Replacing branches by polynomials in vectorizable elementary functions

Olga Kupriianova Christoph Lauter

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6,
F-75005 Paris, France

16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics
Würzburg, Germany
21 - 26 September 2014

Outline

Mathematical libraries

Mathematical library (libm)

- Gives a set of maths functions (exp, log, sin, cos, `pow`, ...)
- Supports important precisions (`float`, `double`, `long double`)
- Limited set of functions: trigonometric, exponentiation and logarithmic, hyperbolic, special functions
- Limited set of precisions / final accuracies
- The only one implementation, hard-coded long time ago.

Glibc libm

Why glibc libm?

- Math library running on all linux-powered machines

- Written by different developer teams
- Some codes were written 20 years ago
- Strange naming conventions

Metalibm approach

f = exp(x);
dom = [-100,100];
target = $2^{(-24)}$;
maxDegree = 6;

f = sin(x)+10;
dom = [-100,100];
target = $2^{(-53)}$;
maxDegree = 6;

f = erf(x);
dom = [-0.85,0.85];
target = $2^{(-42)}$;
maxDegree = 6;

Different flavors of the same function

$\sim$ 50 functions in `libm`

$\sim$ 1 man-month for implementation of a flavor

Metalibm approach

f = exp(x);
dom = [-100,100];
target = $2^{(-24)}$;
maxDegree = 6;
	f = sin(x)+10;
	dom = [-100,100];
	target = $2^{(-53)}$;
	maxDegree = 6;
		f = erf(x);
		dom = [-0.85,0.85];
		target = $2^{(-42)}$;
		maxDegree = 6;

Different flavors of the same function

$\sim$ 50 functions in `libm`

$\sim$ 1 man-month for implementation of a flavor

$\Rightarrow$ impossible to implement all manually

Metalibm approach

f = exp(x);
dom = [-100,100];
target = $2^{(-24)}$;
maxDegree = 6;
	f = sin(x)+10;
	dom = [-100,100];
	target = $2^{(-53)}$;
	maxDegree = 6;
		f = erf(x);
		dom = [-0.85,0.85];
		target = $2^{(-42)}$;
		maxDegree = 6;

Different flavors of the same function

$\sim$ 50 functions in `libm`

$\sim$ 1 man-month for implementation of a flavor

$\Rightarrow$ impossible to implement all manually

$\Rightarrow$ write a code generator

Outline

Function implementation steps

1. Eliminating special cases and values: zeros, infinities, NaNs, etc.
2. Argument reduction: $r \in [\alpha, \beta]$. r lies in a small interval
3. Polynomial approximation: Remez algorithm for minimax approximations, polynomial of low degree
4. Reconstruction

Example

implement $f(x) = e^x$

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

Range reduction

Range reduction

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

Range reduction

Range 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 are the properties for $\mathrm{erf}(x)$, or a function purely defined by ODE?

Range reduction

Range 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 are the properties for $\mathrm{erf}(x)$, or a function purely defined by ODE?

When range reduction does not work

Piecewise-polynomial approximations

Range reduction

Range reduction

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

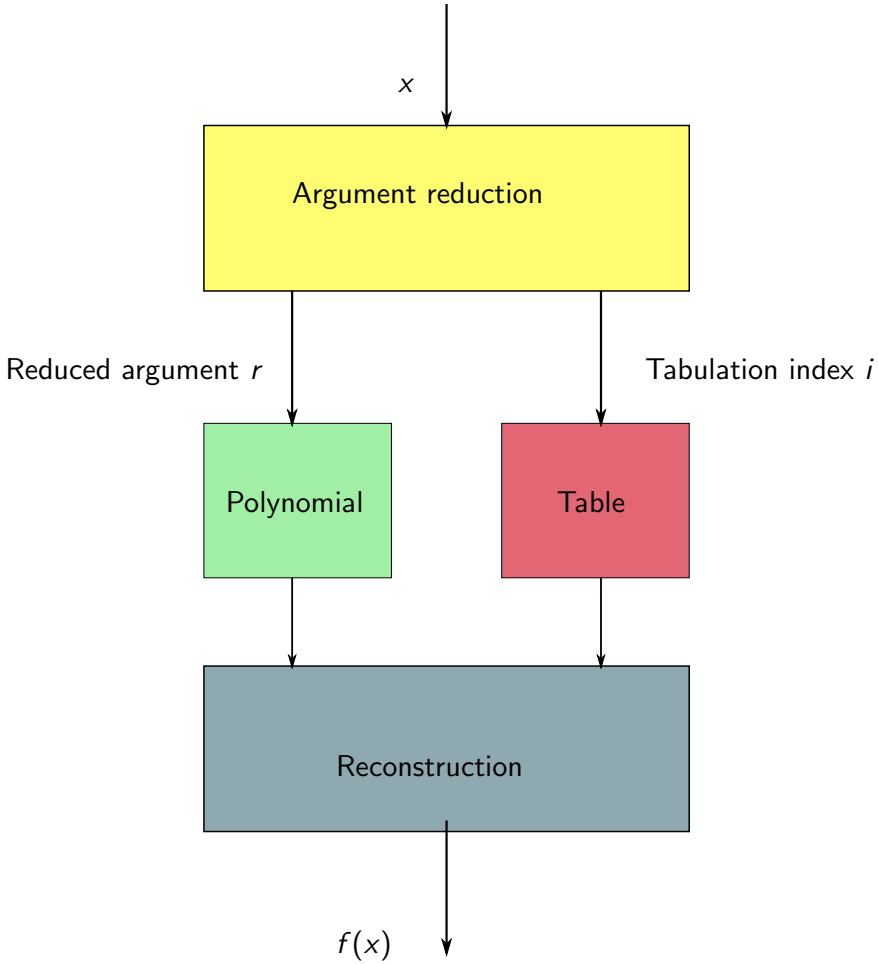
What are the properties for $\mathrm{erf}(x)$, or a function purely defined by ODE?

