Rounding error analysis

Old and nontrivial question [von Neumann, Turing, Wilkinson, ...]

In this lecture, two approaches:

A priori analysis: → Claude-Pierre's lecture

- Goal: bound on $\|\hat{x} - x\|/\|x\|$ for any input and format
- Tool: the many nice properties of floating-point
- Ideal: readable, provably tight bound + short proof

A posteriori, automatic analysis: → this lecture

- Goal: $\hat{x}$ and enclosure of $\hat{x} - x$ for given input and format
- Tool: interval arithmetic based on floating-point
- Ideal: a narrow interval computed fast

Interval arithmetic: implementation using floating-point arithmetic

Implementation using floating-point arithmetic:
use directed rounding modes (cf. IEEE 754 standard)

$$\sqrt{[2,3]} = [\mathrm{RD}(\sqrt{2}), \mathrm{RU}(\sqrt{3})]$$

Advantage: every result is guaranteed, in the sense that the exact, unknown result, belongs to the computed interval result.

Agenda

Agenda

A brief introduction

Interval arithmetic: replace numbers by intervals and compute.

Fundamental theorem of interval arithmetic:
(or "Thou shalt not lie"):
the exact result (number or set) is contained in the computed interval.

No result is lost, the computed interval is guaranteed to contain every possible result.

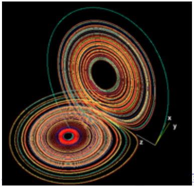
A brief introduction

Interval arithmetic: replace numbers by intervals and compute.
Initially: introduced to take into account roundoff errors (Moore 1966)
and also uncertainties (on the physical data...).
Later: computations "in the large", computations with sets.

Interval analysis: develop algorithms for **reliable (or verified, or guaranteed, or certified) computing**,
that are suited for interval arithmetic,
i.e. different from the algorithms from classical numerical analysis.

A brief introduction: examples of applications

- control the roundoff errors, cf. computational geometry
- solve several problems with verified solutions: linear and nonlinear systems of equations and inequalities, constraints satisfaction, (non/convex, un/constrained) global optimization, integrate ODEs e.g. particules trajectories...
- mathematical proofs: cf. Hales' proof of Kepler's conjecture or Tucker's proof that Lorenz system has a strange attractor or Helfgott's proof of the ternary Goldbach conjecture.



Agenda

A brief introduction

Interval arithmetic: replace numbers by intervals and compute.

Interval: closed connected subset of $\mathbb{R}$.

$\emptyset$, $[-1, 3]$, $]-\infty, 2]$, $[5, +\infty[$ and $\mathbb{R}$ are intervals.

$]-1, 3]$, $]0, +\infty[$ ou $[1, 2] \cup [3, 4]$ are not intervals.

Definitions: operations

Fundamental theorem of interval arithmetic:
(or "Thou shalt not lie"):
the exact result (number or set) is contained in the computed interval.

No result is lost, the computed interval is guaranteed to contain every possible result.

Definitions: operations

$\boldsymbol{x} \diamond \boldsymbol{y} = \textbf{Hull}\{x \diamond y \;:\; x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$

Arithmetic and algebraic operations: use the monotonicity

$$
\begin{array}{rcl}
[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] & = & [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] & = & [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] & = & [\min(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y}), \max(\text{ibid.})] \\
[\underline{x}, \overline{x}]^2 & = & [\min(\underline{x}^2, \overline{x}^2), \max(\underline{x}^2, \overline{x}^2)] \text{ if } 0 \notin [\underline{x}, \overline{x}] \\
 & & [0, \max(\underline{x}^2, \overline{x}^2)] \text{ otherwise}
\end{array}
$$

Interval arithmetic: implementation using floating-point arithmetic

Implementation using floating-point arithmetic:
use directed rounding modes (cf. IEEE 754 standard)

$$\sqrt{[2,3]} = [\mathrm{RD}(\sqrt{2}), \mathrm{RU}(\sqrt{3})]$$

Advantage: every result is guaranteed, in the sense that the exact, unknown result, belongs to the computed interval result.

