Prof. W. Kahan's

# Commentary on *"THE END of ERROR — Unum Computing"*

by John L. Gustafson, (2015) CRC Press

## Contents

# Commentary on *"THE END of ERROR — Unum Computing"*

by John L. Gustafson, (2015) CRC Press

I am going to lose a friend by quoting a *Yiddish* saying appropriate for this very seductive book:

"Almost all True is altogether a Lie."

The first lie is the book's title. It promises that Unum Computing will end computational errors. It won't. These errors have numerous sources besides programmers' correctable mistakes:

[1] Answering the wrong question, correctly or incorrectly.
[2] Errors in models of physical, chemical, biological, economic, … systems.
[3] Discretization errors in approximations to the continuum and to movement.
[4] Errors and uncertainties in measurements and other observations.
[5] Roundoff in arithmetic and in Decimal ↔ Binary conversions.

Correct numbers can lie. Mark Twain attributed to Benjamin Disraeli, perhaps wrongly, …

"There are three kinds of lies: lies, damned lies, and statistics."

So Unum Computing cannot possibly and does not try to put an end to all-too-common errors of type [1], nor of type [2]. Pp. 327-332 assert that unums alleviate *all* errors of the other types, but they can't. Which kinds of errors can unums allay? Many of type [5], and a few of type [4], but provably not *all* of them. Neither need unums perform their task economically when they succeed at it. And they cannot always succeed without human mathematical error-analysis.

Unum Computing resembles interval arithmetic with varying precisions determined more or less automatically at run-time to achieve a prescribed accuracy demand. Good**!** Interval arithmetic came into existence in the late 1950s to cope with the consequences of uncertainties mainly of types [3-4], but succeeded only in relatively simple special cases because of limitations imposed by small (by today's standards) memories and one-pass compilers. Even when freed from these limitations, even if it exploits vast memories and massive parallelism and varying precisions determined at run-time, interval arithmetic remains vitiated by the Curse of High Dimensions. And since Unum Computing is a tarted kind of variable-precision interval arithmetic, …

The same Curse of High Dimensions vitiates Unum Computing.

Alas, errors take so much longer to correct than to commit that I despair of correcting even the worst few errors in Gustafson's book. Its 433 pages offer misunderstandings, misconceptions, misremembered history, misleading comparisons and mistaken mathematics. Where to begin?

The book's Preface begins on p. xiii with an intentionally provocative quotation:

"The purpose of mathematics is to eliminate thought."

Mere hyperbole. Mathematics needs no purpose; but when mathematics does have a purpose it is to promote economy of thought. It can eliminate wasteful thought, not all, and certainly not thoughts that must accompany every approximate computation. Among these thoughts are error-analyses, sometimes trivial, sometimes not. I have long yearned to automate them and spend time instead upon enjoyable kinds of mathematics. No such luck. Unum Computation cannot END ERROR because it cannot end the occasional necessity for an error-analysis. Why not?

# §1: Why Approximation = Sin

In the 1970s D.H. Lehmer, a renowned Number Theorist at Berkeley, used to warn me …
"Acquiescence to rounding errors places you in a state of sin."

That sin occurs when an ideal mathematical algorithm is not distinguished from the computer program intended to implement it, though the program was transliterated faithfully from the algorithm into the computer's programming language. Usually the program's result is as close as desired to what was expected from the algorithm. Occasionally results are wrong enough to cause trouble. Unum Computing purports to eliminate those occasions. Why do they happen anyway?

Some algorithms suffer badly from roundoff at some otherwise innocuous input data. When we recognize such an algorithm we call it "Numerically Unstable" and try to replace it by a different numerically stable algorithm to solve the same mathematical problem. Often an error-analysis can distinguish the stable from the unstable algorithm by taking account of roundoff; but there are complex algorithms in daily use for whose programs no satisfactory error-analyses exist yet. This is why numerical computation can be somewhat uncertain. Most programs are simple enough to have obvious error-analyses, or they invoke subprograms from a tested and reliable library, or they use stable algorithms programmers with Science, Engineering or Mathematics degrees may have learned from a University course on modern Numerical Analysis.

The book END of ERROR … is aimed at programmers who have taken no course on Numerical Analysis or have forgotten it, and yet desire numerical results more than *usually* correct; they want *always* correct. The book promises Unum Computation *always* delivers correct results.

No such promise can be fulfilled.

Deep mathematical and logical properties of real numbers prevent Unum Computation, and any other scheme restricted to perform only arithmetic operations of *at most* some finite precision, no matter how wide that precision may be, from *always* getting correct results out of *every* program transliterated faithfully from an ideal mathematical algorithm that would work correctly with infinitely precise arithmetic. These are the wages of the sin of approximate computation. To know that a computed result is (in)correct may require an error-analysis, which might not exist.

Everything in life is uncertain except death and taxes. Compared with ordinary floating-point of an appropriately chosen precision, the book's Unum Computation often diminishes uncertainty, often exacerbates it, and usually takes longer, sometimes very much longer despite reliance upon parallel computation to an extent that not everyone can afford. The book's examples were chosen to show how well Unum Computation works. To show how badly it can misbehave, examples will be presented herein. None are complicated; most differ little from the book's.

The first example is adapted from the book's pp. 173-6. It exposes a fundamental difficulty in the computation of real numbers in general, most the results of limit-processes. The simplest one is an iteration $\mathbf{x}_{n+1} := \boldsymbol{f}(x_n)$ that always converges to a limit $\mathbf{z} = \boldsymbol{f}(\mathbf{z})$ as $n \to \infty$, and does so at a palpable rate, though the limit $\mathbf{z}$ may depend upon the given initial value $\mathbf{x}_0$. Ideally, iteration can cease when $\mathbf{x}_{n+1} - \mathbf{x}_n$ becomes smaller than some tolerance inferred from the known rate of convergence and the desired accuracy. Thus an ideal mathematical algorithm derived from the text of $\boldsymbol{f}$ and its properties can compute the limit $\mathbf{z}$ provably as accurately as desired.

Let that ideal algorithm be translated into a computer program including a subprogram **F** derived from the text of ***f*** to compute it to any desired accuracy using arithmetic of adequate precision.

When and why should the computed iteration $\mathbf{X}_{n+1} := \mathbf{F}(\mathbf{X}_n)$ be stopped ?

A stopping criterion suited to an ideal iteration $\mathbf{x}_{n+1} := \boldsymbol{f}(x_n)$ might not work for its computed iteration $\mathbf{X}_{n+1} := \mathbf{F}(\mathbf{X}_n)$, especially when the limit **z** depends discontinuously upon $\mathbf{x}_0$ and also every subsequent iterate. It happens to an iteration due to J-M. Muller slightly modified:

$$w_{n+1} := 111 - (1130 - 3000/w_{n-2})/w_{n-1} \; ; \quad w_0 := 2 \; ; \quad w_1 := -4 \; .$$

This is the iteration in the book's pp. 173-6. It can be written in the form $\mathbf{x}_{n+1} := \boldsymbol{f}(x_n)$ by setting

$$\boldsymbol{f}\left(\begin{bmatrix} w \\ v \end{bmatrix}\right) := \begin{bmatrix} 111 - (1130 - 3000/v)/w \\ w \end{bmatrix} \; ; \quad \text{the iteration starts at } \mathbf{x}_0 := \begin{bmatrix} -4 \\ 2 \end{bmatrix} \; .$$

Absent roundoff, $\mathbf{x}_n \to \mathbf{z} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$ about as fast as $(5/6)^n \to 0$ ; but roundoff causes $\mathbf{X}_n \to \begin{bmatrix} 100 \\ 100 \end{bmatrix}$ at least about as fast, ultimately, as $(3/50)^n \to 0$ no matter how many significant digits are carried to compute **F** in ordinary floating-point. 53 sig.bits (like 15 sig.dec.) produces $W_{13} \approx 6.1394$ .

What does Unum Computing get? On p. 175 the book exhibits iterates $w_2$ to $w_{13}$ as *ubounds* (intervals) carrying up to 57 sig.dec. Their widths appear not to shrink; their $w_{13}$ is roughly **[**6.1395, 6.1452**]** . It seems a meager reward for such arduous arithmetic. Unaided by an error-analysis, how long will Unum Computing take to get, say, 13 sig.dec. of the correct limit **z** ?

Forever?

An error-analyst's task goes beyond overestimating error. When an overestimate is excessive, a different way is sought to compute the desired result with an acceptable error at a tolerable cost. Correctly $w_{13} \approx 6.142359$ computed from an error-analyst's replacement of the original function ***f*** by a new $\bar{\boldsymbol{f}}\left(\begin{bmatrix} w \\ v \end{bmatrix}\right) := \begin{bmatrix} 11 - 30/w \\ w \end{bmatrix}$ to get a numerically stable recurrence that computes accurately enough every iterate $\mathbf{x}_n$ in floating-point arithmetic. But iterates are not the desired result; it is their limit **z**. Is there another way to compute it, preferably sooner?

Iteration selects **z** from the roots of an equation "$\mathbf{z} = \boldsymbol{f}(\mathbf{z})$". Unum Computing and interval arithmetic share a laborious scheme to find numerically every solution of an equation in some given part of its domain. That part is partitioned into numerous small regions (*ubounds* or *uboxes*) on each of which the equation's range is (over)estimated. Regions within which the equation can nowhere be satisfied are discarded; the rest are partitioned into smaller subregions, some discarded, and so on until the remainder confine the equation's roots as tightly as desired.

The book calls them a *c-solution*.

The iteration $\mathbf{x}_{n+1} := \boldsymbol{f}(\mathbf{x}_n)$ amounts to a way to choose a fixed-point $\mathbf{z} = \boldsymbol{f}(\mathbf{z})$ of ***f***. As defined above, it has three fixed-points at which $w = v = 5,\ 6$ or $100$. The choice depends upon $\mathbf{x}_0$ discontinuously in a way the c-solution of "$\mathbf{z} = \boldsymbol{f}(\mathbf{z})$" cannot reveal. In general an equation like it could have infinitely many roots **z** from which to choose. Or the domain of ***f*** could be a vector space of high dimension in which $\mathbf{z} = \boldsymbol{f}(\mathbf{z})$ on a continuum whose sufficiently tight c-

solution entails astronomically many tiny subregions (the *Curse of High Dimensions*), but does not reveal the limit of the iteration $\mathbf{x}_{n+1} := \boldsymbol{f}(\mathbf{x}_n)$. That kind of c-solution resembles more nearly an *Answer to the Wrong Question* than the evaluation of the desired limit $\mathbf{z}$. Problems that foil Unum Computation in similar ways will turn up in §4 and §6 below.

Approximate computation of a discontinuous function may seem foolish, yet it happens often. As a function of its elements, the rank of a matrix is discontinuous if less than both dimensions. A square matrix's Jordan Normal Form is discontinuous if it is not diagonal. The greatest common divisor of polynomials with real coefficients varies with them discontinuously when nontrivial.

Such problems must be altered a little to make sense in the context of approximate computation. For instance, instead of asking for the rank of a matrix, compute its distances from the nearest matrices of ranks lower than its least dimension. A seemingly slight alteration of the problem has incurred enormously many more approximate arithmetic operations than the unaltered problem's solution would have entailed if exact (like integer) arithmetic were feasible. Consequently most people resist accepting a complicated and costly answer to what seemed to be a simple question.

Still, to ask Unum Computing to offset human folly may be deemed unfair, so henceforth …

**We shall eschew computations of a discontinuous function at one of its discontinuities.**

That was my motive for an example about which Gustafson's book says on p. 177 …

> "… not many moments of 'high drama' in a book … .
> *Can unums defeat Professor Kahan's monster? …*" [His *italics*.]

