Designing a Correct Numerical Algorithm

Christoph Lauter Guillaume Melquiond

March 27, 2013

Outline

Outline

Motivation: Implementing Mathematical Functions

Our work: Designing methods and tools to help with the floating-point implementation of mathematical functions.

Motivation: Implementing Mathematical Functions

Our work: Designing methods and tools to help with the floating-point implementation of mathematical functions.

Features:

- correct rounding (deterministic, portable, suitable for proofs),

Motivation: Implementing Mathematical Functions

Our work: Designing methods and tools to help with the floating-point implementation of mathematical functions.

Features:

- correct rounding (deterministic, portable, suitable for proofs),
- as fast as crude implementations,

Motivation: Implementing Mathematical Functions

Our work: Designing methods and tools to help with the floating-point implementation of mathematical functions.

Features:

- correct rounding (deterministic, portable, suitable for proofs),
- as fast as crude implementations,
- high confidence in the correctness.

Implementing a Mathematical Function

Peculiarities of such implementations:

- One needs to bound the maximum error
  (and not the average error, as is done for signal processing).

Implementing a Mathematical Function

Peculiarities of such implementations:

- One needs to bound the maximum error
  (and not the average error, as is done for signal processing).
- Functions are straight-line codes,
  so as to avoid costly control flow.

Implementing a Mathematical Function

Peculiarities of such implementations:

- One needs to bound the maximum error
  (and not the average error, as is done for signal processing).
- Functions are straight-line codes,
  so as to avoid costly control flow.
- Accuracy and precision are fixed
  (no multi-precision algorithms *à la* MPFR).

Implementing a Mathematical Function

Peculiarities of such implementations:

- One needs to bound the maximum error
  (and not the average error, as is done for signal processing).
- Functions are straight-line codes,
  so as to avoid costly control flow.
- Accuracy and precision are fixed
  (no multi-precision algorithms *à la* MPFR).
- Algorithm and proof are devised at the same time.

Tools: Sollya and Gappa

Environment for high-confidence code developments

Tools: Sollya and Gappa

Environment for high-confidence code developments:

- Sollya: the mathematical side of the process.

Tools: Sollya and Gappa

Environment for high-confidence code developments:

- Sollya: the mathematical side of the process.
  - Finding approximation polynomials.

Tools: Sollya and Gappa

Environment for high-confidence code developments:

- Sollya: the mathematical side of the process.
    - Finding approximation polynomials.
    - Bounding truncation and quantization errors.

Tools: Sollya and Gappa

Environment for high-confidence code developments:

- Sollya: the mathematical side of the process.
  - Finding approximation polynomials.
  - Bounding truncation and quantization errors.
- Gappa: the floating-point side of the process.

Tools: Sollya and Gappa

Environment for high-confidence code developments:

- Sollya: the mathematical side of the process.
  - Finding approximation polynomials.
  - Bounding truncation and quantization errors.
- Gappa: the floating-point side of the process.
  - Bounding round-off and global errors.

Running Example

Example (Exponential)

A floating-point implementation of the exponential function with a relative accuracy of 2^{-42} (but no correct rounding).

Running Example

Example (Exponential)

A floating-point implementation of the exponential function with a relative accuracy of 2^{-42} (but no correct rounding).

Constraints:

- a C function working on `binary64` numbers,
- for any finite input x and any finite output y,

$$\left| \frac{y}{\exp x} - 1 \right| \leq 2^{-42},$$

- efficiency!

Vocabulary

Definition (Precision and accuracy)

- Precision: number format used for input, output, and computations.
- Accuracy: quality of the results.

Vocabulary

Definition (Validation and verification)

How to ensure that a program/device is fit for a purpose?

- Validation: black box, experimental process.
- Verification: abstraction, mathematical process.

Vocabulary

Definition (Validation and verification)

How to ensure that a program/device is fit for a purpose?

- Validation: black box, experimental process.
- Verification: abstraction, mathematical process.

Definition (Certification and qualification)

- Certification: assessing software safety according to regulations.
- Qualification: making tools suitable for use during certification.

Basic Concepts: Floating-Point Arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.

– IEEE-754 standard for FP arithmetic

Basic Concepts: Floating-Point Arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.

– IEEE-754 standard for FP arithmetic

- Concise specification, suitable for program verification.

Basic Concepts: Floating-Point Arithmetic

Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result.
– IEEE-754 standard for FP arithmetic

- Concise specification, suitable for program verification.
- It is all about real numbers:
  $\circ(x)$ is the real value of the floating-point number the closest to the real number x, given a format and a rounding mode.

Basic Concepts: Interval Arithmetic

Interval evaluations can serve as proofs of bounds,
when they satisfy the containment property:

$$x \in I_x \wedge y \in I_y \quad \Longrightarrow \quad x \diamond y \in I_z \quad \text{if } I_x \diamond I_y \subseteq I_z$$

for $\diamond \in \{+, -, \times, \div\}$. Also for unary functions: $\sqrt{\cdot}$, sin, and so on.

Basic Concepts: Interval Arithmetic

Interval evaluations can serve as proofs of bounds,
when they satisfy the containment property:

$$x \in I_x \wedge y \in I_y \implies x \diamond y \in I_z \quad \text{if } I_x \diamond I_y \subseteq I_z$$

for $\diamond \in \{+, -, \times, \div\}$. Also for unary functions: $\sqrt{\cdot}$, sin, and so on.

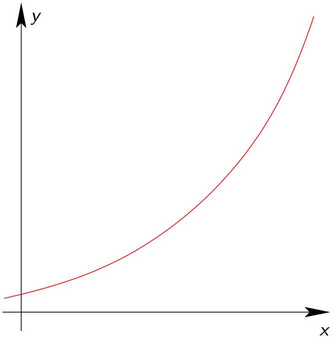
Arithmetic operations on intervals:

- $[a, b] + [c, d] = [a + c, b + d]$,
- $[a, b] - [c, d] = [a - d, b - c]$,
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$,
- $[a, b] \div [c, d] = [a, b] \times [c, d]^{-1}$
  with $[c, d]^{-1} = [1/d, 1/c]$ if $0 \notin [c, d]$.

Outline

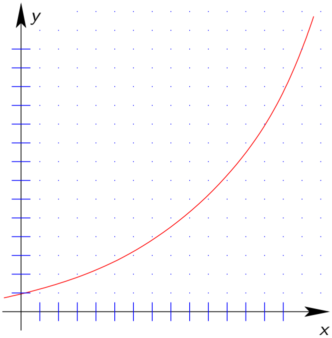
2 Implementation theory
- Functions on computers
- Ingredients for an implementation
- Implementation schemes

Elementary functions implemented on Computers



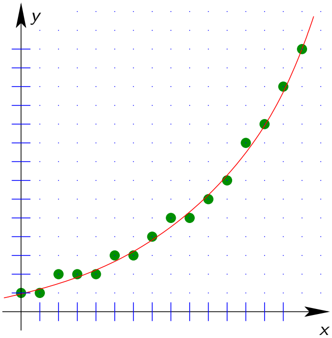
$$x \xrightarrow{f} f(x)$$

Elementary functions implemented on Computers



$$\forall x \in \mathbb{F}_k, \qquad x \xmapsto{f} f(x) \xmapsto{\circ_k}$$

Elementary functions implemented on Computers



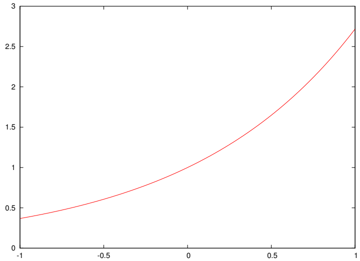
$$\forall x \in \mathbb{F}_k, \qquad x \overset{f}{\longmapsto} f(x) \overset{\circ_k}{\longmapsto} \circ_k(f(x))$$

Disclaimer for this section

- No multi-precision algorithms
  - The input/output precisions are supposed to be known
  - E.g. exp gets implemented on `binary64` with 53 bits
  - Implementation of exp with k bits of output precision is harder.
    - Refer to the MPFR library for that.
- No techniques especially developed for hardware
  - The different `libms` are implemented in software
  - Some processors do integrated specialized hardware
    - but it is less and less used for common `libm` usecases.
    - If it is used, it is on Intel/AMD, where it is microcode, i.e. software.
  - There are some specialized techniques for hardware, such as CORDIC.

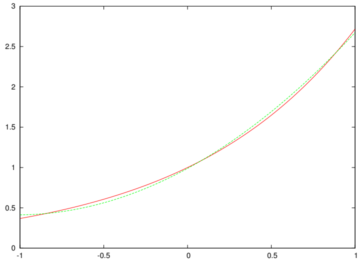
Polynomial approximation

- How to implement $f = \exp$...
- ...on a small domain $I = [-1; 1]$, to start with?



Polynomial approximation

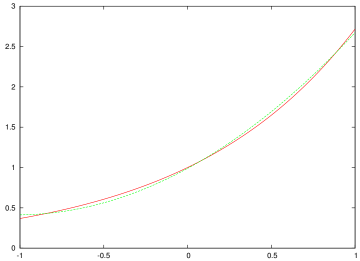
- How to implement $f = \exp\ldots$
- $\ldots$ on a small domain $I = [-1; 1]$, to start with?



- Solution: replace f by an approximation polynomial p

Polynomial approximation

- How to implement $f = \exp$...
- ...on a small domain $I = [-1; 1]$, to start with?



- Solution: replace f by an approximation polynomial p
- Classes of polynomials:
  - Weierstraß tells us that the techniques always work
  - Taylor expansion as a first idea
  - Polynomials that minimize the maximum error on the domain.

Why polynomials?

- Why polynomials and no other approximants?
  - Why no (truncated) continued fractions?
  - Why no approximations with other base functions?

Why polynomials?

- Why polynomials and no other approximants?
  - Why no (truncated) continued fractions?
  - Why no approximations with other base functions?

- A (almost) purely technological answer
  - Fast hardware support for $+$, $\times$ and FMA
  - Less performance for division (19 cycles on i5/i7)
  - The ability to explicitly compute the error for $+$ and $\times$...
    ...but not for sin and log

Why polynomials?

- Why polynomials and no other approximants?
  - Why no (truncated) continued fractions?
  - Why no approximations with other base functions?

- A (almost) purely technological answer
  - Fast hardware support for $+$, $\times$ and FMA
  - Less performance for division (19 cycles on i5/i7)
  - The ability to explicitly compute the error for $+$ and $\times$...
    ...but not for sin and log

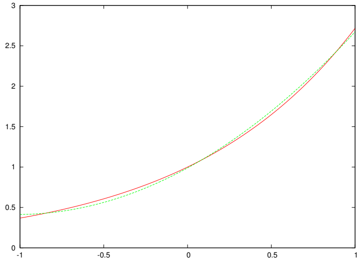
- This answer is not categorical.
  - Some functions are hard to approximate with polynomials: asin
  - We are doing software; things might be different in hardware.
  - There is a rich tool-chain for polynomials and almost nothing for the rest.

Reasons why argument reduction is needed

- Functions f need to be implemented on the whole FP domain
- For `binary64`, $f = \exp$ is defined on $I = [-744.5; 709]$

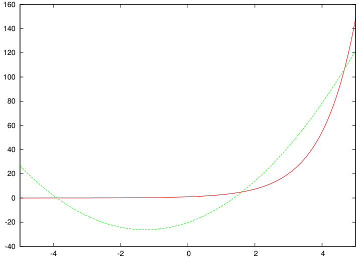
Reasons why argument reduction is needed

- Functions f need to be implemented on the whole FP domain
- For `binary64`, $f = \exp$ is defined on $I = [-744.5; 709]$
- Polynomial approximation alone is not enough:



Reasons why argument reduction is needed

- Functions f need to be implemented on the whole FP domain
- For `binary64`, $f = \exp$ is defined on $I = [-744.5; 709]$
- Polynomial approximation alone is not enough:



- When the domain is large, the error explodes for a given degree
- or: a huge degree is needed to compensate.

Argument reduction

- Argument reduction reduces the range of the function
  - Use of algebraic properties of the function
    - Periodicity: sin
    - Autosimilarity: exp
    - Symmetry: asin
  - Use of the semi-logarithmic character of the FP formats
    - From space, convertToInteger and exp are kind of alike.
  - Fallback: splitting of the domain into subdomains.

Argument reduction

- Argument reduction reduces the range of the function
  - Use of algebraic properties of the function
    - Periodicity: sin
    - Autosimilarity: exp
    - Symmetry: asin
  - Use of the semi-logarithmic character of the FP formats
    - From space, convertToInteger and exp are kind of alike.
  - Fallback: splitting of the domain into subdomains.
- Example for exp:

$$e^x = 2^{\frac{x}{\log 2}} = 2^{\left\lfloor \frac{x}{\log 2} \right\rceil} \cdot 2^{\frac{x}{\log 2} - \left\lfloor \frac{x}{\log 2} \right\rceil} = 2^E \cdot e^{x - E \log 2} = 2^E \cdot e^r$$

Tabulation

- Periodicity and the semi-logarithmic FP are not enough
  - Periodicity: a whole period is to be covered by the polynomial
  - Semi-logarithmic character: a whole binade $1 \leq m < 2$

Tabulation

- Periodicity and the semi-logarithmic FP are not enough
  - Periodicity: a whole period is to be covered by the polynomial
  - Semi-logarithmic character: a whole binade $1 \leq m < 2$
- Idea: use a table
  - precompute the function at discrete points in a table
  - and have the polynomial cover the gaps between these points.

Tabulation

- Periodicity and the semi-logarithmic FP are not enough
  - Periodicity: a whole period is to be covered by the polynomial
  - Semi-logarithmic character: a whole binade $1 \leq m < 2$
- Idea: use a table
  - precompute the function at discrete points in a table
  - and have the polynomial cover the gaps between these points.
- Issue: function needs to be autosimilar so that it stays the same between all discrete points.

Tabulation - Example

Example for exp:

$$\begin{aligned}
e^x &= 2^{\frac{x}{\log 2}} \\
&= 2^{2^{-\lambda} \left\lfloor 2^\lambda \frac{x}{\log 2} \right\rceil} \cdot 2^{\frac{x}{\log 2} - 2^{-\lambda} \left\lfloor 2^\lambda \frac{x}{\log 2} \right\rceil} \\
&= 2^E \cdot 2^{2^{-\lambda}\, i} \cdot e^{x - k\, 2^{-\lambda} \log 2} \\
&= 2^E \cdot 2^{2^{-\lambda}\, i} \cdot e^r
\end{aligned}$$

where

$$k = \left\lfloor 2^\lambda \frac{x}{\log 2} \right\rceil, E = \left\lfloor 2^{-\lambda}\, k \right\rfloor, i = k - 2^\lambda E, r = x - k\, 2^{-\lambda} \log 2.$$

Tabulation - Example

Example for exp:

$$\begin{aligned} e^x &= 2^{\frac{x}{\log 2}} \\ &= 2^{2^{-\lambda} \left\lfloor 2^{\lambda} \frac{x}{\log 2} \right\rfloor} \cdot 2^{\frac{x}{\log 2} - 2^{-\lambda} \left\lfloor 2^{\lambda} \frac{x}{\log 2} \right\rfloor} \\ &= 2^E \cdot 2^{2^{-\lambda} i} \cdot e^{x - k\, 2^{-\lambda} \log 2} \\ &= 2^E \cdot 2^{2^{-\lambda} i} \cdot e^r \end{aligned}$$

where

$$k = \left\lfloor 2^{\lambda} \frac{x}{\log 2} \right\rfloor, E = \left\lfloor 2^{-\lambda} k \right\rfloor, i = k - 2^{\lambda} E, r = x - k\, 2^{-\lambda} \log 2.$$

Here,

- $t : i \mapsto 2^{2^{-\lambda} i}$ is $t : \mathbb{N} \to \mathbb{R}$, hence a table,
- where the index i is bounded by $0 \leq i \leq 2^{\lambda} - 1$ and
- r, the reduced argument, is in a small domain $|r| \leq \frac{1}{2}\, 2^{-\lambda} \frac{1}{\log 2}$.