Operations

Algebraic properties: associativity, commutativity hold, some are lost:

- subtraction is not the inverse of addition, in particular $\boldsymbol{x} - \boldsymbol{x} \neq [0]$
- division is not the inverse of multiplication
- squaring is tighter than multiplication by oneself
- multiplication is only sub-distributive wrt addition
- with floating-point implementation, operations are not associative either

Influence of the expression: first example

$$[1,1]+[2^{100},2^{100}]-[2^{100},2^{100}]?$$

With these parentheses:

$$([1,1]+[2^{100},2^{100}])-[2^{100},2^{100}]=[2^{100},\mathrm{succ}(2^{100})]-[2^{100},2^{100}]=[0,\mathrm{ulp}(2^{100})].$$

With those parentheses:

$$[1,1]+([2^{100},2^{100}]-[2^{100},2^{100}])=[1,1]+[0,0]=[1,1].$$

Both include the results, one is more accurate than the other...

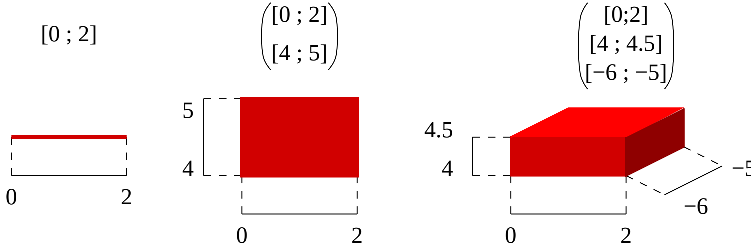
Moral lesson: interval results are always guaranteed to include the exact result, whatever the chosen expression. However their accuracy strongly depends on the chosen expression, on the order of operations.

Agenda

Definitions: intervals, vectors, matrices

Objects:

- intervals of real numbers = closed connected sets of $\mathbb{R}$
  - interval for π: $[3.14159, 3.14160]$
  - data d measured with an absolute error less than $\pm\varepsilon$: $[d-\varepsilon, d+\varepsilon]$
- interval vector: components = intervals; also called *box*



- interval matrix: components = intervals.

Agenda

Definitions: comparisons

How to compare two intervals?

how to compare $[-1, 2]$ and $[0, 3]$? or $[-1, 2]$ and $[0, 1]$?

Several approaches:

- use explicit names: CertainlyLess, PossiblyLess
- use trivalued logic (MPFI): $\boldsymbol{a} < \boldsymbol{b}$ returns
  - -1 if every element of $\boldsymbol{a}$ is $<$ than every element of $\boldsymbol{b}$,
  - $+1$ if every element of $\boldsymbol{a}$ is $>$ than every element of $\boldsymbol{b}$,
  - 0 if $\boldsymbol{a}$ and $\boldsymbol{b}$ overlap.
- use many more relation names, cf. IEEE 1788.

IEEE-1788 standard: comparison relations

- **7 relations:** equal ($=$), subset ($\subset$), less than or equal to ($\leq$), precedes or touches ($\preceq$), interior to, less than ($<$), precedes ($\prec$).

IEEE-1788 standard: comparison relations

- **7 relations:** equal ($=$), subset ($\subset$), less than or equal to ($\leq$), precedes or touches ($\preceq$), interior to, less than ($<$), precedes ($\prec$).
- **Interval overlapping relations:** before, meets, overlaps, starts, containedBy, finishes, equal, finishedBy, contains, startedBy, overlappedBy, metBy, after.

  Again, relations defined by conditions on the bounds.

Agenda

Definitions: function extension

Definition:
an interval extension $\boldsymbol{f}$ of a function f satisfies

$$\forall \boldsymbol{x},\ f(\boldsymbol{x}) \subset \boldsymbol{f}(\boldsymbol{x}),\ \text{and } \forall x,\ f(\{x\}) = \boldsymbol{f}(\{x\}).$$

Elementary functions: again, use the monotony.

$$\begin{array}{lcl}
\exp \boldsymbol{x} & = & [\exp \underline{x}, \exp \overline{x}] \\
\log \boldsymbol{x} & = & [\log \underline{x}, \log \overline{x}] \ \text{ if } \underline{x} \geq 0, [-\infty, \log \overline{x}] \ \text{ if } \overline{x} > 0 \\
\sin[\pi/6, 2\pi/3] & = & [1/2, 1] \\
\ldots & &
\end{array}$$

Definitions: function extension

Example: $f(x) = x^2 - x + 1$ with $x \in [-2, 1]$.
$[-2, 1]^2 - [-2, 1] + 1 = [0, 4] + [-1, 2] + 1 = [0, 7]$.
Since $x^2 - x + 1 = x(x - 1) + 1$, we get $[-2, 1] \cdot ([-2, 1] - 1) + 1 = [-2, 1] \cdot [-3, 0] + 1 = [-3, 6] + 1 = [-2, 7]$.
Since $x^2 - x + 1 = (x - 1/2)^2 + 3/4$, we get $([-2, 1] - 1/2)^2 + 3/4 = [-5/2, 1/2]^2 + 3/4 = [0, 25/4] + 3/4 = [3/4, 7] = f([-2, 1])$.