When range reduction does not work

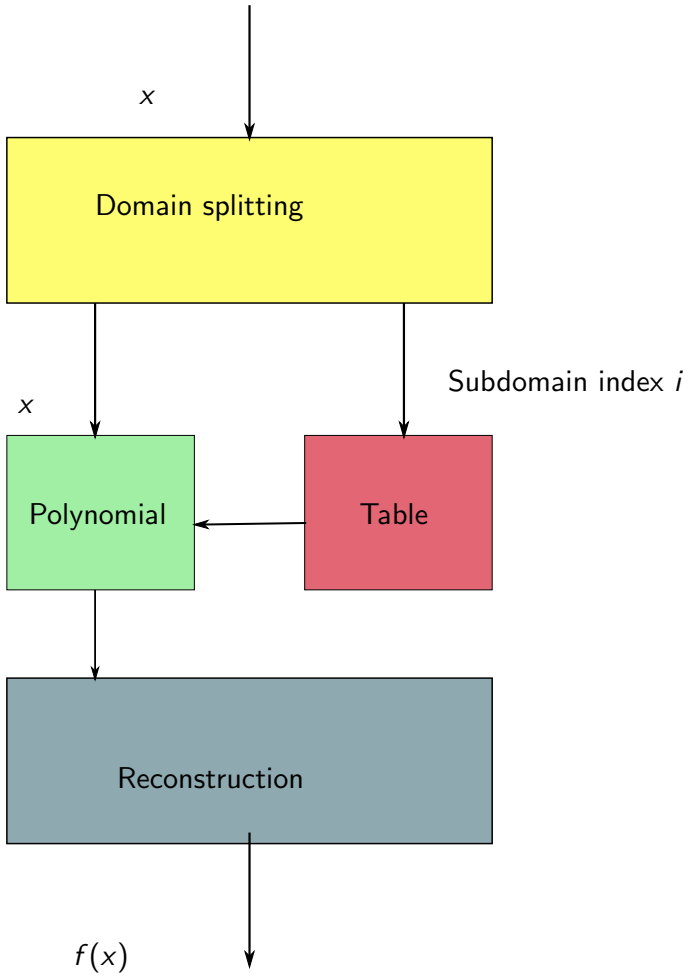
Piecewise-polynomial approximations

- Algorithm to split the domain
- Piecewise polynomial approximation
- Reconstruction is execution of several if-else statements

