Towards a Modern Floating-Point Environment

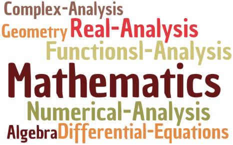
PhD thesis defense

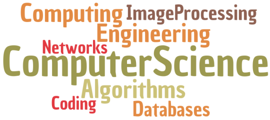
Olga Kupriianova

PEQUAN team, CalSci Dept, LIP6, UPMC

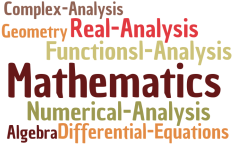
advisors: Jean-Claude Bajard, Christoph Lauter

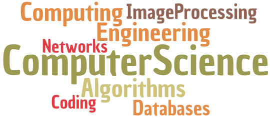
What is Floating-Point Arithmetic





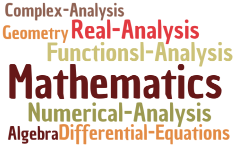
What is Floating-Point Arithmetic

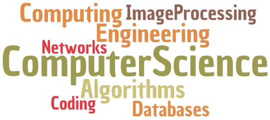




Continuous	**Discrete**
$\mathbb{R}$ numbers	Floating point (FP) numbers
$3/2 = 1.5$	$3/2 = 1.5e0$
$\pi = 3.14159{\color{red}2653589\dots}$	$\pi = 3.14159e0$
$\sqrt{2} = 1.41421{\color{red}3562373\dots}$	$\sqrt{2} = 1.41421e0$

What is Floating-Point Arithmetic





Continuous	**Discrete**
$\mathbb{R}$ numbers	Floating point (FP) numbers
$3/2 = 1.5$	$3/2 = 1.5e0$
$\pi = 3.14159\textcolor{red}{2653589\ldots}$	$\pi = 3.14159e0$
$\sqrt{2} = 1.41421\textcolor{red}{3562373\ldots}$	$\sqrt{2} = 1.41421e0$

$$\pm \underbrace{m_0.m_1m_2\ldots m_{k-1}}_{\text{significand}} \cdot \beta^E$$

E is exponent, k precision, β radix

Rounding

How to represent a real number in machine?

Rounding

How to represent a real number in machine?

$\pi \approx 3.14$	$e \approx 2.71$
$\pi \approx 3.141$	$e \approx 2.718$
$\pi \approx 3.1415$	$e \approx 2.7182$
$\pi \approx 3.14159$	$e \approx 2.71828$
$\pi \approx 3.141592$	$e \approx 2.718281$
$\pi \approx 3.1415926$	$e \approx 2.7182818$

Rounding

How to represent a real number in machine?

$\pi \approx 3.14$	$e \approx 2.71$
$\pi \approx 3.141$	$e \approx 2.718$
$\pi \approx 3.1415$	$e \approx 2.7182$
$\pi \approx 3.14159$	$e \approx 2.71828$
$\pi \approx 3.141592$	$e \approx 2.718281$
$\pi \approx 3.1415926$	$e \approx 2.7182818$

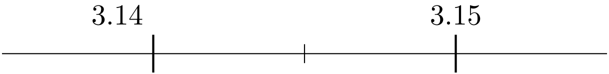
$$\pm \underbrace{m_0.m_1m_2\ldots m_{k-1}}_{\text{significand}} \cdot \beta^E,$$

E is exponent, k precision

How to Round

Rounding modes:

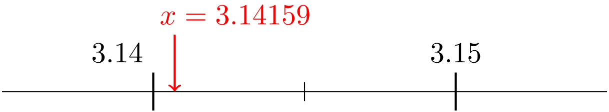
- Rounding Down (RD)
- Rounding Up (RU)
- Rounding toward Zero (RZ)
- Rounding to the Nearest : ties to even (RN), ties to away (RA)



How to Round

Rounding modes:

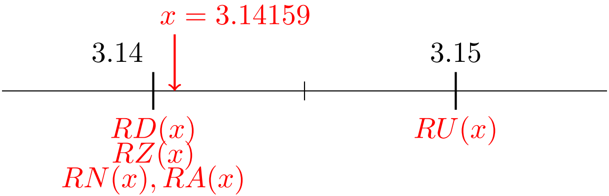
- Rounding Down (RD)
- Rounding Up (RU)
- Rounding toward Zero (RZ)
- Rounding to the Nearest : ties to even (RN), ties to away (RA)



How to Round

Rounding modes:

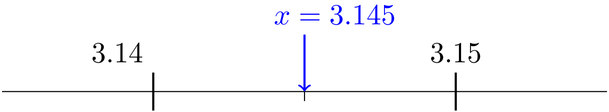
- Rounding Down (RD)
- Rounding Up (RU)
- Rounding toward Zero (RZ)
- Rounding to the Nearest : ties to even (RN), ties to away (RA)



How to Round

Rounding modes:

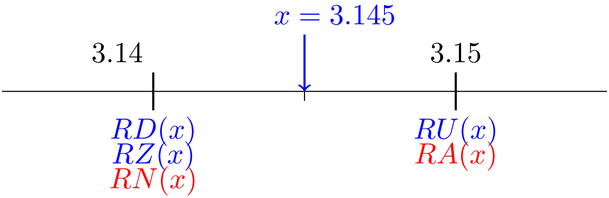
- Rounding Down (RD)
- Rounding Up (RU)
- Rounding toward Zero (RZ)
- Rounding to the Nearest : ties to even (RN), ties to away (RA)



How to Round

Rounding modes:

- Rounding Down (RD)
- Rounding Up (RU)
- Rounding toward Zero (RZ)
- Rounding to the Nearest : ties to even (RN), ties to away (RA)

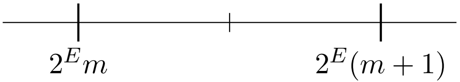


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

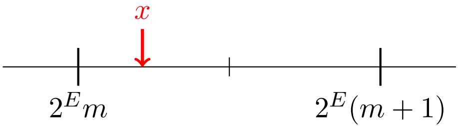


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

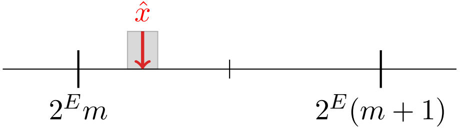


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

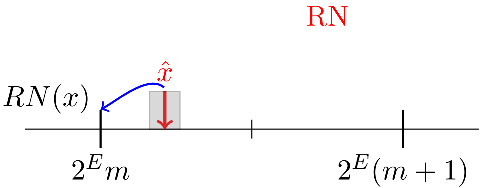


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

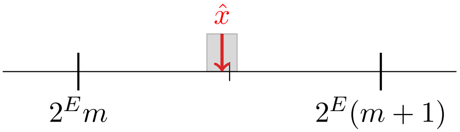


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

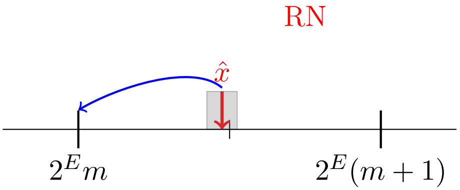


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

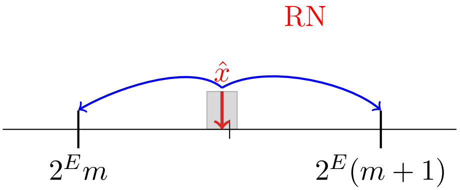


Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$



Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

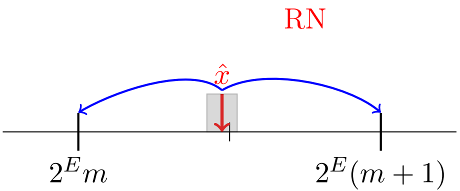


Table Maker's Dilemma (TMD)

