Up to this point.
We defined interval operations from a math point of view.

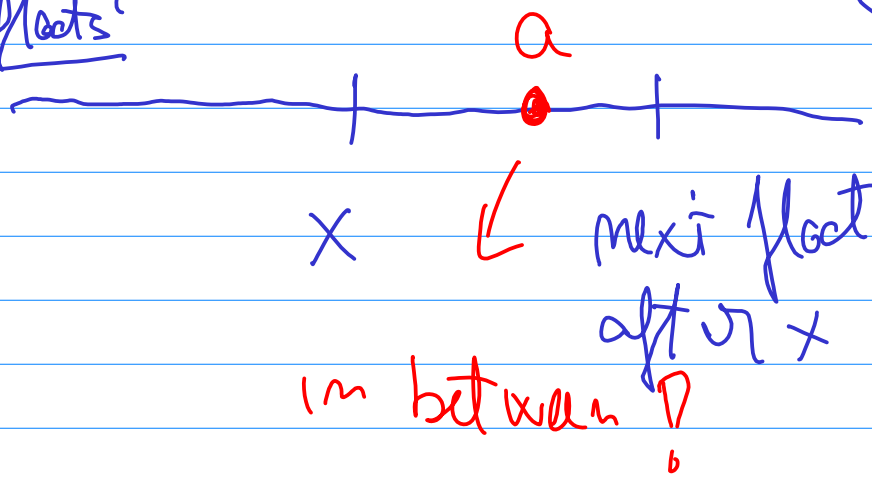
Addition $[a,b] + [c,d] = [a+c, b+d]$

Similarly: subtraction, mult, division, etc.

What problems can we still have numerically when working with float pt. endpoints?

Q1 Work with $[a,b]$, but one of a,b is not exact in fl. point?

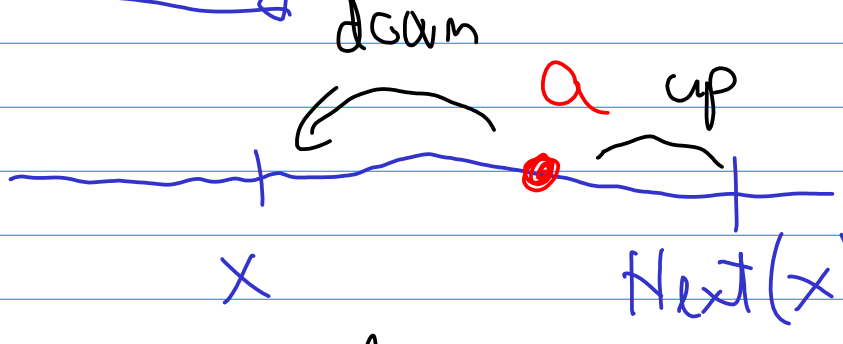
floats!



Q2 $[\ ,\ ] \bullet [\ ,\ ] = [A, B]$

Results endpoints are not exact floating pt. numbers?

Rounding - directed



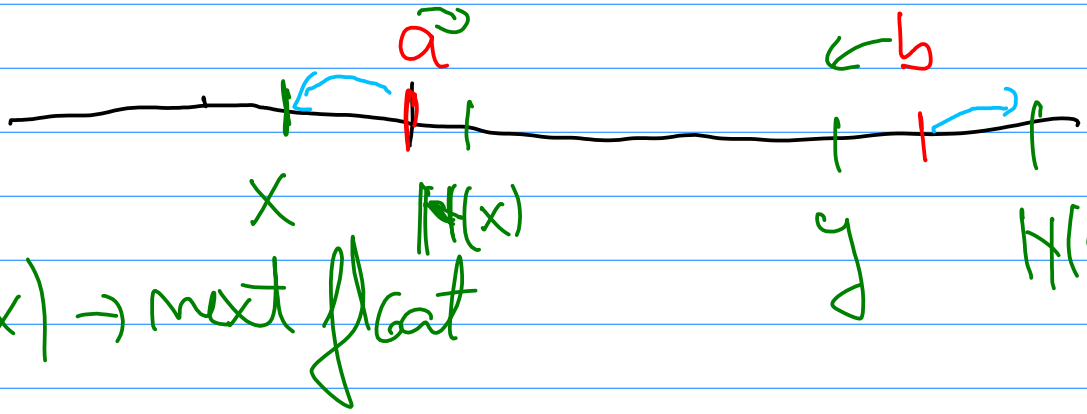
3 options: down(a) = x

up(a) = Next(x)

nearest(a) = nearest float to a

→ nearest float not useful for guaranteeing enclosure of the result.

Graphical examples



N(x) → next float

I = [a, b] → store it using fl. point numbers

Objective: stored interval must contain [a, b]

Nearest rounding:

[a, b] —stored→ [N(x), y]

smaller than [a, b] !

