

A Critique of John L. Gustafson's
THE END of ERROR — Unum Computation
and his
A Radical Approach to Computation with Real Numbers

Prepared for an IEEE Symposium on Computer Arithmetic
ARITH 23

July 10-13, 2016, at the Hyatt Regency in Santa Clara CA

by W. Kahan, Prof. Emeritus (*i.e.*, retired)
Mathematics Dept., and E.E. & Computer Science Dept.
University of California @ Berkeley

This document, formatted for leisurely reading, is posted at
<www.eecs.berkeley.edu/~wkahan/UnumSORN.pdf>

Relevant Documents have been Exchanged:

Gustafson's book was published last year:

THE END of ERROR — Unum Computing (2015, CRC Press, 433 pp.).

A slightly revised version with typos corrected may exist somewhere.

A Radical Approach to Computation with Real Numbers

introduces SORNs and is posted on his web page www.johngustafson.net/... at

...presentations/Unums2.0slides.pptx and ...pubs/RadicalApproach.pdf

48 pages, updated to 23 Apr. 2016

16 pages, downloaded 18 May 2016

I think they greatly exaggerate the merits of schemes proposed therein.

My detailed critiques are posted at

www.eecs.berkeley.edu/~wkahan/EndErErS.pdf and .../SORNers.pdf
and supply analyses to support criticisms presented herein.

Today's two lessons:

A Moment's Mistake
can take far longer —
Months, Years —
to Find and to Fix.

Arithmetic
is a very small part of
Mathematics & Computer Science.

What are *Unums* (“Universal numbers”) ?

Floating-point numbers each with width, range and precision variable at run-time, each tagged to show whether it is exact or uncertain by $\frac{1}{2}ulp$.

Likely to serve in pairs as endpoints of intervals for interval arithmetic.

Alleged to save storage space if each is no wider than a datum needs.

Ideally, widths would vary automatically to yield final results of desired accuracies.

What are *SORNs* (Sets Of Real Numbers) ?

Finite collections of *Extended Real* numbers (including unsigned ∞) and also open intervals between them, referenced by *pointers* instead of their values.

Likely to serve in pairs as endpoints of intervals for interval arithmetic,

Save storage space if pointers are no wider than the data need.

Fastest arithmetic operations each achieved by a table-look-up.

Both schemes are intended for use with massive parallelism, more than thousands-fold, and with *Big Data*, perhaps as “a shortcut to achieving exascale computing”.

Unum and SORN Computation would be Worth their Price,
whatever it is,

IF

the Promises Gustafson has made for them
could *ALWAYS* be fulfilled.

But they can't.

Not even *USUALLY*.

The Promises are Extravagant;
the Virtues of Unums and SORNs have been exaggerated;
and you can't *Always* know whether they have betrayed you.

Offered here will be a few of many examples posted in ...EndErErS.pdf & ...SORNers.pdf

Extravagant Claims for Unum and SORN Computing ?

The book claims that ...

Unum Computation needs no difficult mathematical analysis to prevent mishaps due to ...

- Discretization of the continuum, as when solving differential equations ?
- Underestimating how uncertainties in the given data affect computed results ?
- Roundoff and over/underflow in IEEE 754's floating-point arithmetic ?

Unum and SORN computation are free from Interval Arithmetic's failure modes ?

- The *Dependency Problem*: taking no account of correlations among variables
- The *Wrapping Effect*: intervals grow exponentially too fast in long computations

The book claims that “**Calculus is Evil**”, or at least unnecessary ?

- No need to know it to solve differential equations; use massive parallelism instead?
- No need for modern numerical analysis nor difficult mathematical error-analyses?

No use nor need for IEEE 754's mostly still unsupported diagnostic aids like ...

- “A plethora of NaNs” ?
- Flags inaccessible from most programming languages ?
- Directed roundings (since Unums and SORNs would eliminate roundoff) ?

These claims pander to Ignorance and Wishful Thinking.

Wishful Thinking: We can be Liberated from Error-Analysis ?

Four steps to solve a computational problem ...

- 1• *Choose* or invent an algorithm α , express it in the language of Mathematics, and *prove* that it would work if performed in ideal arithmetic with infinite precision.
- 2• Translate α faithfully into a program P in a computer's programming language like **C** or **FORTRAN** or **MATLAB** or **PYTHON** or
- 3• Execute P to obtain a result R . Usually R is accurate enough, sometimes not.