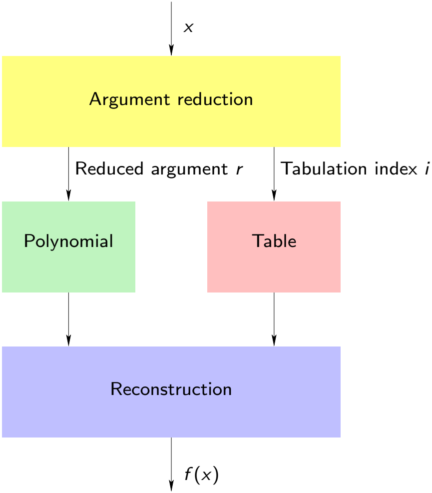
Reconstruction

- Argument reduction "cuts" f into several other functions
  - the function on the reduced domain
  - the functions $i \mapsto t[i]$ computed by the tables
  - the basic operations used in the reduction itself.

Reconstruction

- Argument reduction "cuts" f into several other functions
  - the function on the reduced domain
  - the functions $i \mapsto t[i]$ computed by the tables
  - the basic operations used in the reduction itself.
- The reconstruction phase recovers the original function f.
- Reconstruction code is often easier to write than argument reduction code.
  - Bit-fiddling for argument reduction
  - Pure basic FP operations (adds and muls) for reconstruction.
- However, reconstruction does yield to some error due to roundings.

Algorithm scheme for a function's implementation



A toy exponential

```
// A very crude "toy" implementation of exp(x)
//
// About 42 bits of accuracy. No checks for NaN, Inf whatsoever.
//
double Exp(double x) {
  double z, n, t, r, P, tbl, y;
  uint32_t E, idx, N;
  doubleCaster shiftedN, twoE;

  // Argument reduction
  z = x * TWO_4_RCP_LN_2;              // z = x * 2^4 * 1/ln(2)
  shiftedN.d = z + TWO_52_P_51;        // shiftedN.d = nearestint(z) + 2^52 + 2^51
  n = shiftedN.d - TWO_52_P_51;        // n = nearestint(z) as double
  N = shiftedN.i[LO];                  // N = nearestint(z) as integer
  E = N >> 4;                          // E = floor(2^-4 * N)
  idx = N & 0x0f;                      // idx = N - E * 2^4
  t = n * TWO_M_4_LN_2;                // t = n * 2^-4 * ln(2)
  r = x - t;                           // r = x - t

  // Polynomial approximation         p(r) approximates exp(r)
  P = c0 + r * (c1 + r * (c2 + r * (c3 + r * (c4 + r * c5))));

  // Table access
  tbl = table[idx];                    // tbl = 2^(2^-4 * idx)

  // Reconstruction
  twoE.i[HI] = (E + 1023) << 20;
  twoE.i[LO] = 0;                      // twoE.d = 2^E
  y = twoE.d * (tbl * P);              // y = 2^E * tbl * P

  return y;
}
```

Exceptional cases

- The function $f : \mathbb{R} \to \mathbb{R}$ to implement lives in a FP format

Exceptional cases

- The function $f : \mathbb{R} \to \mathbb{R}$ to implement lives in a FP format
- Floating-point formats have special values:
  - Infinities $\pm\infty$
  - Not-A-Numbers (NaNs), possibly with "payload"
  - Zeros ± 0, with some semantics behind the sign
  - Subnormal numbers

Exceptional cases

- The function $f : \mathbb{R} \rightarrow \mathbb{R}$ to implement lives in a FP format
- Floating-point formats have special values:
  - Infinities $\pm\infty$
  - Not-A-Numbers (NaNs), possibly with "payload"
  - Zeros ± 0, with some semantics behind the sign
  - Subnormal numbers
- An implementation of f must handle these special values
  - in input, branching out before argument reduction
    - NaNs give NaNs with the same payload,
    - Infinities are handled with some limits semantics,
    - Subnormals must be renormalized as appropriate.
  - in output
    - Underflows and overflows where appropriate,
    - Domain errors yield NaNs.

Outline

3. Error analysis
   - Error kinds and propagation
   - Approximation
   - Evaluation
   - Argument reduction, result reconstruction

Error Kinds and Propagation

Definition (Absolute and relative errors)

Let $\tilde{x}$ be a computed value and x an expected value.

- Absolute error: $\delta_x = \tilde{x} - x$.
- Relative error: $\varepsilon_x = (\tilde{x} - x)/x$.

(Gappa's definitions)

Error Kinds and Propagation

Definition (Absolute and relative errors)

Let $\tilde{x}$ be a computed value and x an expected value.

- Absolute error: $\delta_x = \tilde{x} - x$.
- Relative error: $\varepsilon_x = (\tilde{x} - x)/x$.

(Gappa's definitions)

- Relative error rules over floating-point arithmetic.

Error Kinds and Propagation

Definition (Absolute and relative errors)

Let $\tilde{x}$ be a computed value and x an expected value.

- Absolute error: $\delta_x = \tilde{x} - x$.
- Relative error: $\varepsilon_x = (\tilde{x} - x)/x$.

(Gappa's definitions)

- Relative error rules over floating-point arithmetic.
- It needs careful analysis in presence of addition:

$$\frac{(\tilde{x} + \tilde{y}) - (x + y)}{x + y} = \frac{\delta_x + \delta_y}{x + y} \quad \text{when } x + y \to 0.$$

Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.

Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.
- Input error: inaccuracy of sensors.

Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.
- Input error: inaccuracy of sensors.
- Truncation error: due to ignoring terms in formulas.

Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.
- Input error: inaccuracy of sensors.
- Truncation error: due to ignoring terms in formulas.
- Quantization error: representation of mathematical constants as floating-point numbers.

Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.
- Input error: inaccuracy of sensors.
- Truncation error: due to ignoring terms in formulas.
- Quantization error: representation of mathematical constants as floating-point numbers.
- Round-off error: caused by finite-precision computations.

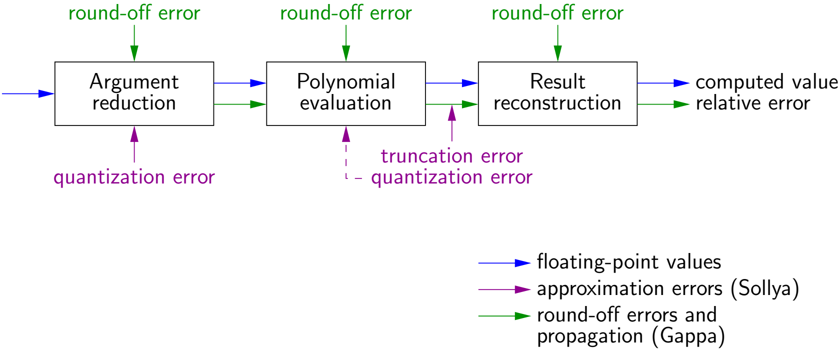
Error Kinds

Definition

- Model error: distance between the physical phenomenon and its mathematical model.
- Input error: inaccuracy of sensors.
- Truncation error: due to ignoring terms in formulas.
- Quantization error: representation of mathematical constants as floating-point numbers.
- Round-off error: caused by finite-precision computations.

Arbitrary classification.
Better look at them from a tool perspective.

Error Kinds


round-off error
round-off error
round-off error
Argument reduction
Polynomial evaluation
Result reconstruction
computed value
relative error
quantization error
truncation error
quantization error
floating-point values
approximation errors (Sollya)
round-off errors and propagation (Gappa)

Polynomial Approximation

Example (Truncation error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.

Expected result: e^r.

Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.

Goal: bound $\hat{P}(r)/e^r - 1$.

Polynomial Approximation

Example (Truncation error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Expected result: e^r.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Goal: bound $\hat{P}(r)/e^r - 1$.

Specificities:

- $r \mapsto \hat{P}(r)/e^r$ is a C^∞ function.
  All the methods from real analysis are available.

Polynomial Approximation

Example (Truncation error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Expected result: e^r.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Goal: bound $\hat{P}(r)/e^r - 1$.

Specificities:

- $r \mapsto \hat{P}(r)/e^r$ is a C^∞ function.
  All the methods from real analysis are available.
- $\hat{P}(r)$ and e^r are close (by design),
  so be wary of tool results.

Polynomial Approximation

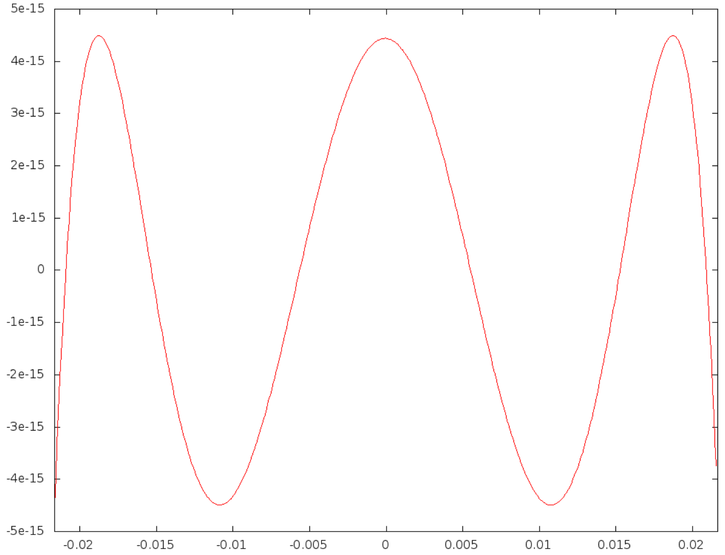
Example (Truncation error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Expected result: e^r.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Goal: bound $\hat{P}(r)/e^r - 1$.

Specificities:

- $r \mapsto \hat{P}(r)/e^r$ is a C^∞ function.
  All the methods from real analysis are available.
- $\hat{P}(r)$ and e^r are close (by design),
  so be wary of tool results.
- Dedicated tool: Sollya.

Relative Truncation Error



Polynomial Evaluation

Example (Round-off error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Computed value: $P(r) = \circ(c_0 + \circ(r \times \circ(c_1 + \ldots)))$.
Goal: bound $P(r)/\hat{P}(r) - 1$.

Polynomial Evaluation

Example (Round-off error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Computed value: $P(r) = \circ(c_0 + \circ(r \times \circ(c_1 + \ldots)))$.
Goal: bound $P(r)/\hat{P}(r) - 1$.

Specificities:

- $r \mapsto P(r)/\hat{P}(r)$ is not even continuous.

Polynomial Evaluation

Example (Round-off error for exponential)

Input domain: $|r| \leq 195103586506733 \cdot 2^{-53} \simeq 2^{-5.5}$.
Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.
Computed value: $P(r) = \circ(c_0 + \circ(r \times \circ(c_1 + \ldots)))$.
Goal: bound $P(r)/\hat{P}(r) - 1$.

Specificities:

- $r \mapsto P(r)/\hat{P}(r)$ is not even continuous.
- Dedicated tool: Gappa.

Polynomial Evaluation

Example (~~Round-off~~ Global error for exponential)

Input domain: $|r| \leq 1951035865067 33 \cdot 2^{-53} \simeq 2^{-5.5}$.

Algorithm with infinitely-precise computations: $\hat{P}(r) = \sum_{i \leq 5} c_i r^i$.

Computed value: $P(r) = \circ(c_0 + \circ(r \times \circ(c_1 + \ldots)))$.

Goal: bound $P(r)/\hat{P}(r) - 1$.

Input error: $|r - \hat{r}| \leq 18464675644882170 3 \cdot 2^{-100} \simeq 2^{-42.6}$.

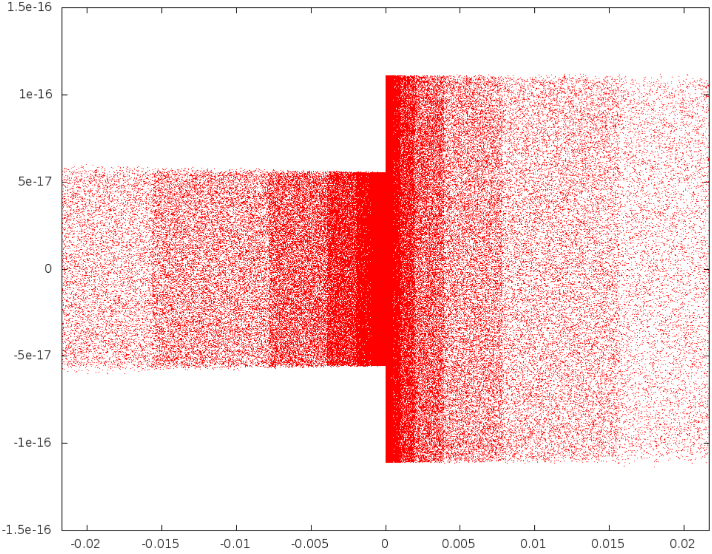
Truncation error: $|\hat{P}(r)/e^r - 1| \leq 30567 \ldots 0435 \cdot 2^{-129} \simeq 2^{-47.6}$.

Goal: bound $P(r)/e^{\hat{r}} - 1$.

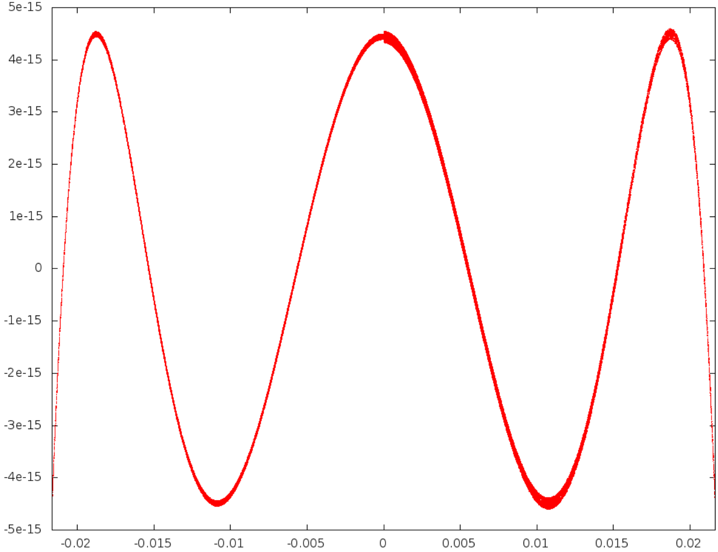
Specificities:

- $r \mapsto P(r)/\hat{P}(r)$ is not even continuous.
- Dedicated tool: Gappa.
- Note: the truncation error comes from Sollya.

Relative Round-Off Error



Relative Round-Off and Truncation Error



Argument Reduction

Where does the bound $|r - \hat{r}| \leq 2^{-42.6}$ come from?

Argument Reduction

Where does the bound $|r - \hat{r}| \leq 2^{-42.6}$ come from?

Example (Input error for polynomial evaluation of exponential)

Input domain: $|x| \leq 800$.
Reduction factor: $n = \lfloor \circ(x \cdot \texttt{TWO\_4\_RCP\_LN\_2}) \rceil$.
Computed value: $r = \circ(x - \circ(n \times \texttt{TWO\_M\_4\_LN\_2}))$.
Expected value: $\hat{r} = x - n \times \ln(2)/16$.
Goal: bound $r - \hat{r}$.