Problem with this definition: infinitely many interval extensions, syntactic use (instead of semantic).

How to choose the best extension? How to choose a good one?

Definitions: function extension

Mean value theorem of order 1 (Taylor expansion of order 1):
$\forall x, \forall y, \exists \xi_{x,y} \in (x,y) \ : \ f(y) = f(x) + (y-x)\cdot f'(\xi_{x,y})$
Interval interpretation:
$\forall y \in \boldsymbol{x}, \forall \tilde{x} \in \boldsymbol{x}, \ f(y) \in f(\tilde{x}) + (y-\tilde{x})\cdot \boldsymbol{f'}(\boldsymbol{x})$
$\Rightarrow f(\boldsymbol{x}) \subset f(\tilde{x}) + (\boldsymbol{x}-\tilde{x})\cdot \boldsymbol{f'}(\boldsymbol{x})$

Mean value theorem of order 2 (Taylor expansion of order 2):
$\forall x, \forall y, \exists \xi_{x,y} \in (x,y) : f(y) = f(x) + (y-x)\cdot f'(x) + \frac{(y-x)^2}{2}\cdot f''(\xi_{x,y})$
Interval interpretation:
$\forall y \in \boldsymbol{x}, \forall \tilde{x} \in \boldsymbol{x}, \ f(y) \in f(\tilde{x}) + (y-\tilde{x})\cdot f'(\tilde{x}) + \frac{(y-\tilde{x})^2}{2}\cdot \boldsymbol{f''}(\boldsymbol{x})$
$\Rightarrow f(\boldsymbol{x}) \subset f(\tilde{x}) + (\boldsymbol{x}-\tilde{x})\cdot f'(\tilde{x}) + \frac{(\boldsymbol{x}-\tilde{x})^2}{2}\cdot \boldsymbol{f''}(\boldsymbol{x})$.

Definitions: function extension

No need to go further:

- it is difficult to compute (automatically) the derivatives of higher order,
  especially for multivariate functions;
- there is no (theoretical) gain in quality.

Theorem:

- for the natural extension $\boldsymbol{f}$ of f, it holds
  $d(f(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x})) \leq \mathcal{O}(w(\boldsymbol{x}))$
- for the first order Taylor extension $\boldsymbol{f_{T_1}}$ of f, it holds
  $d(f(\boldsymbol{x}), \boldsymbol{f_{T_1}}(\boldsymbol{x})) \leq \mathcal{O}(w(\boldsymbol{x})^2)$
- getting an order higher than 3 is impossible without the squaring operation, is difficult even with it...

Agenda

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals
- **1946:** Alan Turing in his project for NPL (cited by Wilkinson)

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals
- **1946:** Alan Turing in his project for NPL (cited by Wilkinson)
- **1931:** Rosalind Cecil Young in her PhD thesis in Cambridge (UK) has used some formulas

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals
- **1946:** Alan Turing in his project for NPL (cited by Wilkinson)
- **1931:** Rosalind Cecil Young in her PhD thesis in Cambridge (UK) has used some formulas
- **1927:** Bradis, for positive quantities, in Russian

Historical remarks

Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals
- **1946:** Alan Turing in his project for NPL (cited by Wilkinson)
- **1931:** Rosalind Cecil Young in her PhD thesis in Cambridge (UK) has used some formulas
- **1927:** Bradis, for positive quantities, in Russian
- **1908:** Young, for some bounded functions, in Italian

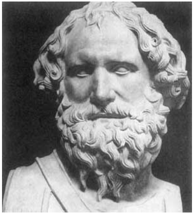
Historical remarks

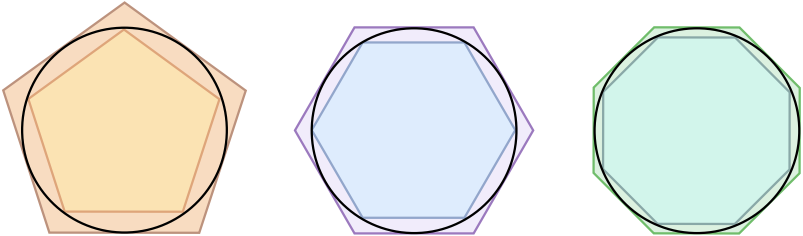
Who invented Interval Arithmetic?