Accuracy and Rounding

Accuracy

Mathematical $x \to \hat{x}$ in FP arithmetic

Due to roundings, $|x - \hat{x}| > 0$

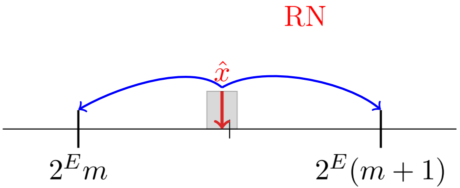


Table Maker's Dilemma (TMD)

How to determine the accuracy

of the approximation $\hat{x}$

Floating-Point Environment

- **IEEE754-1985**: binary FP numbers, rounding modes, arithmetic operations
  In total ~ 70 operations

Floating-Point Environment

- **IEEE754-1985**: binary FP numbers, rounding modes, arithmetic operations
  In total ~ 70 operations
- **IEEE754-2008** : decimal FP numbers and operations, recommendations for mathematical function implementations, heterogeneous operations: mix precisions
  In total ~ 350 operations for binary arithmetic

Floating-Point Environment

- **IEEE754-1985**: binary FP numbers, rounding modes, arithmetic operations
  In total ~ 70 operations
- **IEEE754-2008** : decimal FP numbers and operations, recommendations for mathematical function implementations, heterogeneous operations: mix precisions
  In total ~ 350 operations for binary arithmetic
- **Current situation**: decimal and binary worlds, intersect only in conversion
- **...next revisions (2018?)**: even more operations?

Floating-Point Environment

- **IEEE754-1985**: binary FP numbers, rounding modes, arithmetic operations
  In total $\sim$ 70 operations
- **IEEE754-2008** : decimal FP numbers and operations, recommendations for mathematical function implementations, heterogeneous operations: mix precisions
  In total $\sim$ 350 operations for binary arithmetic
- **Current situation**: decimal and binary worlds, intersect only in conversion
- **...next revisions (2018?)**: even more operations?
  mix binary and decimals in one operation

Floating-Point Environment

- **IEEE754-1985**: binary FP numbers, rounding modes, arithmetic operations
  In total ~ 70 operations
- **IEEE754-2008** : decimal FP numbers and operations, recommendations for mathematical function implementations, heterogeneous operations: mix precisions
  In total ~ 350 operations for binary arithmetic
- **Current situation**: decimal and binary worlds, intersect only in conversion
- **...next revisions (2018?)**: even more operations?
  mix binary and decimals in one operation
  provide more implementations of mathematical functions

Implementation of Mathematical Functions

Why do we need more implementations of mathematical functions?

Implementation of Mathematical Functions

Why do we need more implementations of mathematical functions?

Mathematics

$$\exp(x) = e^x = \lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n$$

$$\log_b(x) = y; \quad b^y = x$$

$$\operatorname{erf}(x) = \int_0^x e^{-t^2} dt$$

Computer Science

Several implementations for each:
exp - correctly rounded
exp - faithfully rounded
exp - with accuracy $\leq 2^{-45}$
...

Implementation of Mathematical Functions

Why do we need more implementations of mathematical functions?

Mathematics

$$\exp(x) = e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$$

$$\log_b(x) = y; \quad b^y = x$$

$$\mathrm{erf}(x) = \int_0^x e^{-t^2} dt$$

Computer Science

Several implementations for each:
exp - correctly rounded
exp - faithfully rounded
exp - with accuracy $\leq 2^{-45}$
...

The existing libraries give *only few* implementations per function

Contributions/Structure

$$2^E m \quad 10^F n$$

$$2^E m \times 10^{F_1} n_1 = 10^{F_2} n_2$$

$$2^{E_1} m_1 + 10^F n = 2^{E_2} m_2$$

1. Mixed-Radix Arithmetic and Arbitrary Precision Base Conversion

2. Automatic Generation of Mathematical Functions (Metalibm)

$$\exp \quad \sin$$

$$\int_0^x \frac{\sin t}{t} dt$$

$$\log$$

Mixed-Radix Arithmetic

- Radix conversion algorithm for Mixed-Radix Arithmetic
- Correctly-rounded conversion from decimal character sequence to a binary FP number (scanf analogue)
- Research for fused multiply-add operation $xy \pm z$

Mixed-Radix Arithmetic

Radix Conversion

- Two-steps algorithm:
  1. Exponent computation is errorless
  2. Compute mantissa with specified accuracy
- Integer-based computations
- Memory consumption is known in advance

Mixed-Radix Arithmetic

- Radix conversion algorithm for Mixed-Radix Arithmetic
- Correctly-rounded conversion from decimal character sequence to a binary FP number (scanf analogue)
- Research for fused multiply-add operation $xy \pm z$

Mixed-Radix Arithmetic

Our scanf version

- User input is arbitrarily long
- No way to precompute the worst cases for the TMD
- Algorithm with integer computations, no memory allocations
- Two-ways scheme is used: fast easy rounding or slow hard
- Two conversions: decimal-binary and binary-decimal
- Gives correctly-rounded (CR) result for all the rounding modes

Mixed-Radix Arithmetic

- Radix conversion algorithm for Mixed-Radix Arithmetic
- Correctly-rounded conversion from decimal character sequence to a binary FP number (scanf analogue)
- Research for fused multiply-add operation (FMA) $xy \pm z$

Mixed-Radix Arithmetic

Mixed-radix FMA

$$xy \pm z$$

$$2^{-4} \cdot 139 \times 10^{-16} \cdot 346 - 2^3 \cdot 42 = 10^E m$$

- Addition and multiplication within *only* one rounding
- Worst cases search: reduce the iterations number with the use of continued fraction theory and variable relations
- Iterations reduced at $\sim 99\%$
- Still about $\sim 16\,000\,000\,000$ iterations
- First results obtained in ~ 10 weeks of computations

Code Generation for Mathematical Functions

1. Introduction & Motivation for Code Generation
2. Manual Function Implementation. Background
3. Metalibm: General Overview
4. Metalibm: Domain Splitting
5. Metalibm: Reconstruction for Vectorizable Code
6. Conclusion and Perspectives

Existing Mathematical Libraries (libm)

Existing implementations

- Intel's Math Kernel Libraries
- AMD's libm
- ARM's mathlib
- libmcr by Sun
- ...
- glibc libm
- CRLibm by ENS Lyon
- newlib
- OpenLibm for Julia
- Yeppp!

Existing Mathematical Libraries (libm)

Existing implementations

- Intel's Math Kernel Libraries
- AMD's libm
- ARM's mathlib
- libmcr by Sun
- ...
- glibc libm
- CRLibm by ENS Lyon
- newlib
- OpenLibm for Julia
- Yeppp!