Argument Reduction

Where does the bound $|r - \hat{r}| \leq 2^{-42.6}$ come from?

Example (Input error for polynomial evaluation of exponential)

Input domain: $|x| \leq 800$.
Reduction factor: $n = \lfloor \circ(x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \rceil$.
Computed value: $r = \circ(x - \circ(n \times \mathtt{TWO\_M\_4\_LN\_2}))$.
Expected value: $\hat{r} = x - n \times \ln(2)/16$.
Goal: bound $r - \hat{r}$.

Specificities:

- $r - \hat{r}$ is not continuous with respect to x;
  the last round-off error vanishes.

Argument Reduction

Where does the bound $|r - \hat{r}| \leq 2^{-42.6}$ come from?

Example (Input error for polynomial evaluation of exponential)

Input domain: $|x| \leq 800$.
Reduction factor: $n = \lfloor \circ(x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \rceil$.
Computed value: $r = \circ(x - \circ(n \times \mathtt{TWO\_M\_4\_LN\_2}))$.
Expected value: $\hat{r} = x - n \times \ln(2)/16$.
Goal: bound $r - \hat{r}$.

Specificities:

- $r - \hat{r}$ is not continuous with respect to x; the last round-off error vanishes.
- Dedicated tool: Gappa.

Argument Reduction

Where does the bound $|r - \hat{r}| \leq 2^{-42.6}$ come from?

Example (Input error for polynomial evaluation of exponential)

Input domain: $|x| \leq 800$.
Reduction factor: $n = \lfloor \circ(x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \rceil$.
Computed value: $r = \circ(x - \circ(n \times \mathtt{TWO\_M\_4\_LN\_2}))$.
Expected value: $\hat{r} = x - n \times \ln(2)/16$.
Goal: bound $r - \hat{r}$.

Specificities:

- $r - \hat{r}$ is not continuous with respect to x;
  the last round-off error vanishes.
- Dedicated tool: Gappa.
- Note: the quantization error comes from Sollya.

Result Reconstruction

Example (Final error for exponential)

Computed value: $y = \circ(2^E \times \circ(tbl \times P(r)))$.
Expected value: $\hat{y} = e^x = 2^E \times (2^{idx/16} \times e^{\hat{r}})$.
Goal: bound $y/\hat{y} - 1$.

- Dedicated tool: Gappa.

Result Reconstruction

Example (Final error for exponential)

Computed value: $y = \circ(2^E \times \circ(tbl \times P(r)))$.
Expected value: $\hat{y} = e^x = 2^E \times (2^{idx/16} \times e^{\hat{r}})$.
Goal: bound $y/\hat{y} - 1$.

- Dedicated tool: Gappa.
- Note: the quantization error on $2^{idx/16}$ comes from Sollya.

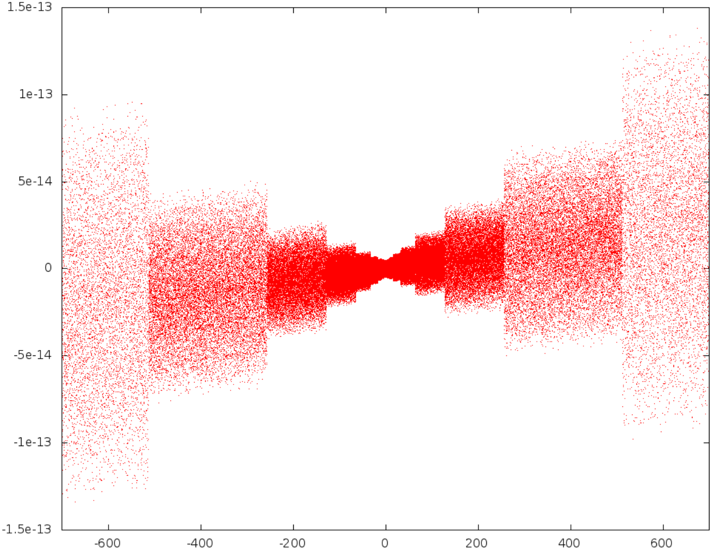
Result Reconstruction

Example (Final error for exponential)

Computed value: $y = \circ(2^E \times \circ(tbl \times P(r)))$.
Expected value: $\hat{y} = e^x = 2^E \times (2^{idx/16} \times e^{\hat{r}})$.
Goal: bound $y/\hat{y} - 1$.

- Dedicated tool: Gappa.
- Note: the quantization error on $2^{idx/16}$ comes from Sollya.
- Disclaimer: we have ignored the case where multiplying by 2^E causes an underflow.

Relative Global Error



Outline

4 Sollya and polynomial approximations
- Polynomial approximation theory
- First ideas for polynomial approximation
- Interpolation polynomials
- Choosing the right interpolation points
- Remez polynomials
- Polynomial approximation with Sollya

Specifying polynomial approximation

- Let $f : \mathbb{R} \to \mathbb{R}$ be a function...
- ...on a *small* domain $I = [a; b] \subset \mathbb{R}$.
- Also fix a target error $\overline{\varepsilon}$ to be satisfied.

Specifying polynomial approximation

- Let $f : \mathbb{R} \to \mathbb{R}$ be a function...
- ...on a *small* domain $I = [a; b] \subset \mathbb{R}$.
- Also fix a target error $\overline{\varepsilon}$ to be satisfied.
- We need to compute a polynomial p of minimal degree n such that the error $\frac{p}{f} - 1$ stays bounded by $\overline{\varepsilon}$ (in magnitude).

Specifying polynomial approximation

- Let $f : \mathbb{R} \to \mathbb{R}$ be a function. . .
- . . . on a *small* domain $I = [a; b] \subset \mathbb{R}$.
- Also fix a target error $\overline{\varepsilon}$ to be satisfied.
- We need to compute a polynomial p of minimal degree n such that the error $\frac{p}{f} - 1$ stays bounded by $\overline{\varepsilon}$ (in magnitude).

Example:

- Function: $f = \exp$
- Domain: $I = \left[-\frac{1}{2}; \frac{1}{2}\right]$
- Target error: $\overline{\varepsilon} = 2^{-5}$ (5 correct bits)

Specifying polynomial approximation

- Let $f : \mathbb{R} \to \mathbb{R}$ be a function...
- ...on a *small* domain $I = [a; b] \subset \mathbb{R}$.
- Also fix a target error $\overline{\varepsilon}$ to be satisfied.
- We need to compute a polynomial p of minimal degree n such that the error $\frac{p}{f} - 1$ stays bounded by $\overline{\varepsilon}$ (in magnitude).

Example:

- Function: $f = \exp$
- Domain: $I = \left[-\frac{1}{2}; \frac{1}{2}\right]$
- Target error: $\overline{\varepsilon} = 2^{-5}$ (5 correct bits)
- $p(x) = 1 + x + \frac{1}{2}\, x^2 \quad n = 2$
- $\|\frac{p}{f} - 1\|_\infty^I \approx 3.05 \cdot 10^{-2} < 2^{-5}$

Specifying polynomial approximation - cont'd

- Put differently, we need to
  - choose a degree n that seems to be sufficient, to
  - compute the coefficients c_i of a polynomial

$$p(x) = \sum_{i=0}^{n} c_i \, x^i$$

  - to consider the error $\frac{p}{f} - 1$ yielded by p
  - and to start over, increasing n, if it is too large.

Specifying polynomial approximation - cont'd

- Put differently, we need to
  - choose a degree n that seems to be sufficient, to
  - compute the coefficients c_i of a polynomial

$$p(x) = \sum_{i=0}^{n} c_i \, x^i$$

  - to consider the error $\frac{p}{f} - 1$ yielded by p
  - and to start over, increasing n, if it is too large.

But how to compute the coefficients c_i of the polynomial p?

Taylor polynomials

- First idea:
  - Let us take $x_0 \in I$
  - $f(x_0)$ approximates $f(x)$ over I
  - $f(x_0) + (x - x_0) \cdot f'(x_0)$ is better
  - $f(x_0) + (x - x_0) \cdot f'(x_0) + \frac{1}{2} \cdot (x - x_0)^2 \cdot f''(x_0)$ works even better

Taylor polynomials

- First idea:
  - Let us take $x_0 \in I$
  - $f(x_0)$ approximates $f(x)$ over I
  - $f(x_0) + (x - x_0) \cdot f'(x_0)$ is better
  - $f(x_0) + (x - x_0) \cdot f'(x_0) + \frac{1}{2} \cdot (x - x_0)^2 \cdot f''(x_0)$ works even better
- This is the idea behind a Taylor expansion at x_0:

$$f(x) = \underbrace{\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} \cdot (x - x_0)^i}_{=:p(x)} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!} \cdot (x - x_0)^{n+1}}_{\text{Lagrange rest resp. error}}$$

Taylor polynomials

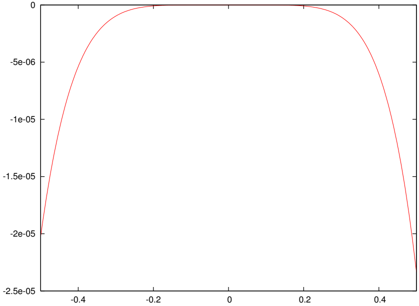
- First idea:
  - Let us take $x_0 \in I$
  - $f(x_0)$ approximates $f(x)$ over I
  - $f(x_0) + (x - x_0) \cdot f'(x_0)$ is better
  - $f(x_0) + (x - x_0) \cdot f'(x_0) + \frac{1}{2} \cdot (x - x_0)^2 \cdot f''(x_0)$ works even better
- This is the idea behind a Taylor expansion at x_0:

$$f(x) = \underbrace{\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} \cdot (x - x_0)^i}_{=:p(x)} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!} \cdot (x - x_0)^{n+1}}_{\text{Lagrange rest resp. error}}$$

- The differential equations defining our functions are simple:
  - Taylor expansions are known for the functions in a `libm`

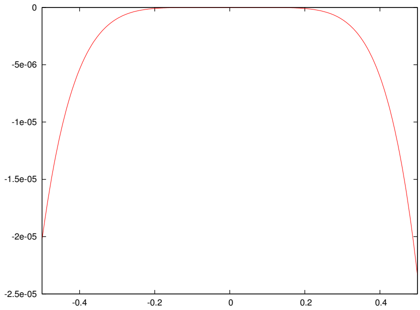
Issues with Taylor polynomials

Error of a Taylor polynomial p of degree 5 with respect to $f = \exp$:



Issues with Taylor polynomials

Error of a Taylor polynomial p of degree 5 with respect to $f = \exp$:



- The Taylor polynomial approximates the function best at the expansion point x_0: the error vanishes at x_0
- The error *explodes* at the extremities of I
- $\Rightarrow$ p should approximate f well over the whole domain I, or, at least, the error should vanish several times on I

Interpolation polynomials

- A polynomial of degree n
  - has $n+1$ coefficients
  - i.e. $n+1$ degrees of freedom
  - It is defined by $n+1$ points $(x_j, p(x_j))$

Interpolation polynomials

- A polynomial of degree n
  - has $n+1$ coefficients
  - i.e. $n+1$ degrees of freedom
  - It is defined by $n+1$ points $(x_j, p(x_j))$

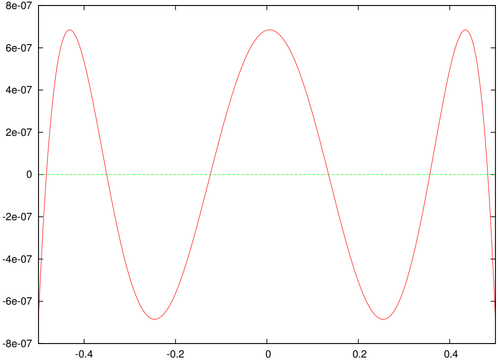
- For the error $p(x) - f(x)$ to vanish at x_j,
  it suffices to have $p(x_j) = f(x_j)$.
- The polynomial p *interpolates* the function f at x_j

Interpolation polynomials

- A polynomial of degree n
  - has $n+1$ coefficients
  - i.e. $n+1$ degrees of freedom
  - It is defined by $n+1$ points $(x_j, p(x_j))$
- For the error $p(x) - f(x)$ to vanish at x_j,
  it suffices to have $p(x_j) = f(x_j)$.
- The polynomial p *interpolates* the function f at x_j
- A interpolation polynomial for f is defined by $n+1$ values x_j
  - There is one value per degree of freedom.
  - The ordinates $f(x_j)$ are trivial for f and x_j.

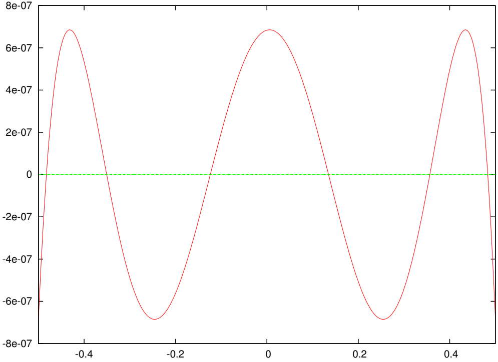
Example of an interpolation polynomial

Error of p, interpolation polynomial of $f = \exp$:



Example of an interpolation polynomial

Error of p, interpolation polynomial of $f = \exp$:



- The error vanishes at the interpolation points x_j
- By choosing the interpolation points, we can constrain the error to oscillate

Computing an interpolation polynomial

How can an interpolation polynomial be computed?

- We have $p(x_j) = \sum_{i=0}^{n} c_i \, x_j^i = f(x_j)$ for $n+1$ points x_j

Computing an interpolation polynomial

How can an interpolation polynomial be computed?

- We have $p(x_j) = \sum_{i=0}^{n} c_i \, x_j^i = f(x_j)$ for $n+1$ points x_j
- Put differently, we have

$$\underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & \cdots & x_1^n \\ 1 & x_2 & \ddots & & & \vdots \\ \vdots & \vdots & & x_j^i & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ 1 & x_{n+1} & \cdots & \cdots & \cdots & x_{n+1}^n \end{pmatrix}}_{=:A} \cdot \underbrace{\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{pmatrix}}_{=:\vec{p}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{n+1}) \end{pmatrix}}_{=:\vec{f}}$$

Computing an interpolation polynomial

How can an interpolation polynomial be computed?

- We have $p(x_j) = \sum_{i=0}^{n} c_i \, x_j^i = f(x_j)$ for $n+1$ points x_j
- Put differently, we have

$$\underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & \cdots & x_1^n \\ 1 & x_2 & \ddots & & & \vdots \\ \vdots & \vdots & & x_j^i & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ 1 & x_{n+1} & \cdots & \cdots & \cdots & x_{n+1}^n \end{pmatrix}}_{=:A} \cdot \underbrace{\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{pmatrix}}_{=:\vec{p}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{n+1}) \end{pmatrix}}_{=:\vec{f}}$$

- We know that Vandermonde matrix A and the r.h.s. $\vec{f}$.
  - $\Rightarrow$ solving that systems yields p
  - Side note: special resolution algorithms for Vandermonde

A chicken and egg problem

- We know that right-hand side $\vec{f}$.

A chicken and egg problem

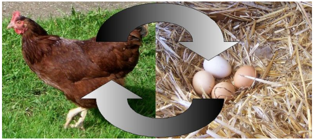
- We know that right-hand side $\vec{f}$.
- So we can compute $f(x_j)$ for all x_j.

