\Delta}$ s. t. $f(x) - T(x) \in \boldsymbol{\Delta}, \forall x \in [a, b]$
- Makino and Berz: use Taylor Models[3] for computing $(T, \boldsymbol{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$
- Idea used by (Kramer 1996), (Harrison 1997): functions manually handled, one by one.

[3] Have been formally proved by R. Zumkeller

Ad-hoc techniques

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
  - polynomial approximation T
  - interval $\mathbf{\Delta}$ s. t. $f(x) - T(x) \in \mathbf{\Delta}, \forall x \in [a, b]$
- Makino and Berz: use Taylor Models[3] for computing $(T, \mathbf{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$
- Idea used by (Kramer 1996), (Harrison 1997): functions manually handled, one by one.
- None of these techniques correctly handles removable discontinuities.

[3] Have been formally proved by R. Zumkeller

Ingredients of our algorithm

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
  - polynomial approximation T
  - interval $\mathbf{\Delta}$ s. t. $f(x) - T(x) \in \mathbf{\Delta}, \forall x \in [a, b]$
- Makino and Berz: use Taylor Models[3] for computing $(T, \mathbf{\Delta})$.
  Modify them to handle removable discontinuities.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

Note: Next, we'll focus on computing $(T, \mathbf{\Delta})$.

[3] Have been formally proved by R. Zumkeller

Ingredients of our algorithm

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
  - polynomial approximation T
  - interval $\mathbf{\Delta}$ s. t. $f(x) - T(x) \in \mathbf{\Delta}, \forall x \in [a, b]$

- Makino and Berz: use Taylor Models[3] for computing $(T, \mathbf{\Delta})$.
  Modify them to handle removable discontinuities.

- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty \rightsquigarrow$ Relative error case slightly more technical

Note: Next, we'll focus on computing $(T, \mathbf{\Delta})$.

[3] Have been formally proved by R. Zumkeller

Taylor Models

Idea: Consider Taylor approximations

Taylor Models - How do we obtain them?

Idea: Consider Taylor approximations

Let $n \in \mathbb{N}$, $n+1$ times differentiable function f over $[a, b]$ around x_0.

- $f(x) = \underbrace{\sum_{i=0}^{n} \frac{f^{(i)}(x_0)(x-x_0)^i}{i!}}_{T(x)} + \underbrace{\Delta_n(x,\xi)}_{\text{remainder}}$
- $\Delta_n(x,\xi) = \frac{f^{(n+1)}(\xi)(x-x_0)^{n+1}}{(n+1)!}$, $x \in [a, b]$, ξ lies strictly between x and x_0

Taylor Models - How do we obtain them?

Idea: Consider Taylor approximations

Let $n \in \mathbb{N}$, $n+1$ times differentiable function f over $[a, b]$ around x_0.

- $$f(x) = \underbrace{\sum_{i=0}^{n} \frac{f^{(i)}(x_0)(x - x_0)^i}{i!}}_{T(x)} + \underbrace{\Delta_n(x, \xi)}_{\text{remainder}}$$
- $\Delta_n(x, \xi) = \dfrac{f^{(n+1)}(\xi)(x - x_0)^{n+1}}{(n+1)!}$, $x \in [a, b]$, ξ lies strictly between x and x_0

- How to compute the coefficients $\dfrac{f^{(i)}(x_0)}{i!}$ of $T(x)$?
- How to compute an interval enclosure $\boldsymbol{\Delta}$ for $\Delta_n(x, \xi)$?

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \to u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \to v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k \, v_{i-k}$

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \rightarrow u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \rightarrow v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k \, v_{i-k}$

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \to u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \to v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

$$f(x) \to [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \ldots, u_0\, v_4 + u_1\, v_3 + u_2\, v_2 + u_3\, v_1 + u_4\, v_0]$$

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k \, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \to u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \to v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

$$f(x) \to [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \dots, u_0\, v_4 + u_1\, v_3 + u_2\, v_2 + u_3\, v_1 + u_4\, v_0]$$

Automatic Differentiation - Point intervals

Compute $f^{(i)}(x_0)$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k\, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \rightarrow u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \rightarrow v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

$$f(x) \rightarrow [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \ldots, u_0\, v_4 + u_1\, v_3 + u_2\, v_2 + u_3\, v_1 + u_4\, v_0]$$

Automatic Differentiation - Larger intervals

Compute $f^{(i)}([a,b])$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k\, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}(0)$

$$\sin(x) \to u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \to v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

$$f(x) \to [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \ldots, u_0\, v_4 + u_1\, v_3 + u_2\, v_2 + u_3\, v_1 + u_4\, v_0]$$

