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.
Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
- Previous techniques
Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  - Previous techniques
- Taylor Models and Enhancements
Outline

- Origin and Motivation of Supremum Norms Computations
- Previous techniques
- Taylor Models and Enhancements
- Chebyshev Models
Supremum Norms and Rigorous Polynomial Approximations

Outline

- Origin and Motivation of Supremum Norms Computations
  - 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

$\parallel \varepsilon \parallel_\infty = \sup_{x \in [a, b]} \{|\varepsilon (x)|\}$
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 $[a, b]$. 
  - 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 $\varepsilon(x) = p(x)/f(x) - 1$. 

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:

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

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

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

\[ \varepsilon(x) = f(x) - p(x) \quad \text{s.t.} \quad \|\varepsilon\|_\infty = \sup_{x \in [a, b]} \{|\varepsilon(x)|\} \quad \text{is as small as possible (Remez algorithm)} \]
Example:

\[ f(x) = e^{1/\cos(x)}, \ x \in [0, 1], \ p(x) = \sum_{i=0}^{10} c_i x^i, \]
\[ \varepsilon(x) = f(x) - p(x) \text{ s.t. } \|\varepsilon\|_{\infty} = \sup_{x \in [a, b]} \{|\varepsilon(x)|\} \text{ is as small as possible (Remez algorithm)} \]
Our problem - How to find $\ell$ and $u$ such that $\|\varepsilon\|_{\infty} \in [\ell, u]$?
Our problem - How to find $\ell$ and $u$ such that $\|\varepsilon\|_\infty \in [\ell, u]$?

$\ell$ - lower bound for $\|\varepsilon\|_\infty$

( easily obtained by numerical methods)

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

$\varepsilon = f - p$

$u$ - upper bound for $\|\varepsilon\|_\infty$

$\ell$ - lower bound for $\|\varepsilon\|_\infty$
(easily obtained by numerical methods)
Our problem - How to find $\ell$ and $u$ such that $\|\varepsilon\|_\infty \in [\ell, u]$?

$\eta$ - quality (input parameter)

$\ell \eta$  

$\ell$ - lower bound for $\|\varepsilon\|_\infty$  
(easily obtained by numerical methods)

$u$ - upper bound for $\|\varepsilon\|_\infty$

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

- $\eta$ - quality (input parameter)
- $\ell$ - lower bound for $\|\epsilon\|_\infty$
- $u$ - upper bound for $\|\epsilon\|_\infty$
- (easily obtained by numerical methods)

$\epsilon = f - p$

Actual difficulty: certified upper bound $u$
Our approach

- Automatic: no parameter requires to be manually adjusted.

\[
\begin{align*}
\text{Automatic: no parameter requires to be manually adjusted.}
\end{align*}
\]
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 $\eta$ 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 $\eta$ 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 $\eta$ 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.

\(^{1}\text{Hansen '92, Kearfott '96, Messine '97}\)
State of the art

It is a univariate global optimization problem.

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

\footnote{Hansen '92, Kearfott '96, Messine '97}
State of the art

It is a univariate global optimization problem.

- Purely numerical algorithms: find the zeros of $\varepsilon'$ (e.g., Newton’s algorithm).
  $\leadsto$ 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\(^2\))

\[ \pi \in [3.1415, 3.1416] \]

Interval Arithmetic Operations

E.g. \([1, 2] + [-3, 2] = [-2, 4] \]

Range bounding for functions

E.g. \(x \in [-1, 2], f(x) = x^2 - x + 1\)

\[ F(X) = X^2 - X + 1 \]

\[ F([-1, 2]) = [-1, 2]^2 - [-1, 2] + [1, 1] = [-1, 6] \]

\[ x \in [-1, 2], f(x) \in [-1, 6], but \text{Im}(f) = [\frac{3}{4}, 3] \]

\(^2\text{http://gforge.inria.fr/projects/mpfi/}\)
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. \(x \in [-1, 2], f(x) = x^2 - x + 1\)
  \(F(X) = X^2 - X + 1\)
  \(F([-1, 2]) = [-1, 2]^2 - [-1, 2] + [1, 1]\)
  \(F([-1, 2]) = [0, 4] - [-1, 2] + [1, 1]\)
  \(F([-1, 2]) = [-1, 6]\)
  \(x \in [-1, 2], f(x) \in [-1, 6], \text{ but } \text{Im}(f) = [3/4, 3]\)

\(^2\text{http://gforge.inria.fr/projects/mpfi/}\)
Example (IA - Dependency phenomenon):

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

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

Using IA, \( \varepsilon(x) \in [-233, 298] \), but \( \|\varepsilon(x)\|_\infty \approx 3.8325 \cdot 10^{-5} \)
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: $(T, \Delta)$
  - polynomial approximation $T$
  - interval $\Delta$ s. t. $f(x) - T(x) \in \Delta$, $\forall x \in [a, b]$
Ad-hoc techniques

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

- Makino and Berz: use Taylor Models\(^3\) for computing $(T, \Delta)$.