All these libms are *static*
Only few implementations per function
Limited dictionary of functions

Existing Mathematical Libraries (libm)

Existing implementations

- Intel's Math Kernel Libraries
- AMD's libm
- ARM's mathlib
- libmcr by Sun
- ...
- glibc libm
- CRLibm by ENS Lyon
- newlib
- OpenLibm for Julia
- Yeppp!

All these libms are *static*
Only few implementations per function
Limited dictionary of functions

How to get
$\exp(x)$ with 40 accurate bits

Existing Mathematical Libraries (libm)

Existing implementations

- Intel's Math Kernel Libraries
- AMD's libm
- ARM's mathlib
- libmcr by Sun
- ...
- glibc libm
- CRLibm by ENS Lyon
- newlib
- OpenLibm for Julia
- Yeppp!

All these libms are *static*
Only few implementations per function
Limited dictionary of functions

How to get
$\exp(x)$ with 40 accurate bits
$\int_0^x \frac{\sin t}{t} dt$ with 25 bits

One Size Does not Fit All

Current request

- Several performance/accuracy options
  ("quick and dirty", faithful, correctly-rounded)
- Several portability options (SIMD instructions)

One Size Does not Fit All

Current request

- Several performance/accuracy options ("quick and dirty", faithful, correctly-rounded)
- Several portability options (SIMD instructions)

Some people are still not happy

- More performance, less compliance
  - degraded accuracy
  - reduced domain
- Functions not from standard libm (e.g. $\int_0^x \frac{\sin t}{t} dt$, dilogarithm)

One Size Does not Fit All

Current request

- Several performance/accuracy options
  ("quick and dirty", faithful, correctly-rounded)
- Several portability options (SIMD instructions)

Some people are still not happy

- More performance, less compliance
  - degraded accuracy
  - reduced domain
- Functions not from standard libm (e.g. $\int_0^x \frac{\sin t}{t} dt$, dilogarithm)

Who is going to write all these variants?

Task

Rough Computations

- 3 precisions from IEEE754
- ~ 50 functions in libm
- ~ 5 accuracies for each precision
- ~ 5 various approximations

Task

Rough Computations

- 3 precisions from IEEE754
- ~ 50 functions in libm
- ~ 5 accuracies for each precision
- ~ 5 various approximations

I have to implement **1300** functions...

Task

Rough Computations

- 3 precisions from IEEE754
- ~ 50 functions in libm
- ~ 5 accuracies for each precision
- ~ 5 various approximations

I have to implement **1300** functions...

Each function implementation takes ~ 1 man-month

Task

Rough Computations

- 3 precisions from IEEE754
- ~ 50 functions in libm
- ~ 5 accuracies for each precision
- ~ 5 various approximations

I have to implement **1300** functions...

Each function implementation takes ~ 1 man-month
It will take me more then **100** years to rewrite the libm

Task

Rough Computations

- 3 precisions from IEEE754
- ~ 50 functions in libm
- ~ 5 accuracies for each precision
- ~ 5 various approximations

I have to implement **1300** functions...

Each function implementation takes ~ 1 man-month
It will take me more then **100** years to rewrite the libm

Generate parametrized implementations of mathematical functions
Metalibm

How to Implement a Mathematical Function

Elementary functions have transcendental values

$$\exp(1) = 2.71828182845904523536028747135266\ldots$$
$$\log_2(10) = 3.32192809488736234787031942948939\ldots$$

How to Implement a Mathematical Function

Elementary functions have transcendental values

$$\exp(1) = 2.718281828459045235360287471352 66\ldots$$
$$\log_2(10) = 3.321928094887362347870319429489 39\ldots$$

Compute polynomial approximations

How to Implement a Mathematical Function

Elementary functions have transcendental values

$$\exp(1) = 2.7182818284590452353602874713526\ldots$$
$$\log_2(10) = 3.3219280948873623478703194294893\ldots$$

Compute polynomial approximations
Lagrange, Newton, Chebyshev

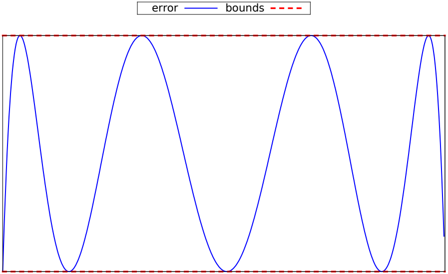
How to Implement a Mathematical Function

Elementary functions have transcendental values

$$\exp(1) = 2.71828182845904523536028747135266\ldots$$
$$\log_2(10) = 3.32192809488736234787031942948939\ldots$$

Compute polynomial approximations
Lagrange, Newton, Chebyshev

Remez algorithm

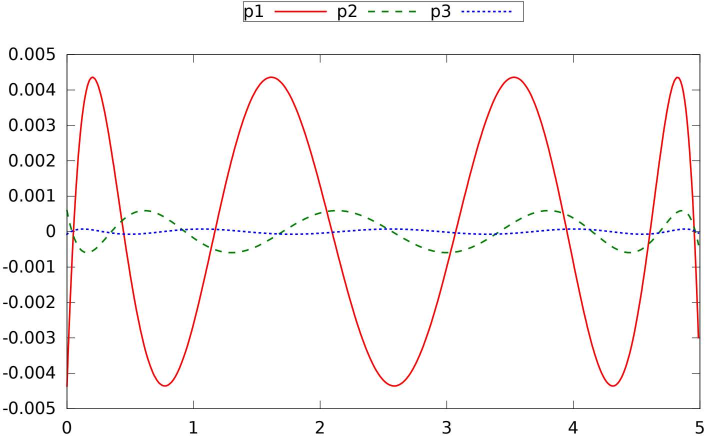


Approximation Polynomials

$$f = \exp$$

Compute different approximations on $[0, 5]$

$\deg(p_1) = 7$, $\deg(p_2) = 8$, $\deg(p_3) = 9$



Approximation Polynomials

$$f = \exp$$

Compute different approximations on $[0, 5]$

$\deg(p_1) = 7$, $\deg(p_2) = 8$, $\deg(p_3) = 9$

Larger domain, larger degree

Conclusion: approximate *only* on small domains

Argument Reduction. Example

Example

Implement $f(x) = e^x$

Exploit the algebraic property $e^{a+b} = e^a \cdot e^b$

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

$$r \in \left[-\frac{\log(2)}{2}, \frac{\log(2)}{2} \right]$$

Argument Reduction. Example

Example

Implement $f(x) = e^x$
Exploit the algebraic property $e^{a+b} = e^a \cdot e^b$
P. Tang's table-based method

$$e^x = 2^m \cdot 2^{i/2^t} \cdot e^{r^*}, \quad r^* \in \left[-\frac{\log(2)}{2 \cdot 2^t}, \frac{\log(2)}{2 \cdot 2^t}\right]$$

$$m \in \mathbb{Z}, \quad t \in \mathbb{Z}, \quad 0 \le i \le 2^t - 1, \quad 2^{i/2^t} \text{ in table}$$

Argument Reduction. Does It Always Work?