- **1962:** Ramon Moore defines IA in his PhD thesis and then a rather exhaustive study of IA in 1966 – Kantorovich in Russian
- **1958:** Tsunaga, in his MSc thesis in Japanese
- **1956:** Warmus
- **1951:** Dwyer, in the specific case of closed intervals
- **1946:** Alan Turing in his project for NPL (cited by Wilkinson)
- **1931:** Rosalind Cecil Young in her PhD thesis in Cambridge (UK) has used some formulas
- **1927:** Bradis, for positive quantities, in Russian
- **1908:** Young, for some bounded functions, in Italian
- **3rd century BC:** Archimedes, to compute an enclosure of π!

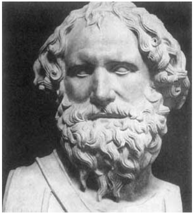
Cf. http://www.cs.utep.edu/interval-comp/, click on *Early papers by Others*.

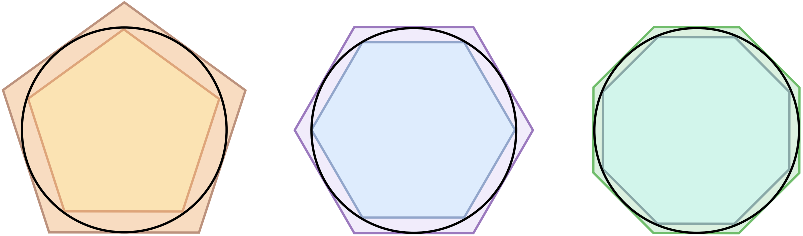
Archimedes and an enclosure of π





Archimedes and an enclosure of π





$$3 + \frac{10}{71} \simeq 3.1408 \leq \pi \leq 3 + \frac{1}{7} \simeq 3.1429.$$

Historical remarks

Childhood until the seventies.

Popularization in the 1980, German school (U. Kulisch).

IEEE-754 standard for floating-point arithmetic in 1985: directed roundings are standardized and available (?).

Since the nineties: interval **algorithms**.

IEEE-1788 standard for interval arithmetic in 2015.

Agenda

Agenda

Cons: overestimation (1/2)

The result encloses the true result, but it is too large:
overestimation phenomenon.
Two main sources: variable dependency and wrapping effect.

(Loss of) Variable dependency:

$$\boldsymbol{x} - \boldsymbol{x} = \{x - y \ : \ x \in \boldsymbol{x}, y \in \boldsymbol{x}\} \neq \{x - x \ : \ x \in \boldsymbol{x}\} = \{0\}.$$

Cons: overestimation (2/2)

Wrapping effect

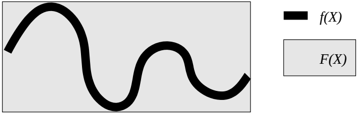
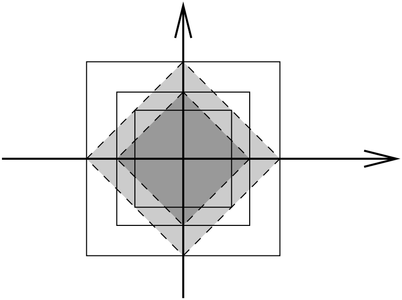


image of $f(\boldsymbol{x})$
with $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$



2 successive rotations of $\pi/4$
of the little central square

Cons: complexity

Complexity: most problems are NP-hard (Gaganov, Rohn, Kreinovich. . .)

- evaluate a function on a box. . . even up to ε
- solve a linear system. . . even up to $1/4n^4$
- determine if the solution of a linear system is bounded

Cons: efficiency (1/3)

Efficiency

Implementation using floating-point arithmetic:
use directed roundings, towards $\pm\infty$.

Programming languages did not give access to the rounding modes ($\rightarrow$ asm).