Mathematical Function Implementation



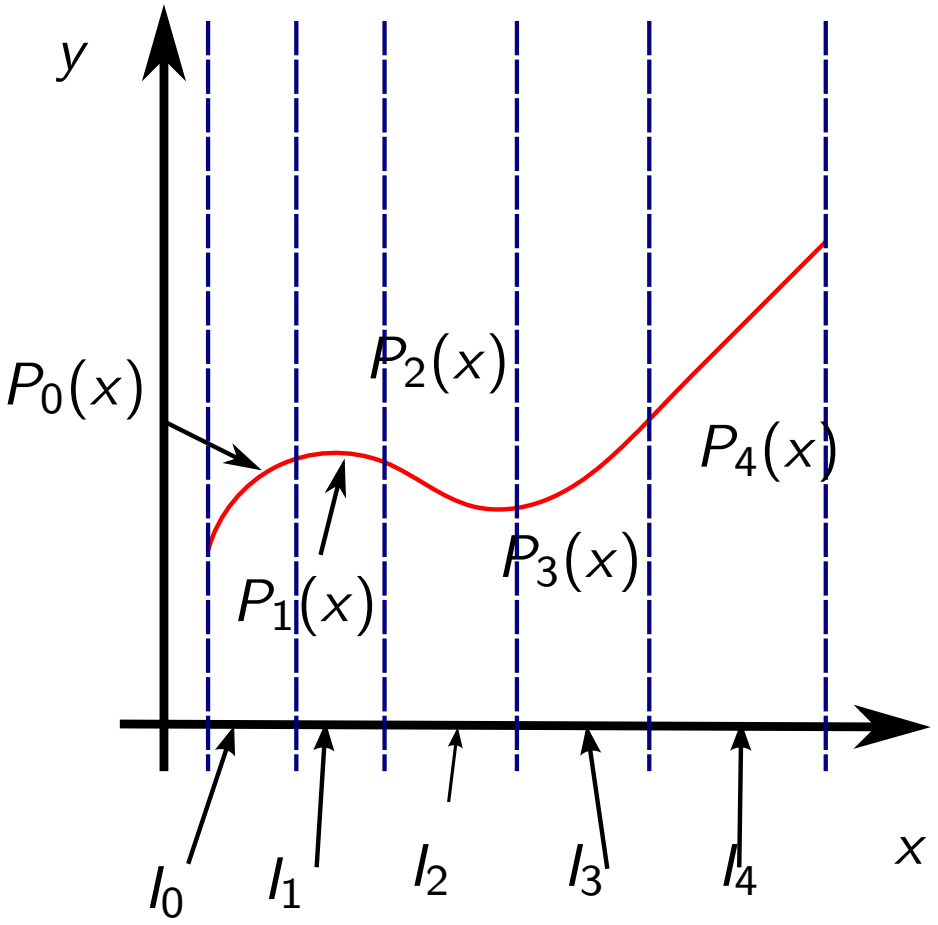
Mathematical Function Implementation



Outline

Problems in reconstruction

Determine the Corresponding Interval



Reconstruction Example

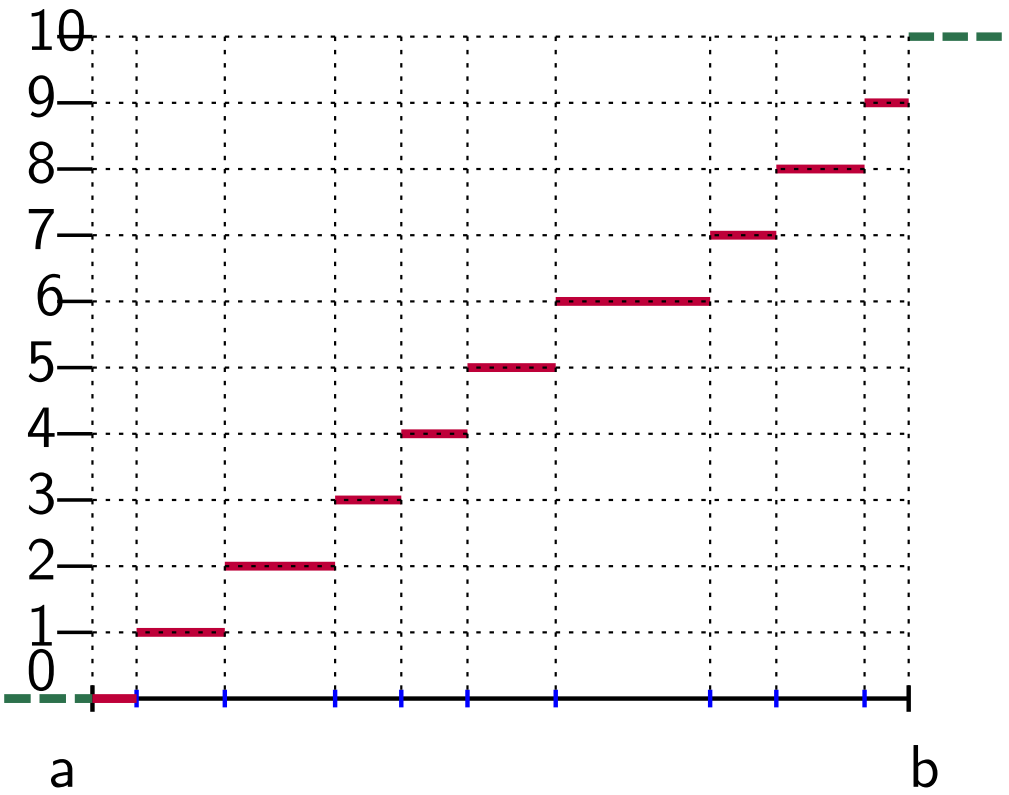
```
i=31;
if (x < arctan_table[i][A].d) i−= 16;
else i+=16;
if (x < arctan_table[i][A].d) i−= 8;
else i+= 8;
if (x < arctan_table[i][A].d) i−= 4;
else i+= 4;
if (x < arctan_table[i][A].d) i−= 2;
else i+= 2;
if (x < arctan_table[i][A].d) i−= 1;
else i+= 1;
if (x < arctan_table[i][A].d) i−= 1;
xmBihi = x−arctan_table[i][B].d;
xmBilo = 0.0;
```