My "*monster*" showed how floating-point of every preassigned precision could deliver the same wrong result almost everywhere for a continuous function. Frustrating Unum Computation was not the *monster*'s purpose. Besides, I advised that "Numerical distress due solely to roundoff is relieved too often by increased precision for its use when available to be deterred by this example despite its worrisome simplicity." Let's look at this example the book calls a "*monster*":

Real variables x, y, z ;
Real Function T(z) := { If z = 0 then 1 else ( exp(z) – 1 )/z } ;
Real Function Q(y) := | y – √(y² + 1) | – 1/( y + √(y² + 1) ) ;
Real Function G(x) := T( Q(x)² ) ;
For Integer n = 1 to 9999 do Display{ n , G(n) } end do.

( I don't know why the book changed my names from T to E and G to H .)

Ideal real arithmetic, free from roundoff, produces $Q(x) = 0$ and $G(x) = 1$ for every $x > 0$. Approximate arithmetic almost always produces something else tiny for $Q(x)$, if not a rounding error then $-1/(2x)$ when $x$ is so huge that "$x^2 + 1$" rounds the 1 away. Then $\exp(Q(x)^2)$ rounds to 1 and $G(x) := T( Q(x)^2 )$ ends up wrongly as 0 instead of 1. Almost always.

This disconcerting example comes from §6 of my web-page posting Mindless.pdf. It is simple enough for a few computer programs (one called "Herbie" was devised at the University of Washington in Seattle) to scan the text of functions T, Q and G, diagnose their error, and offer

to cure it as would a human error-analyst. Diagnosis exposes a culprit: The analytic function T*(z)* is a *Divided Difference* with a *Removable Singularity* at z = 0, but its removal is thwarted by roundoff in exp*(z)* when |z| is tiny and not zero. A cure was presented in my posting just after the diagnosis. Assuming that the Math. library's exp and log are accurate within less than a unit in the arithmetic's last digit carried (as should always be the case nowadays), the following substitute for T was supplied:

```
Real Function  T(Real z) := {                    … Precautions against
        t := exp( z ) ;                          … premature over/underflow,
        If ( t ≠ 1 ) then  t := ( t – 1 )/log( t ) ;   … superfluous here, have been
        Return( t ) ;          }                 … omitted for simplicity's sake.
```

With this substitution, G(x) = 1 for every x > 0 and for all supported precisions, single, double and quadruple, of floating-point arithmetic. The revised program counteracts roundoff in exp.

After this example my posting Mindless.pdf goes on to say …

> "Ironically, if multi-precision Interval Arithmetic were used naively to compute G(n) either from its initial formula or from its accurate program, the results at every precision would be intervals so excessively wide as could not distinguish the accurate program from the inaccurate one."

The book runs my initial formula G (my "*monster*") in interval arithmetic to get G ∈ [-∞, ∞] on p. 177, but nowhere mentions my accurate version. Instead the book substitutes this for T :

```
Real Function  T(z) := { If  z ≈ 0  then  1  else  ( exp(z) – 1 )/z } ;
```

This version of T differs from my *monster*'s in just one crucial symbol, which yields G = 1 correctly; but it side-steps a fundamental question:

> How small must a computed value z be to be deemed indistinguishable from zero?

The answer to this question requires an error-analysis or a lucky guess. Gustafson was lucky. Had my *monster* been designed to thwart Unum Computing, my G would have been different:

```
Real Function  G°(x) :=  T( Q(x)^2 + (10.0^-300)^(100000·(x+1)) ) ;
        For  Integer  n = 1 to 9999  do  Display{ n ,  G°(n) }  end do.
```

Without roundoff, the ideal value G°(x) ≈ 1.0 for all real x. Rounded floating-point gets 0.0 almost always for all practicable precisions. What, if anything, does Unum Computing get for G°(n) ? And how long does it take? It cannot be soon nor simply 1. Why does the addition of a negligible deeply underflowed quantity to the argument of T cause neither of my programs to behave differently but distresses the text's program? We'll see soon below.

The computation's undiagnosed culprit is Q. In the early 1970s, when the *monster* was born, no automated algebra system like *Macsyma*, *Maple*, *Mathematica*, … could *Simplify* (deduce that) Q(y) = 0 for all real y in the absence of roundoff. Now some systems, like *Maple*, are smart enough to do it. To outsmart them, Q can be replaced by a different expression provably zero although no current computer program can find the proof. The question arose in the 1950s:

> Given any collection $\{\zeta_j\}$ of at least two real numbers, perhaps transcendental,
> do real algebraic numbers $\{\alpha_j\}$ exist, not all zero, satisfying $\sum_j \alpha_j \cdot \zeta_j = 0$ ?

The conjecture then was that the question is undecidable in general. The conjecture persists so far as I know. In 1968 D. Richardson in pp. 511-520 of J. Symbolic Logic **33** showed how, if the conjecture is true, any algorithm intended to decide which expressions simplify to zero can be thwarted by an expression built from integers, finitely many algebraic operations, absolute value, and one transcendental constant $\log(2)$ .

In short, whether an expression simplifies to zero can be very unobvious. What an expression delivers when computed approximately can be unpredictable without an error-analysis, which might not exist. Replacing the predicate "$z = 0$" by "$z \approx 0$" implies a tolerance whose appropriate choice is generally unobvious without an error-analysis, which might not exist.

Gustafson has dodged the zero-question by changing utterly the meaning of "$\approx$". When he asserts "$x \approx y$" he means that *ubounds* x and y overlap in at least one point. Consequently he would assert "$[0, 100000000] \approx 0$". An infinitesimal addition like the one in my revised G° would alter that predicate to something like "$(0, 100000000.000\ldots001) \approx 0$" whose open *ubound* (interval) now excludes the single point 0 , rendering the predicate *false*.

In a numerical computation designed well, a predicate like "$x \approx y$" would select one of two paths through a program that both produce an adequate result when the predicate is nearly true or nearly false. This is what my more accurate version of T does promptly. The book's version of T must take an exceptionally long time, or produce a wide unsatisfactory *ubound*, or both.

We have lived through such experiences before. For instance, in the mid-1960s K. Iverson's programming language APL was implemented on an IBM 360/50 in the basement of IBM's Watson Research Laboratory in Yorktown Heights. The implementers decided that the predicate "$x = y$" should always be interpreted as if it were "$x \approx y$ within a tolerance CT". Here the *Comparison Tolerance* CT was a *System Variable* which, like the origin for indexing arrays, the programmer could alter from what the system supplied by default. In 1967 the implementers admitted to me that, had they known earlier what they had come to know by then, they would not have done it. Almost no programmer would figure out how to alter CT ; and after they ignored it some of their programs would misbehave badly. Gustafson's "$\approx$" will revive that experience.

On p. 178 just after his Unum Computation of the "*monster*", his book reproduces without attribution a painting of the young hero David standing with his sling over the corpse of Goliath.

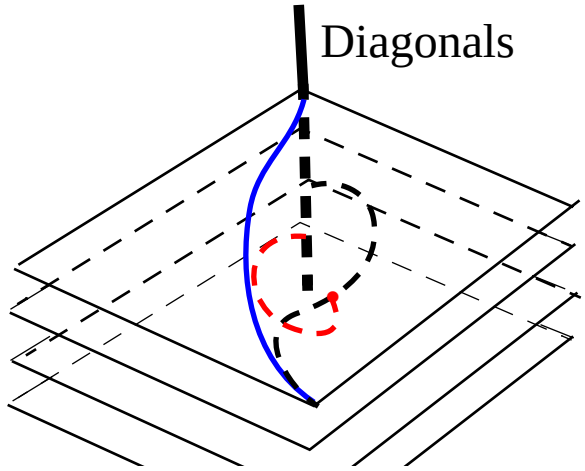
More hyperbole.

## §2: Oh, Ye'll take the Low Road and I'll take the High Road …

Both roads take you to Scotland, though the "high road" may take you there sooner.

Surprisingly many numerical programs deliver practically the same results by utterly different intermediate paths, some of them sometimes taking substantially longer than others, all chosen by accidents of roundoff. Examples include Gaussian elimination with pivotal exchanges to solve linear systems, QR and Jacobi iterations to solve eigenproblems, and Newton's and other iterations to solve polynomial and other nonlinear equations, especially in high dimensions.

Let's focus upon a specific example: QR iteration to compute the eigenvalues and eigenvectors of a real symmetric matrix. First it is reduced by orthogonal *Similarities* (matrix multiplications) to a tridiagonal form with the same eigenvalues within tolerable rounding errors. Then many QR iterations are performed to reduce the tridiagonal to a diagonal matrix with the same eigenvalues within tolerable rounding errors. When some off-diagonal elements are small but not yet small enough to disregard, the iteration is virulently unstable in so far as the iteration's progress from one tridiagonal to the next becomes an accident of roundoff/roundoff. The picture below shows (smoothed) the program's roundoff-determined path through the space of tridiagonal matrices:




All the tridiagonal matrices in a sheet have the same eigenvalues, obvious in diagonals.

The eigenvalues of adjacent sheets differ, inconsequentially, in only their last few bits.

Curved paths are followed during a program's computation of the set of eigenvalues when …

… no rounding error occurs: "C"

… the usual rounding errors occur: "?"

… a rounding error is altered: "G"

This can happen when the same data is presented to the same program but compiled by different compilers (or different "optimizations") leading to the performance of associative operations, like two additions or a multiplication-and-division, in different orders. No great harm is done to the eigenvalues though they may come out in an inconsequentially different order.

How will Unum Computing handle this kind of near-inconsequential indeterminacy? If a book's algorithm or a published Fortran program is transliterated to Unum Computing, will it expend extra time recomputing in higher precision until the indeterminacy is overcome? Or will this indeterminacy be converted to uncertainty (ubounds) and propagated to the results? Who makes the choice? Without an error-analysis, how can higher precision be justified when each matrix element is uncertain in its last bit, though this uncertainty affects only the last few bits of the computed eigenvalues?

Unum Computing proffers easy answers to questions about computational errors, not necessarily helpful answers.

# §3: Interval and Ubound Evaluations of a Polynomial

We wish to evaluate a polynomial, say $\pi(\xi) := \alpha + \beta\cdot\xi + \delta\cdot\xi^2$ to keep it simple, given intervals for its coefficients and for its argument $\xi$. On pp. 225-232 of *THE END of ERROR — Unum Computing* John L. Gustafson asserts "**a general approach for evaluating polynomials with interval arguments without any information loss is presented here for the first time**." [His **bold-face**.] On p. 227 the book says "Perhaps one reason this approach does not seem to appear in the literature for interval arithmetic is that it makes heavy use of the ubound ability to express whether an endpoint is open or closed" No, that is not the reason. His scheme is unused partly because it is inefficient and partly because of a mistaken assertion on p. 226:

" Notice that the new coefficients each use a, b, c, … at most once.
That means they can be computed without information loss. …"

Actually, information ***can be*** lost when the coefficients are intervals (or nontrivial *unums*), rather than the exact scalars used in the text's simple illustrative examples. That information loss will be demonstrated below. Consequently better ways than the book's exist to evaluate the interval range of a polynomial, even using unums.

First let us see how information gets lost. Given that $\xi \in [x, X]$ (which means $x \le \xi \le X$), the book partitions the interval $[x, X]$ into subintervals each narrow enough that its process, that overestimates the range of the given polynomial $\pi$ on each subinterval, produces acceptably close overestimates. "Acceptably close" means that no overestimate is wider than the true range by more than an implied tolerance. The process shifts the $\xi$-origin to subintervals' ends. The book claims that, because each shift's every new coefficient uses the given coefficients "at most once", no information can get lost. Here is a counter-example to that claim:

Given that $\alpha \in [a, A]$, $\beta \in [b, B]$ and $\delta \in [d, D]$, we infer that $\pi(\xi) \in [p(\xi), P(\xi)]$ thus:

If $\xi \ge 0$ then $\{ p(\xi) := a + b\cdot\xi + d\cdot\xi^2 ; \quad P(\xi) := A + B\cdot\xi + D\cdot\xi^2 \}$
else $\{ p(\xi) := a + B\cdot\xi + d\cdot\xi^2 ; \quad P(\xi) := A + b\cdot\xi + D\cdot\xi^2 \}$ .