How can we know *for sure* whether R is accurate enough ?

- 4• If P 's arithmetic is *Floating-Point*, whatever its precision(s), we need a proof-like *Error-Analysis* to determine *for sure* if P implements α accurately enough.
 If an Error-Analysis exists, it may be obvious, or it may be obscure.
 If an Error-Analysis exists, it could cost more than R is worth.



Wishful Thinking:

Error-Analysis is unnecessary if P is executed in SORN or Unum arithmetic ?

Disappointment:

Like Interval Arithmetic, SORN and Unum Computation can grossly overestimate, sometimes by many orders of magnitude, the uncertainty in computed results. *How?*

Let $\mathbf{f}(\mathbf{x})$ be the function to be computed; let $\mathbf{f}(\mathbf{x})$ be the algorithm or formula chosen to compute $\mathbf{f}(\mathbf{x})$; and let $\mathbf{F}(\mathbf{x})$ be (the result from) the program that implements $\mathbf{f}(\mathbf{x})$.

When executed in floating-point arithmetic, $\mathbf{F}(\mathbf{x})$ could occasionally be arbitrarily wrong for all we know without an error-analysis. What can be done about that? (*Later*)

When executed in SORN or Unum or Interval arithmetic, $\mathbf{F}(\mathbf{x})$ is an *interval* that must enclose the ideal mathematical value of $\mathbf{f}(\mathbf{x})$. *What if $\mathbf{F}(\mathbf{x})$ is far too big?*

- If big width is due to roundoff, redo computation with *appropriately* higher precision. Unum Computation lets precision be increased by the program. Often it works.
- If big width of $\mathbf{F}(\mathbf{X})$ is due to uncertain data \mathbf{X} but seems far too big, partition \mathbf{X} into smaller subregions $\bar{\mathbf{X}}$ and replace $\mathbf{F}(\mathbf{X})$ by the *Union* of all intervals $\mathbf{F}(\bar{\mathbf{X}})$. Often this union $\cup_{\bar{\mathbf{X}} \in \mathbf{X}} \mathbf{F}(\bar{\mathbf{X}})$ is smaller than $\mathbf{F}(\mathbf{X})$ if every $\bar{\mathbf{X}}$ is tiny enough.

BUT NEITHER RECOMPUTATION **ALWAYS** SUCCEEDS.



Two of the Failure Modes:

- How can increasing Unums' precision fail to overcome roundoff ?

It happens to futile attempts to compute accurately an expression at its discontinuity.

The program may fall into an infinite sequence of ever increasing precisions.

It's unnecessary if discontinuity alters only the path to the program's goal; it can obstruct common matrix computations, like eigenvalues.

Examples: see pp. 4 - 5 & p. 8 of .../EndErErS.pdf .

- How can the union $\cup_{\bar{x} \in \mathbf{X}} \mathbf{F}(\bar{x})$ fail to be smaller than over-size $\mathbf{F}(\mathbf{X})$?

Example: $R(x, y) := \frac{(x-y) \cdot (x+y)}{x^2 + y^2}$ and $S(x, y) := 1 - \frac{2}{1 + (x/y)^2}$ are two expressions

for the same rational function. In IEEE 754 floating-point $R(x, y)$ is the more accurate when $|x/y| \approx 1$; but $S(x, y)$ is not degraded by over/underflow *etc.*

Let \mathbf{Q} be the open square $((0, 1), (0, 1)) = \{ (x, y) : 0 < x < 1 \ \& \ 0 < y < 1 \}$.

It is representable in both SORN and Unum arithmetic as a pair of open intervals.

Subdivide \mathbf{Q} into a union of smaller rectangles $\bar{\mathbf{Q}}$. BUT however tiny they are,

$\cup_{\bar{\mathbf{Q}} \in \mathbf{Q}} \mathbf{R}(\bar{\mathbf{Q}}) = (-\infty, +\infty)$ stays far wider than the correct interval $\mathbf{S}(\mathbf{Q}) = [-1, +1]$.