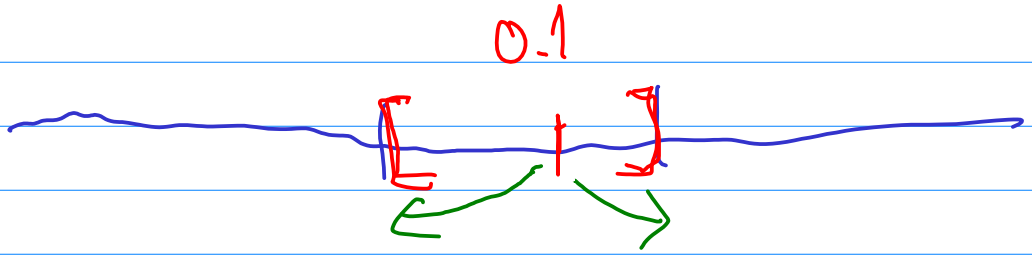
Directed rounding

[a, b] → [x, N(y)] → [down(a), up(b)]

Remember: $0.1 = \frac{1}{10}$ not exact in binary

$= \frac{1}{2^5}$

Work with 0.1 in a validated comp



Before doing anything: 0.1

$[down(0.1), up(0.1)]$

$z \in \mathbb{R}$ $down(z) \rightarrow$ biggest float $\leq z$

$up(z) \rightarrow$ smallest float $\geq z$

Question: Complexity of $down(z)$ and $up(z)$?

A binary point never equals $0.1 = \frac{1}{10}$

$\Rightarrow$ an interval $[b, b]$ never encloses $\frac{1}{10}$.

b → binary, fl. pt

0.1 enclosure $\rightarrow [down(0.1); up(0.1)]$

print(down(0.1)) $\rightarrow$ 0.1(- ...)

print(up(0.1)) $\rightarrow$ 0.1(+ ...)

Enclose input before computations!

$z \rightarrow [down(z), up(z)]$

Operations: the resulting interval for a math operation follows the same principle: lower endpoint ↘ rounded down

upper endpoint ↗ rounded up.

Addition: $X = [a,b]$, $Y = [c,d]$

$X+Y = [a+c, b+d] \subseteq [down(a+c), up(b+d)]$

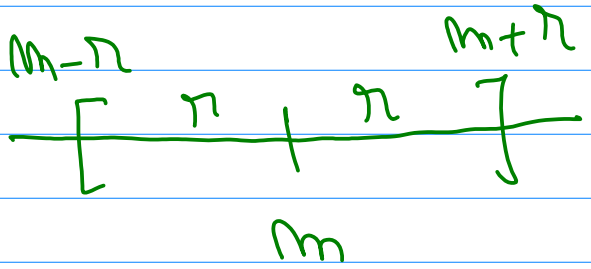
↗ inclusion/enclosure.

$X-Y \subseteq [down(a-d), up(b-c)]$

$0 \in [c,d] \Rightarrow \frac{1}{[c,d]} = \left[\frac{1}{d}, \frac{1}{c}\right]$

$$\frac{1}{[c,d]} \subseteq \left[\text{down}\left(\frac{1}{d}\right), \text{up}\left(\frac{1}{c}\right)\right]$$

Center, radius perspective



$$[m-r, m+r] + [n-s, n+s]$$

$$\overset{\text{exact}}{=} \left[m+n-(r+s),\ m+n+(r+s) \right]$$

If rounding is necessary:

Lower: $m+n \to \text{down}(m+n)$, $r+s \to \text{up}(r+s)$

Upper: $m+n \to \text{up}(m+n)$, $r+s \to \text{up}(r+s)$

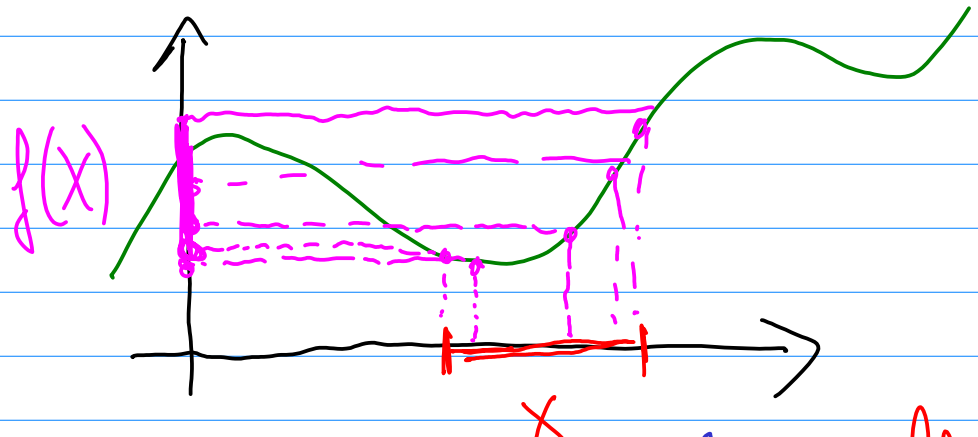
Solving an equation of the form $f(x) = 0$

Validated comp.