Useful algebraic properties:

$$\exp(a+b) = \exp(a) \cdot \exp(b)$$
$$\log(ab) = \log(a) + \log(b)$$
$$\sin(a + 2\pi k) = \sin(a)$$
$$\ldots$$

Argument Reduction. Does It Always Work?

Useful algebraic properties:

$$\exp(a+b) = \exp(a)\cdot\exp(b)$$
$$\log(ab) = \log(a) + \log(b)$$
$$\sin(a + 2\pi k) = \sin(a)$$
$$\dots$$

How to implement Dickman's function?

$$u\rho'(u) + \rho(u-1) = 0, \quad \rho(u) = 1 \text{ for } 0 \le u \le 1$$

Argument Reduction. Does It Always Work?

Useful algebraic properties:

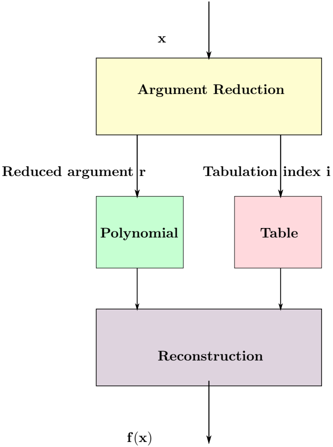
$$\exp(a+b) = \exp(a)\cdot\exp(b)$$
$$\log(ab) = \log(a) + \log(b)$$
$$\sin(a+2\pi k) = \sin(a)$$
$$\ldots$$

How to implement Dickman's function?

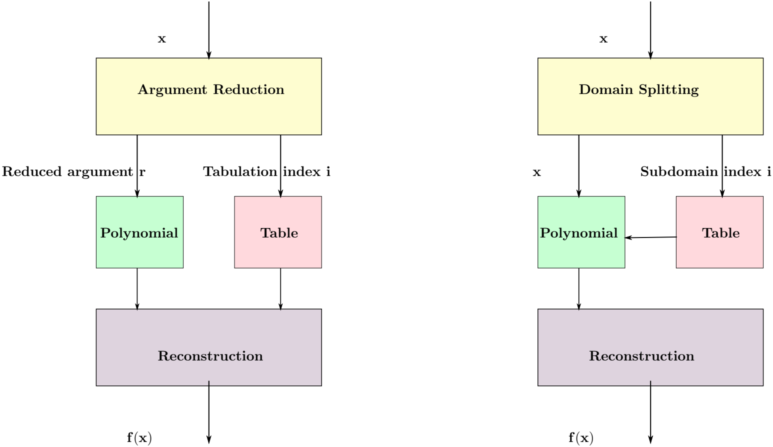
$$u\rho'(u) + \rho(u-1) = 0, \quad \rho(u) = 1 \text{ for } 0 \le u \le 1$$

Piecewise-polynomial approximation

General Scheme

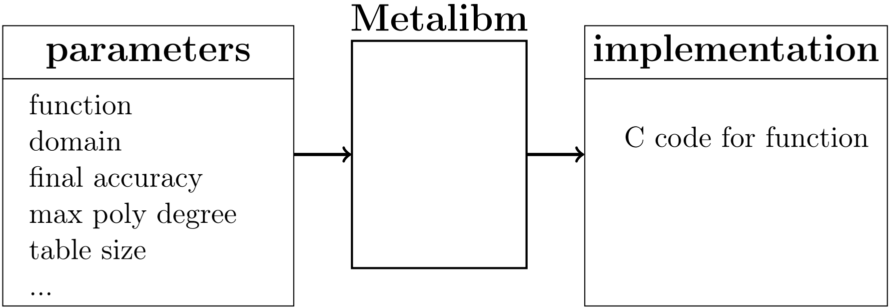


General Scheme



And filtering of special cases

Structure


Metalibm
parameters
function
domain
final accuracy
max poly degree
table size
...
implementation
C code for function

What is Metalibm

Academic Prototype

- Open-Source code generator for parametrized libm functions
- Part of ANR Metalibm Project
- Available at `http://metalibm.org/`

Objectives

- Push-button tool to implement functions $f : \mathbb{R} \to \mathbb{R}$
  - automatic argument reduction
  - automatic polynomial approximation
  - automatic domain splitting
- Support black-box functions
  - no function dictionary
  - specify function by an expression or external code

What is Metalibm

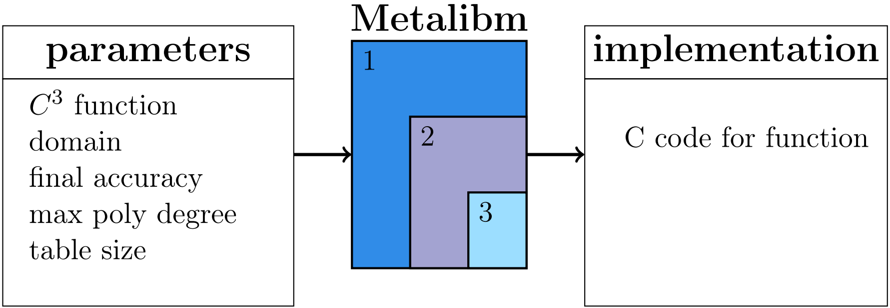
Academic Prototype

- Open-Source code generator for parametrized libm functions
- Part of ANR Metalibm Project
- Available at `http://metalibm.org/`

Objectives

- Push-button tool to implement functions $f : \mathbb{R} \to \mathbb{R}$
  - automatic argument reduction
  - automatic polynomial approximation
  - automatic domain splitting
- Support black-box functions
  - no function dictionary
  - specify function by an expression or external code
    $\Rightarrow$ use only C^3 functions that can be evaluated over intervals