Automatic Differentiation - Larger intervals

Compute $f^{(i)}([a,b])$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}([a,b]) = \sum_{k=0}^{i} u_k\, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}([0,1])$

$$\sin(x) \to u = [\sin(0), \cos(0), -\sin(0), -\cos(0), \sin(0)]$$

$$\cos(x) \to v = [\cos(0), -\sin(0), -\cos(0), \sin(0), \cos(0)]$$

$$f(x) \to [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \dots, u_0\, v_4 + u_1\, v_3 + u_2\, v_2 + u_3\, v_1 + u_4\, v_0]$$

Automatic Differentiation - Larger intervals

Compute $f^{(i)}([a,b])$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibniz formula: $f^{(i)}([a,b]) = \sum_{k=0}^{i} u_k \, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}([0,1])$

$\sin(x) \to U = [[0, 0.85], [0.54, 1], [-0.85, 0], [-1, -0.54], [0, 0.85]]$

$\cos(x) \to U = [[0.54, 1], [-0.85, 0], [-1, -0.55], [0, 0.85], [0.54, 1]]$

$f(x) \to [u_0 \, v_0, u_0 \, v_1 + u_1 \, v_0, \ldots, u_0 \, v_4 + u_1 \, v_3 + u_2 \, v_2 + u_3 \, v_1 + u_4 \, v_0]$

Automatic Differentiation - Larger intervals

Compute $f^{(i)}([a,b])$ - f represented as an expression tree
- Simple formulas for derivatives of "basic functions": $\exp$, $\sin$, etc.
- Leibnitz formula: $f^{(i)}([a,b]) = \sum_{k=0}^{i} u_k\, v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

Given $f(x) = \sin(x)\cos(x), \text{compute } f^{(4)}([0,1])$

$\sin(x) \to U = [[0, 0.85], [0.54, 1], [-0.85, 0], [-1, -0.54], [0, 0.85]]$

$\cos(x) \to U = [[0.54, 1], [-0.85, 0], [-1, -0.55], [0, 0.85], [0.54, 1]]$

$f(x) \to [u_0\, v_0, u_0\, v_1 + u_1\, v_0, \ldots, [0, 13.5]]$ But $f^{(4)}([0,1]) = [0, 8]$

What happens when f is a composite function?

The interval bound $\mathbf{\Delta}$ for $\Delta_n(x, \xi)$ can be largely overestimated.

What happens when f is a composite function?

The interval bound $\mathbf{\Delta}$ for $\Delta_n(x, \xi)$ can be largely overestimated.

Example :

$f(x) = e^{1/\cos x}$, over $[0, 1]$, $n = 13$, $x_0 = 0.5$.
$f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]$

What happens when f is a composite function?

The interval bound $\boldsymbol{\Delta}$ for $\Delta_n(x,\xi)$ can be largely overestimated.

Example :

$f(x) = e^{1/\cos x}$, over $[0,1]$, $n = 13$, $x_0 = 0.5$.
$f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]$

- Automatic differentiation and Lagrange formula:
  $\boldsymbol{\Delta} = [-1.93 \cdot 10^2,\ 1.35 \cdot 10^3]$

What happens when f is a composite function?

The interval bound $\boldsymbol{\Delta}$ for $\Delta_n(x, \xi)$ can be largely overestimated.

Example :

$f(x) = e^{1/\cos x}$, over $[0, 1]$, $n = 13$, $x_0 = 0.5$.
$f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]$

- Automatic differentiation and Lagrange formula:
  $\boldsymbol{\Delta} = [-1.93 \cdot 10^2,\ 1.35 \cdot 10^3]$
- Cauchy's Estimate
  $\boldsymbol{\Delta} = [-9.17 \cdot 10^{-2},\ 9.17 \cdot 10^{-2}]$

What happens when f is a composite function?

The interval bound $\boldsymbol{\Delta}$ for $\Delta_n(x, \xi)$ can be largely overestimated.

Example :

$f(x) = e^{1/\cos x}$, over $[0, 1]$, $n = 13$, $x_0 = 0.5$.
$f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]$

- Automatic differentiation and Lagrange formula:
  $\boldsymbol{\Delta} = [-1.93 \cdot 10^2,\ 1.35 \cdot 10^3]$
- Cauchy's Estimate
  $\boldsymbol{\Delta} = [-9.17 \cdot 10^{-2},\ 9.17 \cdot 10^{-2}]$
- Taylor Models
  $\boldsymbol{\Delta} = [-9.04 \cdot 10^{-3},\ 9.06 \cdot 10^{-3}]$

Taylor Models Philosophy

For bounding the remainders:

- For “basic functions” use Lagrange formula.
- For “composite functions”use a two-step procedure:
  - compute models (T, I) for all basic functions;
  - apply algebraic rules with these models, instead of operations with the corresponding functions.