→ Find an enclosure for a 0.

. Find $[a, b]$ s.t. $\exists x \in [a, b], f(x) = 0$

→ Show that f is never 0 on $[a, b]$



$f(X) = \{f(x) : x \in X\}$ = all values taken by the function f on X.

Impractical $f(X)$ is not exact

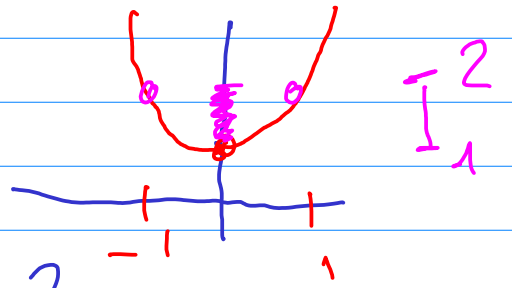
→ f is not "simple"

Instead we have an interval evaluation F such that $f(X) \subseteq F(X)$

$$f(X) \subseteq F(X)$$

math object ↙ ↘ algorithmic object.

Algorithmic object should contain the math object!



Example:

$X = [-1;1]$, $f(x) = x^2 + 1$

$$[1;2] = f(X) \subseteq X \cdot X + 1 = [-1;1] + 1 = [0;2]$$

$X \cdot X$ X interval — we don't assume equal numbers but in $x^2 = x \cdot x$ the two numbers are equal!

Sometimes more info. is required in the algorithm to get a TIGHTER enclosure.

It is true that $f(X) \subset [0,2]$

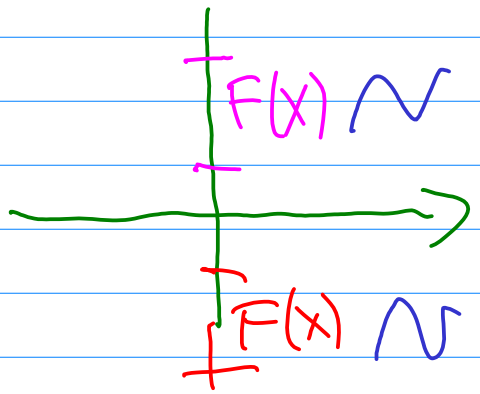
$f(X) \subset [1,2]$

The enclosure $f(X) \subset [1,2]$ shows that $\boxed{f > 0}$ on X.

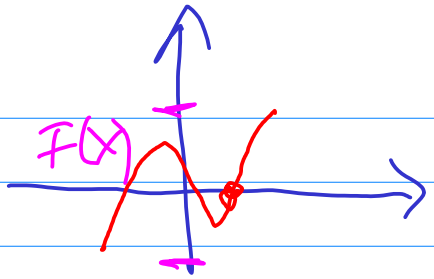
A numerical validated computation proves a math inequality.

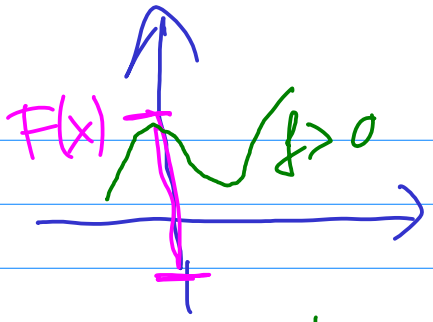
Two situations: $f(X) \subseteq F(X)$

if $0 \notin F(X) \Rightarrow f$ is never 0 on X



If $0 \in F(X)$ we cannot say anything concerning $\{f = 0\}$





F(X) may be too large and $0 \in F(X)$ may be irrelevant.

- Validated computing can prove inequalities (strict)
- Showing an equality through valid. computing is difficult.

Evaluation is inclusion isotonic if

$$X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$$

Example: one function 3 methods

$f(x) = x \cdot (1-x)$ on $[0,1]$

$$x - x^2 = x(1-x) = \frac{1}{4} - \left(x - \frac{1}{2}\right)^2$$

	Intervals	Enclosure
$x - x^2$	$[0,1] - [0,1]$	$[-1,1]$
$x(1-x)$	$[0,1] \cdot [0,1]$	$[0;1]$
$\frac{1}{4} - (x-\frac{1}{2})^2$ $\in [-\frac{1}{2};\frac{1}{2}]$	$\frac{1}{4} - [0,\frac{1}{4}]$	$[0,\frac{1}{4}]$

Best = Smallest

Real answer

