Why are Users of Interval Arithmetic so Often Disappointed?

Prepared for a UC Berkeley
Bebop Meeting

Wed. Mar. 4, 2020

This document will be posted at people.eecs.berkeley.edu/~wkahan/4Bebop/4Mar20.pdf

Why are Users of Interval Arithmetic so Often Disappointed?

And Why Should We Care?

Scientists, Engineers and their Programmers should care because
Interval Arithmetic and weaker cheaper versions of it
like Significance Arithmetic, UNUMs, …
are often advocated as easy answers to questions like …

How accurate are my program's results?

Why are they less accurate than I had expected or hoped?

Is inaccuracy due to an error (BUG) in my program,
or is its algorithm Numerically Unstable,
or is my data badly Ill-Conditioned?

These questions may lack easy answers.

Why are Users of Interval Arithmetic so Often Disappointed?

The short answer is

Grotesque Over-Estimates of Errors

when Interval Arithmetic is used **naively**. It can happen when all a program's Floating-Point (*Real*) variables are merely re-declared to be *Intervals*.

Here is how:

I: Ignored Anti-Correlations among variables.

II: The Rush to Judgement.

III: The Curse of Big Dimensions.

IV: Unavoidable Intentional Approximations

V: Uncertain Data

What can be done instead **Sometimes ?**

I: Ignored Anti-Correlations among variables.

In ordinary Floating-Point arithmetic, if all a polynomial's zeros are known, a product of factors is a good and often more accurate way to compute it. *Eg*:

$$\Pi(x) := \alpha\cdot(x - z_1)\cdot(x - z_2)\cdot((x - x_3)^2 + y_3^2)\cdot(x - z_4) .$$

Not necessarily so for Interval Arithmetic; *eg*:

$\Pi(x) := (x - 1)\cdot(x + 1)$; $P(x) := x^2 - 1$; $\mathbf{X} \in [-\delta, \delta]$ for $0 < \delta << 1$.

$\Pi(\mathbf{X}) \in [-1 - \delta^2 - 2\delta, \quad -1 - \delta^2 + 2\delta]$; $P(\mathbf{X}) \in [-1, \quad -1 + \delta^2]$.

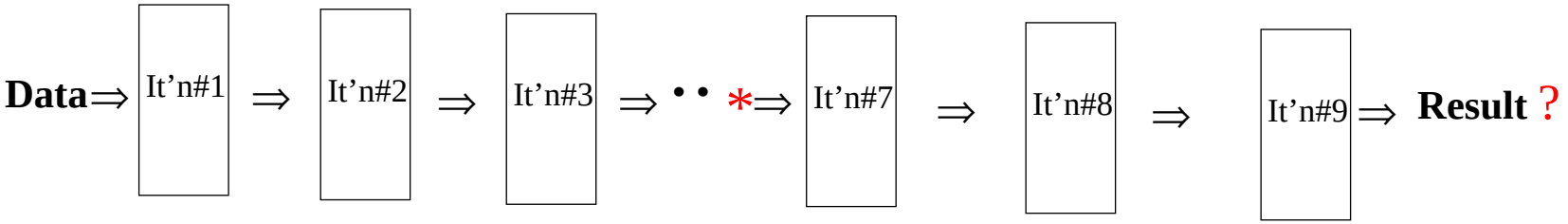
Now $\arccos(\Pi(\mathbf{X}))$ malfunctions though $\arccos(P(\mathbf{X}))$ is unexceptional.

I.A. treats each appearance of " x " as if it were independent of the others.

Similar misbehavior exaggerates errors in matrix computations associated with elastic structures and discretized boundary-value problems where data items are far fewer than the matrix elements computed from them.

II: The Rush to Judgement.

Programs that ***iterate*** upon data look like this:



Suppose an intermediate result marked by * differs utterly from what would have been computed had no rounding errors been committed. Does that imply

Result must be **Wrong** ?

It depends upon the algorithm. For some, **Result** would be wrong. For other algorithms in widespread daily use, **Result** would be quite satisfactory, almost as accurate as if no roundng error had occurred. *See* **IntvlQR.pdf** . However,

Interval Arithmetic can deliver undeservedly wide intervals implying that the **Result** is grossly inaccurate although ordinary arithmetic would have been satisfactory.

I.A. does not know about a correlation at point * where high-dimensional data can very nearly conserve a low-dimensional relationship despite roundoff.