Generation Levels



1 - Properties detection
2 - Domain splitting
3 - Approximation

Property Detection: Example on Exponential

Task

Generate $f(x)$ on $[a, b]$ with accuracy $\bar{\varepsilon}$

Generation hypothesis

$f(x)$ is of type β^x, unknown β

Property Detection: Example on Exponential

Task

Generate $f(x)$ on $[a, b]$ with accuracy $\bar{\varepsilon}$

Generation hypothesis

$f(x)$ is of type β^x, unknown β

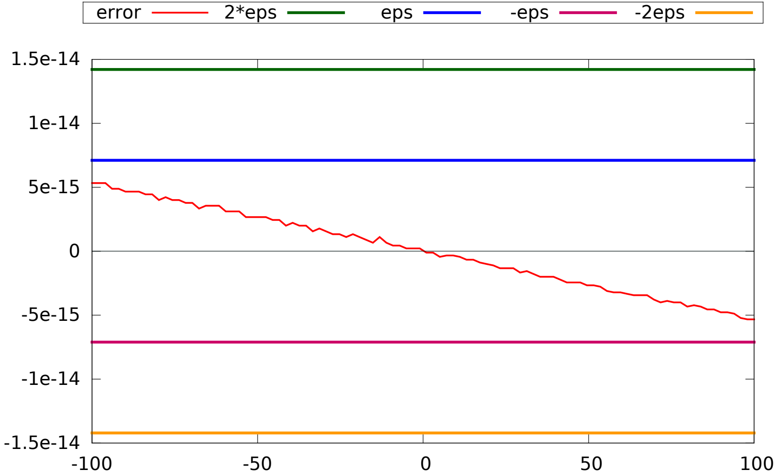
Finding the base

$\beta = \exp\left(\frac{\log(f(\xi))}{\xi}\right)$, for some $\xi \in [a, b]$

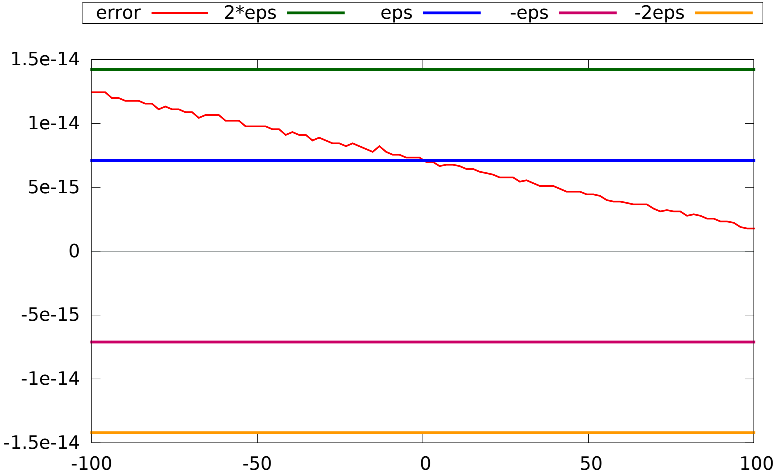
Decision of acceptance

$$\varepsilon = \left\| \frac{\beta^x}{f(x)} - 1 \right\|_\infty^{[a,b]} \qquad |\varepsilon| < |\bar{\varepsilon}|$$

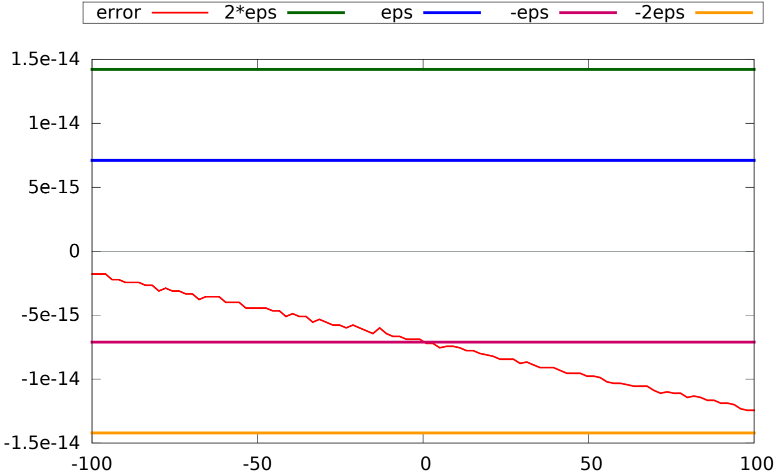
Property Detection: Example on Exponential



Property Detection: Example on Exponential



Property Detection: Example on Exponential



Property Detection

Currently detectable properties

- Exponential $f(a+b) = f(a)f(b)$
- Logarithmic $f(ab) = f(a) + f(b)$
- Periodic $f(x+C) = f(x)$
- Symmetric $f(x) = f(-x)$ and $f(x) = -f(x)$
- Sinh-like family $f(x) = \beta^x - \beta^{-x}$

When property detection does not work

Argument reduction

Step is based on mathematical properties:
$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

When property detection does not work

Argument reduction

Step is based on mathematical properties:
$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

What properties do we know for $\mathrm{erf}(x)$, or a black-box function?

When property detection does not work

Argument reduction

Step is based on mathematical properties:
$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

What properties do we know for $\mathrm{erf}(x)$, or a black-box function?

When argument reduction does not work

Piecewise-polynomial approximation

When property detection does not work

Argument reduction

Step is based on mathematical properties:
$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

What properties do we know for $\mathrm{erf}(x)$, or a black-box function?

When argument reduction does not work

Piecewise-polynomial approximation

- Algorithm to split the domain
- Reconstruction becomes an execution of if-statements

Problem Statement

Task:
Split the domain $[a, b]$ into $I_0, I_1, \ldots, I_N$
Approximation degrees on I_k: $d_k \leq d_{\max}$

function	f
initial domain	$[a, b]$
accuracy	$\bar{\varepsilon}$
degree bound	$d_{\max}$

Problem Statement

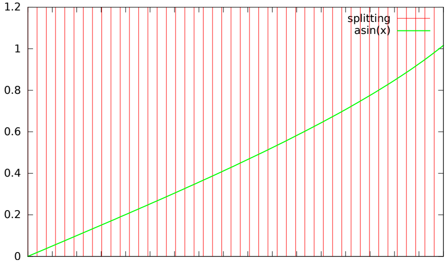
function	f
initial domain	$[a, b]$
accuracy	$\bar{\varepsilon}$
degree bound	$d_{\max}$

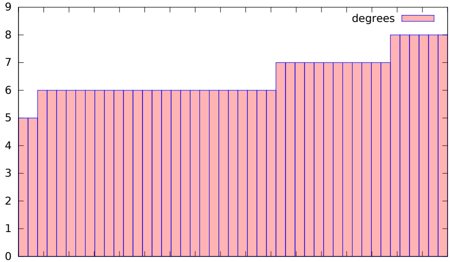
Task:
Split the domain $[a, b]$ into $I_0, I_1, \ldots, I_N$
Approximation degrees on I_k: $d_k \leq d_{\max}$

Naive Solution:
Split domain into N equal parts, N is large.

Problem Statement

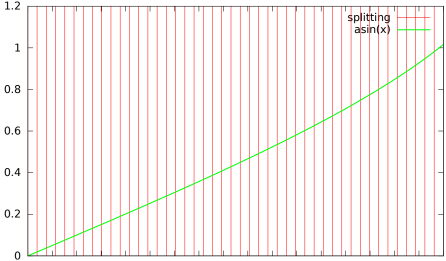
$$f = \mathrm{asin},\ [a,b] = [0, 0.85],\ d_{\max} \leq 8,\ \bar{\varepsilon} = 2^{-52}$$

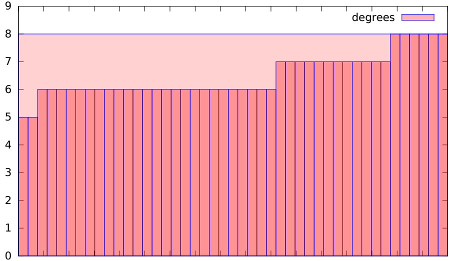




45 intervals

$$f = \text{asin},\ [a,b] = [0, 0.85],\ d_{\max} \leq 8,\ \bar{\varepsilon} = 2^{-52}$$





45 intervals

Problem Statement

The quantity of intervals $N \to \min$

Classic problem:

having f, $[a, b]$ and d
find a polynomial p and accuracy ε

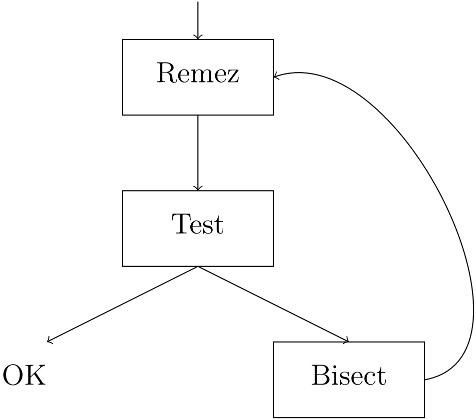
We need:

having f, $[a, b]$, bounds $\bar{\varepsilon}$ and $d_{\max}$
find I and polynomial p

Solution:

compute Remez polynomials and check the constraints?