Code sample for arctan function from `crlibm` library

Polynomial Approach

To determine the subinterval we build a mapping function

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

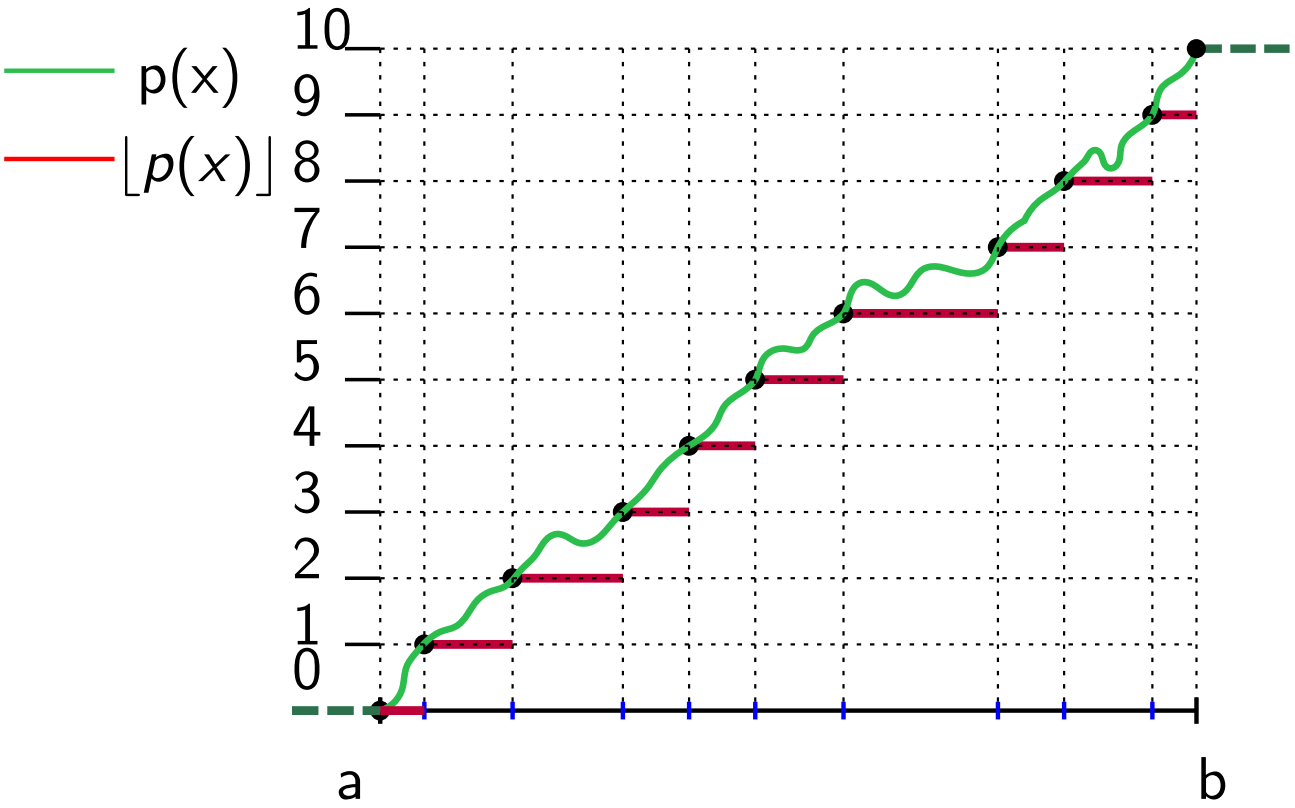


Polynomial Approach

To determine the subinterval we build a mapping function

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

$$p(x) : \lfloor p(x) \rfloor = P(x)$$

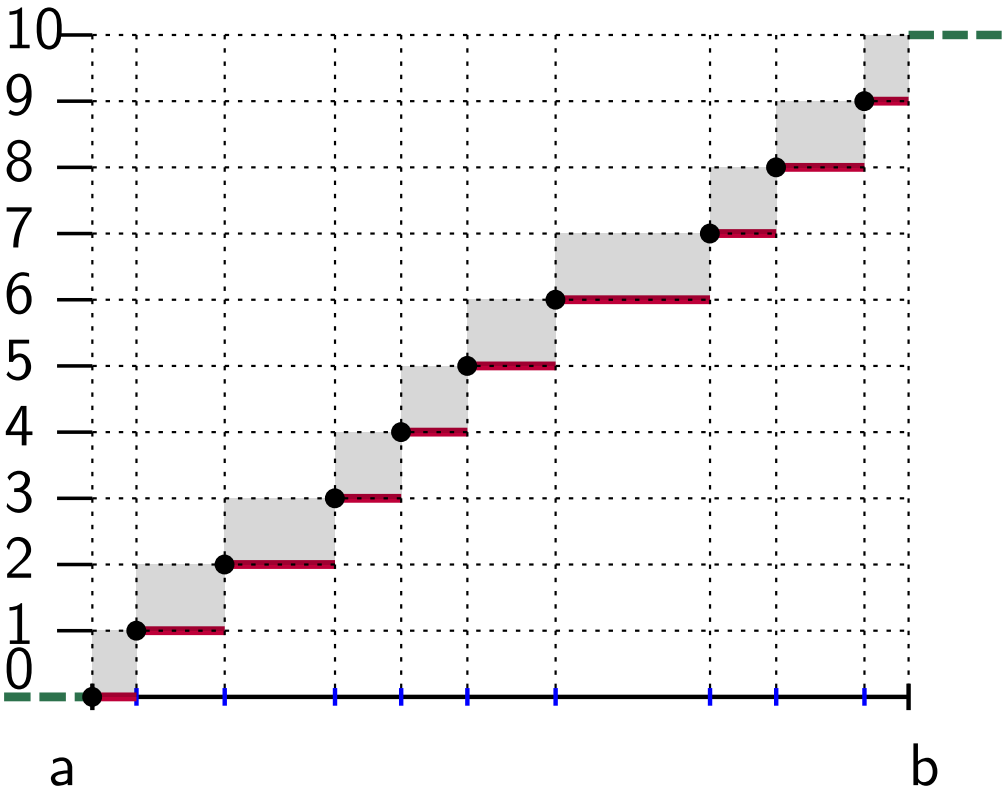


Polynomial Approach

To determine the subinterval we build a mapping function

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

$$p(x) : \lfloor p(x) \rfloor = P(x)$$



Classic interpolation problem:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^n \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