An origin-shift to, say, $\xi = 2$ produces a polynomial $\phi(\zeta) = \pi(\zeta+2)$ with these coefficients:

$$\phi(\zeta) := (\alpha + 2\beta + 4\delta) + (\beta + 4\delta)\cdot\zeta + \delta\cdot\zeta^2 .$$

Here each of $\phi$'s new coefficients uses each of the given coefficients "at most once".

Now we infer that $\phi(\zeta) \in [f(\zeta), F(\zeta)]$ thus:

If $\zeta \ge 0$ then $\{ f(\zeta) := (a + 2b + 4d) + (b + 4d)\cdot\zeta + d\cdot\zeta^2 ; \quad F(\zeta) := (A + 2B + 4D) + (B + 4D)\cdot\zeta + D\cdot\zeta^2 \}$
else $\{ f(\zeta) := (a + 2b + 4d) + (B + 4D)\cdot\zeta + d\cdot\zeta^2 ; \quad F(\zeta) := (A + 2B + 4D) + (b + 4d)\cdot\zeta + D\cdot\zeta^2 \}$.

Since $\phi(-1) = \pi(+1)$, we might hope that $[f(-1), F(-1)] = [p(+1), P(+1)]$, but actually

$$f(-1) = (a + 2b + 4d) - (B + 4D) + d \;<\; p(+1) = a + b + d \quad \text{and}$$
$$P(+1) = A + B + D \;<\; F(-1) = (A + 2B + 4D) - (b + 4d) + D$$

unless the given coefficients' intervals are degenerate with $b = B$ and $d = D$.

The loss of information (excessively wide $[f(\zeta), F(\zeta)]$) can be avoided by taking the precaution to correlate the signs of $\zeta$ and $\xi$, but I have not found that precaution in the text's process.

Anyway, there is a process more efficient on average than Gustafson's.

Having constructed the almost-polynomials p and P that barely constrain $\pi(\xi) \in [p(\xi), P(\xi)]$, we can determine the enclosure of $[p(\xi), P(\xi)]$ for $\xi \in [x, X]$ by computing the extrema of p and P in that $\xi$-interval. The computation is easy if $\pi$ is cubic or quadratic because internal extrema of p and P occur at internal zeros of their derivatives. This is true also for a polynomial of higher degree, but then the challenge is to compute *all* the extrema, which requires that *all* the internal zeros of the derivative be located. The accuracy of those zeros is not challenging, because a polynomial varies so slowly around an internal extremum that it is easy to compute accurately enough around there by interval arithmetic carrying enough precision.

The challenge is to get *all* of the extrema.

The derivative of a polynomial is a polynomial of lower degree. Each of a polynomial's simple real zeros lies between two adjacent extrema of opposite signs. This reduces the location of all extrema of a polynomial to the location of all extrema of that polynomial's second derivative, whence follow locations of all the zeros of the first derivative. Downward recursion reduces the challenge to finding all the real zeros internal to $[x, X]$ of a cubic. Bracketed between extrema, real zeros are computable efficiently by numerical processes like Newton's or Secant iteration even if zeros are (nearly) repeated. For further explanation see my lecture notes posted at

www.eecs.berkeley.edu/~wkahan/Math128/RealRoots.pdf , §7 and §10.

Low accuracy suffices; except for the polynomials p and P, each zero of a derivative need only be accurate enough to determine the sign of an extremum. Thus all of them will be found. But if extrema of p and P are needed to high *relative* accuracy, and if any are nearly zero, then M. Mignotte's theorems from the early 1980s imply that surprisingly high precision may be needed.

At first sight the foregoing recursive process seems to require the numerical computation of too many zeros, roughly $n^2/2$ for a given polynomial $\pi$ of degree n. That can happen, but not usually. A famous theorem (due to Erdos?) says that among polynomials of degree n with real coefficients drawn independently and randomly from many a distribution centered at 0, the expected number of real zeros is a modest multiple of $\log(n)$. That theorem suggests that the foregoing recursive process might typically compute $n \cdot \log(n)$ zeros, but this far exceeds the numbers I have observed.

My experience is limited. I have seen very little demand for interval evaluations of high-degree polynomials expressed as sums of monomials with diverse coefficients. Either almost all the coefficients are the same, or the polynomials are defined by recurrences like those for orthogonal polynomials. These two situations require methods quite different from the ones discussed so far.

There is a demand for realistic estimates of uncertainties in the computed zeros of polynomials that never appear explicitly. These zeros are the eigenvalues of matrices whose many elements are derived from fewer parameters uncertain within specified tolerances. These determine a region in parameter-space that can be partitioned into small boxes each of which provides a matrix whose eigenvalues can be computed accurately enough using arithmetic of sufficiently high precision. Thus an image in eigenvalue-space of that region in parameter-space can be estimated. A close estimate's cost can be horrendous, even with parallelism; it is the Curse of High Dimensions.

Computations like those keep error-analysts busy;
unums will not render them no longer employable.

# §4: "Calculus considered evil: Discrete Physics"

This is the title for the book's ch. 21, which begins on p. 311 with half a page occupied by a picture of a raccoon saying

I HAVE INVENTED SOMETHING
EVIL

I WILL CALL IT CALCULUS

DIYLOL.COM

Gustafson goes on to say …

> "… Calculus and computers make for strange bedfellows, and their combination can destroy the validity of results. Calculus deals with infinitesimal quantities; computers do not calculate with infinitesimals. …"

This quotation tells me that he suffers from a misapprehension that bewildered Bishop Berkeley in the early 18th century, when he complained about the "ghosts of vanished quantities".

Calculus does *not* deal with infinitesimal quantities, though it easily could. Several decades ago Abraham Robinson formulated an extension to algebra with real variables by including also rules for infinitesimals, rendering them tractable by computerized algebra systems like *Mathematica*. This extension does shorten some mathematical proofs, but it is unnecessary for scientists' and engineers' applications of the calculus, as a Cal Tech. graduate in Applied Math. should know.

Expressions like "dy/dx" and "∫…dx" that appear to involve quotients and infinite sums of infinitesimals are actually shorthand for processes that approximate limits as closely as desired if allocated enough time. Equations "dy/dx = …" and "∫…dx = …" often offer faster ways to compute their left-hand sides' limits. Sometimes they equate limits that seemed unrelated before. The limit-processes would take many more words to mention if abbreviations were unavailable.

I don't know how seriously Gustafson wants readers to take his disparagement of the calculus. He is not joking. On p. 330 he includes among the advantages of Unum Computing this claim:

> " • An arbitrarily precise solution method for nonlinear ordinary differential equations that uses no calculus, just elementary algebra, geometry and Newtonian physics."

He continues on p. 331 with a peroration that deprecates floating-point computation as mere guesswork, which indeed it would be in the absence of error-analyses.

> "… Why should anyone continue to use floats when unum arithmetic can mimic floats but also has so many advantages? And why should we tolerate numerical methods that are hard to write and treacherous to use, when bounded methods like ubox sets can produce results that are complete solutions of maximum expressible accuracy?"

This quotation's "treacherous to use" tells me that Gustafson suffers from a misconception widespread in the 1950s and supported by assertions from luminaries, like Von Neumann, who deprecated floating-point. It was deemed so refractory to error-analysis that nobody tried to do it. Of course, one consequence was unreliable floating-point computation. That changed in the late 1950s when, motivated partly by a comment Turing threw away in a paper published in 1949, we found ways to analyze floating-point error. Some were called "Backward Error-Analyses".

Gustafson has contempt for backward error-analysis. He expresses his derision on p. 76:

"It is a variation on the Original Sin, and amusingly, it puts the blame back on the computer user.

> "I cannot give you the answer you requested, and it is all your fault, because you *should* have asked a slightly different question. I gave you a perfect answer to the question you should have asked." [His box]

Gustafson has mistaken an explanation for an excuse, somewhat like a quotation from Mme. De Staël misrendered in English as "To understand all is to forgive all." His is a common mistake.

Backward error-analysis never excused wrong answers. When feasible (which is not always) it maps the rounding errors in a program to end-figure perturbations in the data for the mathematical function the program was intended to compute. Now the consequences of the program's roundoff can be appraised by a perturbation analysis of the mathematical function no longer complicated by the program's details. When a user's data are uncorrelatedly uncertain by much more than end-figure perturbations, that program's roundoff becomes inconsequential; this is the ideal situation. Otherwise higher precision arithmetic must be used for the computation, or a better algorithm (which may be hard to find).

• • • • • • •

The time has come for another test of Gustafson's claims for Unum Computing. It is a simple nonlinear differential equation he claims can be solved using "… no calculus, just elementary algebra, …" for algorithm development, and no human's error-analysis. A. Tissier posed the problem on p. 694 of the *Amer. Math. Monthly* **94** (1987). His differential equation

$$\text{“ } dy/dx = x - 1/y \text{ ”}$$

has a solution $y(x)$ that stays positive and bounded for all finite $x \geq 0$ provided the initial value $y(0)$ is just right. Call that just-right initial value $y^\circ$. If $y(0) > y^\circ$ then $y(x) \to +\infty$ as $x \to +\infty$. If $0 < y(0) < y^\circ$ then a finite $x^\circ > 0$ exists at which $y(x) \to 0$ as $x \to x^\circ$ from below. The task is to find $y^\circ$ correct to, say, 15 sig.dec. without recourse to higher transcendental functions.

In 1988 or 1989 I tried that computation on a new Intel i302 system with i386 and i387 chips whose floating-point I had helped design. The i302 came with an Intel compiler for Fortran 77 produced by people who tried to humor me, so it worked very much as I desired. After an error-analysis of the differential equation, including two elementary differential inequalities, I tested a numerical method that converged at 6th order to as much of the differential equation's solution as needed to get $y^\circ \approx 1.2835987104636$ provably correct to 15 sig.dec. The program performed fewer than 1,600,000 rational arithmetic operations $(+, -, \cdot, \div)$ starting from integers; no $\sqrt{}$ nor $a^b$, and no log except to count compactly (not compute) leading zeros in differences between approximations. The program's text occupies 5751 bytes, mostly explanatory comments.

For corroboration, another 6th order program computed the same $y^\circ$ performing fewer than 1,300,000 rational operations but including two irrational constants, $4^{1/3}$ and $4^{1/5}$, and 7832 bytes of text. Later the i386 and i387 were updated to Cyrix clones, so I have resurrected the programs and rerun them to get the same results but faster partly because the Cyrix CPU has a tiny 1KB on-chip cache twice as big as needed to hold all my programs' variables.

Given the foregoing "guess" $y^\circ$, how much will Unum Computing cost in human effort and computer time to corroborate the accuracy of $y^\circ$ using "no calculus, just elementary algebra" ?

Infinitely more than it cost me.

• • • • • • •

Is Tissier's problem too artificial to serve as a fair test of Unum Computing without Calculus?

If so, here is a problem taken from Photo-Chemical Kinetics, but simplified by the absorption of rate-constants into variables to render them dimension-free. Variables $u(\tau)$ and $v(\tau)$ represent concentrations of two pollutant gases in the atmosphere. They react, turning v into u at a rate proportional to v, but also decomposing both at a rate proportional to their product $u \cdot v$. Their differential equations are …

$$du/d\tau = (1 - u)\cdot v \quad \text{and} \quad dv/d\tau = -(1 + u)\cdot v \,.$$

As time $\tau \to +\infty$, the reaction approaches a steady state: $v(\tau)$ decreases to 0 while $u(\tau)$ tends monotonically to a limit determined by given positive initial values $u(0) := u°$ and $v(0) := v°$.