$\cup_{\bar{\mathbf{Q}} \in \mathbf{Q}} \mathbf{R}(\bar{\mathbf{Q}}) = (-\infty, +\infty)$ stays far wider than the correct interval $S(\mathbf{Q}) = [-1, +1]$.

How often does misbehavior this bad occur ?

It is an instance of the *Dependency Problem* familiar in Interval Arithmetic circles.

The book's chs. 16 & 18.2 claim to overcome the Problem; p. 12 ofpdf & pp. 44-6 ofpptx claim SORN arithmetic has No Dependency Problem. Claims are mistaken.

Recall: $\mathbf{f}(\mathbf{x})$ is to be computed using algorithm $\mathbf{f}(\mathbf{x})$ implemented as program $\mathbf{F}(\mathbf{x})$.

- $\mathbf{F}(\mathbf{x})$ may work fairly well with floating-point but misbehave with Interval Arithmetic, as does $\mathbf{R}(\mathbf{Q})$. How could you know this in advance without knowing $S(\mathbf{Q})$?
- $\mathbf{f}(\mathbf{x})$ may be a numerically precarious algorithm to compute $\mathbf{f}(\mathbf{x})$ at slightly uncertain data \mathbf{x} no matter how \mathbf{F} is programmed. Does an algorithm better than \mathbf{f} exist?
Example: the *Incenter* of a tetrahedron; as volume shrinks $\mathbf{F} \rightarrow \infty$; see p. 26 in www.eecs.berkeley.edu/~wkahan/MathH110/Cross.pdf.
- $\mathbf{f}(\mathbf{x})$ may be the solution of an equation $\mathbf{a}(\mathbf{x}, \mathbf{f}) = \mathbf{o}$ whose coefficients depend on \mathbf{x} . If \mathbf{x} is uncertain so are they, but correlated in a way all interval arithmetics ignore. The equation's solution may react far worse to perturbations than does $\mathbf{f}(\mathbf{x})$.
E.g: deflections of loaded elastic structures, crash-tests, least-squares, ...

A remedy: Use higher precision, not SORN/Interval Arithmetic.

A Third Failure Mode:

The *Wrapping Effect* is familiar in the Interval Arithmetic community. It can cause intervals to grow too fast exponentially with a computation's length and dimension.

- Without mentioning it the book suggests that it does not afflict Unum Computation; see p. 306 for the claim “no exponential growth in the error, only linear growth.”
- SORN arithmetic's “Uncertainty grows *linearly* in general” [pp. 41-2 of ...pptx]

Actually SORN/Unum/Interval arithmetics *can* and do generate intervals that grow too fast exponentially with a computation's length n and dimension d .



Example: simplified Dynamical System's Reachable Set: $\mathbf{x}_n := H \cdot \mathbf{x}_{n-1} = H^n \cdot \mathbf{x}_0$, $n > 6$

$H := 20$ -by- 20 Hadamard matrix; every element has magnitude $1/\sqrt{20}$, but $H^2 = I$. Initialize interval $\mathbf{X}_0 :=$ Unit HyperCube. Compute $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_{2n}$ in turn. Dimension $d = 20$; Computation length = $2n$. True $\mathbf{X}_{2n} = \mathbf{X}_0$; no growth at all. SORN/Unum/Interval Arithmetics produce \mathbf{X}_{2n} excessively too big by $20^{n-1/2}$.

To reduce grossly excessive to moderately excessive, say 400 times too big, \mathbf{X}_0 has to be subdivided into at least 20^{20n-50} tiny hypercubes $20^{5/2-n}$ on a side, well past the capability of “... mindless ... large-scale parallel computing”. [book p. 219]

“Never Wrong” \neq “Always Right”

A proponent of SORN/Unum/Interval arithmetic may claim that it is *Never Wrong* since it delivers an interval that *Always* encloses the True Result (if one exists).

But we’ve seen delivered intervals vastly wider than the True Result deserves.

How bad is that ?

No harm is done by intervals *known* to be much too wide;
these will be disregarded, if computed at all. (Interval arithmetic isn’t popular.)

Harm *is* done by vastly oversized intervals believed deserved by the data.

- A worthwhile project may be abandoned unnecessarily.
- Extra work may be undertaken only because interval arithmetic was believed.