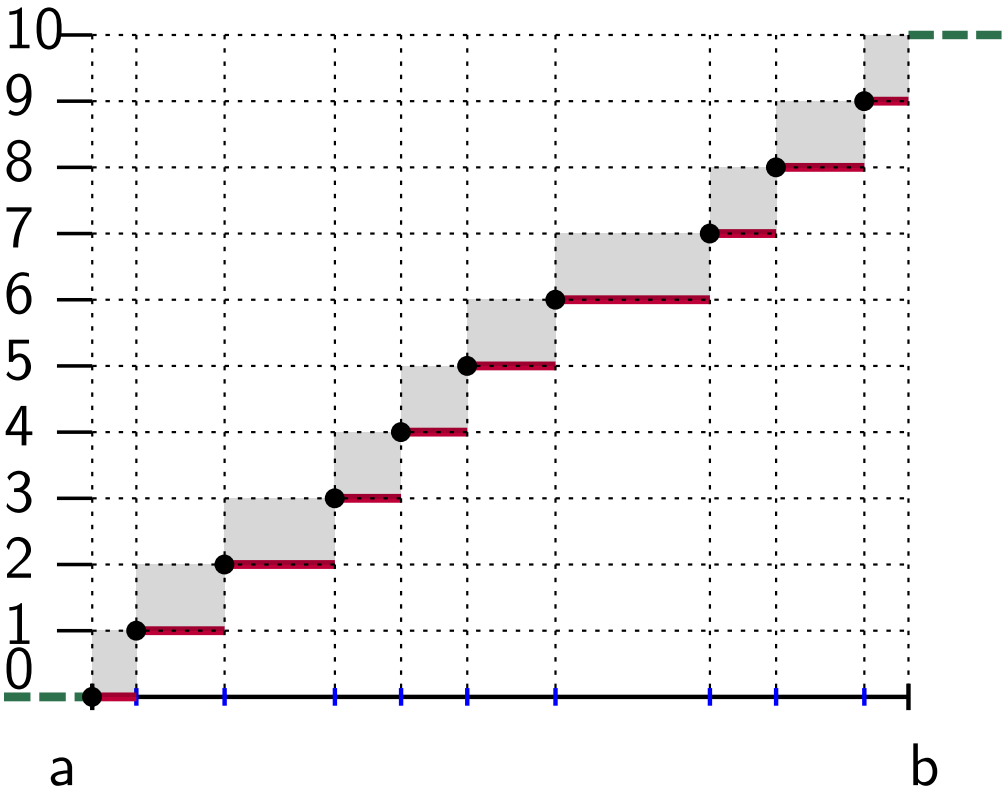
Linear system with Vandermonde matrix

Polynomial Approach

To determine the subinterval we build a mapping function

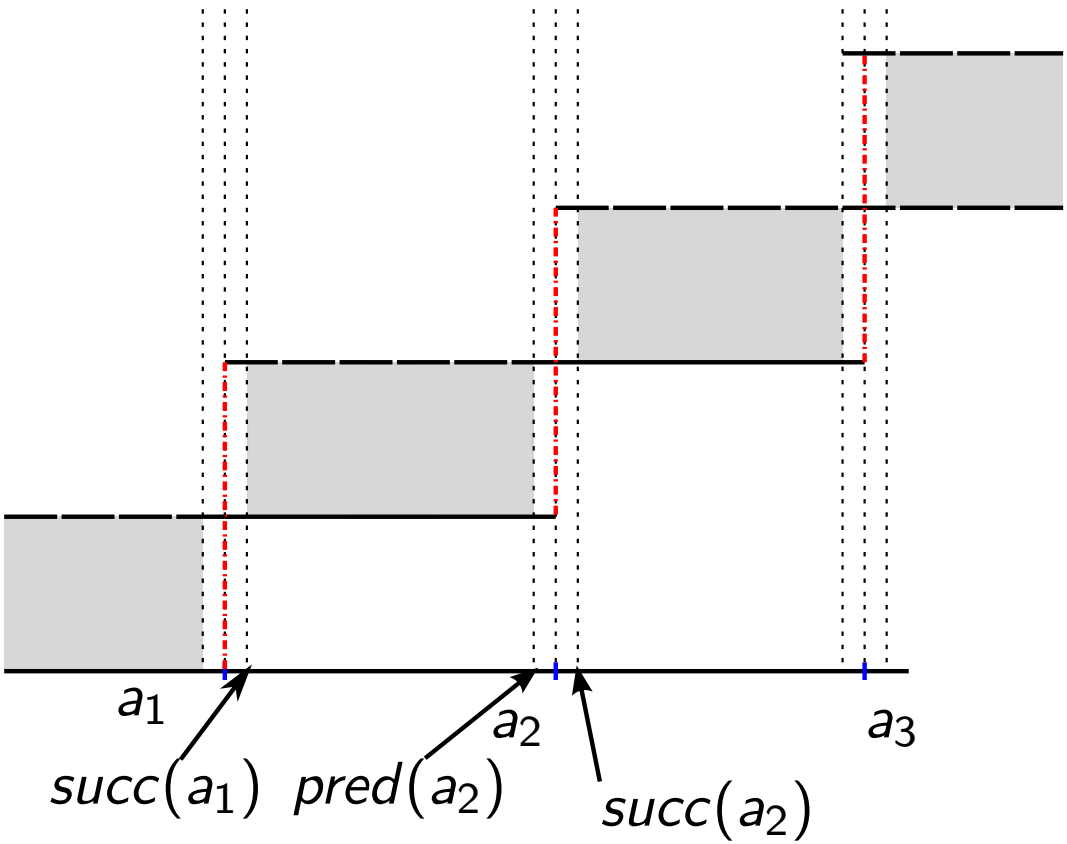
$$P(x) = k, x \in I_k$$

$$p(x) : \lfloor p(x) \rfloor = P(x)$$



a posteriori condition checking

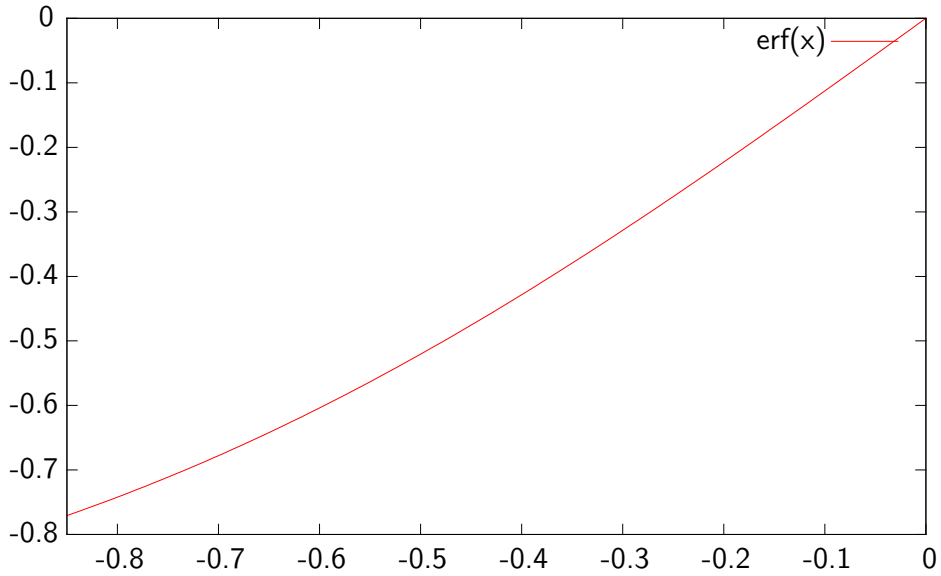
Taking FP into account