A chicken and egg problem

- We know that right-hand side $\vec{f}$.
- So we can compute $f(x_j)$ for all x_j.
- WTF? We are implementing a procedure to compute f!

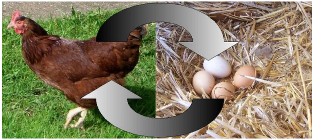
A chicken and egg problem

- We know that right-hand side $\vec{f}$.
- So we can compute $f(x_j)$ for all x_j.
- WTF? We are implementing a procedure to compute f!



- There is a chicken and egg problem:
  - We first need code to evaluate the $f(x_j)$
  - In a second step, we can compute interpolation polynomials to put them into code that computes f

A chicken and egg problem



- We know that right-hand side $\vec{f}$.
- So we can compute $f(x_j)$ for all x_j.
- WTF? We are implementing a procedure to compute f!

- There is a chicken and egg problem:
  - We first need code to evaluate the $f(x_j)$
  - In a second step, we can compute interpolation polynomials to put them into code that computes f

- Practically:
  - We write multi-precision procedures for the functions
  - Those are based on *inefficient* Taylor polynomials in any case.
  - Then, we use these code in order develop a `libm`

Which interpolation points to choose?

- The choice made for x_j has an influence on the error:
  Let p be a polynomial interpolating f at the x_j. Hence we have

$$f(x) = p(x) + \frac{1}{(n+1)!}\, f^{(n+1)}(\xi) \prod_{j=0}^{n} (x - x_j)$$

Which interpolation points to choose?

- The choice made for x_j has an influence on the error:
  Let p be a polynomial interpolating f at the x_j. Hence we have

$$f(x) = p(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{j=0}^{n} (x - x_j)$$

- Equidistant points are not optimal
- Chebyshev tried to minimize the error

Which interpolation points to choose?

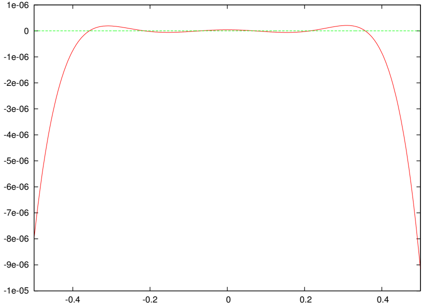
- The choice made for x_j has an influence on the error:
  Let p be a polynomial interpolating f at the x_j. Hence we have

$$f(x) = p(x) + \frac{1}{(n+1)!}\, f^{(n+1)}(\xi) \prod_{j=0}^{n} (x - x_j)$$

- Equidistant points are not optimal
- Chebyshev tried to minimize the error
- Remez has finally given an iterative algorithm
  - to compute the polynomial that is minimal w.r.t. the maximal error
  - The algorithm "just" chooses the right interpolation points

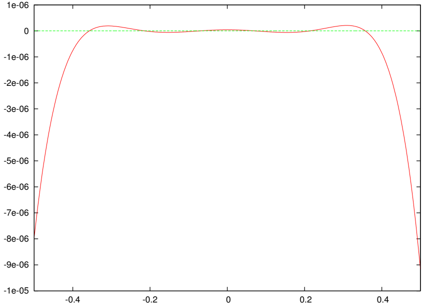
Equidistant points

Error of p interpolating $f = \exp$ at equidistant points:



Equidistant points

Error of p interpolating $f = \exp$ at equidistant points:



- This is better than Taylor: the error is 2.5 times less
- The error starts to oscillate
- The error still explodes at the extremities
  $\Rightarrow$ we should try to put more points near the extremities

Chebyshev interpolation nodes

- The error is to be minimized for the maximum norm $\| \quad \|_\infty$:

$$f(x) - p(x) = \frac{1}{(n+1)!}\, f^{(n+1)}(\xi) \prod_{j=0}^{n} (x - x_j)$$

- P. L. Chebyshev: try to minimize

$$\left\| \prod_{j=0}^{n} (x - x_j) \right\|_\infty$$

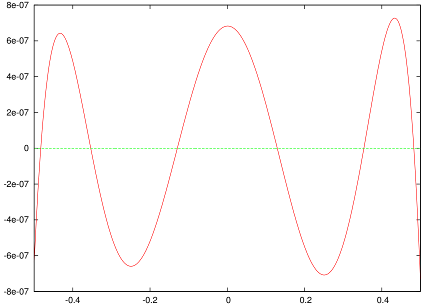
- Particular points for which this is the case over $I = [a; b]$:

$$x_j = a + \frac{b-a}{2} \cdot \left(\cos\left(\frac{2j-1}{2\,(n+1)} \right) + 1 \right)$$

- These are the Chebyshev interpolation nodes.

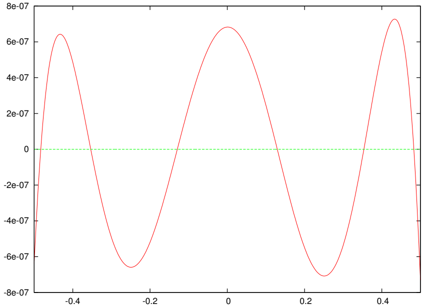
Chebyshev interpolation nodes - example

Error of p interpolating $f = \exp$ at the Chebyshev nodes:



Chebyshev interpolation nodes - example

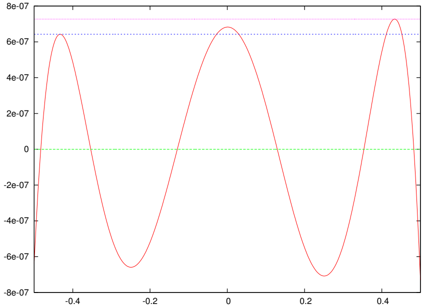
Error of p interpolating $f = \exp$ at the Chebyshev nodes:



- This is $10\times$ better than for equidistant points!

Chebyshev interpolation nodes - example

Error of p interpolating $f = \exp$ at the Chebyshev nodes:



- This is $10\times$ better than for equidistant points!
- It is not yet optimal: see plot
- There are functions where this sub-optimality is huge.

Remez polynomials

- Theorem (Chebyshev/ La Vallée-Poussin):
  The approximation polynomial is optimal iff all the extrema are the same height.

Remez polynomials

- Theorem (Chebyshev/ La Vallée-Poussin):
  The approximation polynomial is optimal iff all the extrema are the same height.

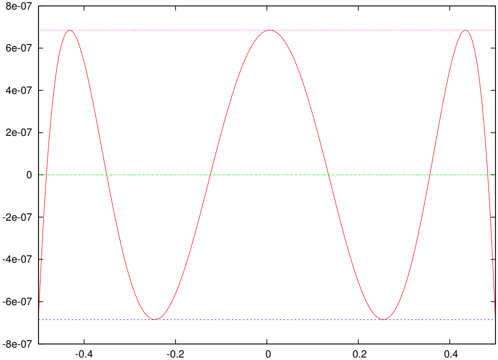
- E. Ya. Remez:
  - Directly interpolate $f(x) + (-1)^j \cdot \varepsilon$
    where ε is the height of the sought extrema.
  - While the extrema are not at the same height, exchange the interpolation points with the points where the real extrema are located.

Remez polynomials

- Theorem (Chebyshev/ La Vallée-Poussin):
  The approximation polynomial is optimal iff all the extrema are the same height.
- E. Ya. Remez:
  - Directly interpolate $f(x) + (-1)^j \cdot \varepsilon$
    where ε is the height of the sought extrema.
  - While the extrema are not at the same height, exchange the interpolation points with the points where the real extrema are located.
- This is the Remez algorithm
- It converges towards the polynomial minimal undermaximum norm
- That is why it is also called minimax algorithm

Example: Remez polynomial

Erreur of Remez polynomial p of degree 5 w.r.t $f = \exp$:



- The error is just a little smaller than for Chebyshev
- All the extrema are now at the same height

Yet another issue → fpminimax

- Taylor, Chebyshev, Remez: polynomials with real coefficients

$$p(x) = \sum_{i=0}^{n} c_i \, x^i \quad c_i \in \mathbb{R}$$

- On computers, there are only FP numbers to store the c_i.

Yet another issue → fpminimax

- Taylor, Chebyshev, Remez: polynomials with real coefficients

$$p(x) = \sum_{i=0}^{n} c_i \, x^i \quad c_i \in \mathbb{R}$$

- On computers, there are only FP numbers to store the c_i.
- When rounding coefficient by coefficient, $\tilde{c}_i = \circ_{c_i}$, we destroy all the optimization work done by the Remez algorithm.
  - The oscillations disappear.
  - The error explodes.

Yet another issue → fpminimax

- Taylor, Chebyshev, Remez: polynomials with real coefficients
$$p(x) = \sum_{i=0}^{n} c_i \, x^i \quad c_i \in \mathbb{R}$$
- On computers, there are only FP numbers to store the c_i.
- When rounding coefficient by coefficient, $\tilde{c}_i = \circ_{c_i}$, we destroy all the optimization work done by the Remez algorithm.
  - The oscillations disappear.
  - The error explodes.
- Need to compute approximation polynomials with FP coefficients
$$p(x) = \sum_{i=0}^{n} c_i \, x^i \quad c_i \in \mathbb{F}_k$$
- This is a hard discrete optimization problem
- Since 2006, there has been heuristic algorithms based on lattice reduction.

Sollya: Functions and domains

```
> f = exp(x);              /* Define the function */
> dom = [-1/4;1/4];        /* Define the domain */
> f;                       /* Display f */
exp(x)
> diff(f);                 /* Differentiate f */
exp(x)
> f(x + 2)                 /* Compose f with x + 2 */
exp(2 + x)
> dom;                     /* Display the domain */
[-0.25;0.25]
> inf(dom);                /* Lower bound of the domain */
-0.25
> sup(dom);                /* Upper bound of the domain */
0.25
> plot(f, dom);            /* Plot the function */
>
```

Sollya: The remez command

```
> f = exp(x);                 /* Define the function */
> dom = [-1/4;1/4];           /* Define the domain */
> n = 5;                      /* Set degree n to 5 */
> p = remez(f, n , dom); /* Remez polynomial */
> p;
1.00000001061667289247812...
>
```

Sollya: Looking at the error

```
> f = exp(x);                /* Define the function */
> dom = [-1/4;1/4];          /* Define the domain */
> n = 5;                     /* Set degree n to 5 */
> p = remez(f, n , dom);     /* Remez polynomial */
> err = p/f - 1;             /* Define the rel. error */
> plot(err, dom);            /* Plot the error */
```

Sollya: Optimizing for absolute or relative error

```
> f = log(1 + x);            /* Define the function */
> dom = [-1/4;1/4];          /* Define the domain */
> n = 5;                     /* Set degree n to 5 */
> p = remez(f, n , dom);     /* Remez polynomial */
> err = p/f - 1;             /* Define the rel. error */
> plot(err, dom);            /* Plot the error */

> /* Recompute Remez polynomial for rel. error */
> p = remez(1, [| 1,...,n |], dom, 1/f);

> err = p/f - 1;             /* Define the rel. error */
> plot(err, dom);            /* Plot the error */
```

Sollya: Computing and bounding the maximum error

```
> f = exp(x);                  /* Define the function */
> dom = [-1/4;1/4];            /* Define the domain */
> n = 5;                       /* Set degree n to 5 */
> p = remez(1, n , dom, 1/f); /* Remez polynomial */
> err = p/f - 1;               /* Define the rel. error */

> /* Compute supremum norm to get max. error */
> errmax = supnorm(p, f, dom, relative, 2^(-10));

> errmax;
[1.0576...e-8; 1.0586...e-8]

> superrmax = sup(errmax);
> log2(superrmax);
-2.649...e1
```

Sollya: The fpminimax command

```
> f = exp(x);                /* Define the function */
> dom = [-1/4;1/4];          /* Define the domain */
> n = 5;                     /* Set degree n to 5 */

> /* fpminimax with double prec. coeffs. */
> p = fpminimax(f, n, [|D...|], dom);

> err = p/f - 1;             /* Define the rel. error */
> errmax = supnorm(p, f, dom, relative, 2^(-10));
> supermax = sup(errmax);
> log2(superrmax);
-2.649...e1

> display = hexadecimal;
Display mode is hexadecimal numbers.
> for i from 0 to degree(p) do coeff(p,i);
0x1.0000002c62a0cp0
0xf.ffff4495ce78p-4
```

Sollya: Computing the optimal degree

```
> f = exp(x);                /* Define the function */
> dom = [-1/4;1/4];          /* Define the domain */
> targeterr = 2^-20;         /* Define target error */
> nmax = 18;                 /* Set max. degree */

> n = 1;
> okay = false;
> while (!okay && (n <= nmax)) do {
      p = fpminimax(f, n, [|D...|], dom);
      errmax = sup(supnorm(p, f, dom, relative,
           2^(-10)));
      if (errmax <= targeterr) then okay = true;
      n = n + 1;
  };

> okay;
true;
> degree(p);
4
```

Advertisement: Sollya vs. Maple

- Both Maple and Sollya...
  - implement the Remez algorithm
  - have a plot command
  - support some means to estimate the supremum norm $\|p/f - 1\|_\infty$

Advertisement: Sollya vs. Maple

- Both Maple and Sollya...
  - implement the Remez algorithm
  - have a plot command
  - support some means to estimate the supremum norm $\|p/f - 1\|_\infty$
- But Maple...
  - has trouble, in Remez, with relative errors and functions that vanish,
  - implements the plot command with HW FP arithmetic. As the error is often less than 2^{-53}, the plot shows nonsense.
  - Finally, Maple always *underestimates* the supremum norm.

Advertisement: Sollya vs. Maple

- Both Maple and Sollya...
  - implement the Remez algorithm
  - have a plot command
  - support some means to estimate the supremum norm $\|p/f - 1\|_\infty$
- But Maple...
  - has trouble, in Remez, with relative errors and functions that vanish,
  - implements the plot command with HW FP arithmetic. As the error is often less than 2^{-53}, the plot shows nonsense.
  - Finally, Maple always *underestimates* the supremum norm.
- $\rightsquigarrow$ Use Sollya!

Outline

5 Gappa and error bounds
- Gappa's overview
- Gappa's inner workings
- Debugging proofs
- Proof hints

Gappa

Objective: help users verify/analyze their numerical applications.

Gappa

Objective: help users verify/analyze their numerical applications.

Design decisions:

- The tool verifies enclosures of mathematical expressions.
- These expressions can contain rounding operators to express limitations and properties of datatypes.
- Formal proofs are generated to provide confidence in the results.

Gappa

Objective: help users verify/analyze their numerical applications.

Design decisions:

- The tool verifies enclosures of mathematical expressions.
- These expressions can contain rounding operators to express limitations and properties of datatypes.
- Formal proofs are generated to provide confidence in the results.

How does it work?

- Interval arithmetic for propagating enclosures.
- Theorems about rounded values and round-off errors.
- Rewriting rules for tightening computed enclosures.

Bounding Expressions by Numeric Intervals

Basic element: an enclosure $e \in I$.

- e is an expression on real numbers:
  $$e ::= number \mid -e \mid \circ(e) \mid e + e \mid e \times e \mid \sqrt{e} \mid \ldots$$
- $I = [a, b]$ is an interval with dyadic rational bounds: $m \times 2^n$.

Bounding Expressions by Numeric Intervals

Basic element: an enclosure $e \in I$.

- e is an expression on real numbers:
  $$e ::= \mathit{number} \mid -e \mid \circ(e) \mid e + e \mid e \times e \mid \sqrt{e} \mid \ldots$$
- $I = [a, b]$ is an interval with dyadic rational bounds: $m \times 2^n$.