Taylor Models - Operations: Addition

Given two Taylor Models for f_1 and f_2, over $[a, b]$, degree n:
$f_1(x) - P_1(x) \in \mathbf{\Delta}_1$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a, b]$.

Addition
$(P_1, \mathbf{\Delta}_1) + (P_2, \mathbf{\Delta}_2) = (P_1 + P_2, \mathbf{\Delta}_1 + \mathbf{\Delta}_2)$.

Taylor Models - Operations: Multiplication

Given two Taylor Models for f_1 and f_2, over $[a,b]$, degree n:
$f_1(x) - P_1(x) \in \mathbf{\Delta}_1$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a,b]$.

Multiplication
We need algebraic rule for: $(P_1, \mathbf{\Delta}_1) \cdot (P_2, \mathbf{\Delta}_2) = (P, \mathbf{\Delta})$ s.t.
$f_1(x) \cdot f_2(x) - P(x) \in \mathbf{\Delta}$, $\forall x \in [a,b]$

Taylor Models - Operations: Multiplication

Given two Taylor Models for f_1 and f_2, over $[a, b]$, degree n:
$f_1(x) - P_1(x) \in \mathbf{\Delta}_1$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a, b]$.

Multiplication
We need algebraic rule for: $(P_1, \mathbf{\Delta}_1) \cdot (P_2, \mathbf{\Delta}_2) = (P, \mathbf{\Delta})$ s.t.
$f_1(x) \cdot f_2(x) - P(x) \in \mathbf{\Delta}$, $\forall x \in [a, b]$

$$f_1(x) \cdot f_2(x) \in \underbrace{P_1 \cdot P_2}_{} + \underbrace{\boldsymbol{P_2} \cdot \mathbf{\Delta}_1 + \boldsymbol{P_1} \cdot \mathbf{\Delta}_2 + \mathbf{\Delta}_1 \cdot \mathbf{\Delta}_2}_{\boldsymbol{I_2}}.$$

$$\underbrace{(P_1 \cdot P_2)_{0 \ldots n}}_{P} + \underbrace{(P_1 \cdot P_2)_{n+1 \ldots 2n}}_{\boldsymbol{I_1}}$$

$$\mathbf{\Delta} = \boldsymbol{I_1} + \boldsymbol{I_2}$$

Taylor Models - Operations: Multiplication

Given two Taylor Models for f_1 and f_2, over $[a,b]$, degree n:
$f_1(x) - P_1(x) \in \mathbf{\Delta}_1$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a,b]$.

Multiplication
We need algebraic rule for: $(P_1, \mathbf{\Delta}_1) \cdot (P_2, \mathbf{\Delta}_2) = (P, \mathbf{\Delta})$ s.t.
$f_1(x) \cdot f_2(x) - P(x) \in \mathbf{\Delta}$, $\forall x \in [a,b]$

$$f_1(x) \cdot f_2(x) \in \underbrace{P_1 \cdot P_2}_{} + \underbrace{\boldsymbol{P_2} \cdot \mathbf{\Delta}_1 + \boldsymbol{P_1} \cdot \mathbf{\Delta}_2 + \mathbf{\Delta}_1 \cdot \mathbf{\Delta}_2}_{\boldsymbol{I_2}}.$$

$$\underbrace{(P_1 \cdot P_2)_{0 \ldots n}}_{P} + \underbrace{(P_1 \cdot P_2)_{n+1 \ldots 2n}}_{\boldsymbol{I_1}}$$

$$\mathbf{\Delta} = \boldsymbol{I_1} + \boldsymbol{I_2}$$

In our case, for bounding "$\boldsymbol{P}$s": IA evaluation.

Taylor Models - Operations: Composition

Given TMs for f_1 over $[c, d]$, for f_2 over $[a, b]$, degree n:
$f_1(y) - P_1(y) \in \mathbf{\Delta}_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a, b]$.

Taylor Models - Operations: Composition

Given TMs for f_1 over $[c,d]$, for f_2 over $[a,b]$, degree n:
$f_1(y) - P_1(y) \in \mathbf{\Delta}_1$, $\forall y \in [c,d]$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a,b]$.

Remark: $(f_1 \circ f_2)(x)$ is f_1 evaluated at $y = f_2(x)$.
We need: $f_2([a,b]) \subseteq [c,d]$, checked by $\boldsymbol{P_2} + \mathbf{\Delta}_2 \subseteq [c,d]$

Taylor Models - Operations: Composition

Given TMs for f_1 over $[c, d]$, for f_2 over $[a, b]$, degree n:
$f_1(y) - P_1(y) \in \mathbf{\Delta}_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a, b]$.