Cons: efficiency (2/3)

Efficiency

Overhead in execution time:

- in theory, at most 4, or 8, cf.

$$[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = [\quad \min(RD(\underline{x} \times \underline{y}), RD(\underline{x} \times \overline{y}), RD(\overline{x} \times \underline{y}), RD(\overline{x} \times \overline{y})),\\ \max(RU(\underline{x} \times \underline{y}), RU(\underline{x} \times \overline{y}), RU(\overline{x} \times \underline{y}), RU(\overline{x} \times \overline{y}))]$$

Cons: efficiency (3/3)

Efficiency

Overhead in execution time:

- in practice, around 20: changing the rounding modes implies flushing the pipelines (on most architectures and implementations),
- or even up to 100 or to 1000, when compared to highly optimized codes such as BLAS,
- but less and less so, with new architectures and static rounding modes: GPU, Xeon Phi Knights Landing (Skylake and Cannonlake).

Not to mention issues related to multithreading. . .

Agenda

Pros: automatic bounds

Cf. Octave interval:

- once the cornercases are determined,
- very easy to get bounds: just plug in interval arithmetic.

Reminder: for Kahan's algorithm, Claude-Pierre could establish

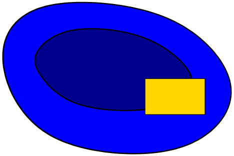
$$\frac{|\hat{r} - r|/|r|}{2u} = \frac{1}{1+2u} = 1 - 2u + O(u^2).$$

With a dozen lines of code, it was possible to establish that

$$\frac{(\boldsymbol{r} - r)/r}{2u} \in [0, 3] \text{ for the naive method.}$$

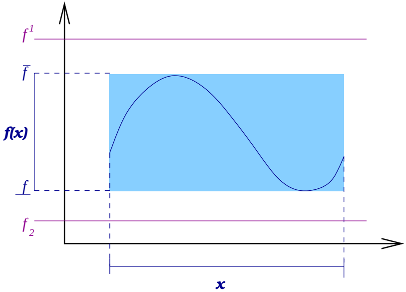
Pros: set computing

Behaviour safe? controllable? dangerous?



always controllable.

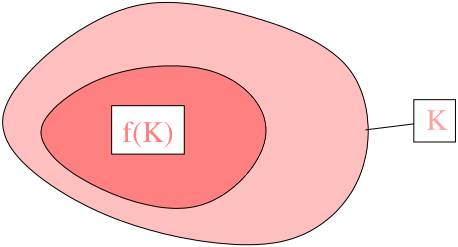
On $\boldsymbol{x}$, are the extrema of the function f $> f^1$, $< f_2$?



No if $f(\boldsymbol{x}) = [\underline{f}, \overline{f}] \subset [f_2, f^1]$.

Pros: Brouwer-Schauder theorem

A function f which is continuous on the unit ball B and which satisfies $f(B) \subset B$ has a fixed point on B.



The theorem remains valid if B is replaced by a compact K and in particular an interval.

Agenda

Reference for this section

W. Kahan: *How Futile is Mindless Assessment of Roundoff in Floating-Point Computation?*, 2006.

Agenda

Five approaches detailed in Kahan's paper

1. Repeat the computation in arithmetics of increasing precision, increase it until as many as desired of the results' digits agree.
2. Repeat the computation in arithmetic of the same precision but rounded differently, say *Down*, and then *Up*, and maybe *Towards Zero* too, besides *To Nearest*, and compare three or four results.
3. Repeat the computation a few times in arithmetic of the same precision rounding operations randomly, some *Up*, some *Down*, and treat results statistically.
4. Repeat the computation a few times in arithmetic of the same precision but with slightly different input data each time, and see how widely results spread.
5. Perform the computation in *Significance Arithmetic*, or in *Interval Arithmetic*.

The mindless use of these approaches is qualified as "futile" by Kahan.

Multiple Precision Interval Arithmetic

Almost foolproof is extendable-precision Interval Arithmetic.
Let's be almost foolproof: let's use MPFI today.

What is MPFI?

- based on MPFR library: arbitrary precision:
- MPFR stands for *Multiple Precision Reliable Floating-point library*:
- MPFI stands for *Multiple Precision reliable Floating-point Interval library*:
- the computing precision of each operation can be specified:
- no limit apart from the memory of your computer.