$p(x) \in [k, k+1)$, where $x \in [succ(a_k), pred(a_{k+1})] \in I_k$, $0 \leq k \leq n-1$

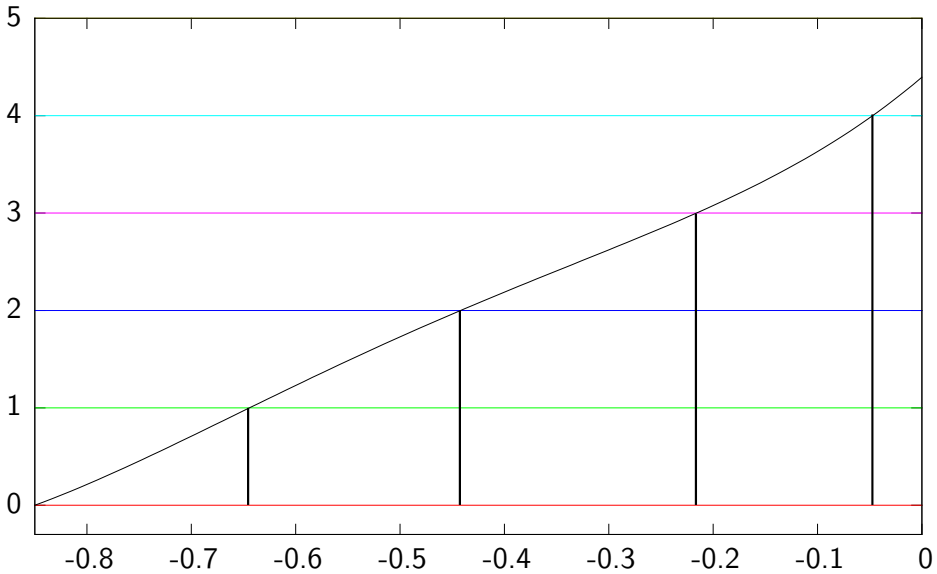
$p(x) \in [k-1, k+1)$, where $x = a_k$

Example of Polynomials



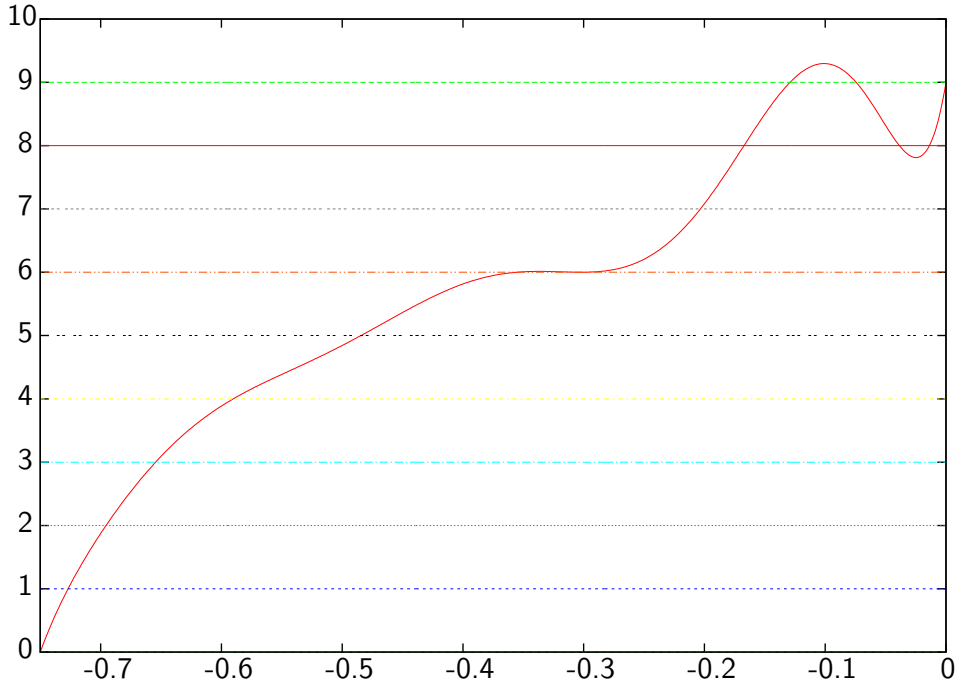
f=erf(x);
dom = [-0.85; 0];
$\bar{\varepsilon} = 2^{-55}$;
maxDegree = 10;

Example of Polynomials



f=erf(x);
dom = [-0.85; 0];
$\bar{\varepsilon} = 2^{-55}$;
maxDegree = 10;