If we wished to compute the steady-state value $u(+\infty)$ we could construe it as the limit of an iteration whose fixed-points constitute a continuum, as was mentioned on pp. 4-5 in §1 above. The limit depends continuously upon the iteration's starting point. An application of Differential and Integral Calculus provides a fast way to compute $u(+\infty)$ at the cost of at most a few hundred floating-point operations. Computing $u(+\infty)$ does not answer a more interesting question:

How long does $v(\tau)$ take to decrease from $v°$ to, say, 1% of $v°$ ?

Let $T(u°, v°)$ answer that question. What is the cost, in human effort plus machine computation, of a program that computes $T(u°, v°)$ to, say, 3 sig.dec. without invoking the allegedly evil Differential and Integral Calculus? Infinitely more than the program cost me …:

When I used them to produce a MATLAB program for my old Mac Quadra, it computed each $T(u°, v°)$ with fewer than 10,000 floating-point operations according to MATLAB's *flops* count. The program occupies less than 3 KB and runs entirely in the µ68040's 8 KB cache.

**Table 1: Decay times $T(u°, v°)$ for $v(\tau)$ from $v°$ to $v°/100$**

| | $v° = 0.001$ | $v° = 0.01$ | $v° = 0.1$ | $v° = 1.0$ | $v° = 10.0$ |
|---|---|---|---|---|---|
| $u° = 0.001$ | 4.597 | 4.5652 | 4.2926 | 3.197 | 2.3675 |
| $u° = 0.01$ | 4.5561 | 4.5254 | 4.2615 | 3.1886 | 2.3669 |
| $u° = 0.1$ | 4.1841 | 4.1624 | 3.9709 | 3.104 | 2.3616 |
| $u° = 1.0$ | 2.3026 | 2.3026 | 2.3026 | 2.3026 | 2.3026 |
| $u° = 10.0$ | 0.4187 | 0.4189 | 0.4211 | 0.4446 | 0.9284 |

To produce this table efficiently my program uses the reaction's conservation of a transcendental relation between u and v. I see no way to find it without using that Calculus deemed evil.

At issue above are not the virtues nor deficiencies of Unum Computing.

At issue is Gustafson's perverse anti-intellectual disdain for the past few centuries' mathematical analyses. His scorn is a gratuitous distraction from a fair evaluation of Unum Computing's costs and benefits.

## §5: What does Unum Computing cost?

On today's computers the cost, in time and power dissipation, of memory management often dominates arithmetic in the execution-time costs of a computation. Its cost in human effort — data-gathering, analysis, (re)programming and debugging — can dominate execution-time costs in engineering, scientific and statistical work; but let's reconsider human costs later. For now, let's focus on the execution-time costs that the book says Unum Computing will save. These savings are summarized on pp. 193-4 where a bogus analogy is invoked to combat criticism:

> `The biggest objection to unums is likely to come from the fact that they are variable in size, at least when they are stored in packed form.`

> The biggest objection to unums is likely to come from the fact that they are variable in size, at least when they are stored in packed form.

"What you see above are two identical sentences, one with a monospaced typeface (`Courier`) and the other with a variable-width typeface (Times) of exactly the same font size. Notice how much less space the variable-width typeface uses. There was a time (before Xerox PARC …) that computers *always* displayed monospaced font, since system designers thought it inconceivably difficult to manage text display with letters of variable width. … Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces. … "

"Fewer bits means unums will be faster than floats in any computer that is bandwidth-limited …"

Bunkum**!** Gustafson has confused the way text is printed, or displayed on today's bit-mapped screens, with the way text is stored in files and in DRAM memory by word-processor software. Look for yourself. You will see strings of constant-width ASCII or Unicode characters plus a sprinkling of milestones and escape-characters. Milestones mark the separations between lines, paragraphs and pages. Escape characters supply formatting information about fonts, sizes, styles (**bold**, *italic*, underlined, …) and how justified (left, centered, right, …), *etc*. The files may include fonts. In compressed files, like .pdf files, storage economy is achieved by exploiting redundancy in texts, not by storing variable-width characters packed together tightly. Text stored in variable-width characters would occupy more DRAM memory, not less, as we shall see.

To say "… system designers thought it inconceivably difficult …" is to misremember history by forgetting Nroff and Troff, the UNIX software that drove expensive printers to print text with variable-width characters in the 1970s. And these could be viewed on an expensive IBM 8500-series (if I remember rightly) bit-mapped display. It was expensive because video memory and its processor occupied many chips on some boards. Character-based displays were far cheaper and simpler to drive, requiring only a 9-pin connector. I still have one somewhere. Later in the 1980s, costs of 1-chip CPUs and VRAM memory chips came down enough that a bit-mapped color VGA display could be driven from a small card and a 15-pin connector; and WYSIWYG word processors showed how variable-width characters would look when printed. Now DRAMs are cheap and big enough, and CPUs fast enough, to look up characters' widths in a font table as they fly to the screen or page. What used to be expensive was never "inconceivably difficult".

• • • • • • •

When the book counts how much a computation with unums costs, it is displayed it in a box thus:

| | |
|---|---|
| Numbers moved | 178152 |
| Unum bits moved | 4043801 |
| **Average bits per number** | 22.7 |

This box appears on p. 191; others are on pp. 115, 175, 178, 183, 187 and 265. The box shown here has the biggest of all costs the book displays. It is for a Fourier Transform during which an array of unums are repeatedly read, modified and stored back. Some costs have been overlooked.

Can you see those costs omitted from the cost-box?

• • • • • • • Pause. • • • • • • •

How much does a unum cost to fetch? After it has been modified, its width may increase; then how much will finding a new site to store it cost? Can costs of address computations be ignored?

The book brushes the last question aside on pp. 40-41:

"… does the programmer then have to manage the variable fraction and exponent sizes?
**No.** That can be done automatically by the computer."

So it can. For an additional price paid in nanoseconds and picojoules (as listed on his p. 6).

This price is tolerable for computations small enough to fit entirely in the CPU's multi-megabyte on-chip cache. There arithmetic dissipates more time and power than does memory movement, especially when the thousands of extra transistors needed for unum arithmetic, plus their wires and pipeline stages, are taken into account. The book's worked examples of Unum Computation all seem small enough to fit in a modern CPU's on-chip cache. Perhaps this explains why cost-boxes have not been supplied for the book's bigger computations of "c-solutions" and orbits.

A computation costly for "any computer that is bandwidth-limited" must entail *Big Data* — vast collections of numbers that reside in DRAM or beyond, perhaps in the *Cloud*. Moreover, data destined only to be read, not altered, can cost relatively few more address bits besides the data bits; the variability of unums' widths will not much worsen the cost of randomly fetching them.

Address computations for Unum Computations become costly when the values of unum-valued variables can have their widths increased at run-time, as happens in the Fourier Transform. An elaboration of the book's analogy above will help expose these hidden costs. Suppose a sentence

"Things are seldom what they seem."

is #374 in a corpus of 49215 sentences stored packed together consecutively in DRAM or in a file. Each paragraph has an address, say 32 bits wide, that points to the start of the paragraph's first sentence; and each sentence begins with a field that counts the sentence's characters. To fetch a sentence, the computer must find its paragraph, unless it is already known, by loading and searching a table, and then skip along sentences until #374 is reached. Say it changes to

"Things are seldom what they seem; skim milk poses oft as cream."

Where will it be stored? If squeezed where it went before, all subsequent sentences will first be moved down to make way. Otherwise #374 must be pushed onto a *Heap* and its new address put into the old place. Later, *Garbage Collection* (file or memory defragmentation) may occur.

Omitting hidden costs paid for address (re)computations is disingenuous.

# §6: Never Wrong ?

Unum Computation, like interval arithmetic, tends to produce pessimistic estimates of computed results' uncertainties due to uncertain data. Such an estimate is not deemed "Wrong" so long as it encloses the range of the true results, no matter how pessimistic a computed enclosure may be. Grossly excessive pessimism is useless or, worse, misleading, albeit not deemed "Wrong".

Grossly excessive pessimism is often lessened by a *Subdivide-and-Conquer* scheme that costs more computation, sometimes vastly more computation. However, no such scheme, lacking a human's mathematical error-analysis, can ***always*** reduce pessimism from grossly to moderately excessive. Consequently a naive user of Unum Computation cannot know whether his result's oversized uncertainty is deserved by the data or is an incidental by-product of an ill-chosen way to compute what he wishes to know. We shall see how four attempts to alleviate his uncertainty fail.

**Failure Mode I: The Curse of High Dimensions**

Given an interval (ubox) $\mathbf{X}$ containing a d-dimensional uncertain vector $\mathbf{x}$, we wish to compute the uncertainty inherited by $\mathbf{y} := \mathcal{H}(\mathbf{x})$ from a continuous function implemented as a program $\mathcal{H}$. The image $\mathcal{Y} = \mathcal{H}(\mathbf{X})$ is a connected region in y-space. The shape of $\mathcal{Y}$ could be almost arbitrary; it could resemble a pretzel. It may lie inside a computed ubox far larger than necessary.

The Subdivide-and-Conquer scheme diminishes that pessimism by a factor typically near $1/k$ by subdividing the ubox $\mathbf{X}$ by a factor $1/k$ in each dimension to get $k^d$ smaller uboxes; then it (re)computes their unum images to get $k^d$ uboxes whose union contains the desired image $\mathcal{Y}$. If enough processors are available, those $k^d$ images can be computed in parallel. If their union seems still too big, increase $k$. Thus does the Curse of High Dimensions loom over the scheme.

The Subdivide-and-Conquer scheme purports to cope with a phenomenon called "the Wrapping Effect", but for a price. To illustrate the effect in a simple way, we consider a repetitive process

$$\mathcal{H}(\mathbf{x}) := \mathbf{h}^{[N]}(\mathbf{x}) := \mathbf{h}(\mathbf{h}(\mathbf{h}(\ldots\mathbf{h}(\mathbf{h}(\mathbf{h}(\mathbf{x})))\ldots))) \text{ N times}$$

for a big integer $N > 6$. The unum version of $\mathbf{h}$ will produce a ubox $\mathbf{Y}_1 \supseteq \mathbf{h}(\mathbf{X})$ too big by some factor, say $\Lambda > 1$, but not much bigger. Repetition produces a ubox $\mathbf{Y}_N \supseteq \mathcal{Y} = \mathcal{H}(\mathbf{X})$ too big by a huge factor $\Lambda^N$ unless something intervenes to stop the process prematurely. It could stop if $\mathbf{h}$ is nonlinear and has a singularity that $\mathbf{h}(\mathbf{Y}_{N-3})$ evades but $\mathbf{Y}_{N-2} \supseteq \mathbf{h}(\mathbf{Y}_{N-3})$ encloses.

Lest a singularity complicate a simple illustration, let's choose a linear function for $\mathbf{h}$:

$$\mathbf{h}(\mathbf{x}) := H\cdot\mathbf{x} \text{ where matrix } H := \text{hadamard}(20)/\sqrt{20}\,.$$

Here hadamard(n) is MATLAB's Hadamard matrix; this H is a symmetric 20-by-20 matrix whose every element is one of $\pm 1/\sqrt{20}$. Note that $H^2 = I$, the identity matrix. Next let $\mathbf{X}$ be a hypercube-shaped ubox around an uncertain $\mathbf{x}$. The smallest rectangular ubox around $\mathbf{h}(\mathbf{X})$ is a hypercube bigger by a factor $\Lambda := \sqrt{20} \approx 4.47$. Now $\mathbf{Y}_N \supseteq \mathcal{Y} = \mathcal{H}(\mathbf{X}) = H^N\cdot\mathbf{X}$ but grows too big by an enormous factor $20^{(N-1)/2}$. Subdivision by a factor $1/k$ to reduce this gross pessimism to mere moderate pessimism, say a factor of 400, would need $k \geq 20^{(N-5)/2}$ and $k^d \geq 20^{10\cdot(N-5)}$, which is a humongous number of recomputations, well beyond the capabilities of parallelism.