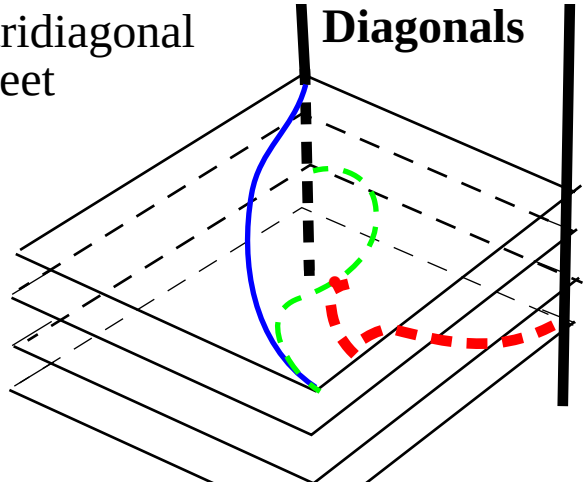
Example: QR-Iteration on Symmetric Tridiagonal Matrices

The elements of N-byN symmetric tridiagonals fill a 2N–1-dimensional space. Each matrix has N eigenvalues; any choice of those puts elements on an N–1-dimensional manifold (like a surface), maybe wrinkled if eigenvalues repeat.

Each QR-iteration-step moves the data on that manifold towards one of N**!** points where diagonal matrices exhibit the sought eigenvalues, except for roundoff. It has been proved to do little damage beyond altering the order in which eigenvalues are exhibited on a different diagonal.

All symmetric tridiagonal matrices in a sheet have the same eigenvalues.

Adjacent sheets differ by practically negligible roundoff.



Paths followed during a program's computation of eigenvalues with …

… no rounding errors

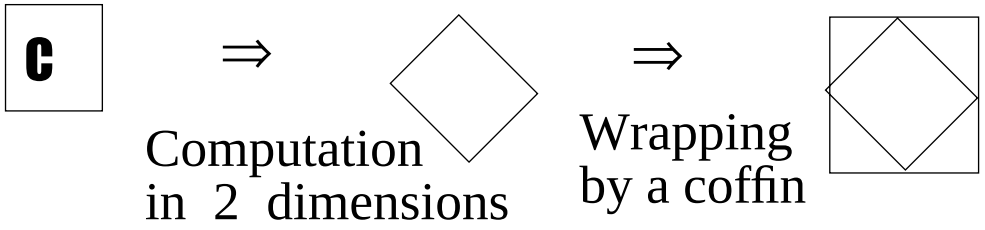
… the usual rounding errors

… and more severe rounding errors

Interval Arithmetic's intervals are at least as wide as the differences between paths with and without roundoff, thus deeming accurate results inaccurate.

III: The Curse of Big Dimensions.

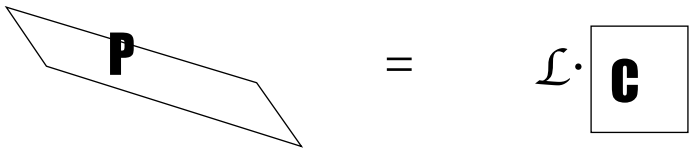
Vectors of intervals can represent only *Coffins*.



Wrapping the result of a computation that merely rotates can cause a coffin **C** to grow by $\sqrt{2}$.

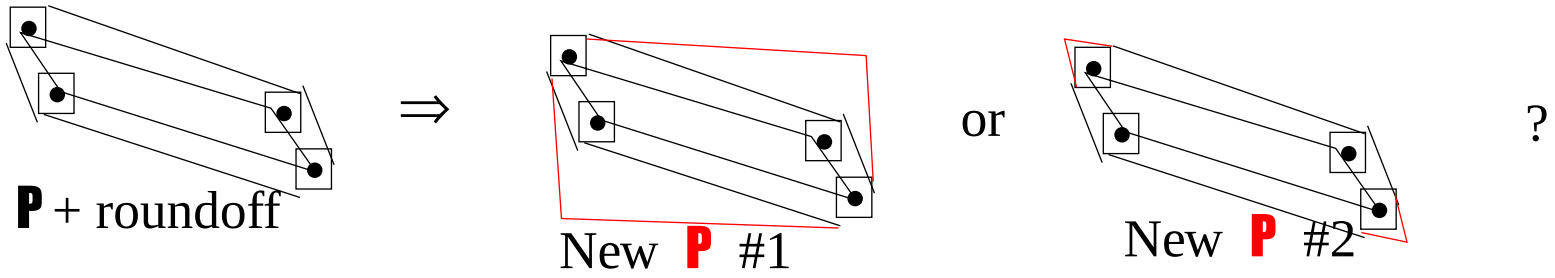
Computations in N dimensions can encounter growth by $\sqrt{N}$; see H.txt . And that is without additional interval expansion by rounding errors. Repeated compute-and-wrap-by-a-coffin can incur exponentially excessive growth.

In the late 1960s F. Krückeberg advocated using general parallelepipeds instead of coffins. (A parallelepiped **P** := $\mathcal{L}$·**C** for some invertible linear map $\mathcal{L}$.)