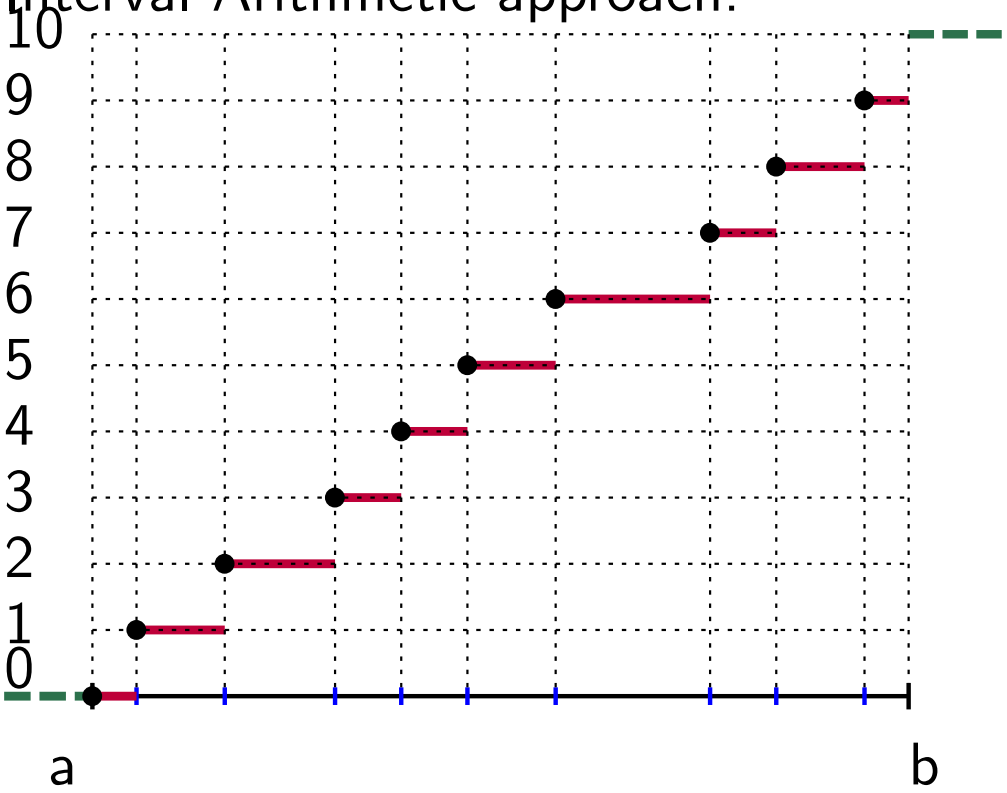
A Posteriori Condition Check Fails



$$\mathsf{f} = \mathsf{asin}(\mathsf{x});$$
$$\bar{\varepsilon} = 2^{-48};$$

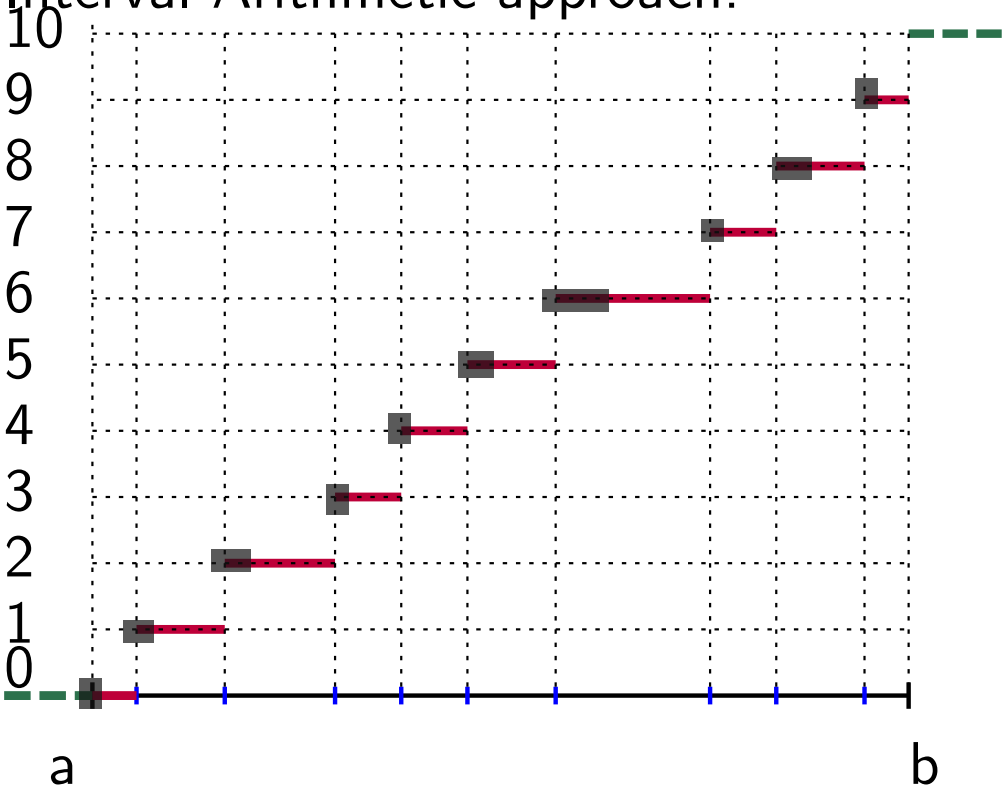
Towards an a priori Approach

Interval Arithmetic approach:



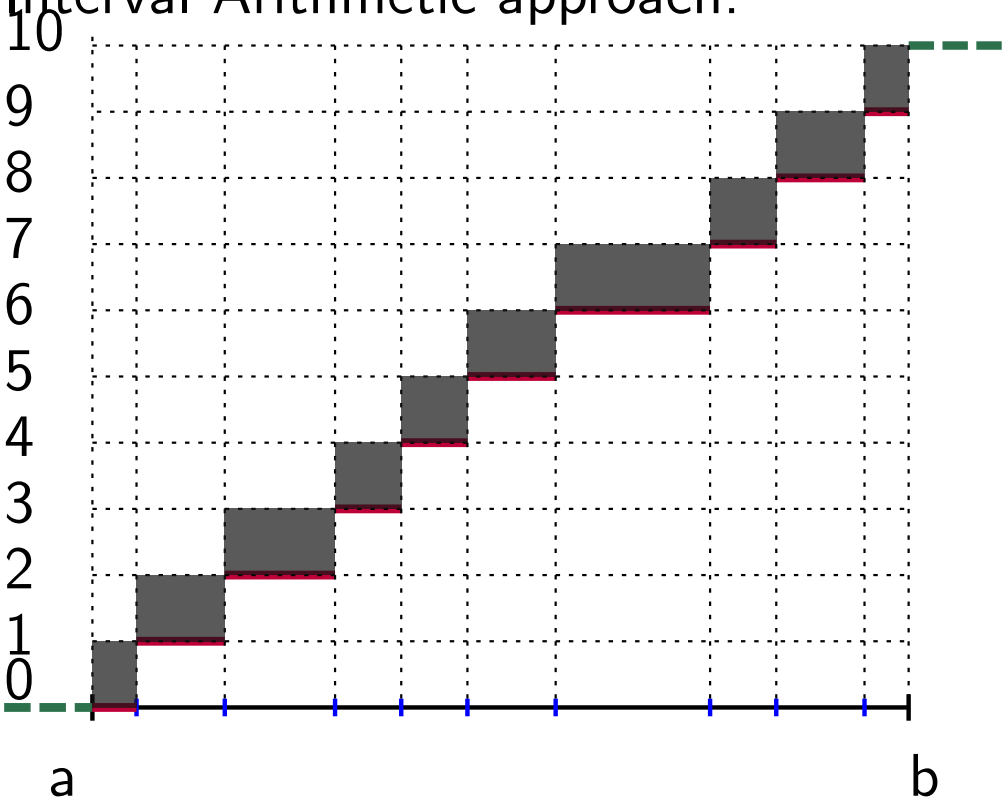
Towards an a priori Approach

Interval Arithmetic approach:



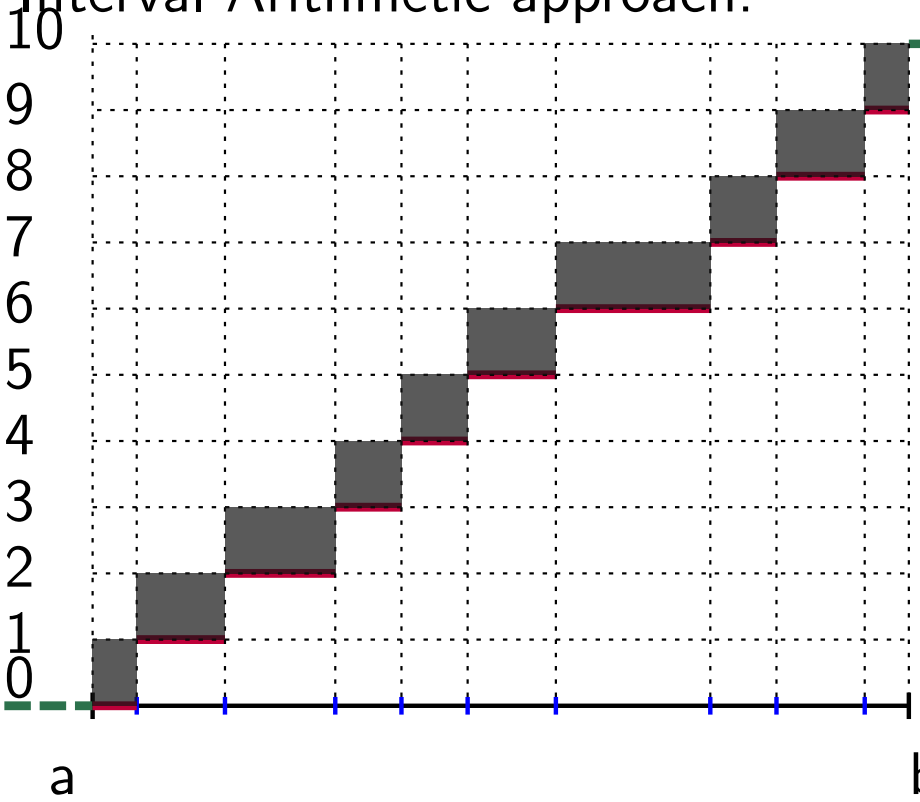
Towards an a priori Approach

Interval Arithmetic approach:



Towards an a priori Approach

Interval Arithmetic approach:



Interval algebraic problem

$$\begin{pmatrix} 1 & \mathbf{x_0} & \cdots & \mathbf{x_0^n} \\ 1 & \mathbf{x_1} & \cdots & \mathbf{x_1^n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{x_n} & \cdots & \mathbf{x_n^n} \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \mathbf{y_0} \\ \mathbf{y_1} \\ \vdots \\ \mathbf{y_n} \end{pmatrix}$$

$$\forall A \in \bar{A}, \exists b \in \bar{b} : \; Ax = b$$

Conclusions

- Polynomial approach allows to avoid branching
- Good start on generation on vectorizable implementations
- Works in $\sim 30\%$ cases
- *A posteriori* conditions $\rightarrow$ *a priori*

Q/A

Thank you for your attention!
Questions?