Gustafson is aware of the Wrapping Effect. He devotes ch. 16.2 (pp. 215-219) to dismissing it with a trivial example ($d = 2$, $k^d = 24$) and an appeal to massive parallelism, concluding …

> "This is the essence of the ubox approach. Mindless, brute-force application of large-scale parallel computing …" (p. 219)

My example $\mathcal{H}(\mathbf{x}) = H^N \cdot \mathbf{x}$ is trivial too but does not succumb to "Mindless, brute-force …".

Nontrivial examples abound. For instance suppose a nonlinear $\mathbf{h}(\mathbf{x})$ simulates a minute's motion of a few asteroids and lots of artificial satellites and space junk in motion near the earth. Such a simulation is intended to help predict and, if possible, avoid costly collisions. Starting values of $\mathbf{x}$ are a little uncertain because of errors in observations. Starting from a globular region $\mathbf{X}$ in which the uncertain initial $\mathbf{x}$ lies, $\mathbf{h}(\mathbf{h}(\ldots\mathbf{h}(\mathbf{h}(\mathbf{X}))\ldots))$ evolves first into a slowly tumbling cigar, then a banana, and ultimately into a ring or pretzel after many orbits. The unum version of $\mathbf{h}$ wraps a coffin, a ubox with edges parallel to coordinate axes, around a tilted cigar. Some corners of the coffin spread faster than the cigar, tilt, and must be wrapped again. Exponential growth.

The text's ch. 20 (pp. 287-310) advocates Gustafson's elaborate scheme to compute orbits after disparaging (pp. 292-4) a "traditional method" (4th order Runge-Kutta) that ***no*** experienced practitioners have been using for orbit calculations. The cost of his scheme is nowhere reckoned, and it is performed for only a small fraction of a two-body orbit. Two-body orbits can be plotted without any numerical equation-solving, so their uncertainty grows linearly, not exponentially. See www.eecs.berkeley.edu/~wkahan/Math128/KeplerOrbits.pdf for a much simpler scheme.

Along orbits of three or more bodies, Gustafson's expansion factor $\Lambda$ exceeds 1 only slightly for each short step $\mathbf{h}$. I think this explains his mistaken assertion on p. 306 that "the expansion of uncertainty is roughly linear, not something that grows catastrophically fast." At the beginning of exponential growth it would appear linear. Subdivision of the initial ubox $\mathbf{X}$ to attenuate that growth by a factor $1/k$ would cost $k^{6N}$ parallel recomputations to simulate the motions of $N$ items around the earth; there are thousands of items. I doubt that his scheme is practicable.

"The unum method may show empirical validation of Kahan's observation" says the book on p. 307. This refers without explanation to my schemes that compute tumbling (hyper)ellipsoids that wrap around tumbling cigars more tightly than coffins can, and thus prolong the simulations for at least several orbits before growing excessively pessimistic. The ellipsoids grow too fast but not exponentially too fast. Six-dimensional ellipsoids, one per item being simulated, cost at least six times as much arithmetic and storage as the items' simulation; perhaps this explains why my ellipsoidal scheme has not yet become popular.

How do people get along without unums nor ellipsoids? Numerous samples of initial values $\mathbf{x}$ are drawn from the range of uncertainties, and simulations are computed in parallel, one for each sample, using high-order numerical methods like "Symplectic Integrators" that conserve energy and momenta well enough to avoid the inward or outward spirals derided on p. 292. Arithmetic precision is adequate to render roundoff utterly negligible compared with uncertainties in the data and the equations of motion. If a close encounter is observed among the samples simulated, the simulations are redone with samples distributed more densely around initial conditions that led to the close encounter. Thus do useful predictions become available in real time without unums.

**Failure Mode II: Unbounded Phantom c-Solutions**

"Self-Validating Computation", a method favored by the Interval Arithmetic community, is worth mentioning here though Gustafson does not mention it, perhaps because his "c-solutions" seem more general at first sight. Suppose a solution $\mathbf{z}$ to some problem is sought; perhaps $\mathbf{z}$ is a solution of some equation. Infinitely many equations have exactly the same set of solutions, if any. Self-validating computation works when an equation $\mathbf{z} = \boldsymbol{f}(\mathbf{z})$ can be found that exhibits $\mathbf{z}$ as a fixed-point of a sufficiently *contractive* map $\boldsymbol{f}$. This usually means that there is a matrix norm $\|\ldots\|$ and a constant $\lambda$ such that the Jacobian matrix $\boldsymbol{f}'(\mathbf{x})$ of first partial derivatives satisfies $\|\boldsymbol{f}'(\mathbf{x})\| < \lambda < 1$ for all $\mathbf{x}$ in some palpable neighborhood of $\mathbf{z}$. If $\boldsymbol{f}$ is a little uncertain then $\lambda$ must be appreciably less than 1 so that the interval arithmetic iteration $\mathbf{X}_{n+1} := \boldsymbol{f}(\mathbf{X}_n)$, possibly subdivided, will converge to a region $\mathbf{Z}$ provably enclosing at least one solution $\mathbf{z}$.

Alas, not every equation $Æ(\mathbf{z}) = \mathbf{o}$ is equivalent to an equation $\mathbf{z} = \boldsymbol{f}(\mathbf{z})$ with a contractive map $\boldsymbol{f}$. An instance is an equation whose Jacobian matrix $Æ'(\mathbf{x})$ is not known beforehand to be *singular* at $\mathbf{x} = \mathbf{z}$ (*i.e.*, $\det(Æ'(\mathbf{z})) = 0$). Here is a didactic example for column 2-vectors $\mathbf{x}$:

$$Æ(\mathbf{x}) := \begin{bmatrix} -6 \\ -4 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ 0 & 2 \end{bmatrix} \cdot \mathbf{x} + \begin{bmatrix} \mathbf{x}^T \cdot C_1 \cdot \mathbf{x} \\ \mathbf{x}^T \cdot C_2 \cdot \mathbf{x} \end{bmatrix} /2 \quad \text{in which} \quad C_1 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \quad \text{and} \quad C_2 = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix} .$$

Newton's iterating function $\boldsymbol{f}(\mathbf{x}) := \mathbf{x} - Æ'(\mathbf{x})^{-1} \cdot Æ(\mathbf{x})$ would be a contractive map with a fixed-point $\mathbf{z} = \begin{bmatrix} -6 \\ 4 \end{bmatrix}$ at which $Æ(\mathbf{z}) = \mathbf{o}$, and to which the iteration $\mathbf{x}_{n+1} := \boldsymbol{f}(\mathbf{x}_n)$ converged almost always, except for an inconvenience: $Æ'(\mathbf{z}) = \mathbf{O}$ so $\boldsymbol{f}(\mathbf{z})$ is indeterminate. Worse, $Æ(\mathbf{x}) = \mathbf{o}$ all along the line £ whose equation is $[1,\ 1]\cdot\mathbf{x} = -2$. Therefore no Self-Validating Computation will find a contractive map for any of the infinitely many solutions $\mathbf{z}$ of the equation $Æ(\mathbf{z}) = \mathbf{o}$.

The book's's c-solutions would reveal all solutions $\mathbf{z}$ within any ubox $\mathbf{X}$ big enough. After $\mathbf{X}$ has been subdivided into smaller uboxes, those within which unum evaluations of $Æ(\mathbf{x})$ cannot vanish will be rejected and the remainder further subdivided, and so on, until all the remaining uboxes strung along £ are as small as desired. The process is lengthy, but …

> "This is the essence of the ubox approach. Mindless, brute-force application of large-scale parallel computing …" (p. 219)

Before rejoicing at the success of Gustafson's c-solutions, we must consider the possibility that the coefficients of the function $Æ(\mathbf{x})$ are a little uncertain. This means that $Æ(\mathbf{x}) + \Delta Æ(\mathbf{x})$ is practically indistinguishable from $Æ(\mathbf{x})$ if $\|\Delta Æ(\mathbf{x})\|$ is small enough but not zero. However, no matter how small "small enough" may be, infinitely many such functions $Æ(\mathbf{x}) + \Delta Æ(\mathbf{x})$ never vanish at any real $\mathbf{x}$. This nonexistence does not alter the c-solutions. Then what do they mean?

If ostensibly negligible perturbations of a complicated equation can cause its solution(s) to flicker in and out of existence, how will unum computation warn us about this kind of misbehavior?

Apparently not via c-solutions.

• • • • • • •

**Failure Mode III: Persistent "c-Solutions" that Do Not Exist**

During the search for c-solutions of an equation $Æ(z) = 0$, what must be done with a ubound or ubox X when $Æ(X)$ encounters an invalid operation, perhaps division by zero? A policy that rejects X and searches elsewhere for c-solutions risks rejecting valid solutions of the equation. For instance, $Æ(x) := 3/(x+1) - 2/(x-1) + 1/(x-1)^2$ encounters invalid operations during the evaluation of $Æ([0, 4])$; but rejection of interval $X = [0, 4]$ would reject both finite solutions $z = 2$ and $z = 3$ of $Æ(z) = 0$. Instead, X must be partitioned into subintervals until each is small enough to localize either a solution z or a singularity of Æ as accurately as desired.

Ubounds and uboxes need not include their entire boundaries. Consequently they can avoid divisions by zero in some cases. For instance, the reciprocal of the interval $0 < x \le 1$ is the unbounded interval $1/x \ge 1$, both representable as ubounds. Alas, this capability fails to preclude aberrant behavior near singularities. For instance, "c-solutions" of equations involving rational functions can converge onto arbitrarily tiny uboxes that enclose no solutions of the given equations. Here is a didactic example of the phenomenon designed to be understood easily:

Let $R(x, y) := (x - y)\cdot(x + y)/(x^2 + y^2)$ literally. DO NOT "SIMPLIFY" IT !

We seek c-solutions $(x, y)$, if any, of two equations: $R(x, y) = 1.125$ and $R(y, x) = -1.125$.

Because $-1 \le R(x, y) = -R(y, x) \le +1$ at the plane's every finite point $(x, y)$ other than $(0, 0)$,
no real solution $(x, y)$ exists.

However, subdivision of the plane into uboxes, no matter how tiny, cannot reject every ubox with a corner relatively near enough to $(0, 0)$, though it be excluded from every ubox. Thereon Unum Computation, like interval arithmetic, gets overly wide intervals for R. For example,

$X := [0.01, 0.02]$, $Y := [0, 0.01]$, $R(X, Y) \subset [0, 6]$.

$X := [0.01, 0.011]$, $Y := [-0.001, 0.001]$, $R(X, Y) \subset [0.664, 1.44]$.

Thus, while c-solutions are sought, ever tinier uboxes are found on which Unum Computation overestimates the range of R to include 1.125; and these tiny uboxes $(X, Y)$ converge to but never overlap the singularity at $(0, 0)$. Only a mathematical analysis of the formula for R can defend against acceptance of one of these tiny uboxes as an approximation to a sought solution.

The given equations can be handled properly, but not Gustafson's "Mindless, brute-force" way. One way rewrites $R(x, y) = 1 - 2/(1 + (x/y)^2)$, which works only because R is so simple. A second way multiplies all rational equations by their denominators to convert them to polynomial equations free from singularities; but doing so can introduce spurious c-solutions. The spurious solution here, $(x, y) = (0, 0)$, is easy to recognize as a singularity of R. The singularities of more general equations may be harder to distinguish from legitimate solutions by mere numerical evaluations without mathematical analysis.

C-solutions can solve every equation only by "solving" also some that have no solution.