\(^3\)Have been formally proved by R. Zumkeller
Ad-hoc techniques

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

- Makino and Berz: use Taylor Models\(^3\) for computing $(T, \Delta)$.

- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

\(^3\)Have been formally proved by R. Zumkeller
Ad-hoc techniques

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

- Makino and Berz: use Taylor Models\(^3\) for computing $(T, \Delta)$.

- $\|f - p\|_{\infty} \leq \|f - T\|_{\infty} + \|T - p\|_{\infty}$

- 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: $(T, \Delta)$
  - polynomial approximation $T$
  - interval $\Delta$ s. t. $f(x) - T(x) \in \Delta$, $\forall x \in [a, b]$

- Makino and Berz: use Taylor Models\(^3\) for computing $(T, \Delta)$.

\[ \|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty \]

- 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: \((T, \Delta)\)
  - polynomial approximation \(T\)
  - interval \(\Delta\) s. t. \(f(x) - T(x) \in \Delta, \forall x \in [a, b]\)

- Makino and Berz: use Taylor Models\(^3\) for computing \((T, \Delta)\).
  Modify them to handle removable discontinuities.

- \(\|f - p\|_{\infty} \leq \|f - T\|_{\infty} + \|T - p\|_{\infty}\)

Note: Next, we’ll focus on computing \((T, \Delta)\).

\(^3\)Have been formally proved by R. Zumkeller
Ingredients of our algorithm

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

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

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

Note: Next, we’ll focus on computing $(T, \Delta)$.

\(^3\)Have been formally proved by R. Zumkeller
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 \( x_0 \).

\[
f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)(x - x_0)^i}{i!} + \Delta_n(x, \xi)
\]

\( \Delta_n(x, \xi) = \frac{f^{(n+1)}(\xi)(x - x_0)^{n+1}}{(n + 1)!}, \ x \in [a, b], \ \xi \) 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 $x_0$.

\[
f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)(x - x_0)^i}{i!} + \Delta_n(x, \xi)
\]

\[
\Delta_n(x, \xi) = \frac{f^{(n+1)}(\xi)(x - x_0)^{n+1}}{(n + 1)!}, \ x \in [a, b], \ \xi \text{ lies strictly between } x \text{ and } x_0
\]

- How to compute the coefficients $\frac{f^{(i)}(x_0)}{i!}$ of $T(x)$?
- How to compute an interval enclosure $\Delta$ 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 \( f(x) = \sin(x) \cos(x) \), 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 $f(x) = \sin(x) \cos(x)$, 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 $f(x) = \sin(x) \cos(x)$, 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 $f(x) = \sin(x) \cos(x)$, 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 - 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 $f(x) = \sin(x) \cos(x)$, 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 - 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: $f^{(i)}(x_0) = \sum_{k=0}^{i} u_k \cdot v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

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

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

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

$f(x) \rightarrow [u_0 \cdot v_0, u_0 \cdot v_1 + u_1 \cdot v_0, \ldots, u_0 \cdot v_4 + u_1 \cdot v_3 + u_2 \cdot v_2 + u_3 \cdot v_1 + u_4 \cdot v_0]$
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 $f(x) = \sin(x) \cos(x)$, 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: $f^{(i)}([a, b]) = \sum_{k=0}^{i} u_k v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

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

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

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

$f(x) \rightarrow [u_0 v_0, u_0 v_1 + u_1 v_0, \ldots, u_0 v_4 + u_1 v_3 + u_2 v_2 + u_3 v_1 + u_4 v_0]$
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: $f^{(i)}([a, b]) = \sum_{k=0}^{i} u_k v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

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

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

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

$f(x) \rightarrow [u_0 v_0, u_0 v_1 + u_1 v_0, \ldots, u_0 v_4 + u_1 v_3 + u_2 v_2 + u_3 v_1 + u_4 v_0]$
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: $f^{(i)}([a, b]) = \sum_{k=0}^{i} u_k v_{i-k}$
- For composite functions, recursively apply operations (addition, multiplication, composition)

Example:

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

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

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

$f(x) \rightarrow [u_0 v_0, u_0 v_1 + u_1 v_0, \ldots, [0, 13.5]]$ But $f^{(4)}([0, 1]) = [0, 8]$
What happens when $f$ is a composite function?

The interval bound $\Delta$ for $\Delta_n(x, \xi)$ can be largely overestimated.
What happens when $f$ is a composite function?