These enclosures are appropriate to express questions that usually arise when verifying numerical applications:

- no overflow, no invalid operations, etc
  - variable domain: $\tilde{x} \in I$,
- accuracy of computed values
  - absolute error: $\tilde{x} - x \in I$,
  - relative error: $(\tilde{x} - x)/x \in I$.

Rounding Operators

Example (Floating-point arithmetic)

`float<53,-1074,ne>(x)` is a number

- writable with 53 bits,
- multiple of 2^{-1074},
- closest to x when rounding to nearest, with tie break to even mantissas.

In other words, it is the default `binary64` rounding of x.
It can also be written `float<ieee_64,ne>(x)`.

Rounding Operators

Example (Floating-point arithmetic)

`float<53,-1074,ne>(x)` is a number

- writable with 53 bits,
- multiple of 2^{-1074},
- closest to x when rounding to nearest, with tie break to even mantissas.

In other words, it is the default `binary64` rounding of x.
It can also be written `float<ieee_64,ne>(x)`.

Example (Fixed-point arithmetic)

`fixed<-16,dn>(x)` is a number

- multiple of 2^{-16},
- closest to x when rounding toward $-\infty$.

In other words, it is $2^{-16} \cdot \lfloor x \cdot 2^{16} \rfloor$.

Syntax

```
# 1. Macros for expressions and rounding operators
one = 1;
one_third = float<ieee_32,ne>(1 / 3);
@D = float<ieee_64,ne>;
y = D(1 + D(x * one_third));
y_too D= one + x * one_third;

# 2. Logical formula to prove
{
  x in [1,2]
->
  one + 1 in [2,2] /\
  y - (1 + x * (1 / 3)) in ?     # question mode
}

# 3. Hints for Gappa
```

Syntax

```
# 1. Macros for expressions and rounding operators
one = 1;
one_third = float<ieee_32,ne>(1 / 3);
@D = float<ieee_64,ne>;
y = D(1 + D(x * one_third));
y_too D= one + x * one_third;

# 2. Logical formula to prove
{
  x in [1,2]
->
  one + 1 in [2,2] /\
  y - (1 + x * (1 / 3)) in ?     # question mode
}

# 3. Hints for Gappa
```

```
$ gappa test.g
Warning: y is being renamed to y_too at line 6 column 29

Results for x in [1, 2]:
one + one in [1b1 {2, 2^(1)}, 1b1 {2, 2^(1)}]
y_too - (one + x * (one / 3)) in [384307162470066815b-85 {9.93411e-09,
     2^(-26.585)}, 11453246219b-59 {1.98682e-08, 2^(-25.585)}]
```

Gappa Script for Exponential

```
@rnd = float<ieee_64,ne>;

TWO_4_RCP_LN_2 = rnd(2.308312065422341419207441504113376140594482421875e1);
TWO_M_4_LN_2 = rnd(4.332169878499658038917274893719877582043409347534 18e-2);

c0 = rnd(1.0000000000000004440892098500626161694526672363 28125);
c1 = rnd(0.99999999999999813482531862973701208829879760742 1875);
c2 = rnd(0.49999999982837961542969651418388821184635162353 5156);
c3 = rnd(0.16666666680658745458565306307718856260180473327 6367);
c4 = rnd(4.16676435273481901577774522138497559353709220886 23e-2);
c5 = rnd(8.33319249495430465490830584940340486355125904083 25e-3);

x = rnd(x_);   # the input
n = int<ne>(rnd(x * TWO_4_RCP_LN_2));
r rnd= x - n * TWO_M_4_LN_2;   # the reduced argument
Mr = x - n * Mln2div16;

p rnd= c0 + r * (c1 + r * (c2 + r * (c3 + r * (c4 + r * c5))));
Mp = c0 + Mr * (c1 + Mr * (c2 + Mr * (c3 + Mr * (c4 + Mr * c5))));

y rnd= tbl * p;   # the value computed by the implementation
My = Mtbl * Mp;

{
  # all the hypotheses, including bounds by Sollya
  |x| <= 800 /\
  |TWO_M_4_LN_2 -/ Mln2div16| <= 1b-53 /\
  |tbl -/ Mtbl| <= 3449743230720423263703614822092606179232957043428 5b-218 /\
  Mtbl in [1,2] /\
  |Mp -/ f| <= 305678033371193414370043 5b-129
->
  # what is the relative global error?
  y -/ Mtbl * f in ?
}
```

Dependent Expressions and Interval Evaluation

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that:

- domains of u, v, $\tilde{u}$, and $\tilde{v}$ are $[1, 100]$;
- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \leq 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \leq 0.2$.

Dependent Expressions and Interval Evaluation

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that:

- domains of u, v, $\tilde{u}$, and $\tilde{v}$ are $[1, 100]$;
- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \leq 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \leq 0.2$.

Interval evaluation:

$$
\begin{aligned}
\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} &\in \frac{[1,100]\times[1,100]-[1,100]\times[1,100]}{[1,100]\times[1,100]} \\
&\in \frac{[1,10000]-[1,10000]}{[1,10000]} \\
&\in [-9999, 9999]
\end{aligned}
$$

Dependent Expressions and Interval Evaluation

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that:

- domains of u, v, $\tilde{u}$, and $\tilde{v}$ are $[1, 100]$;
- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \leq 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \leq 0.2$.

Interval evaluation:

$$\begin{aligned}
\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} &\in \frac{[1,100]\times[1,100]-[1,100]\times[1,100]}{[1,100]\times[1,100]} \\
&\in \frac{[1,10000]-[1,10000]}{[1,10000]} \\
&\in [-9999, 9999] \qquad \text{Bad!}
\end{aligned}$$

Naive interval arithmetic does not track dependencies between values.

Rewriting Expressions to Exhibit Dependencies

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that

- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \le 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \le 0.2$.

Solution: make dependencies explicit.

$$\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} = \frac{\tilde{u}-u}{u} + \frac{\tilde{v}-v}{v} + \frac{\tilde{u}-u}{u} \times \frac{\tilde{v}-v}{v}$$

Rewriting Expressions to Exhibit Dependencies

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that

- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \leq 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \leq 0.2$.

Solution: make dependencies explicit.

$$\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} = \frac{\tilde{u}-u}{u} + \frac{\tilde{v}-v}{v} + \frac{\tilde{u}-u}{u} \times \frac{\tilde{v}-v}{v}$$

Interval evaluation:

$$\begin{aligned} \frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} \quad &\in \quad [-0.1, 0.1] + [-0.2, 0.2] + [-0.1, 0.1] \times [-0.2, 0.2] \\ &\in \quad [-0.32, 0.32] \end{aligned}$$

Rewriting Expressions to Exhibit Dependencies

Example

Compute the range of $\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v}$ knowing that

- values are related: $\left|\frac{\tilde{u}-u}{u}\right| \le 0.1$ and $\left|\frac{\tilde{v}-v}{v}\right| \le 0.2$.

Solution: make dependencies explicit.

$$\frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} = \frac{\tilde{u}-u}{u} + \frac{\tilde{v}-v}{v} + \frac{\tilde{u}-u}{u} \times \frac{\tilde{v}-v}{v}$$

Interval evaluation:

$$\begin{aligned} \frac{\tilde{u}\cdot\tilde{v}-u\cdot v}{u\cdot v} &\in [-0.1, 0.1] + [-0.2, 0.2] + [-0.1, 0.1] \times [-0.2, 0.2] \\ &\in [-0.32, 0.32] \end{aligned}$$

Gappa automatically rewrites expressions in order to get tight enclosures when computing error bounds.

Gappa's Inner Workings

1. Massage the user goal into a simple logical formula:
$$e_1 \in I_1 \ \wedge \ \cdots \ \wedge \ e_n \in I_n \quad \Longrightarrow \quad e_{n+1} \in I_{n+1}.$$
2. Guess expressions and instances of theorems potentially useful as intermediate steps for bounding $e_1, \cdots, e_{n+1}$.
3. Assuming that $e_1 \in I_1, \cdots, e_n \in I_n$ hold, perform a saturation on the selected theorems until the enclosure $e_{n+1} \in I_{n+1}$ is proved.

   (Keep track of the theorems as they are applied.)
4. Generate a formal proof.

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv \quad x \in I$	`x in I`

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv\ x \in I$	`x in I`
$\mathtt{ABS}(x, I)$	$\equiv\ \lvert x \rvert \in I$	

Expression Properties

Gappa handles more than just enclosures:

$\texttt{BND}(x, I)$	$\equiv$	$x \in I$	`x in I`
$\texttt{ABS}(x, I)$	$\equiv$	$\lvert x \rvert \in I$	
$\texttt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I, \quad x = y \cdot (1 + \varepsilon)$	`x -/ y in I`

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\mathtt{ABS}(x, I)$	$\equiv$	$\lvert x\rvert \in I$		
$\mathtt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\mathtt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\mathtt{ABS}(x, I)$	$\equiv$	$\lvert x \rvert \in I$		
$\mathtt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\mathtt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`
$\mathtt{FLT}(x, p)$	$\equiv$	$\exists m, e \in \mathbb{Z},$	$x = m \cdot 2^e \wedge \lvert m \rvert < 2^p$	`@FLT(x,p)`

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\mathtt{ABS}(x, I)$	$\equiv$	$\lvert x\rvert \in I$		
$\mathtt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\mathtt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`
$\mathtt{FLT}(x, p)$	$\equiv$	$\exists m, e \in \mathbb{Z},$	$x = m \cdot 2^e \wedge \lvert m\rvert < 2^p$	`@FLT(x,p)`
$\mathtt{NZR}(x)$	$\equiv$	$x \neq 0$		

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\mathtt{ABS}(x, I)$	$\equiv$	$\lvert x \rvert \in I$		
$\mathtt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\mathtt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`
$\mathtt{FLT}(x, p)$	$\equiv$	$\exists m, e \in \mathbb{Z},$	$x = m \cdot 2^e \wedge \lvert m \rvert < 2^p$	`@FLT(x,p)`
$\mathtt{NZR}(x)$	$\equiv$	$x \neq 0$		
$\mathtt{EQL}(x, y)$	$\equiv$	$x = y$		`x = y`

Expression Properties

Gappa handles more than just enclosures:

$\mathtt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\mathtt{ABS}(x, I)$	$\equiv$	$\lvert x \rvert \in I$		
$\mathtt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\mathtt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`
$\mathtt{FLT}(x, p)$	$\equiv$	$\exists m, e \in \mathbb{Z},$	$x = m \cdot 2^e \wedge \lvert m \rvert < 2^p$	`@FLT(x,p)`
$\mathtt{NZR}(x)$	$\equiv$	$x \neq 0$		
$\mathtt{EQL}(x, y)$	$\equiv$	$x = y$		`x = y`

Variants: `x <= c`, `x >= c`, `|x| <= c`, `|x -/ y| <= c`.

Expression Properties

Gappa handles more than just enclosures:

$\texttt{BND}(x, I)$	$\equiv$	$x \in I$		`x in I`
$\texttt{ABS}(x, I)$	$\equiv$	$\lvert x\rvert \in I$		
$\texttt{REL}(x, y, I)$	$\equiv$	$\exists \varepsilon \in I,$	$x = y \cdot (1 + \varepsilon)$	`x -/ y in I`
$\texttt{FIX}(x, e)$	$\equiv$	$\exists m \in \mathbb{Z},$	$x = m \cdot 2^e$	`@FIX(x,e)`
$\texttt{FLT}(x, p)$	$\equiv$	$\exists m, e \in \mathbb{Z},$	$x = m \cdot 2^e \wedge \lvert m\rvert < 2^p$	`@FLT(x,p)`
$\texttt{NZR}(x)$	$\equiv$	$x \neq 0$		
$\texttt{EQL}(x, y)$	$\equiv$	$x = y$		`x = y`

Variants: `x <= c`, `x >= c`, `|x| <= c`, `|x -/ y| <= c`.

Question mode: replace "`in I`" by "`in ?`".

What Could Possibly Go Wrong?

Example (Wrong)

The relative round-off error is bounded:

```
{ float<ieee_64,ne>(x) -/ x in ? }
```

What Could Possibly Go Wrong?

Example (Wrong)

The relative round-off error is bounded:

```
{ float<ieee_64,ne>(x) -/ x in ? }
```

Example (Correct)

But only outside the subnormal range:

```
{ |x| in [1e-6,1e6] ->
  |float<ieee_64,ne>(x) -/ x| <= 1b-53 }
```

What Could Possibly Go Wrong?

Example (Wrong)

The relative round-off error of the FP addition is bounded:

```
{ float<ieee_64,ne>(x + y) -/ (x + y) in ? }
```

What Could Possibly Go Wrong?

Example (Wrong)

The relative round-off error of the FP addition is bounded:

```
{ float<ieee_64,ne>(x + y) -/ (x + y) in ? }
```

Example (Correct)

But only if the inputs are floating-point numbers:

```
@rnd = float<ieee_64,ne>;
x = rnd(x_); y = rnd(y_);
{ |rnd(x + y) -/ (x + y)| <= 1b-53 }
```

What Could Possibly Go Wrong?

Example (Wrong)

The relative round-off error of the FP addition is bounded:

```
{ float<ieee_64,ne>(x + y) -/ (x + y) in ? }
```

Example (Correct)

But only if the inputs are floating-point numbers:

```
@rnd = float<ieee_64,ne>;
x = rnd(x_); y = rnd(y_);
{ |rnd(x + y) -/ (x + y)| <= 1b-53 }
```

Example (Minimal)

Actually, that is how Gappa does the proof:

```
@rnd = float<53,-1074,ne>;
{ @FIX(z,-1074) -> |rnd(z) -/ z| <= 1b-53 }
```

What Could Possibly Go Wrong?

Example (Too difficult)

```
{ int<dn>(x + y) - (y + x) in ? }
```

What Could Possibly Go Wrong?

Example (Too difficult)

```
{ int<dn>(x + y) - (y + x) in ? }
```

Example (Too easy)

Gappa is guided by the syntax and structure of expressions:

```
{ int<dn>(x + y) - (x + y) in [-1,0] }
```

What Could Possibly Go Wrong?

Example (Dependencies)

```
{ x in [0,1] -> x * (1 - x) in ? }    # [0,1]
```

What Could Possibly Go Wrong?

Example (Dependencies)

```
{ x in [0,1] -> x * (1 - x) in ? }    # [0,1]
```

Example (Sub-intervals)

```
{ x in [0,1] -> x * (1 - x) in ? }    # [0,0.375]
$ x;    # $ x in 4;
# x in [0,0.25] [0.25,0.5] [0.5,0.75] [0.75,1]
```

What Could Possibly Go Wrong?

Example (Dependencies)

```
{ x in [0,1] -> x * (1 - x) in ? }    # [0,1]
```

Example (Sub-intervals)

```
{ x in [0,1] -> x * (1 - x) in ? }    # [0,0.375]
$ x;    # $ x in 4;
# x in [0,0.25] [0.25,0.5] [0.5,0.75] [0.75,1]
```

Example (Rewriting)

```
{ x in [0,1] -> x * (1 - x) in [0,0.25] }
x * (1 - x) -> 1/4 - (x - 1/2) * (x - 1/2);
```

What Could Possibly Go Wrong?

Example (Exponential)

Why does the script for exponential fail?

```
{ |x| <= 800 /\
  |TWO_M_4_LN_2 -/ Mln2div16| <= 1b-53 /\
->
  r in ? /\    # |r| <= 2^(10.6)
  p in ? }     # |p| <= 2^(46.3)
```

Due to overestimation, p possibly crosses zero;
the relative error between p and $\hat{p}$ is thus meaningless!