• • • • • • •

**Failure Mode IV: Illegitimate Unbounded c-Solutions**

Given the Cartesian coordinates **o**, **u**, **v**, **w** of the four vertices of a tetrahedron $\nabla$, we seek its *Incenter* **c**; it is equidistant from $\nabla$'s four faces and on the same side of each face as its opposite vertex. This description translates to three linear equations for the coordinates of column **c** thus:

$\mathbf{p} := (\mathbf{v}-\mathbf{u})\times(\mathbf{w}-\mathbf{u})/\|(\mathbf{v}-\mathbf{u})\times(\mathbf{w}-\mathbf{u})\|$ ; … a column when **u**, **v** and **w** are columns

$M := [\, \mathbf{v}\times\mathbf{w}/\|\mathbf{v}\times\mathbf{w}\| + \mathbf{p}, \quad \mathbf{w}\times\mathbf{u}/\|\mathbf{w}\times\mathbf{u}\| + \mathbf{p}, \quad \mathbf{u}\times\mathbf{v}/\|\mathbf{u}\times\mathbf{v}\| + \mathbf{p}\,]^T$ ; … 3-by-3

$\mathbf{m} := [\mathbf{u},\ \mathbf{v},\ \mathbf{w}]^T\cdot\mathbf{p} = \mathbf{u}^T\cdot\mathbf{p}\cdot[1,\ 1,\ 1]^T$ ;

Solve $M\cdot\mathbf{c} = \mathbf{m}$ for **c**.

Matrix M has $\det(M) \neq 0$ provided tetrahedron $\nabla$ is non-degenerate (has nonzero volume $\mathbf{u}^T\cdot\mathbf{v}\times\mathbf{w}/6$), and then $\mathbf{c} = M^{-1}\cdot\mathbf{m}$. The incenter of a degenerate tetrahedron $\nabla$ is the limit of incenters of any non-degenerate tetrahedra that collapse continuously onto $\nabla$.

Numerical Example: $[\mathbf{u},\ \mathbf{v},\ \mathbf{w}] := \begin{bmatrix} 4182 & 5168 & 4791 \\ 5168 & 6388 & 5922 \\ 4791 & 5922 & 5490 \end{bmatrix}$ has $\mathbf{u}^T\cdot\mathbf{v}\times\mathbf{w} = 36$, so $\det(M) \neq 0$ and the incenter of $\nabla$ is $\mathbf{c} \approx [4789.4057,\ 5920.0275,\ 5488.1688]^T$. However, if each integer entry in the coordinates of $\nabla$ is *independently* uncertain by $\pm 1/2$ then the ubox $\mathbf{X} \supseteq \nabla$ must include some degenerate tetrahedra, and then Gustafson's c-solutions of the equations $M\cdot\mathbf{c} = \mathbf{m}$ derived from **X** must stretch off to infinity though no incenter wanders very far from the **c** exhibited above.

## Infinite pessimism**!**

Infinite pessimism is undeserved. It arises from the choice of a numerically troublesome method to compute incenters. The equation "$M\cdot\mathbf{c} = \mathbf{m}$" is far more sensitive than is the geometry to ill-oriented perturbations. A far better numerical method is a simple explicit formula for **c** that satisfies the equation "$M\cdot\mathbf{c} = \mathbf{m}$" without ever constructing it. The simple formula is hard to find and little known; see p. 26 of www.eecs.berkeley.edu/~wkahan/MathH110/Cross.pdf .

*Do not confuse* ignorance *with* stupidity:
"Against Stupidity even the Gods struggle in vain." J.C.F. von Schiller, 1759 - 1805

A programmer who derives the equation "$M\cdot\mathbf{c} = \mathbf{m}$" to solve for **c** is far from stupid; he is unlucky or too impatient or too beset by deadlines to pore through texts on vectors and geometry.

Unum Computation is no defence against the mistake of choosing an algebraically correct but numerically precarious algorithm, and then accepting grossly oversized computed uncertainties as if they were deserved by the desired results.

• • • • • • •

The four failure modes exhibited above cast a long dark shadow over assertions implying a kind of infallibility for Unum Computation and for Interval Arithmetic. They may beguile the unwary:

"Never Wrong" does not imply "Always Right", so

Unum Computation can't be THE END OF ERROR.

# §7: The Price Paid for Willful Ignorance

On p. 194 Gustafson writes

> "… Since unum bounds resemble interval arithmetic, how do we know they will not suffer the same fate as traditional intervals in producing bounds that are much looser than they should be and need to be? If we already have an algorithm designed for floats, how do we make it work with unums without requiring that the programmer learn about interval arithmetic and its hazards? There is a general solution for this, the *ubox* approach."

Actually, as we have just seen, unum bounds ***can*** suffer from the same excessive pessimism as can traditional intervals. Nevertheless, he goes on in his chs. 15.1 - 15.5, pp. 195 - 210, to demonstrate how his ubox approach estimates the area of a circular disk. His demonstration does injustice to unums and to intervals because it is predisposed to adhere strictly to his *mantra*:

> "This is the essence of the ubox approach. Mindless, brute-force application of large-scale parallel computing …" (p. 219)

Consequently the demonstration panders to ignorant readers who believe they have …

- No need to know any Calculus; it is "evil".
- No need to know about contemporary numerical analysis; it is "deeply unsatisfying".
- No need to know about any arithmetic operations beyond grade-school's $(+, -, \cdot, \div)$.
- No need to know about the costs of arithmetic, data structures, and communications, nor how costs grow when more than one or two sig.dec. of accuracy are needed.

"The deeply unsatisfying nature of classical error bounds" is the heading for ch. 15.2 (pp. 197-9) containing complaints about a formula misquoted on p. 198 thus: [His box]

$$\text{error} \le (b - a)\cdot h^2\cdot|f''(\xi)|/24$$

This "error" is the difference between integral $\int_a^b f(x)\cdot dx$ and its approximation by the *Midpoint Rule* after the interval from a to b has been broken into subintervals each of width h, and $f$ has been sampled at the midpoint of every subinterval. Here $f''(\xi)$ is the second derivative …

> "… at some unspecified point $\xi$ between a and b. To compute the second derivative, we first have to know calculus to figure out what the function $f''$ is, and then we have to somehow find the *maximum possible absolute value* of that derivative over the range from a to b."

Gustafson has exposed his misunderstanding. Alas, he shares it with too many mathematicians assigned to lecture about Numerical Analysis. This misunderstanding will be cleared up later.

Pp. 199-210 demonstrate his ubox approach to the estimation of $\pi/4 = \int_0^1 \sqrt{(1 - x^2)}\cdot dx$ as the area of a quarter of the unit disk. This area is bounded by $x^2 + y^2 \le 1$ inside the square wherein $0 \le x \le 1$ and $0 \le y \le 1$. Initially the square is partitioned into $K^2$ uboxes each a small square of side-length $h = 1/K$ for a chosen positive integer K. The demonstration's $K = 16$. Next the uboxes in the quarter disk are counted. There are two counts: Lo counts the uboxes entirely within the quarter disk; Hi counts the uboxes that intersect with the quarter disk. Whether a ubox deserves to be counted can be decided by computing the predicate "$x^2 + y^2 \le 1$" at the ubox's

upper-right corner for Lo, lower-left for Hi; the demonstration uses a more complicated unum procedure to compute the predicates. Anyway, all $K^2$ uboxes' predicates can be computed in parallel quickly. Then $4\cdot Lo/K^2 < \pi < 4\cdot Hi/K^2$. The demonstration's estimates for $K = 16$ are …

" $2.859375 < \pi < 3.34375$ " .

"*Parturient montes, nascetur ridiculus mus*." Horace (65 - 8 BC) *Ars Poetica*
["The mountains heave in labor to bring forth a silly mouse."]

256 uboxes' predicates seem like too much work for so little as one sig.dec. of accuracy. What would 5 sig. dec. cost? The book doesn't say. Let's find out:

It so happens that $Hi - Lo = 2\cdot K - 1$, so the width of that interval estimate for $\pi$ is $8/K - 4/K^2$. To achieve at least 5 sig.dec. of accuracy would require $K > 80{,}000$ roughly. This would require predicates for humongously many, over $64\cdot 10^8$, uboxes if "Mindless brute-force …" were the only option. It isn't. At the bottom of p. 209 the book mentions "*Grid refinement*" but offers no program for it, leaving unknown its two costs:

<> The time taken to write a parallel version of Grid refinement, with load balancing.
<> How many uboxes' predicates the program must compute, as it depends upon K.

At least $2\cdot K - 1$ uboxes have to be located; these are the ones whose predicates "$x^2 + y^2 \le 1$" are true at the lower left corner, false at the upper right. They can be located by tracing the arc of the quarter circle, a purely sequential process. For 5 sig.dec., $2\cdot K - 1 > 160{,}000$. Too huge.

"Mindless brute-force" is a costly way to compute any but the crudest estimates of integrals; and sometimes it cannot provide any estimates at all. For instance, take an ellipse's circumference:

This circumference is needed to compute the weight of steel tubing intended for a racing bicycle. The tubing's cross-section is elliptical instead of circular to reduce weight while retaining enough resistance to anticipated loads. Let $x^2 + y^2/4 = 1$ be the ellipse's equation. A little calculus gets an expression $L = 4\cdot\int_0^1 \sqrt{(1 + (dy/dx)^2)}\cdot dx = 4\cdot\int_0^1 \sqrt{((1 + 3\cdot x^2)/(1 - x^2))}\cdot dx$ for the circumference L of the ellipse. It is an *improper* integral because the integrand rises to $+\infty$, so "Mindless brute-force" would have to count infinitely many uboxes. It can't. We will compute L later.

• • • • • • •

Now let's clear up the misunderstanding of the misquoted formula in the box above. It should say

$$\text{(Midpoint Rule)} - \int_a^b f(x)\cdot dx = (b - a)\cdot h^2\cdot f''(\xi)/24 \quad \text{and}$$
$$\int_a^b f(x)\cdot dx - \text{(Trapezoidal Rule)} = (b - a)\cdot h^2\cdot f''(\eta)/12 .$$

Here $f''(\xi)$ and $f''(\eta)$ are differently weighted averages of the second derivative $f''(x)$ over x between a and b. The weights are positive but not constant. If $f''(x)$ is bounded throughout the range of integration, each Rule's error ultimately approaches zero no slower than $h^2$ as h — the width of every subinterval in the interval of integration — approaches zero. An algorithm suggested by the formulas repeatedly doubles the number of subintervals, combines new samples of $f$ with old, and gets new Trapezoidal and Midpoint estimates each nearer the integral than the old by factors ultimately at most 1/4. Other than that $f''$ be bounded, it need not be known to estimate the integral as closely as desired, though unpredictably many samples may be needed.

The boxed formulas tell anyone who seeks guaranteed interval or ubound estimates for an integral something very much worth knowing:

> If $f''(x)$ does not reverse sign between $a$ and $b$, then $\int_a^b f(x)\cdot dx$ lies between its estimates provided by the Midpoint and Trapezoidal Rules.

The two estimates differ by $(b-a)\cdot h^2\cdot f''(\omega)/8$ for another positively weighted average $f''(\omega)$. If $f''(x)$ reverses sign at some $x = z$ between $a$ and $b$, compute the integral as a sum of two:

$$\int_a^b f(x)\cdot dx = \int_a^z f(x)\cdot dx + \int_z^b f(x)\cdot dx .$$

A computer program can turn the program for $f(x)$ into a program for $f''(x)$ and locate a zero $z$ within an error smaller than $h^2/|b-a|$, which is small enough to not matter. The Rules' samples of $f(x)$ can be computed in parallel in batches, each batch twice as big as the previous one.