Remark: $(f_1 \circ f_2)(x)$ is f_1 evaluated at $y = f_2(x)$.
We need: $f_2([a, b]) \subseteq [c, d]$, checked by $\boldsymbol{P_2} + \mathbf{\Delta}_2 \subseteq [c, d]$

$f_1(y) \in P_1(y) + \mathbf{\Delta}_1$

Taylor Models - Operations: Composition

Given TMs for f_1 over $[c, d]$, for f_2 over $[a, b]$, degree n:
$f_1(y) - P_1(y) \in \mathbf{\Delta}_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \mathbf{\Delta}_2$, $\forall x \in [a, b]$.

Remark: $(f_1 \circ f_2)(x)$ is f_1 evaluated at $y = f_2(x)$.
We need: $f_2([a, b]) \subseteq [c, d]$, checked by $\boldsymbol{P_2} + \mathbf{\Delta}_2 \subseteq [c, d]$

$f_1(f_2(x)) \in P_1(f_2(x)) + \mathbf{\Delta}_1$

Taylor Models - Operations: Composition

Given TMs for f_1 over $[c, d]$, for f_2 over $[a, b]$, degree n:
$f_1(y) - P_1(y) \in \boldsymbol{\Delta}_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \boldsymbol{\Delta}_2$, $\forall x \in [a, b]$.

Remark: $(f_1 \circ f_2)(x)$ is f_1 evaluated at $y = f_2(x)$.
We need: $f_2([a, b]) \subseteq [c, d]$, checked by $\boldsymbol{P_2} + \boldsymbol{\Delta}_2 \subseteq [c, d]$

$f_1(f_2(x)) \in P_1(P_2(x) + \boldsymbol{\Delta}_2) + \boldsymbol{\Delta}_1$

Extract polynomial and remainder: P_1 can be evaluated using only additions and multiplications: Horner's algorithm

TMs for functions with removable discontinuities

Example: $F(x) = \dfrac{\sin(x)}{\log(1+x)}$ over $[-\dfrac{1}{2}, \dfrac{1}{2}]$.

Taylor expansion exists, but using classical tm arithmetic:
we need a tm for $\dfrac{1}{y}$ over $[\log(\frac{1}{2}), \log(\frac{3}{2})] \ni 0$ which cannot be computed.

TMs for functions with removable discontinuities

Keep remainders in "relative" way:

TM for $f(x) = \sin(x)$ in 0, order $n+1$, over $[-\frac{1}{2}, \frac{1}{2}]$.
TM for $g(x) = \log(1+x)$ in 0, order $n+1$, over $[-\frac{1}{2}, \frac{1}{2}]$.

$$\frac{f}{g} = \frac{x \times (T_f(x) + r_f x^n)}{x \times (T_g(x) + r_g x^n)}$$

Example:

$$\frac{f}{g} = \frac{x \times (1 - \frac{1}{6}x^2 + r_f x^3)}{x \times (1 - \frac{1}{2}x + \frac{1}{3}x^2 + r_g x^3)}$$

TMs for functions with removable discontinuities

Need: tm of order n for $F = \frac{f}{g}$ over I, knowing that $x_0 \in I$ root of order $k > 0$ of f and g.

Idea: Keep remainders in "relative" way: $\mathbf{r} = \mathbf{\Delta} \times (x - x_0)^{n+k}$

- Compute tms of order $n + k$ for f and g in x_0.
- Formally simplify $\frac{f}{g}$ by $(x - x_0)^k$, then compute using regular tm arithmetic
- All operations can be easily extended to work with "relative remainders“

So far, some key points in our supnorm algorithm...

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
- use modified Taylor Models for computing $(T, \mathbf{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

So far, some key points in our supnorm algorithm...

- f replaced with a rigorous polynomial approximation: $(T, \mathbf{\Delta})$
- use modified Taylor Models for computing $(T, \mathbf{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

Note: Our algorithm can fail if $\mathbf{\Delta}$ can not be made small enough.
Solution: Cut the interval into sub-intervals or use ChebModels [4]

[4] N. Brisebarre, M. Joldes, Chebyshev interpolation polynomial-based tools for rigorous computing. In Proceedings of the 2010 international Symposium on Symbolic and Algebraic Computation (Munich, Germany, July 25 - 28, 2010). ACM, New York, NY, 147-154

So far, some key points in our supnorm algorithm...