What Could Possibly Go Wrong?

Example (Exponential)

Why does the script for exponential fail?

```
{ |x| <= 800 /\
  |TWO_M_4_LN_2 -/ M1n2div16| <= 1b-53 /\
->
  r in ? /\    # |r| <= 2^(10.6)
  p in ? }     # |p| <= 2^(46.3)
```

Due to overestimation, p possibly crosses zero;
the relative error between p and $\hat{p}$ is thus meaningless!

Example (Sub-intervals)

```
$ x;
```

gives $|r| \leq 2^{8.6}$ and $|p| \leq 2^{36.3}$. So there are dependencies!

Fixing the Proof of Exponential

Example (Argument reduction)

$n = \lfloor \circ(x \cdot \texttt{TWO\_4\_RCP\_LN\_2}) \rceil$.

$\hat{r} = x - n \cdot \hat{c}$ with $\hat{c} = \ln 2/16$.

Note: x appears twice in $\hat{r}$, hence the overestimation.

Fixing the Proof of Exponential

Example (Argument reduction)

$n = \lfloor \circ(x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \rceil$.
$\hat{r} = x - n \cdot \hat{c}$ with $\hat{c} = \ln 2/16$.
Note: x appears twice in $\hat{r}$, hence the overestimation.

Reducing dependencies:

$$\hat{r} \quad = \quad x - x \cdot \mathtt{TWO\_4\_RCP\_LN\_2} \cdot \hat{c} - (n - x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \cdot \hat{c}$$

Fixing the Proof of Exponential

Example (Argument reduction)

$n = \lfloor \circ(x \cdot \texttt{TWO\_4\_RCP\_LN\_2}) \rceil$.
$\hat{r} = x - n \cdot \hat{c}$ with $\hat{c} = \ln 2/16$.
Note: x appears twice in $\hat{r}$, hence the overestimation.

Reducing dependencies:

$$\begin{aligned}
\hat{r} &= x - x \cdot \texttt{TWO_4_RCP_LN_2} \cdot \hat{c} - (n - x \cdot \texttt{TWO_4_RCP_LN_2}) \cdot \hat{c} \\
&= x \cdot (1 - \texttt{TWO_4_RCP_LN_2} \cdot \hat{c}) - (n - x \cdot \texttt{TWO_4_RCP_LN_2}) \cdot \hat{c}
\end{aligned}$$

Fixing the Proof of Exponential

Example (Argument reduction)

$n = \lfloor \circ(x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \rceil$.
$\hat{r} = x - n \cdot \hat{c}$ with $\hat{c} = \ln 2/16$.
Note: x appears twice in $\hat{r}$, hence the overestimation.

Reducing dependencies:

$$\begin{aligned}\hat{r} &= x - x \cdot \mathtt{TWO\_4\_RCP\_LN\_2} \cdot \hat{c} - (n - x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \cdot \hat{c} \\ &= x \cdot (1 - \mathtt{TWO\_4\_RCP\_LN\_2} \cdot \hat{c}) - (n - x \cdot \mathtt{TWO\_4\_RCP\_LN\_2}) \cdot \hat{c}\end{aligned}$$

Example (Rewriting hint for Gappa)

```
# any property on the lhs can be obtained from the rhs
x - n * Mln2div16 -> x * (1 - TWO_4_RCP_LN_2 *
   Mln2div16) - (n - x * TWO_4_RCP_LN_2) * Mln2div16;
```

Proof Hints

- Prove by splitting the range of x
  - into 4 parts: `$ x;`
  - into 10 parts: `$ x in 10;`
  - at points 1, 10, 11: `$ x in (1,10,11);`

Proof Hints

- Prove by splitting the range of x
  - into 4 parts: `$ x;`
  - into 10 parts: `$ x in 10;`
  - at points 1, 10, 11: `$ x in (1,10,11);`
- Prove a property on y by bisecting the interval of x: `y $ x;`

Proof Hints

- Prove by splitting the range of x
  - into 4 parts: `$ x;`
  - into 10 parts: `$ x in 10;`
  - at points 1, 10, 11: `$ x in (1,10,11);`
- Prove a property on y by bisecting the interval of x: `y $ x;`
- Compute and use the error between y and x: `y ~ x;`

Proof Hints

- Prove by splitting the range of x
  - into 4 parts: `$ x;`
  - into 10 parts: `$ x in 10;`
  - at points 1, 10, 11: `$ x in (1,10,11);`
- Prove a property on y by bisecting the interval of x: `y $ x;`
- Compute and use the error between y and x: `y ~ x;`
- Use y when looking for properties of x
  - always: `x -> y;`
  - only when a constraint is met: `x -> y {z >= 3};`

Outline

6 Supremum norm
- Verified supremum norms
- Previous approaches for supremum norms
- Sollya's supremum norm approach
- Supremum norm algorithm
- Supremum norms on polynomials
- Taylor Models
- Use of the supremum norm algorithm in Sollya

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.
  Let $\text{mode} \in \{\text{absolute}, \text{relative}\}$ (ε defined accordingly).

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.
  Let mode $\in \{\text{absolute}, \text{relative}\}$ (ε defined accordingly).

Find ℓ and u such that

$$\begin{cases} \|\varepsilon\|_\infty & \in & [\ell, u] \\ (u - \ell)/\ell & \leq & \eta. \end{cases}$$

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.
  Let mode $\in \{\text{absolute}, \text{relative}\}$ (ε defined accordingly).

Find ℓ and u such that

$$\begin{cases} \|\varepsilon\|_\infty & \in & [\ell, u] \\ (u-\ell)/\ell & \leq & \eta. \end{cases}$$

- These properties must be rigorously fulfilled.

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.
  Let mode $\in \{\text{absolute}, \text{relative}\}$ (ε defined accordingly).

Find ℓ and u such that

$$\begin{cases} \|\varepsilon\|_\infty & \in & [\ell, u] \\ (u-\ell)/\ell & \leq & \eta. \end{cases}$$

- These properties must be rigorously fulfilled.
- Particular difficulty: removable discontinuities
  e.g., when $f(x) = \sin(x)$ and $p(x) = x\, q(x)$.

Verified supremum norms

- Let f be a function given by an expression,
  p a polynomial,
  $[a, b]$ an interval.
  Let $\eta > 0$ be a quality parameter.
  Let mode $\in \{\text{absolute}, \text{relative}\}$ (ε defined accordingly).

Find ℓ and u such that

$$\begin{cases} \|\varepsilon\|_\infty & \in & [\ell, u] \\ (u-\ell)/\ell & \leq & \eta. \end{cases}$$

- These properties must be rigorously fulfilled.
- Particular difficulty: removable discontinuities
  e.g., when $f(x) = \sin(x)$ and $p(x) = x\,q(x)$.
- The algorithm should be able to generate a formal certificate.

State of the art

It is a univariate global optimization problem.

State of the art

It is a univariate global optimization problem.

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

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.

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 algorithm: (Hansen 1992), (Kearfott 1996), (Messine 1997).

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 algorithm: (Hansen 1992), (Kearfott 1996), (Messine 1997).

Principle: branch and bound. Maintain ℓ such that $\ell \leq \|\varepsilon\|_\infty$.

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 algorithm: (Hansen 1992), (Kearfott 1996), (Messine 1997).

Principle: branch and bound. Maintain ℓ such that $\ell \leq \|\varepsilon\|_\infty$.

Reduce the width:

$\rightsquigarrow$ bisection, Interval Newton's algorithm, Hansen's algorithm.

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 algorithm: (Hansen 1992), (Kearfott 1996), (Messine 1997).

Principle: branch and bound. Maintain ℓ such that $\ell \leq \|\varepsilon\|_\infty$.

Reduce the width:

$\rightsquigarrow$ bisection, Interval Newton's algorithm, Hansen's algorithm.

Eliminate sub-intervals:

$\rightsquigarrow$ If $\varepsilon(I) \subseteq [-\ell, \ell]$, eliminate I.

$\rightsquigarrow$ If $\varepsilon'(I) \not\ni 0$ eliminate I.

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 algorithm: (Hansen 1992), (Kearfott 1996), (Messine 1997).

Principle: branch and bound. Maintain ℓ such that $\ell \leq \|\varepsilon\|_\infty$.

Reduce the width:

$\rightsquigarrow$ bisection, Interval Newton's algorithm, Hansen's algorithm.

Eliminate sub-intervals:

$\rightsquigarrow$ If $\varepsilon(I) \subseteq [-\ell, \ell]$, eliminate I.

$\rightsquigarrow$ If $\varepsilon'(I) \not\ni 0$ eliminate I.

Increase ℓ:

$\rightsquigarrow$ If $\varepsilon(I) = [\alpha,\ \beta]$ with $\alpha \geq \ell$, set $\ell \leftarrow \alpha$.

$\rightsquigarrow$ Idem if $\beta \leq -\ell$.

Dependency phenomenon

- GO algorithms relies on the fact that $\varepsilon(I)$, $\varepsilon'(I)$, etc. give relevant information.
- In our case this assumption is not fulfilled due to the dependency phenomenon.

Dependency phenomenon

- GO algorithms relies on the fact that $\varepsilon(I)$, $\varepsilon'(I)$, etc. give relevant information.
- In our case this assumption is not fulfilled due to the dependency phenomenon.

Example: $I = [0, 1]$, $f = \exp(x)$, $\deg(p) = 13$, $\|p - f\|_\infty \leq 2\text{e}{-19}$.

Dependency phenomenon

- GO algorithms relies on the fact that $\varepsilon(I)$, $\varepsilon'(I)$, etc. give relevant information.
- In our case this assumption is not fulfilled due to the dependency phenomenon.

Example: $I = [0, 1]$, $f = \exp(x)$, $\deg(p) = 13$, $\|p - f\|_\infty \leq 2\text{e}{-19}$.

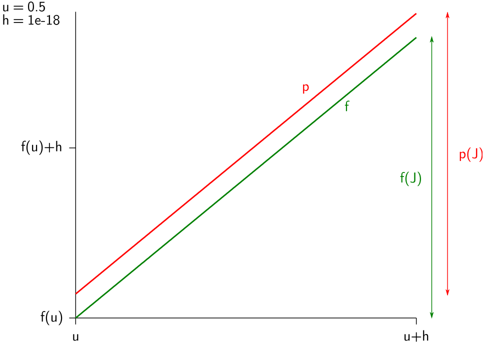
Let $h \ll 1$. On $J = [u, u + h]$:
$\varepsilon(J)$ computed through $p(J) - f(J)$.

Since $h \ll 1$,
$f(J) \simeq [f(u), f(u) + \mathcal{O}(h)]$.

Dependency phenomenon

- GO algorithms relies on the fact that $\varepsilon(I)$, $\varepsilon'(I)$, etc. give relevant information.
- In our case this assumption is not fulfilled due to the dependency phenomenon.

Example: $I = [0, 1]$, $f = \exp(x)$, $\deg(p) = 13$, $\|p - f\|_\infty \leq 2\text{e}{-19}$.



Let $h \ll 1$. On $J = [u, u + h]$:
$\varepsilon(J)$ computed through $p(J) - f(J)$.

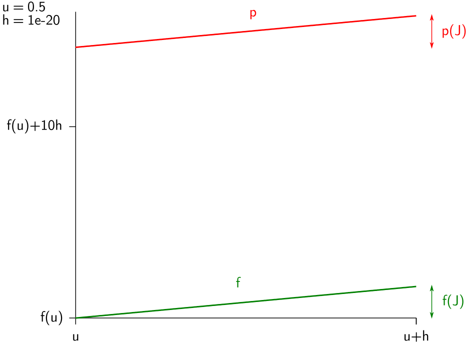
Since $h \ll 1$,
$f(J) \simeq [f(u), f(u) + \mathcal{O}(h)]$.

First case: $h \gg \|\varepsilon\|_\infty$

Dependency phenomenon

- GO algorithms relies on the fact that $\varepsilon(I)$, $\varepsilon'(I)$, etc. give relevant information.
- In our case this assumption is not fulfilled due to the dependency phenomenon.

Example: $I = [0,1]$, $f = \exp(x)$, $\deg(p) = 13$, $\|p - f\|_\infty \leq 2\text{e}{-19}$.



Let $h \ll 1$. On $J = [u, u+h]$:
$\varepsilon(J)$ computed through $p(J) - f(J)$.

Since $h \ll 1$,
$f(J) \simeq [f(u), f(u) + \mathcal{O}(h)]$.

First case: $h \gg \|\varepsilon\|_\infty$
Second case: $h \ll \|\varepsilon\|_\infty$

Dependency phenomenon (2)

This phenomenon appears at several orders:

$$\begin{aligned}
\|\varepsilon\|_\infty &= 1.4\text{e}{-19}, \\
\|\varepsilon'\|_\infty &= 5.6\text{e}{-17}, \\
\|\varepsilon''\|_\infty &= 7.3\text{e}{-15}, \\
\left\|\varepsilon^{(3)}\right\|_\infty &= 5.6\text{e}{-13}, \\
\left\|\varepsilon^{(4)}\right\|_\infty &= 3.0\text{e}{-11}, \\
\left\|\varepsilon^{(5)}\right\|_\infty &= 1.2\text{e}{-9}, \\
\left\|\varepsilon^{(6)}\right\|_\infty &= 3.8\text{e}{-8}.
\end{aligned}$$

Dependency phenomenon (2)

This phenomenon appears at several orders:

$$\begin{aligned}
\|\varepsilon\|_\infty &= 1.4\text{e}{-19},\\
\|\varepsilon'\|_\infty &= 5.6\text{e}{-17},\\
\|\varepsilon''\|_\infty &= 7.3\text{e}{-15},\\
\left\|\varepsilon^{(3)}\right\|_\infty &= 5.6\text{e}{-13},\\
\left\|\varepsilon^{(4)}\right\|_\infty &= 3.0\text{e}{-11},\\
\left\|\varepsilon^{(5)}\right\|_\infty &= 1.2\text{e}{-9},\\
\left\|\varepsilon^{(6)}\right\|_\infty &= 3.8\text{e}{-8}.
\end{aligned}$$

In conclusion: general-purpose techniques of GO useless.

Ad-hoc techniques

- Use an intermediate polynomial $T \simeq f$.
- The dependency in $p - f$ becomes a cancellation in the coefficients of $p - T$.

Ad-hoc techniques

- Use an intermediate polynomial $T \simeq f$.
- The dependency in $p - f$ becomes a cancellation in the coefficients of $p - T$.
- Idea used by (Krämer 1996), (Harrison 1997): functions manually handled, one by one.

Ad-hoc techniques

- Use an intermediate polynomial $T \simeq f$.
- The dependency in $p - f$ becomes a cancellation in the coefficients of $p - T$.
- Idea used by (Krämer 1996), (Harrison 1997): functions manually handled, one by one.
- Makino and Berz: use Taylor Models for computing T. More systematic, but does not guarantee the final quality of $[\ell, u]$.

Ad-hoc techniques

- Use an intermediate polynomial $T \simeq f$.
- The dependency in $p - f$ becomes a cancellation in the coefficients of $p - T$.
- Idea used by (Krämer 1996), (Harrison 1997): functions manually handled, one by one.
- Makino and Berz: use Taylor Models for computing T. More systematic, but does not guarantee the final quality of $[\ell, u]$.
- None of these techniques correctly handle the removable discontinuities.

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 the removable discontinuities in usual cases.

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 the removable discontinuities in usual cases.
- Could generate a complete formal proof without much effort.