Let's compute Gustafson's integral $\pi = 4\cdot\int_0^1 \sqrt{(1 - x^2)}\cdot dx$ using the Midpoint and Trapezoidal Rules. Although the integrand's second derivative never reverses sign, it is unbounded; this causes the gap between the two Rules to shrink like $h^{3/2}$ instead of $h^2$, producing the interval

$$3.141580 \le \pi \le 3.141596 \text{ after } K = 4097 \text{ samples of } \sqrt{((1-x)\cdot(1+x))} .$$

To produce an integrand free from infinite derivatives and inflection points (where the second derivative reverses sign) let us substitute $x = \sqrt{(1 - w^2)}$ and perform some extra calculus and algebra to get a very proper integral $\pi = 4\cdot\int_0^{1/\sqrt{2}} dw/\sqrt{(1 - w^2)}$ and then compute the interval …

$$3.141582 \le \pi \le 3.141613 \text{ after } K = 257 \text{ samples of } 1/\sqrt{(1 - w^2)} .$$

A little of the "evil" Calculus has beaten "Mindless brute-force" by orders of magnitude.

• • • • • • •

The most powerful methods of numerical integration have not (yet) been adapted to produce fully guaranteed interval estimates. These powerful methods were first suggested in the 1960s by a physicist Charles Schwartz in Berkeley. Then his methods were further developed in the 1970s and 1980s by Takahashi and Mori in Japan. For details and pointers to long bibliographies see

D.H. Bailey, K. Jayabalan, X.S. Li "A Comparison of Three High-Precision Quadrature Schemes" pp. 317-329 of *Experimental Mathematics* **14** #3 (2005).

A primitive version of these methods was stuffed into some 1980s hand-held calculators; see

W. Kahan "Handheld Calculator Evaluates Integrals" pp. 23-32 of *The Hewlett-Packard Journal* Aug. 1980.

This article includes advice about avoiding the hazards of fast non-interval integration schemes.

Because my calculator can cope with mildly improper integrals it gets

$$\pi = 4\cdot\int_0^1 \sqrt{(1 - x^2)}\cdot dx \approx 3.14159 \pm 0.00002 \text{ after } 31 \text{ samples of } \sqrt{((1-x)\cdot(1+x))} ; \text{ and}$$

$$\pi = 4\cdot\int_0^{1/\sqrt{2}} dw/\sqrt{(1 - w^2)} \approx 3.14159 \pm 0.000014 \text{ after } 31 \text{ samples of } 1/\sqrt{(1 - w^2)} .$$

Improper integrals like the ellipse's $L = 4\cdot\int_0^1 \sqrt{\left((1 + 3\cdot x^2)/(1 - x^2)\right)}\cdot dx$ pose severe challenges for numerical integration schemes because they have to avoid sampling an integrand at its pole, lest $\infty$ overwhelm everything else, but must sample densely enough near the pole to appraise its contribution to the integral. The appraisal's accuracy is limited to a fraction of the arithmetic's precision depending upon the pole's strength unless the pole rises at $0.0$, in which case over/-underflow may cramp the appraisal's accuracy. Better remedies are suggested in my 1980 article.

One of them is a change to the variable of integration. In L the substitution $x = 1 - w^2$ yields
$L = 4\cdot\int_0^1 \sqrt{((12\cdot(1 - w^2)^2 + 4)/(2 - w^2))}\cdot dw \approx 9.68844822 \pm 0.00000003$ after 127 samples
on the calculator. A tedious interval estimate is feasible because the integrand's second derivative reverses sign at only one point $w \approx 0.6746936853\ldots$; I have not programmed it.

An experimental MATLAB program that combines the calculator's stopping criterion with the sampling strategy of Takahashi and Mori has coped with mildly improper integrals. It gets …
$L = 4\cdot\int_0^1 \sqrt{((1 + 3\cdot x^2)/(1 - x^2))}\cdot dx \approx 9.6884482 \pm 0.0000006$ after 129 samples, but better
$L = 4\cdot\int_0^1 \sqrt{((12\cdot(1 - w^2)^2 + 4)/(2 - w^2))}\cdot dw \approx 9.68844822054768 \pm 10^{-14}$ after 257 samples.

Despite the appearances of "±…", the foregoing estimates obtained from relatively few samples are not produced by interval arithmetic. Gustafson would call them "guesses". They are very good guesses computed by programs that can be foiled; my 1980 article shows how. Whether the risk is tolerable depends upon the value of a prompt result and the costs of other options, and these depend upon the accuracy desired. For high accuracy, here are some indications of the costs of almost surely correct results soon *vs*. three ways to get certainly correct results later:

Errors $\to 0$ like $\exp(-\text{Const}\cdot K)$ for Takahashi-Mori methods drawing K samples.

Error-bounds $\to 0$ like $K^{-2}$ for interval arithmetic in Trapezoidal and Midpoint Rules.
Error-bounds $\to 0$ like $K^{-1}$ for Gustafson's square-counting, but following the arc.
Error-bounds $\to 0$ like $K^{-1/2}$ for Gustafson's "Mindless brute-force" square-counting.

• • • • • • •

To whom is Gustafson trying to sell "Mindless brute-force"?

It may be an appropriate way to estimate the content (area or volume) of a region whose boundary is very complicated, like a Rorschach test. For such a task the costs and benefits claimed for Unum Computation *vs*. short precision floating-point are practically irrelevant.

No. His pitch seems aimed at someone who wishes to compute an integral without having to know anything about contemporary numerical analysis, and without having to look up *Numerical Quadrature* in a book or a software Math. library.

To sell *THE END OF ERROR* to that person panders to ignorance.

# §8: Flogging a swing

The book's long chapter 19, "Pendulums done correctly" pp. 273-286, begins with a photo of a little girl enjoying a swing on a sunny late autumn day. Leaves have fallen off trees or changed color. The caption under the photo reads

> "When physicists analyze pendulums, they prefer to talk about 'small oscillations.'
> Have you ever met a child who didn't prefer the large kind?"
>
> [Has Gustafson ever met a physicist who ***prefers*** to talk about small oscillations?]

Gustafson's snide caption sets the chapter's tone. Without ever exhibiting a differential equation (since Calculus is evil), he sneers at a linearization $d^2\theta/d\tau^2 \approx -\theta$ of the pendulum's differential equation $d^2\theta/d\tau^2 = -\sin(\theta)$. The linearized pendulum's deflection from the vertical, $\theta \approx \sin(\tau)$, does err when bigger than infinitesimal. Let $\Theta$ be $|\theta|$'s maximum amplitude. If $\Theta \le \pi/2$, the pendulum's period $P(\Theta)$ does exceed the linearized period $2\pi$ by very roughly $\tan^2(\Theta/2)$. His method will eliminate this error while tolerating errors that are worse, as we shall see.

The chapter explains his more accurate computation of the pendulum's motion using only "grade school algebra. Without deep human thought but with brute force computing, … This shows why it may be time to overthrow a century of numerical analysis." (p. 281.) Actually, he uses more than grade school algebra's rational operations $(+, -, \cdot, \div)$ because he uses middle school's $\sqrt{\ }$ and high school's $\sin(\theta)$ or $\cos(\theta)$ frequently; and he takes *something crucial* for granted:

He *knows* that the pendulum's total energy, kinetic plus potential, is conserved.

*How does he know that?* The concepts of kinetic and potential energies came into existence with the Calculus, and it is needed to deduce that their total is conserved by a friction-free pendulum. In dimensionless units, the conserved total energy is $(d\theta/d\tau)^2 + 4\cdot\sin^2(\theta/2) = 4\cdot\sin^2(\Theta/2)$. To confirm this, differentiate the left-hand side and invoke the differential equation for $d^2\theta/d\tau^2$.

Determined by initial conditions, the total energy tells us that $\theta$ will vary between $\pm\Theta$, and that the time elapsed between two deflections $\theta(\tau^\circ) = \theta^\circ$ and $\theta(\tau) = \theta$ can be computed from …

$$\tau(\theta) - \tau(\theta^\circ) = \int_{\theta^\circ}^{\theta} \frac{|d\alpha|}{\sqrt{4\sin^2(\Theta/2) - 4\sin^2(\alpha/2)}} = \int_{\theta^\circ}^{\theta} \frac{|d\alpha|}{\sqrt{4\cdot\sin((\Theta-\alpha)/2)\cdot\sin((\Theta+\alpha)/2)}} \ .$$

This is an *Improper* integral. As $\theta$ runs back and forth between $\pm\Theta$, the variable of integration $\alpha$ reaches $\pm\Theta$ and the integrand peaks up to $+\infty$, though the integral remains finite. This singularity reflects what happens when, at the extremes $\theta = \pm\Theta$ of its swing, the pendulum's $\theta$ reverses but time $\tau$ doesn't. The substitution $\alpha = \pm(\Theta - 2\xi^2)$ removes the singularity. For instance, the integral for the pendulum's period $P(\Theta) := 2(\tau(\Theta) - \tau(-\Theta))$ turns into …

$$P(\Theta) = 4\cdot\int_0^{\Theta} \frac{|d\alpha|}{\sqrt{4\cdot\sin((\Theta-\alpha)/2)\cdot\sin((\Theta+\alpha)/2)}} = \int_0^{\sqrt{\Theta/2}} \frac{8d\xi}{\sqrt{\sin(\Theta-\xi^2)\cdot\sin(\xi^2)/\xi^2}} \ .$$

This last integral is easy to evaluate numerically for any given positive numerical value $\Theta < \pi$. The $[\int_y^x]$ key on my old hp-15C shirt-pocket calculator, carrying 10 sig.dec. (though each

keystroke may use a 13 sig.dec. scratchpad), gets 9 sig.dec. of $P(\Theta)$ with an error-estimate. For example, $P(10^{-6}) = 6.283185307 \pm 1.34\cdot10^{-9}$ ; $P(\pi/3) = 6.743001419 \pm 1.51\cdot10^{-9}$ ;
$P(\pi/2) = 7.416298709 \pm 9.6\cdot10^{-10}$ ; $P(3) = 16.15553937 \pm 9.1\cdot10^{-9}$ . ( $P(\pi) = +\infty$ )
(Multiply integrals by $\pi/180$ if angles $\Theta$, $\alpha$ and $\theta$ are specified in degrees instead of radians.)

Without ever exhibiting an integral, the book does compute interval estimates of the integrand for elapsed times $\tau(\theta+\Delta\theta) - \tau(\theta)$ as areas under curves obtained by approximating $d\tau/d\theta$ as a function of $\theta$ over short sub-intervals $[\theta, \theta+\Delta\theta]$ taking account of total energy's conservation:
$d\theta/d\tau = \pm2\sqrt{\left( \sin^2(\Theta/2) - \sin^2(\theta/2) \right)}$ as an *Interval* over a short sub*Interval* $[\theta, \theta+\Delta\theta]$ .
Also used is an interval estimate of $d^2\theta/d\tau^2 = -\sin(\theta)$ over the subinterval. This estimate may be needed in case the interval $[d\theta/d\tau]$ includes 0 , just as the substitution $\alpha = \pm(\Theta - 2\xi^2)$ was used above to remove the integrand's singularity. Thus the book gets a quadratic in $\delta\tau$ with interval coefficients, namely $\delta\theta = [d\theta/d\tau]\cdot\delta\tau + [d^2\theta/d\tau^2]\cdot\delta\tau^2/2$ , to estimate the distance $\delta\theta$ from $\theta$ towards $\theta+\Delta\theta$ traversed by the pendulum in any sufficiently short time $\delta\tau$ . For an interval that covers the time $\Delta\tau$ taken to traverse all of the subinterval $[\theta, \theta+\Delta\theta]$ , solve a quadratic equation $\Delta\theta = [d\theta/d\tau]\cdot\Delta\tau + [d^2\theta/d\tau^2]\cdot\Delta\tau^2/2$ for $[\Delta\tau]$ . [Middle school algebra, not grade school.]

It all seems an extremely elaborate way to estimate $\Delta\tau = \int_\theta^{\theta+\Delta\theta} d\alpha/\sqrt{(\dots)}$ as an interval $[\Delta\tau]$ about $\Omega(\Delta\theta^3)$ wide. The book offers no estimate of $[\Delta\tau]$'s width. Instead we find on p. 277

*All the traversal time bounds can be computed in parallel.*

> "This is a stunning result, because it means we not only get rigorous bounds on the physical behavior, but we can use *as many processors as we have* in a computer system to get any desired answer quality." [His *italics*.]