Without an error-analysis or an alternative computation for comparison,
SORN/Unum/Interval Computation’s failure modes are difficult to diagnose.

To make matters worse, SORN arithmetic lacks *Algebraic Integrity*,

thus undermining a programmer’s faith that Arithmetic \approx Algebra.

What is Algebraic Integrity ?

IEEE 754 Floating-point has it:

In the absence of roundoff,

if several different *rational expressions* for the same *function* produce different values when evaluated numerically, at most *two* different values can be produced, and either the two values are $\pm\infty$ or else **at least one is NaN**, which is easy to detect.

SORN arithmetic lacks Algebraic Integrity, and boasts that it has no NaN, and ... “No rounding errors No exceptions” [... .pptx p. 3,pdf p. 2]. BUT ...

Different SORN expressions for a rational function can produce different SORNs :

Example: As rational functions, $u(t) := 2t/(1+t) = v(t) := 2 - (2/t)/(1+1/t)$;
 $x(t) := (1+u(t)^2)/(2+u(t)^2) = y(t) := (1+v(t)^2)/(2+v(t)^2) = z(t) := 1 - 1/(2+v(t)^2)$;

SORN arithmetic gets $x(0) = 1/2$, $y(0) = (0, \infty]$, $z(0) = [1/2, 1]$, with *no roundoff* nor indication of anything amiss about $y(0)$ or $z(0)$.

As *functions*, $x(t) = y(t) = z(t)$; but when evaluated as arithmetic *expressions* SORN arithmetic gets $x(0) = 1/2$, $y(0) = (0, \infty]$, $z(0) = [1/2, 1]$, with no indication of anything amiss about $y(0)$ or $z(0)$. They have “lost information”.


How? Where other arithmetics would produce a NaN for $0/0$, $\infty - \infty$, $0 \cdot \infty$, ∞/∞ , *etc.*

SORN arithmetic produces Ω , the set of all Extended Reals. $\Omega^2 = [0, \infty]$.

Why would different expressions for the same function appear in a program?

Over different subdomains of the function’s domain, different expressions may be less vulnerable to “loss of information”, or have different costs of evaluation.

On subdomains’ boundaries, the different expressions should agree within roundoff.

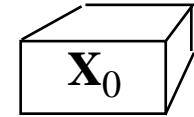
If one expression malfunctions, the program can try another,
 but only if it detects the malfunction.

An arithmetic system that hides malfunctions must produce misleading results occasionally.

We shall see it happen to SORN arithmetic.

C-Solutions:

An important application of Parallel Interval Arithmetic :



Search for *all* solutions \mathbf{z} of an equation “ $\mathfrak{a}(\mathbf{z}) = \mathbf{o}$ ” within a given *coffin* \mathbf{X}_0

given an interval program $\mathfrak{A}(\mathbf{X})$ for $\mathfrak{a}(\mathbf{x})$ satisfying $\mathfrak{A}(\mathbf{X}) \supseteq \mathfrak{a}(\mathbf{X})$ and, except for roundoff, $\text{width}(\mathfrak{A}(\mathbf{X})) \rightarrow 0$ as $\text{width}(\mathbf{X}) \rightarrow 0$.

(A *coffin*, called a *ubox* in the book, is a vector of intervals.)

Procedure: $\mathfrak{A}(\mathbf{X})$ excludes $\mathbf{o} \Rightarrow$ coffin \mathbf{X} cannot contain a solution \mathbf{z} , so discard \mathbf{X} .

Partition \mathbf{X}_0 into small coffins $\bar{\mathbf{X}}$, and discard all those that cannot contain any \mathbf{z} .

Partition all remaining small coffins into smaller coffins; discard " " " ... " " .

Repeat until every remaining coffin is tiny enough, or none are left.

The book calls the remaining coffin(s) a *C-Solution* of “ $\mathfrak{a}(\mathbf{z}) = \mathbf{o}$ ”.

Example: $\mathfrak{a}(x) := 3/(x + 1) - 2/(x - 1) + 1/(x - 1)^2$. Start search at $\mathbf{X}_0 := [0, 4]$.

For $\mathfrak{A}(\mathbf{X}_0)$, Unum & Interval arithmetic get NaN; SORN arithmetic gets Ω .