How a numerical algorithm gives a rigorous bound

- Numerical algorithms give relevant information:
  1. They are often very reliable.

How a numerical algorithm gives a rigorous bound

- Numerical algorithms give relevant information:
  1. They are often very reliable.
  2. They give a rigorous lower bound ℓ.

- Algorithm:

Numerically find the zeros of ε': $L = [z_1, \ldots, z_k]$;
for $i \leftarrow 1$ **to** k
— $[a_i, b_i] \leftarrow |\varepsilon([z_i, z_i])|$;
end
$\ell \leftarrow \max |a_i|$;
return ℓ;

How a numerical algorithm gives a rigorous bound

- Numerical algorithms give relevant information:
  1. They are often very reliable.
  2. They give a rigorous lower bound ℓ.

- Algorithm:

Numerically find the zeros of ε': $L = [z_1, \ldots, z_k]$;
for $i \leftarrow 1$ **to** k
— $[a_i, b_i] \leftarrow |\varepsilon([z_i, z_i])|$;
end
$\ell \leftarrow \max |a_i|$;
return ℓ;

- It is very easy to increase the accuracy of ℓ.

How a numerical algorithm gives a rigorous bound

- Numerical algorithms give relevant information:
  1. They are often very reliable.
  2. They give a rigorous lower bound ℓ.

- Algorithm:

Numerically find the zeros of ε': $L = [z_1, \ldots, z_k]$;
for $i \leftarrow 1$ **to** k
— $[a_i, b_i] \leftarrow |\varepsilon([z_i, z_i])|$;
end
$\ell \leftarrow \max |a_i|$;
return ℓ;

- It is very easy to increase the accuracy of ℓ.
- The actual difficulty is in finding a rigorous upper bound $\rightsquigarrow$ (in other words:) prove the actual accuracy of ℓ.

Estimation of the accuracy of ℓ

- Accuracy of ℓ easy to estimate (reliably but not rigorously).

Estimation of the accuracy of ℓ

- Accuracy of ℓ easy to estimate (reliably but not rigorously).
- Let z be an approximate zero, and $z^\star \simeq z$ the exact zero. Thus:

$$\varepsilon(z) = \varepsilon(z^\star) + (z - z^\star)\,\varepsilon'(z^\star) + \frac{(z - z^\star)^2}{2}\,\varepsilon''(\xi), \quad \text{where} \quad \xi \simeq z.$$

Estimation of the accuracy of ℓ

- Accuracy of ℓ easy to estimate (reliably but not rigorously).
- Let z be an approximate zero, and $z^\star \simeq z$ the exact zero. Thus:

$$\varepsilon(z) = \varepsilon(z^\star) + (z - z^\star)\,\varepsilon'(z^\star) + \frac{(z-z^\star)^2}{2}\,\varepsilon''(\xi), \quad \text{where} \quad \xi \simeq z.$$

Estimation of the accuracy of ℓ

- Accuracy of ℓ easy to estimate (reliably but not rigorously).
- Let z be an approximate zero, and $z^\star \simeq z$ the exact zero. Thus:

$$\varepsilon(z) = \varepsilon(z^\star) + (z - z^\star)\,\varepsilon'(z^\star) + \frac{(z - z^\star)^2}{2}\,\varepsilon''(\xi), \quad \text{where} \quad \xi \simeq z.$$

- This leads to the following estimation:

$$\ell \simeq \|\varepsilon\|_\infty + \frac{(z - z^\star)^2}{2}\,\varepsilon''(z).$$

Estimation of the accuracy of ℓ

- Accuracy of ℓ easy to estimate (reliably but not rigorously).
- Let z be an approximate zero, and $z^\star \simeq z$ the exact zero. Thus:

$$\varepsilon(z) = \varepsilon(z^\star) + (z - z^\star)\,\varepsilon'(z^\star) + \frac{(z - z^\star)^2}{2}\,\varepsilon''(\xi), \quad \text{where} \quad \xi \simeq z.$$

- This leads to the following estimation:

$$\ell \simeq \|\varepsilon\|_\infty + \frac{(z - z^\star)^2}{2}\,\varepsilon''(z).$$

- Algorithm computeLowerBound(ε, I, η) returns ℓ such that

$$\begin{cases} \ell \leq \|\varepsilon\|_\infty & \text{rigorously,} \\ \left|\frac{\|\varepsilon\|_\infty - \ell}{\ell}\right| \leq \eta & \text{with a high level of confidence.} \end{cases}$$

Our algorithm, absolute error case

- Assumption: procedure `findPolyWithGivenError`(f, I, δ) computing a polynomial T (with a sufficient degree) such that $\|T - f\|_\infty \leq \delta$.

Our algorithm, absolute error case

- Assumption: procedure findPolyWithGivenError(f, I, δ) computing a polynomial T (with a sufficient degree) such that $\|T - f\|_\infty \leq \delta$.
- Our algorithm:

$\ell \leftarrow$ computeLowerBound$(p - f,\ I,\ \eta/32)$;
$m' \leftarrow \ell\,(1 + \eta/2)$; $u \leftarrow \ell\,(1 + 31\eta/32)$; $\delta \leftarrow 15\ell\,\eta/32$;
$T \leftarrow$ findPolyWithGivenError$(f,\ I,\ \delta)$;
$s_1 \leftarrow m' - (p - T)$; $s_2 \leftarrow m' - (T - p)$;
if showPositivity$(s_1,\ I) \wedge$ showPositivity$(s_2,\ I)$
then return $(\ell,\ u)$;
else return $\perp$;

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta')$.
  Hence (most likely)

$$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta')$.
  Hence (most likely)
  $$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$
- Problem: how can we prove it?

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta')$.
  Hence (most likely)
  $$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$
- Problem: how can we prove it?
- Idea: proving polynomial inequalities is easier.

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta')$.
  Hence (most likely)

$$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$

- Problem: how can we prove it?
- Idea: proving polynomial inequalities is easier.
- Let $T \leftarrow \texttt{findPolyWithGivenError}(f,\, I,\, \delta)$.
  By triangle inequality (most likely)

$$\|p - T\|_\infty \leq \ell\,(1 + \eta') + \delta. \tag{1}$$

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f, I, \eta')$.
  Hence (most likely)

$$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$

- Problem: how can we prove it?
- Idea: proving polynomial inequalities is easier.
- Let $T \leftarrow \texttt{findPolyWithGivenError}(f, I, \delta)$.
  By triangle inequality (most likely)

$$\|p - T\|_\infty \leq \ell\,(1 + \eta') + \delta. \quad (1)$$

- This inequality can be formally checked.

Absolute error case, proof

- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f, I, \eta')$.
  Hence (most likely)

$$\|p - f\|_\infty \leq \ell\,(1 + \eta').$$

- Problem: how can we prove it?
- Idea: proving polynomial inequalities is easier.
- Let $T \leftarrow \texttt{findPolyWithGivenError}(f, I, \delta)$.
  By triangle inequality (rigorously)

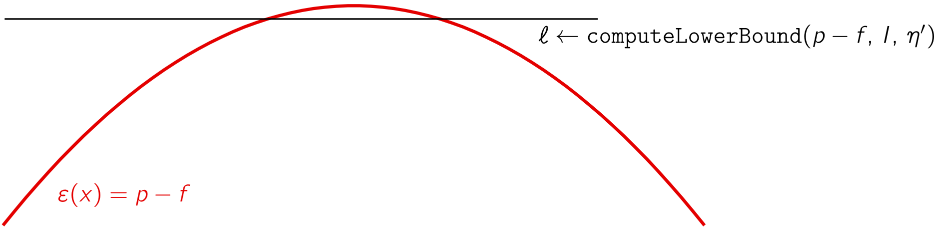
$$\|p - T\|_\infty \leq \ell\,(1 + \eta') + \delta. \tag{1}$$

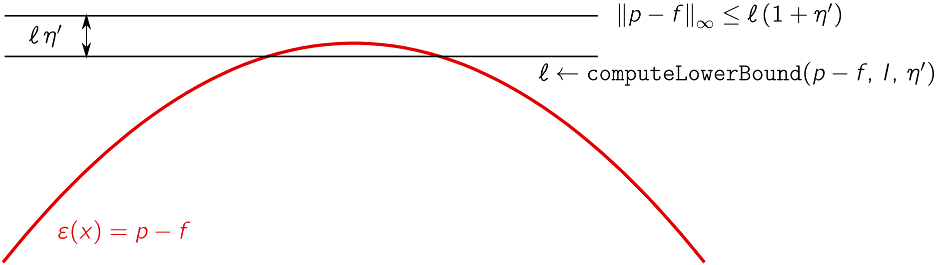
- This inequality can be formally checked.

Absolute error case, proof

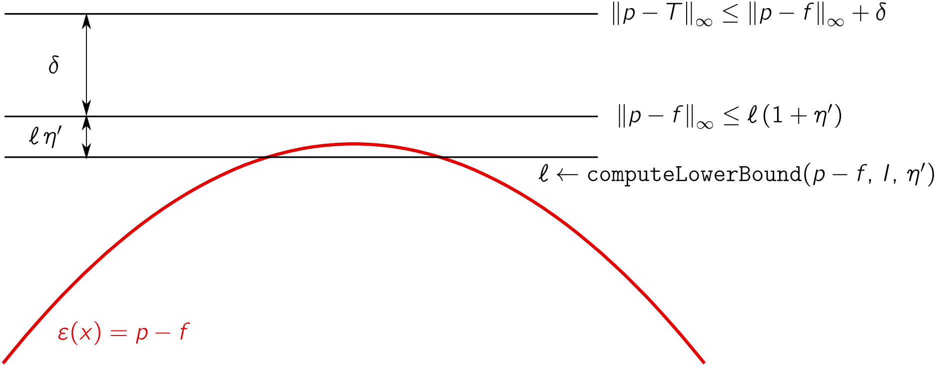
- Let $\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta')$.
  Hence (most likely)
  $$\|p - f\|_\infty \le \ell\,(1 + \eta').$$
- Problem: how can we prove it?
- Idea: proving polynomial inequalities is easier.
- Let $T \leftarrow \texttt{findPolyWithGivenError}(f,\, I,\, \delta)$.
  By triangle inequality (rigorously)
  $$\|p - T\|_\infty \le \ell\,(1 + \eta') + \delta. \qquad (1)$$
- This inequality can be formally checked.
  $\rightsquigarrow$ Using Equation (1), we get the rigorous bound:
  $$\begin{array}{rcl} \|p - f\|_\infty & \le & \|p - T\|_\infty + \|T - f\|_\infty \\ & \le & \ell\,(1 + \eta') + 2\delta. \end{array}$$


$\varepsilon(x) = p - f$

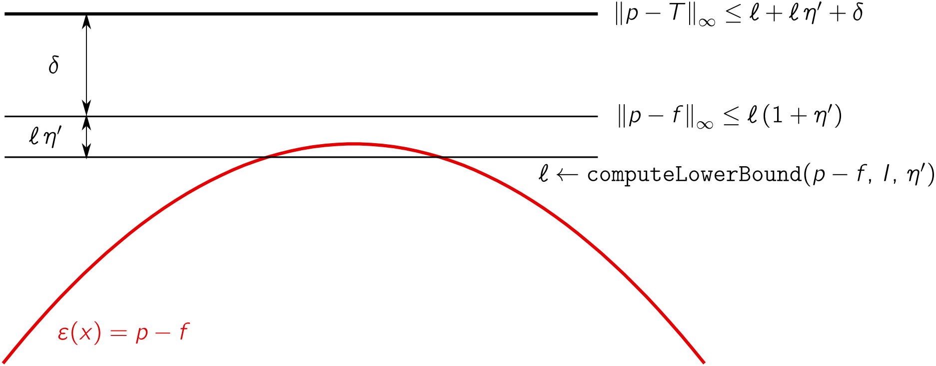

$\ell \leftarrow \texttt{computeLowerBound}(p - f,\ I,\ \eta')$
$\varepsilon(x) = p - f$


$\|p - f\|_\infty \leq \ell\,(1+\eta')$
$\ell\,\eta'$
$\ell \leftarrow \texttt{computeLowerBound}(p - f,\ I,\ \eta')$
$\varepsilon(x) = p - f$

$T \leftarrow \texttt{findPolyWithGivenError}(f, I, \delta)$

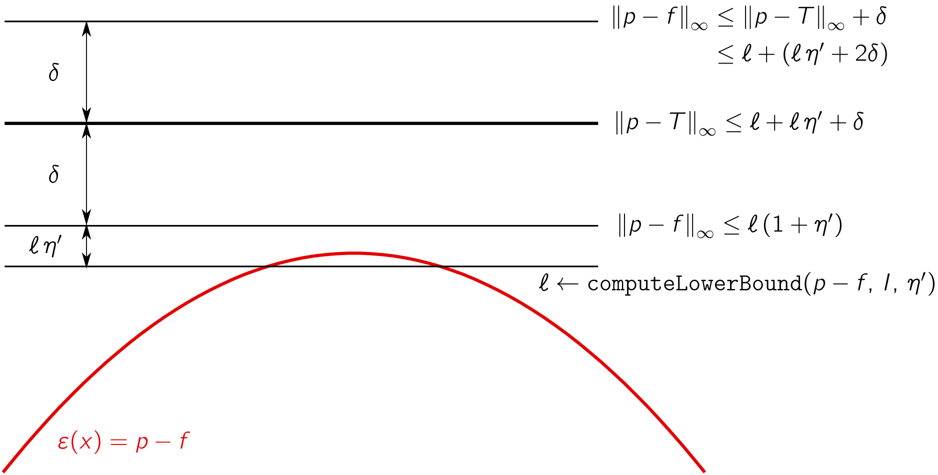


$T \leftarrow \texttt{findPolyWithGivenError}(f, I, \delta)$



$T \leftarrow$ `findPolyWithGivenError`(f, I, δ)

$\delta = \ell\,\mu$ with $\eta' + 2\mu \leq \eta$



Our algorithm: absolute error case

$\ell \leftarrow \texttt{computeLowerBound}(p - f,\, I,\, \eta/32);$

$m' \leftarrow \ell\,(1 + \eta/2); \quad u \leftarrow \ell\,(1 + 31\eta/32); \quad \delta \leftarrow 15\ell\,\eta/32;$

$T \leftarrow \texttt{findPolyWithGivenError}(f,\, I,\, \delta);$

$s_1 \leftarrow m' - (p - T); \quad s_2 \leftarrow m' - (T - p);$

if $\texttt{showPositivity}(s_1,\, I) \wedge \texttt{showPositivity}(s_2,\, I)$

then return $(\ell,\, u)$;

else return $\perp$;

Cases of failure of the algorithm

- The algorithm may fail:
  - One does not achieve to find T such that $\|f - T\|_\infty \leq \delta$.
  - The estimated accuracy of ℓ was wrong.

Cases of failure of the algorithm

- The algorithm may fail:
  - One does not achieve to find T such that $\|f - T\|_\infty \leq \delta$.
  - The estimated accuracy of ℓ was wrong.
- Important point: the algorithm never lies.

Cases of failure of the algorithm

- The algorithm may fail:
  - One does not achieve to find T such that $\|f - T\|_\infty \leq \delta$.
  - The estimated accuracy of ℓ was wrong.
- Important point: the algorithm never lies.
- Failure cases were never encountered in practice.

Cases of failure of the algorithm

- The algorithm may fail:
  - One does not achieve to find T such that $\|f - T\|_\infty \leq \delta$.
  - The estimated accuracy of ℓ was wrong.
- Important point: the algorithm never lies.
- Failure cases were never encountered in practice.
- Possible solutions in case of failure:
  - Cut the interval into sub-intervals.
  - Call `computeLowerBound` with a smaller parameter (e.g. $\eta/1024$).