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

Example:

$$f(x) = \frac{e^1}{\cos x}, \quad \text{over} \ [0, 1], \ n = 13, \ x_0 = 0.5.$$  
$$f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]$$
What happens when $f$ is a composite function?

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

Example:

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

- Automatic differentiation and Lagrange formula:
  $$\Delta = [-1.93 \cdot 10^2, \ 1.35 \cdot 10^3]$$
What happens when $f$ is a composite function?

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

Example:

\[
f(x) = e^{1/\cos x}, \text{ over } [0, 1], \ n = 13, \ x_0 = 0.5.
\]

\[
f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]
\]

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

- Cauchy’s Estimate
  \[
  \Delta = [-9.17 \cdot 10^{-2}, 9.17 \cdot 10^{-2}]
  \]
What happens when \( f \) is a composite function?

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

**Example:**

\[
f(x) = e^{1/\cos x}, \text{ over } [0, 1], \ n = 13, \ x_0 = 0.5.
\]

\[
f(x) - T(x) \in [0, 4.56 \cdot 10^{-3}]
\]

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

- **Cauchy’s Estimate**
  \[
  \Delta = [-9.17 \cdot 10^{-2}, 9.17 \cdot 10^{-2}]
  \]

- **Taylor Models**
  \[
  \Delta = [-9.04 \cdot 10^{-3}, 9.06 \cdot 10^{-3}]
  \]
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.
Given two Taylor Models for $f_1$ and $f_2$, over $[a, b]$, degree $n$: 
$f_1(x) - P_1(x) \in \Delta_1$ and $f_2(x) - P_2(x) \in \Delta_2$, $\forall x \in [a, b]$.

Addition

$(P_1, \Delta_1) + (P_2, \Delta_2) = (P_1 + P_2, \Delta_1 + \Delta_2)$. 
Given two Taylor Models for $f_1$ and $f_2$, over $[a, b]$, degree $n$: $f_1(x) - P_1(x) \in \Delta_1$ and $f_2(x) - P_2(x) \in \Delta_2$, $\forall x \in [a, b]$.

Multiplication

We need algebraic rule for: $(P_1, \Delta_1) \cdot (P_2, \Delta_2) = (P, \Delta)$ s.t. $f_1(x) \cdot f_2(x) - P(x) \in \Delta$, $\forall x \in [a, b]$.
Taylor Models - Operations: Multiplication

Given two Taylor Models for \( f_1 \) and \( f_2 \), over \([a, b]\), degree \( n\):
\[ f_1(x) - P_1(x) \in \Delta_1 \] and \[ f_2(x) - P_2(x) \in \Delta_2, \forall x \in [a, b]. \]

Multiplication
We need algebraic rule for: \((P_1, \Delta_1) \cdot (P_2, \Delta_2) = (P, \Delta)\) s.t.
\[ f_1(x) \cdot f_2(x) - P(x) \in \Delta, \forall x \in [a, b], \]
\[ f_1(x) \cdot f_2(x) \in P_1 \cdot P_2 + P_2 \cdot \Delta_1 + P_1 \cdot \Delta_2 + \Delta_1 \cdot \Delta_2. \]
\[ (P_1 \cdot P_2)_{0...n} + (P_1 \cdot P_2)_{n+1...2n} \]
\[ \Delta = I_1 + I_2 \]
Given two Taylor Models for $f_1$ and $f_2$, over $[a, b]$, degree $n$: $f_1(x) - P_1(x) \in \Delta_1$ and $f_2(x) - P_2(x) \in \Delta_2$, $\forall x \in [a, b]$.

Multiplication
We need algebraic rule for: $(P_1, \Delta_1) \cdot (P_2, \Delta_2) = (P, \Delta)$ s.t. $f_1(x) \cdot f_2(x) - P(x) \in \Delta$, $\forall x \in [a, b]$

\[
f_1(x) \cdot f_2(x) \in P_1 \cdot P_2 + P_2 \cdot \Delta_1 + P_1 \cdot \Delta_2 + \Delta_1 \cdot \Delta_2.
\]

\[
(P_1 \cdot P_2)_{0...n} + (P_1 \cdot P_2)_{n+1...2n}
\]

\[
\Delta = I_1 + I_2
\]

In our case, for bounding “Ps”: IA evaluation.
Taylor Models - Operations: Composition

Given TMs for \( f_1 \) over \([c, d]\), for \( f_2 \) over \([a, b]\), degree \( n\):

\[ f_1(y) - P_1(y) \in \Delta_1, \forall y \in [c, d] \quad \text{and} \quad f_2(x) - P_2(x) \in \Delta_2, \forall x \in [a, b]. \]
Given TMs for $f_1$ over $[c, d]$, for $f_2$ over $[a, b]$, degree $n$:
$f_1(y) - P_1(y) \in \Delta_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \Delta_2$, $\forall x \in [a, b]$.