Despite the NaN *we must not discard* \mathbf{X}_0 . Repeated subdivision converges to tiny intervals around $z = 1$, $z = 2$ and $z = 3$. But only $\mathfrak{a}(2) = \mathfrak{a}(3) = 0$.

Unum/Interval's $\mathfrak{A}(1)$ is NaN. SORN's $\mathfrak{A}(1) = \Omega$ unexceptionally.

C-Solutions can include singularities of \mathfrak{a} unless filtered out.

C-Solutions can include singularities of \mathfrak{a} unless filtered out.

Never-exceptional SORN arithmetic renders singularities difficult to distinguish from ordinary overly wide intervals.

Example: Construct an equation “ $\mathfrak{a}(\mathbf{z}) = \mathbf{o}$ ” using $R(\xi, \eta) := \frac{(\xi - \eta) \cdot (\xi + \eta)}{\xi^2 + \eta^2}$ thus:

$$\mathfrak{a}(\mathbf{x}) := \begin{bmatrix} R(\xi, \eta) - 9/8 \\ R(\eta, \xi) + 9/8 \end{bmatrix} \text{ at } \mathbf{x} = \begin{bmatrix} \xi \\ \eta \end{bmatrix}. \text{ Seek solution } \mathbf{z} \text{ using SORN arithmetic for } \mathfrak{A}(\mathbf{X}).$$

Whenever $\mathbf{o} \in \mathbf{X}$ so does $\mathbf{o} \in \mathfrak{A}(\mathbf{X}) = \begin{bmatrix} \Omega \\ \Omega \end{bmatrix}$. And $\mathbf{o} \in \mathfrak{A}(\mathbf{X})$ whenever one corner of \mathbf{X} is much closer to \mathbf{o} than the others. Consequently the C-Solution process converges to tiny rectangles clustered closely around \mathbf{o} plus, in SORN arithmetic, one enclosing \mathbf{o} .

But $\mathbf{z} = \mathbf{o}$ is not a solution.

$-1 \leq R(\xi, \eta) = -R(\eta, \xi) \leq 1$ except SORN's $R(0, 0) = R(\infty, \infty) = \begin{bmatrix} \Omega \\ \Omega \end{bmatrix}$ instead of NaN, so the equation “ $\mathfrak{a}(\mathbf{z}) = \mathbf{o}$ ” has no solution \mathbf{z} .

In general, C-Solutions can “solve” equations that have no solution.

It would not have happened *here* if Unum/Interval arithmetic replaced SORNs, and *also* $S(\xi, \eta)$ from p. 9 above replaced $R(\xi, \eta)$ to produce narrower intervals.

The book disparages Calculus and Modern Numerical Analysis

- p. 181 Unums **should** work even when used in a naive way, [duty? likely?]
the way floats usually are used.
- p. 216 This is the essence of the ubox approach. [Compensate for ignorance ?]
Mindless, brute-force application of large-scale parallel computing ...
- p. 311 **Calculus considered evil: Discrete physics**
Calculus deals with **infinitesimal** quantities; [NO! ... with limits]
computers do not calculate with infinitesimals.
- p. 273 When physicists analyze pendulums, they **prefer** to talk about "small oscillations".
- p. 316 **Every physical effect can be modeled without rounding error or
sampling [discretization] error if the model is discrete.** [Not what
he does]
- p. 277 Instead, we treat *time* as a function of *location*. [... assuming conservation laws]
..., the time dependency of physical simulation has been misused [By whom?]
as an excuse not to change existing serial software to run in parallel.
[The book's algorithms cannot cope with drag nor friction.]
[cf. EndErErs.pdf pp. 25 -31.]

Book pp. 327-332 An arbitrarily precise solution method for nonlinear ordinary differential equations that uses **no calculus**, just elementary algebra, geometry and Newtonian physics.

Counterexample: $du/d\tau = (1 - u) \cdot v$ and $dv/d\tau = -(1 + u) \cdot v$ from Chemical Kinetics **requires calculus** to reveal Conservation of $u(\tau) + 2 \cdot \log(|u(\tau) - 1|) - v(\tau)$, which simplifies the decay-time of $v(\tau)$ to numerical evaluation of an integral.
[cf. p. 13 of EndErErs.pdf]

Numerical Quadrature is the numerical evaluation of an integral $\int_a^b f(x) dx$.