- f replaced with a rigorous polynomial approximation: $(T, \boldsymbol{\Delta})$
- use modified Taylor Models for computing $(T, \boldsymbol{\Delta})$.
- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

Note: Our algorithm can fail if $\boldsymbol{\Delta}$ can not be made small enough.
Solution: Cut the interval into sub-intervals or use ChebModels [4]

Computing $\|T - p\|_\infty$ not discussed in this talk.

[4] N. Brisebarre, M. Joldes, Chebyshev interpolation polynomial-based tools for rigorous computing. In Proceedings of the 2010 international Symposium on Symbolic and Algebraic Computation (Munich, Germany, July 25 - 28, 2010). ACM, New York, NY, 147-154

Experimental results - Supnorm Algorithm with TMs

f	$[a, b]$	$\deg(p)$	$\deg(T)$	mode	quality $-\log_2 \eta$	time NR	time R
$\exp(x) - 1$	$[-0.25, 0.25]$	5	13	rel.	37.6	14	42
$\log_2(1 + x)$	$[-2^{-9}, 2^{-9}]$	7	17	rel.	83.3	41	103
$\arcsin(x + m)$	$[a_3, b_3]$	22	32	rel.	15.9	270	364
$\cos(x)$	$[-0.5, 0.25]$	15	22	rel.	19.5	93	139
$\exp(x)$	$[-0.125, 0.125]$	25	34	rel.	42.3	337	443
$\sin(x)$	$[-0.5, 0.5]$	9	17	abs.	21.5	13	39
$\exp(\cos^2 x + 1)$	$[1, 2]$	15	44	rel.	25.5	180	747
$\tan(x)$	$[0.25, 0.5]$	10	22	rel.	26.0	47	94
$x^{2.5}$	$[1, 2]$	7	20	rel.	15.5	27	73
$\sin(x)/(\exp(x) - 1)$	$[-2^{-3}, 2^{-3}]$	15	27	abs.	15.5	43	168

Values for example #3:

$$m = \frac{770422123864867}{2^{50}}, \; a_3 = \frac{-205674681606191}{2^{53}}, \; b_3 = \frac{205674681606835}{2^{53}}.$$

In the last columns, "NR" stands for "not rigorous" and "R" stands for "rigorous".

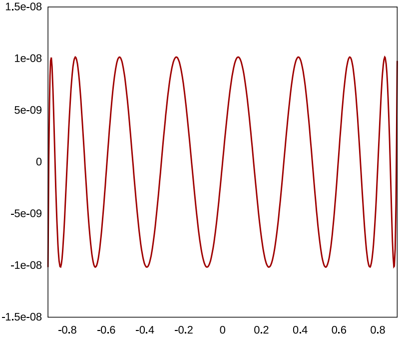
Timings are given in milliseconds on a Core i7-975.

Taylor Models Issues

Example:

$f(x) = \arctan(x)$ over $[-0.9, 0.9]$
$p(x)$ - minimax, degree 15
$\varepsilon(x) = p(x) - f(x)$

$\|\varepsilon\|_\infty \simeq 10^{-8}$



Taylor Models Issues

Example:

$f(x) = \arctan(x)$ over $[-0.9, 0.9]$
$p(x)$ - minimax, degree 15
$\varepsilon(x) = p(x) - f(x)$

$\|\varepsilon\|_\infty \simeq 10^{-8}$

In this case Taylor approximations are not good, we need theoretically a TM of degree 120.

Practically, the computed interval remainder can not be made sufficiently small due to overestimation

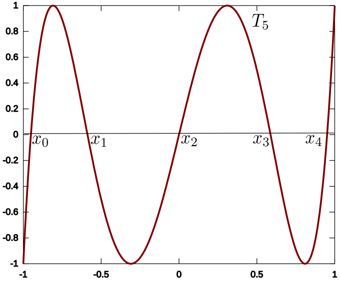
Chebyshev Models

Basic idea:

- Use a polynomial approximation better than Taylor:
  - Chebyshev interpolation polynomial.
  - Chebyshev truncated series.
- Use the two step approach as Taylor Models:
  - compute models (P, I) for basic functions;
  - apply algebraic rules with these models, instead of operations with the corresponding functions (work in Chebyshev basis).

Quick Reminder: Chebyshev Polynomials

Over $[-1, 1]$, $T_n(x) = \cos\left(n \arccos x\right), n \geq 0$.



"Chebyshev nodes": n distinct real roots in $[-1, 1]$ of T_n:
$x_i = \cos\left(\frac{(i+1/2)\,\pi}{n}\right), i = 0, \ldots, n-1.$

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n + 1$.

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n+1$.

Computation of the coefficients

$p_i = \sum_{k=0}^{n} \frac{2}{n+1} f(x_k) T_i(x_k),\ i = 0, \ldots, n$

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n+1$.