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 $P_2 + \Delta_2 \subseteq [c, d]$. 
Given TMs for $f_1$ over $[c, d]$, for $f_2$ over $[a, b]$, degree $n$:
$f_1(y) - P_1(y) \in \Delta_1$, $\forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \Delta_2$, $\forall x \in [a, b]$.

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 $P_2 + \Delta_2 \subseteq [c, d]$

$f_1(y) \in P_1(y) + \Delta_1$
Given TMs for $f_1$ over $[c, d]$, for $f_2$ over $[a, b]$, degree $n$:

$f_1(y) - P_1(y) \in \Delta_1, \forall y \in [c, d]$ and $f_2(x) - P_2(x) \in \Delta_2, \forall x \in [a, b]$.

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 $P_2 + \Delta_2 \subseteq [c, d]$.

$f_1(f_2(x)) \in P_1(f_2(x)) + \Delta_1$
Given TMs for \( f_1 \) over \([c, d]\), for \( f_2 \) over \([a, b]\), degree \( n\):

\[
f_1(y) - P_1(y) \in \Delta_1, \forall y \in [c, d] \text{ and } f_2(x) - P_2(x) \in \Delta_2, \forall x \in [a, b].
\]

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 \( P_2 + \Delta_2 \subseteq [c, d]\)

\[
f_1(f_2(x)) \in P_1(P_2(x) + \Delta_2) + \Delta_1
\]

Extract polynomial and remainder: \( P_1 \) can be evaluated using only additions and multiplications: Horner’s algorithm
TMs for functions with removable discontinuities

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

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

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

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

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

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: \( r = \Delta \times (x - x_0)^{n+k} \)

- 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: \((T, \Delta)\)
- use modified Taylor Models for computing \((T, \Delta)\).
- \( \|f - p\|_{\infty} \leq \|f - T\|_{\infty} + \|T - p\|_{\infty} \)
So far, some key points in our supnorm algorithm...

- $f$ replaced with a rigorous polynomial approximation: $(T, \Delta)$

- Use modified Taylor Models for computing $(T, \Delta)$.

- $\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$

Note: Our algorithm can fail if $\Delta$ cannot be made small enough. Solution: Cut the interval into sub-intervals or use ChebModels.

---

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

- $f$ replaced with a rigorous polynomial approximation: $(T, \Delta)$
- use modified Taylor Models for computing $(T, \Delta)$.

$$\|f - p\|_\infty \leq \|f - T\|_\infty + \|T - p\|_\infty$$

Note: Our algorithm can fail if $\Delta$ 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.

---

# Experimental results - Supnorm Algorithm with TMs

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

Values for example #3:

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

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:

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

\[ \|\varepsilon\|_\infty \approx 10^{-8} \]
Example:

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

\[ \| \varepsilon \|_{\infty} \approx 10^{-8} \]

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

“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**

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

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**

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

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

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

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

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

✗ 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)

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

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

✗ 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

\[ P(x) = \sum_{k=0}^{n} a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} \, dx. \]
Chebyshev Models: using truncated Chebyshev series

\[ P(x) = \sum_{k=0}^{n} a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1 - x^2}} \, dx. \]

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

- recurrence formulae for computing \( a_k \)
Chebyshev Models: using truncated Chebyshev series

\[ P(x) = \sum_{k=0}^{n} a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} \, dx. \]

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

\[ P(x) = \sum_{k=0}^{n} a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} \, dx. \]

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

\[ P(x) = \sum_{k=0}^{n} a_k T_k(x), \text{ where } a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} \, dx. \]

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

\[ \|\varepsilon(x)\|_\infty \approx 3.8325 \cdot 10^{-5} \]
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:** \( \varepsilon(x) = f(x) - p(x) \)

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

\[ \|\varepsilon(x)\|_{\infty} \approx 3.8325 \cdot 10^{-5} \]

Need: TM of degree 30.
CM of degree 13.
Comparison between remainder bounds for several functions:

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

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

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

\[ \| \varepsilon \|_\infty \leq \left( 2 + \frac{2}{\pi} \log(n) \right) \frac{\| \varepsilon \|_\infty}{\Lambda_n} \]

- \( \Lambda_{15} = 3.72... \rightarrow \) 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

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

- Supnorms computation is automatic, efficient, has \textit{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.
  \[\leadsto\] 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.
  - 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. 
  \(\Rightarrow\) 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.
  \(\Rightarrow\) 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.
  ~ 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.
  ~ Work remains for formally proving the rigorous polynomial approximation part.
- Is an approximation of *WORSE* quality really *BETTER*?