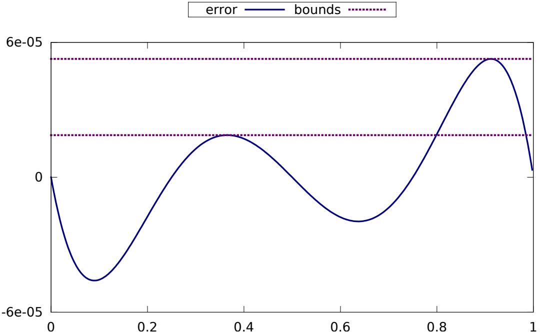
Problem Statement



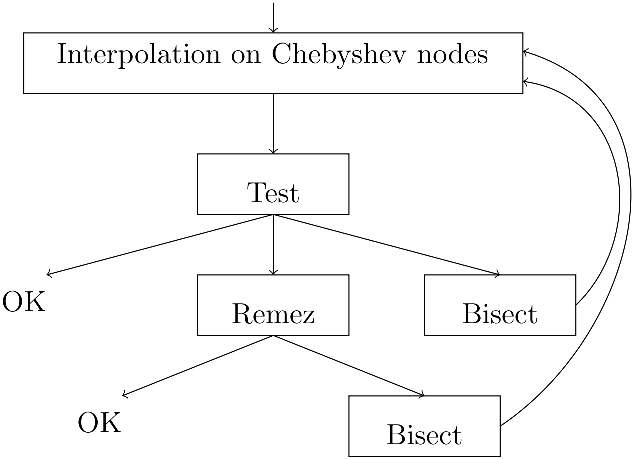
Domain Splitting

Theorem of de la Vallée-Poussin:

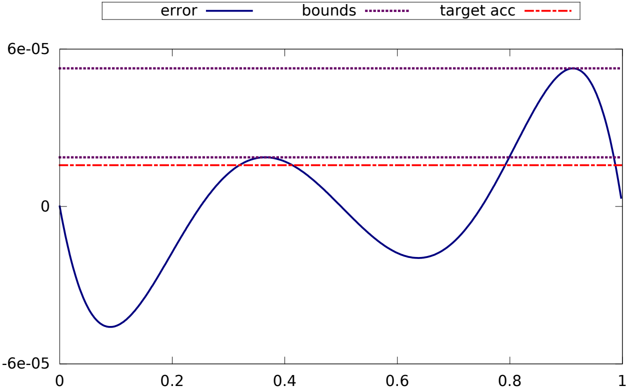
a continuous function f on $[a, b]$,
its approximating polynomial p
find the bounds for optimal error



Domain Splitting

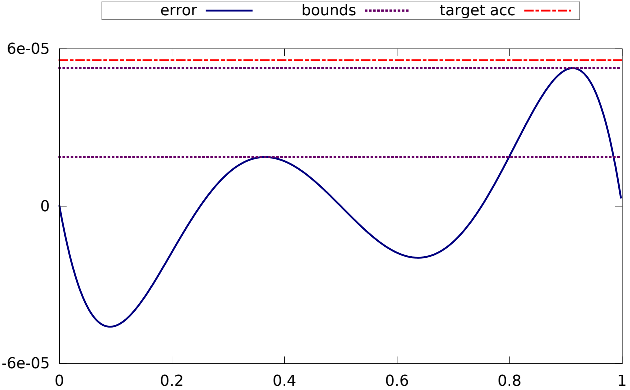


Domain Splitting



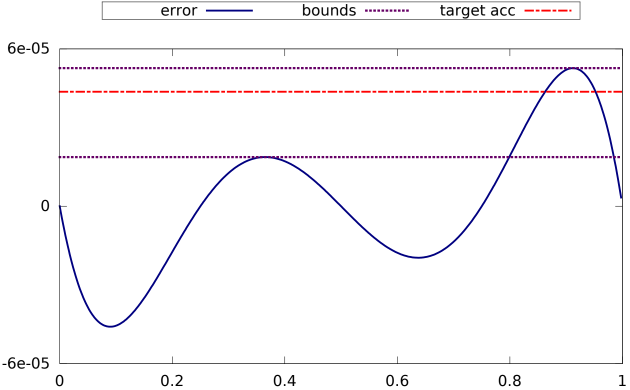
Need a split

Domain Splitting



No split

Domain Splitting

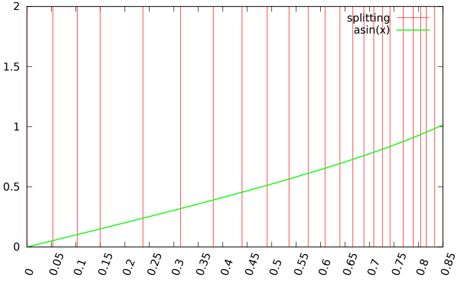


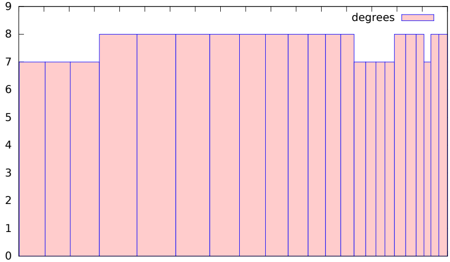
Need Remez

Domain Splitting. Our method

Bisection

$$f = \text{asin},\ [a,b] = [0, 0.85],\ d_{\max} \le 8,\ \bar{\varepsilon} = 2^{-52}$$

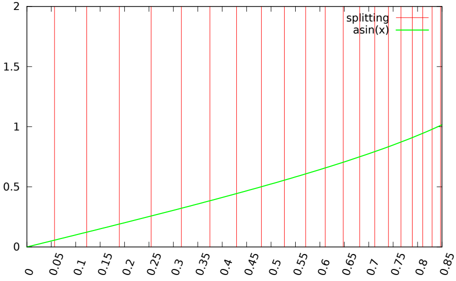


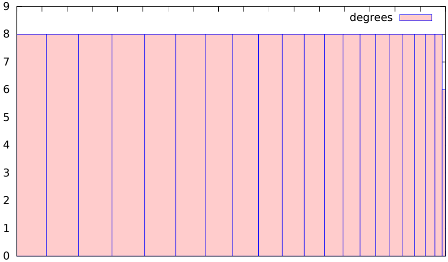


24 intervals

Our optimized method

$$f = \text{asin},\ [a,b] = [0, 0.85],\ d_{\max} \leq 8,\ \bar{\varepsilon} = 2^{-52}$$