Computation of the coefficients

$p_i = \sum_{k=0}^{n} \frac{2}{n+1} f(x_k) T_i(x_k)$, $i = 0, \ldots, n$

Remark: Currently, this step is more costly than in the case of TMs. We can use truncated Chebyshev series instead.

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1,1]$, Chebyshev nodes of order $n+1$.

Computation of the coefficients

Interpolation Error: Lagrange remainder

$\forall x \in [-1,1], \ \exists \xi \in [-1,1] \ \text{ s.t.}$

$f(x) - P(x) = \dfrac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n}(x - x_i).$

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n + 1$.

Computation of the coefficients

Interpolation Error: Lagrange remainder

$\forall x \in [-1, 1], \ \exists \xi \in [-1, 1] \ \text{ s.t.}$

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n}(x - x_i).$$

✓ We should have an improvement of 2^n in the width of the remainder, compared to Taylor remainder.

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n+1$.

Computation of the coefficients

Interpolation Error: Lagrange remainder

$\forall x \in [-1,1], \ \exists \xi \in [-1,1] \ \text{ s.t.}$
$f(x) - P(x) = \dfrac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$

- ✓ We should have an improvement of 2^n in the width of the remainder, compared to Taylor remainder.
- X For composite functions, overestimation of $f^{(n+1)}$

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1,1]$, Chebyshev nodes of order $n+1$.

Computation of the coefficients (for "basic" functions)

Interpolation Error: Lagrange remainder (for "basic" functions)

$\forall x \in [-1,1],\ \exists \xi \in [-1,1]$ s.t.

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n}(x - x_i).$$

- ✓ We should have an improvement of 2^n in the width of the remainder, compared to Taylor remainder.
- X For composite functions, overestimation of $f^{(n+1)}$

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n + 1$.

Computation of the coefficients (for “basic” functions)

Interpolation Error: Lagrange remainder (for “basic” functions)

- For composite functions,use algebraic rules (addition, multiplication, composition) with models

Chebyshev Models: using interpolation polynomial

$P(x) = \sum_{i=0}^{n} p_i T_i(x)$ interpolates f at $x_k \in [-1, 1]$, Chebyshev nodes of order $n + 1$.

Computation of the coefficients (for "basic" functions)

Interpolation Error: Lagrange remainder (for "basic" functions)

- For composite functions,use algebraic rules (addition, multiplication, composition) with models

- Note: Chebfun - "Computing Numerically with Functions Instead of Numbers" (N. Trefethen et al.): Chebyshev interpolation polynomials are already used, but the approach is not rigorous

Chebyshev Models: using truncated Chebyshev series

$$P(x) = {\sum_{k=0}^{n}}' a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} \mathrm{d}x.$$

Chebyshev Models: using truncated Chebyshev series

$$P(x) = {\sum_{k=0}^{n}}' a_k T_k(x), \text{ where } a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\mathrm{d}x.$$

Computation of the coefficients (for “basic” D-finite functions)

- recurrence formulae for computing a_k

Chebyshev Models: using truncated Chebyshev series

$$P(x) = \sum_{k=0}^{n}{}' a_k T_k(x), \text{ where } a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\mathrm{d}x.$$

Computation of the coefficients (for “basic” D-finite functions)

- recurrence formulae for computing a_k Remark: As fast as TMs.

Chebyshev Models: using truncated Chebyshev series

$$P(x) = \sum_{k=0}^{n}{}' a_k T_k(x), \text{ where } a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\mathrm{d}x.$$

Computation of the coefficients (for "basic" D-finite functions)

Truncation Error: Bernstein-like formula (for "basic" D-finite functions)

$$\forall x \in [-1,1],\ \exists \xi \in [-1,1] \ \text{ s.t. } \ f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{2^n (n+1)!}.$$

Chebyshev Models: using truncated Chebyshev series

$$P(x) = {\sum_{k=0}^{n}}' a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} \mathrm{d}x.$$

Computation of the coefficients (for “basic” D-finite functions)

Truncation Error: Bernstein-like formula (for “basic” D-finite functions)

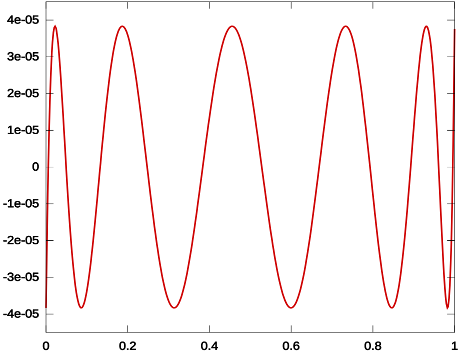
- For composite functions,use algebraic rules (addition, multiplication, composition) with models