Proving the supremum norm of a polynomial

- Absolute error: $\|p - T\|_\infty \leq m'$ if and only if

$$\forall x \in I, \begin{cases} m' - p(x) + T(x) \geq 0 \\ m' + p(x) - T(x) \geq 0. \end{cases}$$

Proving the supremum norm of a polynomial

- Absolute error: $\|p - T\|_\infty \le m'$ if and only if

$$\forall x \in I, \begin{cases} m' - p(x) + T(x) \ge 0 \\ m' + p(x) - T(x) \ge 0. \end{cases}$$

- Relative error: $\|p/T - 1\|_\infty \le m'$ if and only if

$$\forall x \in I, \begin{cases} m' \, |T(x)| - p(x) + T(x) \ge 0 \\ m' \, |T(x)| + p(x) - T(x) \ge 0. \end{cases}$$

Proving the supremum norm of a polynomial

- Absolute error: $\|p - T\|_\infty \le m'$ if and only if

$$\forall x \in I, \begin{cases} m' - p(x) + T(x) \ge 0 \\ m' + p(x) - T(x) \ge 0. \end{cases}$$

- Relative error: $\|p/T - 1\|_\infty \le m'$ if and only if

$$\forall x \in I, \begin{cases} m' \, |T(x)| - p(x) + T(x) \ge 0 \\ m' \, |T(x)| + p(x) - T(x) \ge 0. \end{cases}$$

- Moreover the core of the algorithm proves that

$$\forall x \in I, \left(|f(x)| \ge F \quad \text{and} \quad |f(x) - T(x)| \le \delta \le F \right).$$

Hence, T has a constant sign over I.

Proving the supremum norm of a polynomial

- Absolute error: $\|p - T\|_\infty \leq m'$ if and only if

$$\forall x \in I, \begin{cases} m' - p(x) + T(x) \geq 0 \\ m' + p(x) - T(x) \geq 0. \end{cases}$$

- Relative error: $\|p/T - 1\|_\infty \leq m'$ if and only if

$$\forall x \in I, \begin{cases} m' \, |T(x)| - p(x) + T(x) \geq 0 \\ m' \, |T(x)| + p(x) - T(x) \geq 0. \end{cases}$$

- Moreover the core of the algorithm proves that

$$\forall x \in I, \left(|f(x)| \geq F \quad \text{and} \quad |f(x) - T(x)| \leq \delta \leq F \right).$$

  Hence, T has a constant sign over I.
- In any case, proving the supremum norm of a polynomial is equivalent to proving polynomial inequalities.

Proving a polynomial inequality

In order to prove that $\forall x \in [a, b], q(x) > 0$ where q is a polynomial:

- (Recursively) study the variations of q.

Proving a polynomial inequality

In order to prove that $\forall x \in [a, b], q(x) > 0$ where q is a polynomial:

- (Recursively) study the variations of q.
- Rigorously count the real roots of q inside $[a, b]$:
  $\rightsquigarrow$ Descartes rule of signs, Sturm sequence, etc.

Proving a polynomial inequality

In order to prove that $\forall x \in [a, b], q(x) > 0$ where q is a polynomial:

- (Recursively) study the variations of q.
- Rigorously count the real roots of q inside $[a, b]$:
  $\rightsquigarrow$ Descartes rule of signs, Sturm sequence, etc.
- Sum-of-squares technique: rewrite q as

$$q(x) = \sum_{i=1}^{k} (x-a)^{\alpha_i} \, (b-x)^{\beta_i} \, s_i(x)^2,$$

  where the s_i are polynomials.

Proving a polynomial inequality

In order to prove that $\forall x \in [a,\ b], q(x) > 0$ where q is a polynomial:

- (Recursively) study the variations of q.
- Rigorously count the real roots of q inside $[a,\ b]$:
  $\rightsquigarrow$ Descartes rule of signs, Sturm sequence, etc.
- Sum-of-squares technique: rewrite q as

$$q(x) = \sum_{i=1}^{k} (x-a)^{\alpha_i} (b-x)^{\beta_i} s_i(x)^2,$$

  where the s_i are polynomials.
  - Found by an efficient, possibly heuristic, algorithm.
  - Once found: there just remains a polynomial equality to prove.
  - Particularly interesting for a formal proof.

Computing a rigorous polynomial approximation

- Goal: implement `findPolyWithGivenError`(f, I, δ).
  $\rightsquigarrow$ returning a polynomial T such that $\|T - f\|_\infty \leq \delta$.

Computing a rigorous polynomial approximation

- Goal: implement `findPolyWithGivenError`(f, I, δ).
  $\rightsquigarrow$ returning a polynomial T such that $\|T - f\|_\infty \leq \delta$.

- In practice: write a procedure `findPoly`(f, I, n)
  $\rightsquigarrow$ returning (T, Δ) with $\deg(T) = n$ and $\|T - f\|_\infty \leq \Delta$.

Computing a rigorous polynomial approximation

- Goal: implement `findPolyWithGivenError`(f, I, δ).
  $\rightsquigarrow$ returning a polynomial T such that $\|T - f\|_\infty \leq \delta$.

- In practice: write a procedure `findPoly`(f, I, n)
  $\rightsquigarrow$ returning (T, Δ) with $\deg(T) = n$ and $\|T - f\|_\infty \leq \Delta$.

- `findPolyWithGivenError` obtained from `findPoly` by a simple bisection over n.

Computing a rigorous polynomial approximation

- Goal: implement `findPolyWithGivenError`(f, I, δ).
  $\rightsquigarrow$ returning a polynomial T such that $\|T - f\|_\infty \leq \delta$.
- In practice: write a procedure `findPoly`(f, I, n)
  $\rightsquigarrow$ returning (T, Δ) with $\deg(T) = n$ and $\|T - f\|_\infty \leq \Delta$.
- `findPolyWithGivenError` obtained from `findPoly` by a simple bisection over n.
- This strategy may not terminate
  $\rightsquigarrow$ case of failure of the algorithm.

Taylor Models

- Popular implementation of `findPoly`(f, I, n): Taylor Models.
- Taylor Model of degree n of f over I: $(T, \mathbf{\Delta})$ such that
  1. T is an approximate Taylor polynomial of f.
  2. $\forall x \in I,\ f(x) - T(x) \in \mathbf{\Delta}$.

Taylor Models

- Popular implementation of `findPoly`(f, I, n): Taylor Models.
- Taylor Model of degree n of f over I: $(T, \mathbf{\Delta})$ such that
  1. T is an approximate Taylor polynomial of f.
  2. $\forall x \in I,\ f(x) - T(x) \in \mathbf{\Delta}$.
- A Taylor Model representing any function f given by an expression can be computed by means of composition rules.

Taylor Models

- Popular implementation of `findPoly`(f, I, n): Taylor Models.
- Taylor Model of degree n of f over I: $(T, \mathbf{\Delta})$ such that
  1. T is an approximate Taylor polynomial of f.
  2. $\forall x \in I,\ f(x) - T(x) \in \mathbf{\Delta}$.
- A Taylor Model representing any function f given by an expression can be computed by means of composition rules.
- Problem: the information about the zeros is lost.

Taylor Models

- Popular implementation of `findPoly`(f, I, n): Taylor Models.

- Taylor Model of degree n of f over I: $(T, \boldsymbol{\Delta})$ such that
  1. T is an approximate Taylor polynomial of f.
  2. $\forall x \in I,\ f(x) - T(x) \in \boldsymbol{\Delta}$.

- A Taylor Model representing any function f given by an expression can be computed by means of composition rules.

- Problem: the information about the zeros is lost.

$\rightsquigarrow$ $\dfrac{(x - \frac{x^2}{2}, [-7\text{e}{-3}, 7\text{e}{-3}])}{(x + \frac{x^2}{2}, [-3\text{e}{-3}, 3\text{e}{-3}])}$ leads to an infinite remainder...

... though it represents $\sin(x)/(\exp(x) - 1)$ (perfectly defined by continuity).

Modified Taylor Models

- Develop f at one of its zeros z:

$$f(x) = 0 + a_1\,(x-z) + \cdots + a_n\,(x-z)^n + \mathcal{O}(x-z)^{n+1}.$$

$\rightsquigarrow$ the information $f(z) = 0$ is readable in the development.

Modified Taylor Models

- Develop f at one of its zeros z:

$$f(x) = 0 + a_1 (x - z) + \cdots + a_n (x - z)^n + \mathcal{O}(x - z)^{n+1}.$$

$\rightsquigarrow$ the information $f(z) = 0$ is readable in the development.

- Idea: represent the coefficients of T with small intervals enclosing the actual value.
  $\rightsquigarrow$ If some coefficient is $[0, 0]$, it must be exactly 0.
- Moreover, keep $(x - z)^{n+1}$ factored out in the remainder.

Modified Taylor Models

- Develop f at one of its zeros z:

$$f(x) = 0 + a_1\,(x-z) + \cdots + a_n\,(x-z)^n + \mathcal{O}(x-z)^{n+1}.$$

$\rightsquigarrow$ the information $f(z) = 0$ is readable in the development.

- Idea: represent the coefficients of T with small intervals enclosing the actual value.
  $\rightsquigarrow$ If some coefficient is $[0, 0]$, it must be exactly 0.
- Moreover, keep $(x-z)^{n+1}$ factored out in the remainder.

Modified Taylor Models

A modified Taylor Model of f over I, developed at x_0 is $(T, \mathbf{\Delta})$ where:

1. T has (narrow) interval coefficients.
2. $\forall x \in I,\ \exists \delta \in \mathbf{\Delta},\quad f(x) - T(x - x_0) = (x - x_0)^n\,\delta$

Sollya: Computing and bounding the maximum error

```
> f = exp(x);                 /* Define the function */
> dom = [-1/4;1/4];           /* Define the domain */
> n = 5;                      /* Set degree n to 5 */
> p = remez(1, n , dom, 1/f); /* Remez polynomial */
> err = p/f - 1;              /* Define the rel. error */

> /* Compute supremum norm to get max. error */
> errmax = supnorm(p, f, dom, relative, 2^(-10));

> errmax;
[1.0576...e-8; 1.0586...e-8]

> superrmax = sup(errmax);
> log2(superrmax);
-2.649...e1
```

Outline

7 Conclusion
- Conclusion
- Perspectives
- References

What We Showed Today

How to implement a mathematical function

- that is efficient and rather accurate,

What We Showed Today

How to implement a mathematical function

- that is efficient and rather accurate,
- with an argument reduction and a polynomial approximation found by Sollya,

What We Showed Today

How to implement a mathematical function

- that is efficient and rather accurate,
- with an argument reduction and a polynomial approximation found by Sollya,
- with global error bounds verified by Gappa.

What We Did Not Show Today

For lack of time

How to implement a mathematical function

- that achieves correct rounding (*cf* table-maker dilemma),

What We Did Not Show Today

For lack of time

How to implement a mathematical function

- that achieves correct rounding (*cf* table-maker dilemma),
- that uses floating-point expansions,

What We Did Not Show Today

For lack of time

How to implement a mathematical function

- that achieves correct rounding (*cf* table-maker dilemma),
- that uses floating-point expansions,
- that is based on converging algorithms (*e.g.* Newton),

What We Did Not Show Today

For lack of time

How to implement a mathematical function

- that achieves correct rounding (*cf* table-maker dilemma),
- that uses floating-point expansions,
- that is based on converging algorithms (*e.g.* Newton),
- that is guaranteed not to have unsafe executions (*e.g.* arithmetic overflow, out-of-bound array access),

What We Did Not Show Today

For lack of time

How to implement a mathematical function

- that achieves correct rounding (*cf* table-maker dilemma),
- that uses floating-point expansions,
- that is based on converging algorithms (*e.g.* Newton),
- that is guaranteed not to have unsafe executions (*e.g.* arithmetic overflow, out-of-bound array access),
- that is formally proved to be correct.

What We Could Not Have Shown Today

Even if we had wanted to

How to implement a mathematical function

- for an arbitrary accuracy,

What We Could Not Have Shown Today

Even if we had wanted to

How to implement a mathematical function

- for an arbitrary accuracy,
- with a bound on the average error.

What the Future Holds

- Automated generators of verified implementations.

What the Future Holds

- Automated generators of verified implementations.
  - Function `implementpoly` in Sollya.

What the Future Holds

- Automated generators of verified implementations.
  - Function `implementpoly` in Sollya.
  - Project Metalibm.

What the Future Holds

- Automated generators of verified implementations.
  - Function `implementpoly` in Sollya.
  - Project Metalibm.
- Floating-point functions verified directly on their C code.

Annotated Newton Algorithm for Square Root

```
/*@ requires 0.5 <= x <= 2;
  @ ensures \abs(\result - 1/\sqrt(x)) <= 0x1p-6 * \abs(1/\sqrt(x)); */
double sqrt_init(double x);

/*@ lemma quadratic_newton: \forall real x, t; x > 0 ==>
      \let err = (t - 1 / \sqrt(x)) / (1 / \sqrt(x));
      (0.5 * t * (3 - t * t * x) - 1 / \sqrt(x)) / (1 / \sqrt(x)) ==
        - (1.5 + 0.5 * err) * (err * err); */

/*@ requires 0.5 <= x <= 2;
  @ ensures \abs(\result - \sqrt(x)) <= 0x1p-43 * \abs(\sqrt(x)); */
double sqrt(double x)
{
  int i;
  double t, u;
  t = sqrt_init(x);

  /*@ loop pragma UNROLL 4;
    @ loop invariant 0 <= i <= 3; */
  for (i = 0; i <= 2; ++i) {
    u = 0.5 * t * (3 - t * t * x);
    //@ assert \abs(u - 0.5 * t * (3 - t * t * x)) <= 1;
    /*@ assert \let err = (t - 1 / \sqrt(x)) / (1 / \sqrt(x));
          (0.5 * t * (3 - t * t * x) - 1 / \sqrt(x)) / (1 / \sqrt(x)) ==
          - (1.5 + 0.5 * err) * (err * err); */
    //@ assert \abs(u - 1 / \sqrt(x)) <= 0x1p-10 * \abs(1 / \sqrt(x));
    t = u;
  }

  //@ assert x * (1 / \sqrt(x)) == \sqrt(x);
  return x * t;
}
```

References

- Muller, *Elementary Functions, Algorithms and Implementation*, Birkhäuser, 2nd ed., 2006.
- de Dinechin, CL, Muller, *Fast and correctly rounded logarithms in double-precision*, RAIRO, vol. 41, 85-102, 2007.
- Muller et al., *Handbook of Floating-Point Arithmetic*, Birkhäuser, 2009.
- Boldo, Filliâtre, GM, Combining Coq and Gappa for certifying floating-point programs, *Proceedings of Calculemus 16*, 59-74, Grand Bend, CA-ON, 2009.
- Daumas, GM, *Certification of bounds on expressions involving rounded operators*, ACM TOMS, vol. 37 (1), 1-20, 2010.
- Chevillard, Joldeș, CL, Sollya: an environment for the development of numerical codes, in *Proceedings of ICMS'10*, 28-31, Kobe, Japan, 2010.
- de Dinechin, CL, GM, *Certifying the floating-point implementation of an elementary function using Gappa*, IEEE TOC, vol. 60 (2), 242-253, 2011.
- Chevillard, Harrison, Joldeș, CL, *Efficient and accurate computation of upper bounds of approximation errors*, TCS, vol. 412 (16), 1523-1543, 2011.