18 intervals

Some results

measure	f_1	f_2	f_3	f_4
subdomains in bisection	24	15	9	12
subdomains in our method	18	10	5	8
subdomains saved	25%	30%	44 %	30%
coefficients saved	42	34	30	28
memory saved (bytes)	336	272	240	224
memory saved (%)	23%	35%	38.5%	45%

name	f	$\bar{\varepsilon}$	domain I	$d_{\max}$
f_1	asin	2^{-52}	$[0, 0.85]$	8
f_2	asin	2^{-45}	$[-0.75, 0.75]$	8
f_3	erf	2^{-51}	$[-0.75, 0.75]$	9
f_4	erf	2^{-45}	$[-0.75, 0.75]$	7

Problem Statement

Reconstruction gets tricky with arbitrary domain splitting:

Problem Statement

Reconstruction gets tricky with arbitrary domain splitting:

$f(x) \approx p_k(x)$, $x \in I_k$, $k \in [0, N] \cap \mathbb{Z}$

Problem Statement

Reconstruction gets tricky with arbitrary domain splitting:

$f(x) \approx p_k(x)$, $x \in I_k$, $k \in [0, N] \cap \mathbb{Z}$

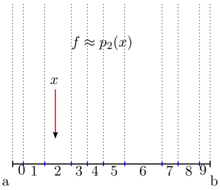
$\forall x \in [a, b]$ find k such that $x \in I_k$

Problem Statement

Reconstruction gets tricky with arbitrary domain splitting:

$f(x) \approx p_k(x)$, $x \in I_k$, $k \in [0, N] \cap \mathbb{Z}$

$\forall x \in [a, b]$ find k such that $x \in I_k$



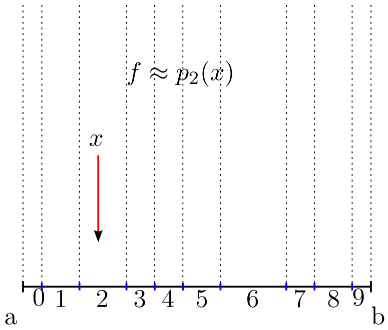
Problem Statement

```
/* compute i so that a[i] < x < a[i+1] */
        i=31;
        if (x < arctan_table[i][A].d) i-= 16;
        else i+=16;
        if (x < arctan_table[i][A].d) i-= 8;
        else i+= 8;
        if (x < arctan_table[i][A].d) i-= 4;
        else i+= 4;
        if (x < arctan_table[i][A].d) i-= 2;
        else i+= 2;
        if (x < arctan_table[i][A].d) i-= 1;
        else i+= 1;
        if (x < arctan_table[i][A].d) i-= 1;
        xmBihi = x-arctan_table[i][B].d;
        xmBilo = 0.0;
```

Listing 1: Code sample for arctan function from `crlibm` library

Vectorizable Reconstruction

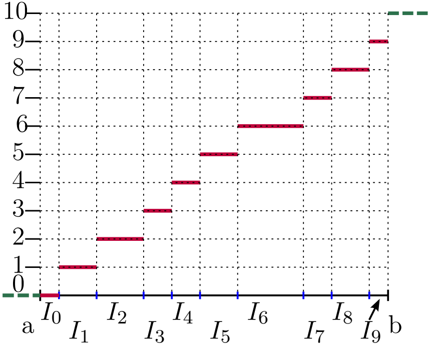
Splitting of
the domain $[a, b]$:



Vectorizable Reconstruction

To determine the subinterval we build a mapping function

$$P(x) = k, x \in I_k$$

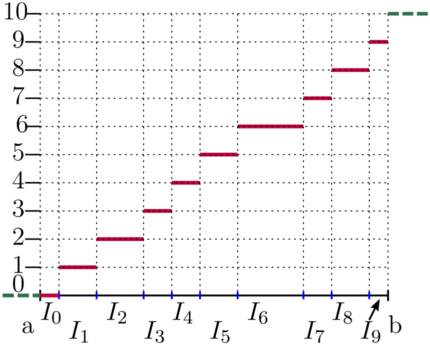


Vectorizable Reconstruction

To determine the subinterval we build a mapping function

$$P(x) = k, x \in I_k$$

$$p(x) : \lfloor p(x) \rfloor = k, x \in I_k$$

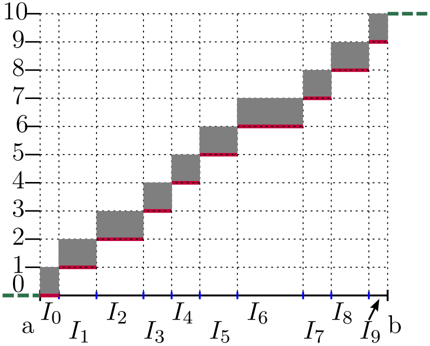


Vectorizable Reconstruction

To determine the subinterval we build a mapping function

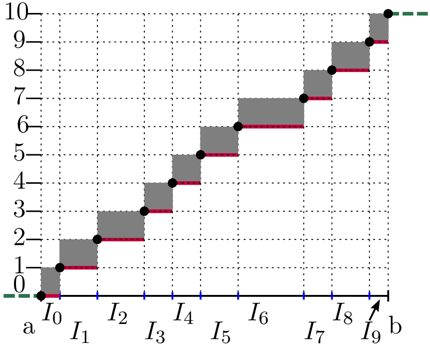
$$P(x) = k, x \in I_k$$

$$p(x) : \lfloor p(x) \rfloor = k, x \in I_k$$



Vectorizable Reconstruction

Interpolation polynomial
with a posteriori condition check

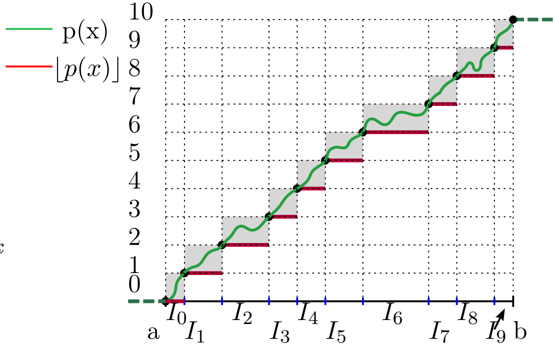


Vectorizable Reconstruction

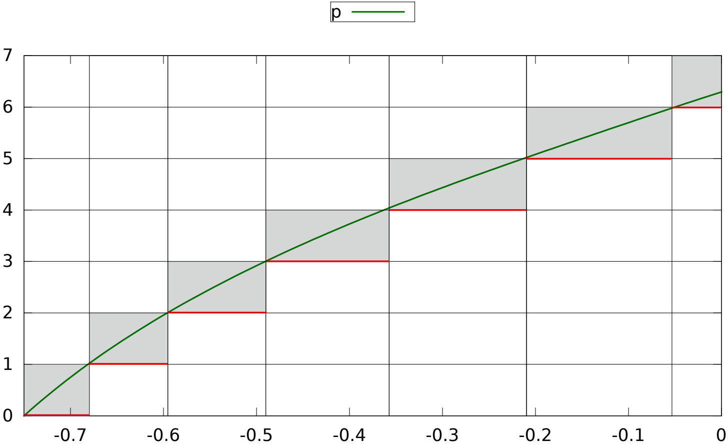
mapping function

$$P(x) = k, x \in I_k$$

$$p(x) : \lfloor p(x) \rfloor = k, x \in I_k$$



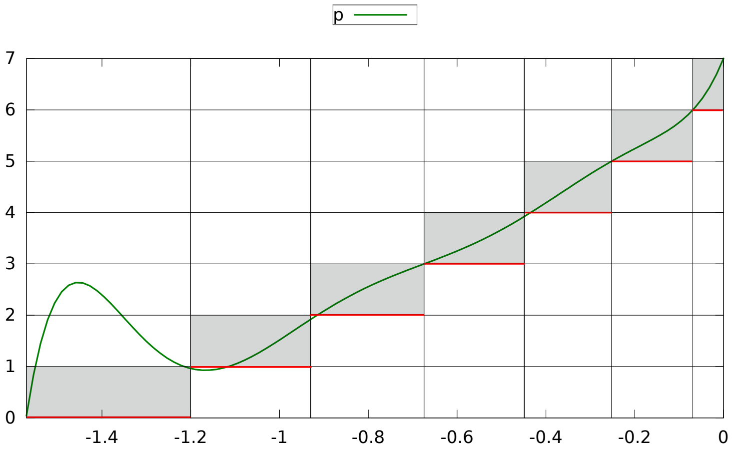
Example



f = asin(x); dom = [-0.75; 0];

$\bar{\varepsilon} = 2^{-48}$; $d_{\max} = 10$;

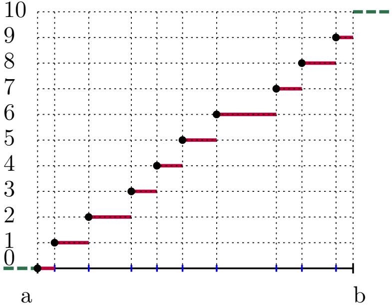
A Posteriori Condition Check Fails



$$f = \mathrm{atan}(x);\ \mathrm{dom} = [-\pi/2;\ 0];$$
$$\bar{\varepsilon} = 2^{-40};\ d_{\max} = 8;$$

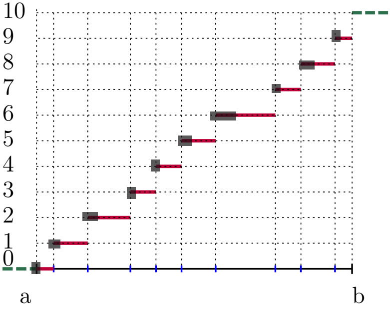
How to Improve our Method

Interval Arithmetic approach:



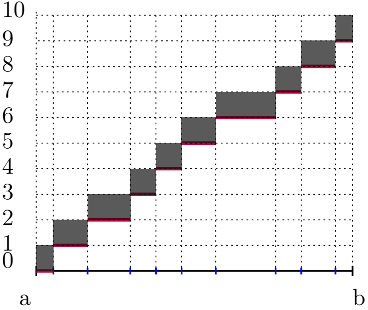
How to Improve our Method

Interval Arithmetic approach:



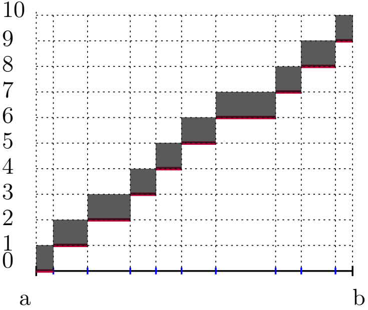
How to Improve our Method

Interval Arithmetic approach:



How to Improve our Method

Interval Arithmetic approach:

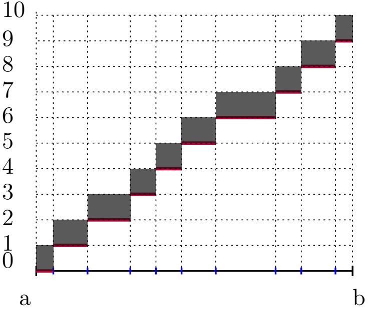


Interval system of linear algebraic equations

$$\begin{pmatrix} 1 & \mathbf{a}_0 & \cdots & \mathbf{a}_0^n \\ 1 & \mathbf{a}_1 & \cdots & \mathbf{a}_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{a}_n & \cdots & \mathbf{a}_n^n \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{pmatrix}$$

How to Improve our Method

Interval Arithmetic approach:



Interval system of linear algebraic equations

$$\begin{pmatrix} 1 & \mathbf{a}_0 & \cdots & \mathbf{a}_0^n \\ 1 & \mathbf{a}_1 & \cdots & \mathbf{a}_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{a}_n & \cdots & \mathbf{a}_n^n \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{pmatrix}$$

tolerance solution: $\Xi_{tol} = \left\{c \in \mathbb{R}^N \,|\, \forall a \in \mathbf{a}, \forall b \in \mathbf{b}, \ Ac = b\right\}$

united solution: $\Xi_{uni} = \left\{c \in \mathbb{R}^N \,|\, \exists a \in \mathbf{a}, \exists b \in \mathbf{b}, \ Ac = b\right\}$

S. Shary HdR thesis "Интервальные алгебраические задачи и их численное решение"

Achievements

- Generate the specified function versions in few minutes
- Detect algebraic properties automatically
- Use the improved (optimized) domain splitting algorithm
- Generation of the vectorizable implementations has started
- Extra bonuses: composite functions and a set of function variants

Code Generation for Mathematical Functions

Conclusion

Paving the road for mixed-radix arithmetic:

- Novel algorithm for radix conversion
- Algorithm for conversion from decimal character sequence to a binary FP number
- Worst cases search for FMA

Code generation for mathematical functions:

- Novel algorithm for domain splitting
- Novel approach to generate vectorizable implementations

Future Work

Mixed-Radix arithmetic

- Finish the worst cases search for FMA
- Start its implementation
- Research on other arithmetic operations
- Obtain test/comparison results for our scanf

Code generation of mathematical functions

- Filtering of special cases
- Improve vectorizable reconstruction:
  - Interval arithmetic for a priori approach
  - Overlapping intervals
  - Connection between reconstruction and domain splitting
- Add more parameters
- Integrate with N.Brunie's version

Perspectives

Long-term Future

- Integrate Metalibm to the glibc/gcc
- Apply the similar algorithms for filter generation
- Formal proof for scanf algorithm

Questions

- N. Brunie, F. de Dinechin, O. Kupriianova, and C. Lauter. Code generators for mathematical functions. In ARITH22 - June 2015, Lyon, France. Proceedings, pp 66-73. Best paper award.
- O. Kupriianova and C. Q. Lauter. A domain splitting algorithm for the mathematical functions code generator. In Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, November, 2014, pp 1271-1275.
- O. Kupriianova and C. Q. Lauter. Replacing branches by polynomials in vectorizable elementary functions. 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, September 2014, Würzburg, Germany.
- O. Kupriianova and C. Q. Lauter. Metalibm: A mathematical functions code generator. In Mathematical Software - ICMS 2014 - 4th International Congress, Seoul, South Korea, August 2014. Proceedings, pp 713–717.
- O. Kupriianova, C. Q. Lauter, and J.-M. Muller. Radix conversion for IEEE754-2008 mixed radix floating-point arithmetic. In Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, November 2013, pp 1134–1138.