Chebyshev Models - Supremum norm example

Example: $\varepsilon(x) = f(x) - p(x)$

$f(x) = e^{1/\cos x}$, over $[0, 1]$, $p(x)$ - minimax, degree 10

$$\|\varepsilon(x)\|_\infty \simeq 3.8325 \cdot 10^{-5}$$



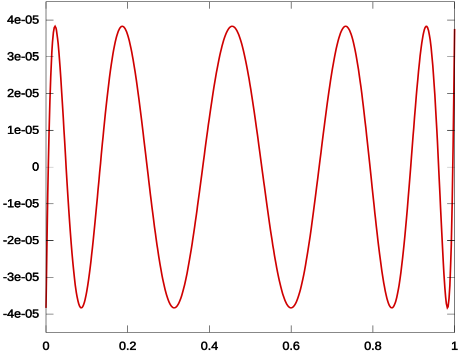
Chebyshev Models - Supremum norm example

Example: $\varepsilon(x) = f(x) - p(x)$

$f(x) = e^{1/\cos x}$, over $[0,1]$, $p(x)$ - minimax, degree 10

$\|\varepsilon(x)\|_\infty \simeq 3.8325 \cdot 10^{-5}$

Need: TM of degree 30.



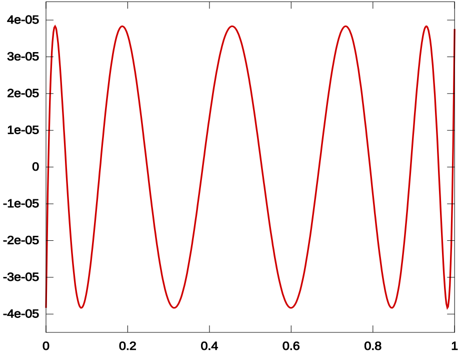
Chebyshev Models - Supremum norm example

Example: $\varepsilon(x) = f(x) - p(x)$

$f(x) = e^{1/\cos x}$, over $[0, 1]$, $p(x)$ - minimax, degree 10

$\|\varepsilon(x)\|_\infty \simeq 3.8325 \cdot 10^{-5}$

Need: TM of degree 30.
CM of degree 13.



CMs vs. TMs

Comparison between remainder bounds for several functions:

$f(x)$, I, n	CM	Exact bound	TM	Exact bound
$\sin(x)$, $[3, 4]$, 10	$1.19 \cdot 10^{-14}$	$1.13 \cdot 10^{-14}$	$1.22 \cdot 10^{-11}$	$1.16 \cdot 10^{-11}$
$\arctan(x)$, $[-0.25, 0.25]$, 15	$7.89 \cdot 10^{-15}$	$7.95 \cdot 10^{-17}$	$2.58 \cdot 10^{-10}$	$3.24 \cdot 10^{-12}$
$\arctan(x)$, $[-0.9, 0.9]$, 15	$5.10 \cdot 10^{-3}$	$1.76 \cdot 10^{-8}$	$1.67 \cdot 10^{2}$	$5.70 \cdot 10^{-3}$
$\exp(1/\cos(x))$, $[0, 1]$, 14	$5.22 \cdot 10^{-7}$	$4.95 \cdot 10^{-7}$	$9.06 \cdot 10^{-3}$	$2.59 \cdot 10^{-3}$
$\frac{\exp(x)}{\log(2+x)\cos(x)}$, $[0, 1]$, 15	$9.11 \cdot 10^{-9}$	$2.21 \cdot 10^{-9}$	$1.18 \cdot 10^{-3}$	$3.38 \cdot 10^{-5}$
$\sin(\exp(x))$,$[-1, 1]$, 10	$9.47 \cdot 10^{-5}$	$3.72 \cdot 10^{-6}$	$2.96 \cdot 10^{-2}$	$1.55 \cdot 10^{-3}$

CMs vs. TMs

Operations complexity:

- ✓ Addition ($O(n)$), Multiplication($O(n^2)$) and Composition ($O(n^3)$) have similar complexity.
- ✓ Initial computation of coefficients for all “basic” D-finite functions is similar ($O(n)$).