Influence of the computing precision (1/2)

Influence on one value:

$$t \notin \mathbb{F} \quad \Rightarrow \quad t \in \left[\mathrm{RD}(t), \mathrm{RU}(t)\right] \quad \Rightarrow \quad \mathrm{RU}(t) - \mathrm{RD}(t) \leq 2u|t|.$$

Influence on one interval operation: the overestimation of the result is proportional to 4 ulp:
$\mathrm{w}(\widehat{\boldsymbol{x} \,\mathrm{op}\, \boldsymbol{y}}) - \mathrm{w}(\boldsymbol{x} \,\mathrm{op}\, \boldsymbol{y}) \leq 4u|\boldsymbol{x} \,\mathrm{op}\, \boldsymbol{y}|)$.

Influence on an interval computation: theoretically, the overestimation of the result is proportional to the ulp:
$\mathrm{w}(\hat{\boldsymbol{x}}) - \mathrm{w}(\boldsymbol{x}) = \mathcal{O}(2^{-p}|\boldsymbol{x}|)$ where p is the computing precision.

DEMO

Influence of the computing precision (2/2)

Influence on an interval computation: in practice,

- use the midpoint-radius representation for thin intervals: the radius accounts for roundoff errors,
- use iterative refinement to reduce the width,
- use higher precision for critical intermediate computations (residual) to hide the effect of the computing precision,

and get $\mathrm{w}(\hat{\boldsymbol{x}}) - \mathrm{w}(\boldsymbol{x}) \simeq 2^{-p}|\boldsymbol{x}|$, i.e. the best possible result.

Examples: linear systems solving, Newton iteration.

Higher precision: extended / arbitrary

Extended precision (double-double, triple-double): (Moler, Priest, Dekker, Knuth, Shewchuk, Bailey. . .)
a number is represented as the sum of 2 (or 3 or . . .) floating-point numbers. Do not evaluate the sum using floating-point arithmetic!
Double-double arith. is implemented using IEEE-754 FP arith.

Arbitrary precision: the precision is chosen by the user, the only limit being the computer's memory.
Arithmetic is implemented in software, e.g. MPFR (**Zimmermann et al.**), MPFI (**Revol, Rouillier et al.), (Yamamoto, Krämer et al.**).

Tradeoff between accuracy and efficiency (and memory):
double-double: accuracy "$\times 2$", ≤ 1 order of magnitude slower
arbitrary prec.: accuracy "∞", $\geq$ 1-2 order of magnitude slower
(provided Higham's rule of thumb applies).

Agenda

Mid-rad representation of intervals

Use of mid-rad representation: better suited for this purpose, as the midpoint corresponds to the floating-point value and the radius accounts for roundoff errors.

with usual precision (floating-point arithmetic available on the processor), cf. IntLab library:
efficient, often does the job.

with arbitrary precision (cf. ARB or Mathemagix library):
for the midpoint and much less precision for the radius.

Mid-rad representations of intervals: operations

Addition:

$$\begin{aligned} z_m &= \mathrm{RN}(x_m + y_m) \text{ and} \\ z_r &= \mathrm{RU}(x_r + y_r + u \cdot |z_m|). \end{aligned}$$

Multiplication:

$$\begin{aligned} z_m &= \mathrm{RN}(x_m \cdot y_m) \text{ and} \\ z_r &= \mathrm{RU}((|x_m| + x_r) \cdot y_r + x_r \cdot |y_m| + u \cdot |z_m|). \end{aligned}$$

Mid-rad representations of intervals: operations in practice

To avoid costly changes of the rounding modes, use RN and inflate the radii.

Addition:

$$\begin{aligned} z_m &= \mathrm{RN}(x_m + y_m) \text{ and} \\ z_r &= \mathrm{RN}((1+4u)\cdot(x_r + y_r + u\cdot|z_m|)). \end{aligned}$$

Multiplication:

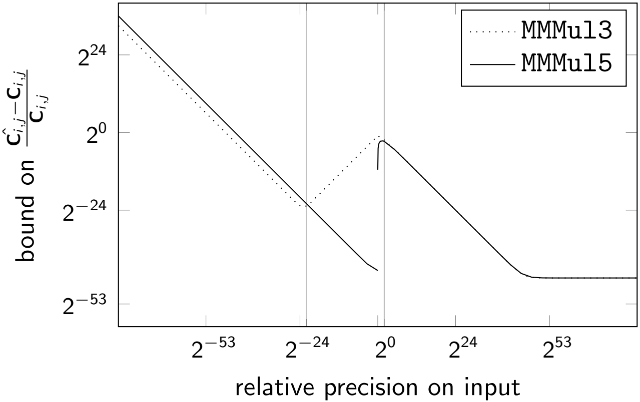
$$\begin{aligned} z_m &= \mathrm{RN}(x_m \cdot y_m) \text{ and} \\ z_r &= \mathrm{RN}((1+4u)\cdot(((|x_m| + x_r)\cdot y_r + x_r\cdot|y_m| + u\cdot|z_m|)). \end{aligned}$$

Mid-rad representations of intervals: efficiency

Efficiency of algorithms using the mid-rad representation:

- **matrix product** within a factor 3 compared to MKL, on a multicore
  (PhD thesis of Philippe Théveny, 2014);
- **linear system solving** within a factor 15 compared to MatLab
  (PhD thesis of Hong Diep Nguyen, 2011).

Mid-rad representation and roundoff errors



Matrix product $\boldsymbol{A} \cdot \boldsymbol{B}$ of size 128×128 with interval coefficients.

For $\boldsymbol{A}$: $\frac{A_{r\,i,j}}{|A_{m\,i,j}|} \leq e$ and e is reached. Ibid for $\boldsymbol{B}$.

Agenda

Affine arithmetic Comba, Stolfi and Figueiredo – Fluctuat

Definition: each input or computed quantity x is represented by
$x = x_0 + \alpha_1\varepsilon_1 + \alpha_2\varepsilon_2 + \cdots + \alpha_n\varepsilon_n$
where x_0, α_1, $\ldots \alpha_n$ are known real / floating-point numbers,
and ε_1, $\varepsilon_2 \ldots \varepsilon_n$ are symbolic variables $\in [-1, +1]$.
Example: $x \in [3, 7]$ is represented by $x = 5 + 2\varepsilon$.

Operations:
$(x + \sum_k \alpha_k\varepsilon_k) + (y + \sum_k \beta_k\varepsilon_k) = (x + y) + \sum_k(\alpha_k + \beta_k)\varepsilon_k$.
$(x+\sum_k \alpha_k\varepsilon_k)\times(y+\sum_k \beta_k\varepsilon_k) = (x\times y)+\sum_k(x\beta_k+y\alpha_k)\varepsilon_k+\gamma_l\varepsilon_l$
with ε_l a new variable.

Roundoff errors: compute δ_l an upper bound of all roundoff errors and add it to γ_l.

Taylor models, polynomial models

Berz, Hoefkens and Makino 1998, Nedialkov, Neher

Principle: represent a function $f(x)$ for $x \in [-1, 1]$ by a polynomial part $p(x)$ and a remainder part (a big bin) $\boldsymbol{I}$ such that $\forall x \in [-1, 1],\ f(x) \in p(x) + \boldsymbol{I}$.

Operations:

- affine operations: straigthforward;
- non-affine operations: enclose the nonlinear terms and add this enclosure to the remainder.

Roundoff errors: determine an upper bound b on the roundoff errors and add $[-b, b]$ to the remainder.

Agenda

Conclusions

Interval arithmetic

- overestimate the result
- are less efficient than floating-point arithmetic (theoretical factor: 4, practical factor: 20 to 100)
  $\Rightarrow$ solve "small" problems,
- still, can automatically enclose roundoff errors
- indeed, can be used mindlessly and still give useful results
- can even be used mindlessly and be foolproof. . . but expensive (wrt memory and time).

Conclusions

Interval algorithms

- can solve problems that other techniques are not able to solve
- provide a simple version of set computing
- give effective versions of theorems which did not seem to be effective (Brouwer)
- can determine all zeros or all extrema of a continuous function
- overestimate the result
- are less efficient than floating-point arithmetic $\Rightarrow$ solve "small" problems.

Existing software and libraries

- IntLab in MatLab
- intPak in Maple: not guaranteed
- IntLib in Fortran: global optimization
- COSY: Taylor models
- Boost
- MPFI
- C-XSC, Fi_lib
- libieeep1788 (1788 compliant)
- interval for Octave (1788 compliant)
- Moore (almost 1788 compliant)
- many specialized libraries, ongoing work for porting to HPC (GPU, MPI)