About this topic, the book has lots to say, all obsolete, misleading and irrelevant:

Book p. 198 $\text{error} \leq (b - a) \cdot |f''(\xi)| \cdot h^2/24$ [Midpoint Rule vs. $\int_a^b f(x) dx$]

... What the hell is that? ... To compute the second derivative we **have to know** calculus ... then we have to **somehow find** the *maximum possible absolute value* ...

This is why the classical error bounds that are still taught to this day are deeply profoundly unsatisfying.

Actually, we don't have to know calculus to invoke *Automatic Symbolic Differentiation* software that transforms program F to a program that interleaves F with F' and F'' . And we don't have to find $\max |f''(\xi)|$ to estimate an integral rigorously.

In ch. 15, the book's crude numerical quadrature ignores modern numerical analysis and consequently costs too much work by orders of magnitude despite parallelism.

Here is how Work grows as Error is diminished by various algorithms:

Book's algorithm	Work = $O(1/\text{Error}^2)$	Interval bounds	EndErErs.pdf pp. 21-2
------------------	------------------------------	-----------------	-----------------------

1960-70's algorithm	Work = $O(1/\sqrt{\text{Error}})$	Interval bounds	EndErErs.pdf p. 23
---------------------	-----------------------------------	-----------------	--------------------

1970-80's algorithm	Work = $O(-\log(\text{Error}))$	Asymptotically	EndErErs.pdf pp. 23-4
---------------------	---------------------------------	----------------	-----------------------

Book p. 281 "... it may be time to overthrow a century of numerical analysis." [Not yet.]

And it's all irrelevant.

The book *THE END of ERROR — Unum Computing* spends more pages advocating ill-advised numerical methods than comparing equitably the costs and benefits of

Unum Computing vs. Interval Arithmetic

with precision roughly variable at run-time and supported by an appropriate programming language and Math. library.

What does Unum Computation cost? *Too much!*

Unums' widths can vary almost arbitrarily at run-time, and they are intended to be packed tightly to minimize time & energy per Unum moved.

Consequently, let's compare Unums of diverse and varying widths vs.

interval arithmetics of precisions 2, 4, 8, 16, ... bytes wide:

- **Arithmetic:** Larger latency because unpacking requires more pipeline stages vs. interval arithmetic's precisions declared upon entry to subprograms whose local variables are then allocated on a stack at call-time.
- **Memory Management:** Its cost is overlooked in the book, which says on pp. 40-41 "... does the programmer have to manage the variable fraction and exponent sizes?
No. That can be done automatically by the computer." **[For a price!]**

Fetching Unums: must cost at least one extra indirect address reference.

Writing Unums: must cost at least one extra indirect address reference *except, if width can change*, must cost more indirection writing to a *Heap* and subsequent Memory Defragmentation/Garbage Collection.

Costs depend crucially upon how the programming language manages diverse widths.

What does SORN Arithmetic cost?

Consider SORN pointers N bits wide, roughly like floating-point's precision.
 N is small; probably $8 \leq N \leq 16$.

Computed SORN intervals usually require pairs of pointers after “information is lost”,
 and may be *Exterior* intervals that include ∞ : *cf.* my 1968 lectures

$$\text{Example: } X = [6, 8]/[-1, 2] = [3, -6] = \{ x \geq 3 \text{ or } x = \infty \text{ or } x \leq -6 \}$$

Allowing for Exterior intervals greatly complicates SORN arithmetic;
 it complicates Interval arithmetic too. Complicated \Rightarrow Slower.

A (too) much faster arithmetic scheme allows *Arbitrary* collections of SORNs,
 each represented by a word 2^N bits wide. If implemented on the CPU chip,
 this faster arithmetic would need area $O(2^{3N})$ — better used for cache.

The slower scheme, using pairs of N -bit pointers, if implemented on the CPU chip,
 needs area $O(N \cdot 2^{2N})$ to run faster than interval arithmetic software using
 standard N -sig.bit floating-point occupying $O(N^2)$ area on-chip.

SORNs could satisfy a demand, if it exists, for low-precision Interval arithmetic. ...