Comparison between remainder bounds for several functions:

$f(x)$, I, n	CM	Exact bound	TM	Exact bound
$\sin(x)$, $[3, 4]$, 10	$1.19 \cdot 10^{-14}$	$1.13 \cdot 10^{-14}$	$1.22 \cdot 10^{-11}$	$1.16 \cdot 10^{-11}$
$\arctan(x)$, $[-0.25, 0.25]$, 15	$7.89 \cdot 10^{-15}$	$7.95 \cdot 10^{-17}$	$2.58 \cdot 10^{-10}$	$3.24 \cdot 10^{-12}$
$\arctan(x)$, $[-0.9, 0.9]$, 15	$5.10 \cdot 10^{-3}$	$1.76 \cdot 10^{-8}$	$1.67 \cdot 10^{2}$	$5.70 \cdot 10^{-3}$
$\exp(1/\cos(x))$, $[0, 1]$, 14	$5.22 \cdot 10^{-7}$	$4.95 \cdot 10^{-7}$	$9.06 \cdot 10^{-3}$	$2.59 \cdot 10^{-3}$
$\frac{\exp(x)}{\log(2+x)\cos(x)}$, $[0, 1]$, 15	$9.11 \cdot 10^{-9}$	$2.21 \cdot 10^{-9}$	$1.18 \cdot 10^{-3}$	$3.38 \cdot 10^{-5}$
$\sin(\exp(x))$,$[-1, 1]$, 10	$9.47 \cdot 10^{-5}$	$3.72 \cdot 10^{-6}$	$2.96 \cdot 10^{-2}$	$1.55 \cdot 10^{-3}$

Quality of approximation compared to minimax

Remark: It is known [Ehlich & Zeller, 1966] that Chebyshev interpolants are "near-best":

$$\|\varepsilon\|_\infty \leq \underbrace{\left(2 + (2/\pi)\log(n)\right)}_{\Lambda_n} \|\varepsilon_{\mathrm{minimax}}\|_\infty$$

- $\Lambda_{15} = 3.72... \rightarrow$ we lose at most 2 bits
- $\Lambda_{30} = 4.16... \rightarrow$ we lose at most 3 bits
- $\Lambda_{100} = 4.93... \rightarrow$ we lose at most 3 bits
- $\Lambda_{100000} = 9.32... \rightarrow$ we lose at most 4 bits

Quality of approximation compared to minimax

No	$f(x)$, I, n	CM	Exact bound	Minimax
1	$\sin(x)$, $[3,\ 4]$, 10	$1.19 \cdot 10^{-14}$	$1.13 \cdot 10^{-14}$	$1.12 \cdot 10^{-14}$
2	$\arctan(x)$, $[-0.25,\ 0.25]$, 15	$7.89 \cdot 10^{-15}$	$7.95 \cdot 10^{-17}$	$4.03 \cdot 10^{-17}$
3	$\arctan(x)$, $[-0.9,\ 0.9]$, 15	$5.10 \cdot 10^{-3}$	$1.76 \cdot 10^{-8}$	$1.01 \cdot 10^{-8}$
4	$\exp(1/\cos(x))$, $[0,\ 1]$, 14	$5.22 \cdot 10^{-7}$	$4.95 \cdot 10^{-7}$	$3.57 \cdot 10^{-7}$
5	$\frac{\exp(x)}{\log(2+x)\ \cos(x)}$, $[0,\ 1]$, 15	$9.11 \cdot 10^{-9}$	$2.21 \cdot 10^{-9}$	$1.72 \cdot 10^{-9}$
6	$\sin(\exp(x))$,$[-1,\ 1]$, 10	$9.47 \cdot 10^{-5}$	$3.72 \cdot 10^{-6}$	$1.78 \cdot 10^{-6}$

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.
- Rigorous polynomial approximations are essential.

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.
- Rigorous polynomial approximations are essential.
  $\rightsquigarrow$ Work in progress: compute CMs for any D-finite function based on the given differential equation (direct application to rigorous ODE solving)

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.
- Rigorous polynomial approximations are essential.
  $\rightsquigarrow$ Work in progress: compute CMs for any D-finite function based on the given differential equation (direct application to rigorous ODE solving)
- The algorithm generates a partial formal proof, together with the supremum norm itself.

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.
- Rigorous polynomial approximations are essential.
  $\rightsquigarrow$ Work in progress: compute CMs for any D-finite function based on the given differential equation (direct application to rigorous ODE solving)
- The algorithm generates a partial formal proof, together with the supremum norm itself.

  $\rightsquigarrow$ Work remains for formally proving the rigorous polynomial approximation part.

Conclusion

- Supnorms computation is automatic, efficient, has *a priori* control of result quality.
- Rigorous polynomial approximations are essential.
  $\rightsquigarrow$ Work in progress: compute CMs for any D-finite function based on the given differential equation (direct application to rigorous ODE solving)
- The algorithm generates a partial formal proof, together with the supremum norm itself.

  $\rightsquigarrow$ Work remains for formally proving the rigorous polynomial approximation part.
- Is an approximation of WORSE quality really BETTER?