But when rounding errors are included, parallelepipeds can also incur growth that is exponentially excessive, though not so fast, and at much higher cost.

• • •



At first sight, New **P** #2 would be preferred to New **P** #1 . But if subsequent computation flattens New **P** further the way **P** has been flattened, that choice will be regretted. And there is a tendency for prolonged computation to flatten.

Pictures in low dimensions can be misleading. Parallelepipeds of extremely high dimensions resemble sea-urchins, with some vertices sticking out like spikes far from the main body despite that all parallelepipeds are convex.

Consequently the substitution of parallelepipeds for coffins cannot by itself avoid exponentially excessive growths of error-bounds in lengthy computations of very high dimensions.

Ellipsoids can do much better than parallelepipeds, but that is a story for another day

IV: Unavoidable Intentional Approximations

I suspect that Interval Arithmetic arose in the late 1950s to assess the accuracy of Inter-Continental Ballistic Missiles. They obey a Differential Equation:

$dy/d\tau = f(\tau, y)$ … vector y includes position coordinates and velocities.

$0 \le \tau \le T$ Typically τ is time. Typically T is given or inferred.

Solution $y(\tau)$ is approximated by $Y(\tau)$ computed from ***iterations*** of a formula:

$Y(\tau+\Delta\tau) := Y(\tau) + \Delta\tau \cdot F(\tau, Y(\tau), \Delta\tau)$ … F is a chosen Numerical Method:

$F(\tau, Y(\tau), \Delta\tau)$ is a computed average of samples of f intended to approximate

$$\int_{\tau}^{\tau+\Delta\tau} f(t, y(t))\,dt/\Delta\tau \quad .$$

The Numerical Method's *Local Error* can be estimated from a power series:

$$y(\tau+\Delta\tau) = y(\tau) + \Delta\tau \cdot F(\tau, y(\tau), \Delta\tau) + \Omega(\tau, y(\tau), \Delta\tau)\cdot\Delta\tau^{\text{œ}+1} + \ldots \quad ‡$$

derived as if $y(\tau)$ and its derivatives were known from its differential equation.

The Method's *Order* $\text{œ} \ge 1$; a smaller $\Delta\tau$ shrinks the error in $Y(T)$ like $\Delta\tau^{\text{œ}}$ at the cost of $T/\Delta\tau$ iterations. Some Methods estimate Ω as a byproduct to allow the local error-estimate to be controlled; see MATLAB's ODE Suite.

Differential Equation: $dy/d\tau = f(\tau, y)$ over $0 \le \tau \le T$.

Numerical Method: $Y(\tau+\Delta\tau) := Y(\tau) + \Delta\tau\cdot F(\tau, Y(\tau), \Delta\tau)$ iterated on $0 \le \tau \le T$.

Local Error: $\Omega(\tau, y(\tau), \Delta\tau)\cdot\Delta\tau^{p+1}$ Estimated from a Power Series:

$$y(\tau+\Delta\tau) = y(\tau) + \Delta\tau\cdot F(\tau, y(\tau), \Delta\tau) + \Omega(\tau, y(\tau), \Delta\tau)\cdot\Delta\tau^{p+1} + \ldots \qquad ‡$$

Naive Interval Arithmetic substitutes **intervals** **Y**, **F** and **Ω** into equation ‡ :

$$\mathbf{Y}(\tau+\Delta\tau) := \mathbf{Y}(\tau) + \Delta\tau\cdot\mathbf{F}(\tau, \mathbf{Y}(\tau), \Delta\tau) + \mathbf{\Omega}(\tau, \mathbf{Y}(\tau), \Delta\tau)\cdot\Delta\tau^{p+1}$$

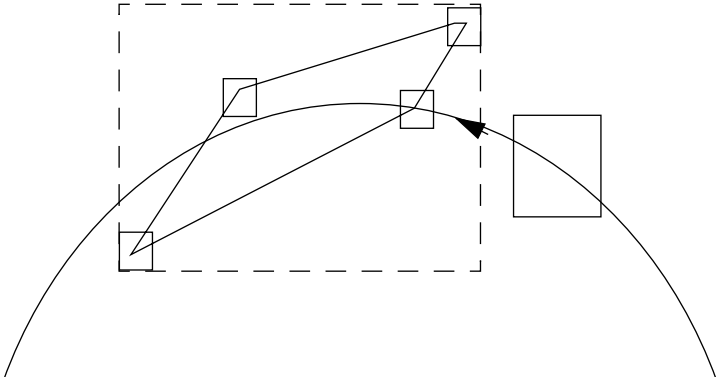
and iterates it. If accuracy is inadequate, the computation is redone with a larger number of smaller steps $\Delta\tau$ hoping the error-bound will shrink like $\Delta\tau^{p}$.