How much Precision is Enough ?

Much of the world's data fits into short words. Most computed results fit into a few digits.

For many a *short* floating-point computation of *low dimensionality*, arithmetic's precision is adequate if it exceeds the accuracy desired in the result.

SORN arithmetic might satisfy that requirement well enough, but
**SORN arithmetic should not be used for lengthy computations
lest it produce intervals vastly too wide, difficult to diagnose.**

An old *Rule-of-Thumb* renders roundoff extremely unlikely to cause embarrassment:

**In all intermediate computation, perform arithmetic carrying somewhat more than
twice as many sig.dec. as are trusted in the data and desired in the final result.**

This rule has long served statistics, optimization, root-finding, geometry, structural analysis and differential equations. Rare exceptions exist, of course. Nothing is perfect.

SORN/Unum/Interval arithmetic purports to insure against betrayal by that rule of thumb, but runs the risk of betrayal by a failure mode of interval arithmetic, as we have seen,

The only sure defence against embarrassment due to roundoff is an error-analysis,
but it might not exist.

Unum and SORN Computation would be Worth their Price,
whatever it is,

IF

the Promises Gustafson has made for them
could *ALWAYS* be fulfilled.

But they can't.

Not even *USUALLY*.

The Promises are Extravagant;
the Virtues of Unums and SORNs have been exaggerated;
and you can't *Always* know whether they have betrayed you.

They can't obviate error-analysis. What can be done instead?

As a scientist or engineer,

I wish not to know how big my errors due to roundoff and discretization aren't.

And I am unwilling to pay much for what I wish not to know.

I dearly need to know ...

- ... that errors due to discretization are negligible
- ... that errors due to roundoff are negligible.
- ... how much uncertainty my results have inherited from uncertain data.

What I need to know is *almost always* revealed by some repeated recomputations:

- Appraise discretization error by refining the discretization.
- Appraise rounding errors by increasing precision, or else [cf. my .../Boulder.pdf] by three recomputations with redirected roundings of all atomic operations.
- *Uncertainty Quantification*, the appraisal of uncertainty inherited from data, requires a difficult and often costly combination of several approaches:
 - » Error analysis, Perturbation analysis, Partial Derivatives,
 - » Recomputation at many samples of intentionally perturbed input data.
 - » Interval arithmetic used skillfully to avoid excessive pessimism.

How shall we debug long intricate scientific and engineering programs using SORN/Unum Arithmetic ?

- Overly wide SORNs and their spurious C-solutions are difficult to diagnose or cure.
- Overly wide Unums due to uncertain data are difficult to diagnose, tedious to cure.

SORN/Unum arithmetics lack IEEE 754's *Flags* ; consequently ...

Overly wide SORNs/Unums due to over/underflow are difficult to diagnose or cure;
lack of flags that point to sites where exceptions first occurred obscures them.

SORNs lack NaNs, lack Algebraic Integrity; cannot easily discover invalid operations.

Unums have just one NaN instead of IEEE 754's "plethora" of them that can
serve as pointers to the program's sites where they were created.

Alas, IEEE 754's diagnostic capabilities are still supported poorly
by programming languages and software development systems, and
by computer architectures that trap floating-point exceptions
into the operating system instead of into the Math. library.

[see my .../Boulder.pdf]

How Best to Enhance the Reliability of Approximate Computation ?

Definitely not by pandering to ignorance.

Probably not by investing a lot of effort in radically different arithmetics.

We can accomplish more than what Unums would accomplish by investing in ...

- Software development systems that support IEEE 754’s diagnostic capabilities.
- Programming languages liberated from FORTRANish expression-evaluation and supporting ...
 - » 2, 4, 8, 16, ... byte wide precisions chosen when a subroutine is called
 - » tagged intervals including Exterior, and Center ± Radius, and Open-ended
 - » coffins and parallelepipeds and ellipsoids
 - » better error-control for a library’s solvers (equations, quadrature, ODEs, ...)

... in that order of priority.



“Work expands to fill the time available for its completion.” C. Northcote *Parkinson’s Law* [1958]

The best gauge of newer faster computers’ worth is

how much more they can do in the same time as before. J.L. *Gustafson’s Law* [1988]

How shall we gauge the reliability of a computing environment ?