But it doesn't.

Not for Celestial Mechanics (Satellites)
Not for Exterior Ballistics (ICBMs)

Why not? Because Coffins suffer from the Curse of Big Dimensions, and grow exponentially until the orbit's curvature causes Coffins to explode. How?

Why Interval Arithmetic's Coffins are Exploded by the Curvature of an Orbits' Trajectories



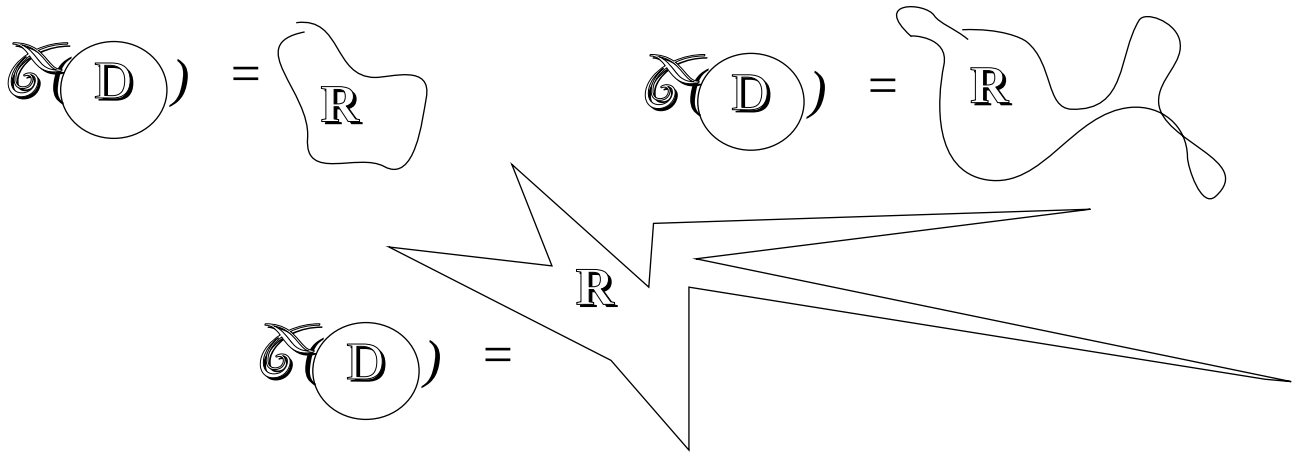
Solutions of the Differential Equation move faster on inside orbits than outside, which performs a ***Shear*** on a Coffin as well as rotating it a little. The Local Errror doesn't add much after a while, but circumscribing the Sheared and slightly Swollen Coffin produces a new Coffin that has grown much bigger.

Ellipsoids last a lot longer though cost a lot more; but that is a long story for another day.

V: Uncertain Data

Computation $\mathfrak{C}$ carries Data D to a result R := $\mathfrak{C}$(D) .
What if D is uncertain, say $\mathbb{D}$? Then $\mathbb{R}$:= $\mathfrak{C}(\mathbb{D})$:

Possibilities:



The shape of $\mathfrak{C}(\mathbb{D})$ can be very nearly arbitrary for all we know at the start.

Starting a naive Interval Arithmetic computation $\mathfrak{C}(\mathbb{D})$ from interval data $\mathbb{D}$ tends to over-estimate $\mathfrak{C}(\mathbb{D})$ for any of the reasons **I - IV** cited previously.

Mathematical and Probabilistic estimates for particular computations $\mathfrak{C}(\mathbb{D})$ appear at conferences whose titles include "***Uncertainty Quantification***".

What can be done instead **Sometimes ?**

For a rigorously guaranteed error-bound:

◊ Refereed & published Error-Analysis, if you can find one that fits.

◊ "*Self-Validating Computation*" mostly in the European Literature, requires that your computation be recast as the *Fixed-Point* of a **Sufficiently Strongly** *Contractive Map*. Not always possible.

◊ Arbitrarilu high variable precision Interval Arithmetic if roundoff is the only source of error.

These are almost prohibitively expensive.
too much for daily answers to our questions:

How accurate are my program's results?

Why are they less accurate than I had expected or hoped?

Is inaccuracy due to an error (BUG) in my program,
or is its algorithm Numerically Unstable,
or is my data badly Ill-Conditioned?

Software tools that cost much less to use have been developed in the distant past,
but their use today would require modifications to
operating systems, compilers, debuggers and Math. libraries.

See people.eecs.berkeley.edu/~wkahan/Boulder.pdf .

The Computing Industry has not sensed a widespread demand for such tools,
perhaps because the beneficiaries of such tools are
unaware that they could exist ***if***
they were demanded and paid for.