Gustafson goes on to disparage purely serial computations that simulate physical phenomena:

> "…, the time dependency of physical simulations has been misused as an excuse not to change existing serial software to run in parallel. It is now time to retire that excuse, …"

Bunkum**!** All that braggadocio merely distracts readers from questions the book never mentions:
- If the "answer quality" is not yet as desired, how much more will a better answer cost?
- Why does Gustafson's scheme malfunction for angles $\Theta > \pi/2$ ? (His example's $\Theta = \pi/3$ .)
- He promised just "grade school algebra". Instead trig functions are computed repeatedly.
- What does he do if he doesn't know what, if anything, the differential equation conserves?

**Quality *vs*. Cost:** Starting from $\theta := -\Theta = 60° = \pi/3$ at time $\tau(-\Theta) := 0$ , the book twice plots sets of interval estimates for the elapsed time $\tau$ , about $2P(\Theta)$ , that $\theta$ takes to reach $-\Theta$ twice more in steps $\Delta\theta$ . The book says the first graph's $\Delta\theta = 10°$, which would take 72 steps; but I could see only 47 steps with $\Delta\theta = 15°$ on p. 286. It says the second graph's $\Delta\theta = (1/16)°$; this takes 11520 steps that merge into a continuous curve on the printed page. Neither graph comes with a cost-box; we cannot know their costs in bits moved. However, we can guess that costs are at least proportional to the numbers of steps. Neither graph comes with a statement of the widths of the interval estimates $[\tau]$ of elapsed times; we will have to estimate their widths.

I think Gustafson's scheme has what numerical analysts call "2nd order": By decreasing a sufficiently small stepsize $\Delta\theta$ to $\Delta\theta/k$, which increases the number of steps by a factor $k$, the computed result's uncertainty or error is decreased by a factor near $1/k^2$ unless roundoff gets in the way. The computation's cost increases by the same factor $k$ as the number of steps unless arithmetic of higher precision is invoked, in which case the cost increases by a bigger factor.

The first graph printed on p. 286 is about 112 mm. long; its last $[\tau]$ is about 1 mm. wide, implying a measured uncertainty of about $\pm 0.06$ in $\tau = 2\cdot P(\pi/3) \approx 6.743$. The second graph's $\Delta\theta = (1/16)°$ is smaller than the first's by a factor $1/k = (1/16)/15 = 1/240$, reducing the first graph's uncertainty $\pm 0.06$ to about $\pm 0.06/240^2 \approx \pm 0.000001$ at the cost of a computation over 240 times longer — 11520 steps — unless vastly many parallel processors were previously idle.

**Malfunction for $\Theta > \pi/2$ (90°):** Gustafson's scheme produces utterly wrong results for swings beyond 90° because, after the swing rises above the level of its pivot, it does not reverse its rising trajectory but drops abruptly or else continues to spin around its pivot. This happens to the young girl on a swing hanging by chains, and to a pendulum hanging by "a string of length *L*" as is prescribed on p. 274. What happens after a drop is hard to predict — perhaps a fall off the swing, perhaps some bouncing, perhaps a break. A rigid pendulum would not drop.

Gustafson has overlooked the case when, for a range of initial energies, the pendulum's bob attached by a string will depart from the circular path he predicts and follow a parabolic ballistic path until it intersects the circle again. His "grade school algebra" has not coped with this case.

> "Why beholdest thou the mote that is in thy brother's eye, but considerest not the beam that is in thy own eye?" *Matthew* 7:3

**Only grade school algebra:** Gustafson's promise to use only "grade school algebra. Without deep human thought but with brute force computing, …" *can* be fulfilled; but understanding how and why will require a little evil Calculus and some of "a century of numerical analysis." First,

$$d\theta/d\tau = \phi\,, \quad d\phi/d\tau = -\sigma\,, \quad d\sigma/d\tau = \gamma\cdot\phi\,, \quad d\gamma/d\tau = -\sigma\cdot\phi$$

are the rigid pendulum's differential equations for the deflection angle $\theta$, its velocity $\phi$, and $\sigma = \sin(\theta)$ and $\gamma = \cos(\theta)$, all functions of time $\tau$ initialized at, say, $\tau = 0$. Of course we expect $\sigma^2 + \gamma^2 = 1$, and this relation will be conserved by our numerical procedures that we shall construct to approximate the four functions. Call their approximations $th \approx \theta(\tau)$, $ph \approx \phi(\tau)$, $sn \approx \sigma(\tau)$ and $cs \approx \gamma(\tau)$; and $\tau$ is the fifth variable. Our first procedure, named up2, updates (overwrites) all five variables; it adds any given $\Delta\tau$ to $\tau$ and then computes four updated values $th \approx \theta(\tau+\Delta\tau)$, *etc*. using no more than grade school algebra. Here it is:

```
Procedure up2(Δτ, τ, th, ph, sn, cs) := {
    τ := τ + Δτ ;
    th := th + ph·Δτ/2 ;
    ph := ph – sn·Δτ/2 ;
    { β := ph·Δτ/2 ;   [cs, sn] := [cs, sn] – [β·cs+sn, β·sn–cs]·2β/(1 + β²) ; } ;
    ph := ph – sn·Δτ/2 ;
    th := th + ph·Δτ/2 ; }.
```

Procedure up2 is *Anadromic* in the sense that two successive calls, up2(Δτ, τ, th, ph, sn, cs) followed immediately by up2(–Δτ, τ, th, ph, sn, cs) , restore all five variables to their former values. Thus up2 conserves a crucial property possessed by every ordinary differential equation:

Running a solution from τ to τ+Δτ and then back from τ+Δτ to (τ+Δτ) – Δτ returns to the solution at τ, retracing the solution's path exactly but for roundoff.

Though most numerical algorithms are not Anadromic, all the ordinary differential equations of classical Mathematical Physics and Physical Chemistry can be approximated by Anadromic numerical methods regardless of what else, if anything, those differential equations conserve. Here up2 conserves $cs^2 + sn^2$, and all using only grade school's rational operations (+, –, ·, ÷).

Like Gustafson's scheme, up2 has 2nd order. This means that if k steps Δτ = T/k are used to advance all five variables from τ = 0 to τ = T, then errors in th*(*T*)* and ph*(*T*)* will be nearly proportional to $T{\cdot}\Delta\tau^2 = T^3/k^2$ provided k is big enough, yet not so big that roundoff interferes.

To render roundoff harmless until th and ph are correct to within several units in the last digits carried by the arithmetic, either store all five variables to twice arithmetic's precision, or else use *Compensated Summation*, a trick published in 1960 used now by savvy Numerical Analysts.

***Digression*:** Compensated Summation will be illustrated by application to a silly sum Gustafson uses on p. 120 to justify what unums do as intervals do, namely, convey numerical uncertainty via their widths. He adds up 1.0 a billion times into a *Float* variable holding 24 sig.bits, about 7 sig.dec., and expresses faux chagrin when most of the addends fall off the variable's right-hand end and get rounded away after that variable gets big enough, which is far less than a billion.

| Crude Program | With Compensated Summation | All in *Float*s |
|---|---|---|
| sum := 0.0 ; | sum := 0.0 ; comp := 0.0 ; | |
| for i = 1 to 1000000000 do **{** | for i = 1 to 1000000000 do **{** | |
| sum := sum + 1.0 ; **}** | comp := comp + 1.0 ; oldsum := sum ; | |
| | sum := oldsum + comp ; | |
| | comp := (oldsum – sum) + comp ; **}** | |
| Printout: sum is 16777216.0 = $2^{24}$ | sum is 1000000000.0 = $10^9$ exactly | |

***End of Digression***

Convergence as Δt → 0 is slow at 2nd order. For smooth solutions of differential equations, higher accuracies are usually achieved sooner with 4th order schemes. Here is such a scheme:

```
Procedure up4(Δτ, τ, th, ph, sn, cs) := {
    δτ := Δτ/6 ;
    for j = 1 to 4 do call up2(δτ, τ, th, ph, sn, cs) ;
    call up2(–2·δτ, τ, th, ph, sn, cs) ;
    for j = 1 to 4 do call up2(δτ, τ, th, ph, sn, cs) ; } .
```

This up4 is the simplest 4th order Anadromic updating scheme, not the most efficient; and it uses only grade school algebra, no repeated calls upon the Math. Library's cos*(…)* nor sin*(…)* .

At first up4 seems constrained to answer the question "Where will the pendulum be at time T ?" using k calls upon up4 with Δτ = T/k and k big enough. However, up4 lets Δτ vary.

Instead of advancing $\tau$ in steps $\Delta\tau$ until $\tau = T$, up4 can be used to advance the numerical approximations $th(\tau)$ and $ph(\tau)$ until they satisfy some other condition. To match the book's results we initialize $th(0) := -\pi/3$, $ph(0) := 0$, $cs(0) := 0.5 = \cos(-\pi/3)$, $sn(0) := -\sqrt{0.75}$, and we stop when $ph(T) = 0 > th(T)$ for the third time, making $T \approx 2\cdot P(\pi/3)$. Knowing only that period $P(|th(0)|) > 2\pi$, we can try $\Delta\tau := 4\pi/k$ for $k \geq 4$ initially, say, and double k as often as necessary until successive re-estimates $T_k$ nearly settle down. How nearly is nearly enough?

While up4's convergence at 4th order implies $(T_k - T_\infty)/(T_{k/2} - T_\infty) \approx 1/2^4$ for *every* k big enough, it implies $(T_{2k} - T_k)/(T_k - T_{k/2}) \approx 1/2^4$ too, and *vice-versa*. These approximations provide an estimate $T_\infty \approx T_{2k} + (T_{2k} - T_k)/(2^4 - 1)$ acceptable when $|T_{2k} - T_k|$ is almost small enough and $(T_{2k} - T_k)/(T_k - T_{k/2}) \approx 1/2^4$.

The following process approximates $2\cdot P(\Theta)$ for a rigid pendulum starting at $th(0) := -|\Theta|$ :

```
Function  T = prd2(Θ, tol) := {  %… approximates  2·P(Θ) ± tol  for  |Θ| < π .
if  |Θ| ≥ π ,  ErrorStop( " prd2(Θ)  needs  |Θ| < π " ) ;
τ° := 0 ;  th° := –|Θ| ;  ph° := 0 ;  cs° := cos(th°) ;  sn° := –√((1 – cs°)·(1 + cs°)) ;
if sn° = 0 ,  return( T := 4·π ) ;
 k := 4 ;  Δτ° := 8·π/k ;   oT := -4 ;   T := -1 ;   K := 0 ;
do { Δτ° := Δτ°/2 ;  Δτ := Δτ° ;  ooT := oT ;  oT := T ;
     τ := τ° ;  oτ := τ ;  th := th° ;  ph := ph° ;  sn := sn° ;  cs := cs° ;  L := -1 ;
     do while ( (L < 2) or (τ ≠ oτ) ) {  oτ := τ ;  %… advance  τ  to  τ+Δτ
          call up4(Δτ, τ, th, ph, sn, cs) ;  K := K+1 ;   %…  K  counts calls on  up4 .
          if ( |L| = 1 ),  { if ((sn < 0) & (ph < 0)),  L := L+1 ; }
            elseif ( L = 0 ),   { if (ph > 0),  L := L+1 ; }
            else  { Δτ := min( Δτ°,  ph/sn ) ; } ; }   %…  after  L = 2  in  2nd period
     T := τ ;   %…  τ  solved  " ph(τ) = 0 "  by cubically convergent  Newton  iteration
   } until ( ( |T – oT| < 10·tol ) & ( |(T – oT)/(oT – ooT) – 2^–4| < 10^–2 ) ) ;
return( T := T + (T – oT)/15 ) .  %…  approximately  2·P(Θ) ± tol .
```

Do up2 and up4 conserve the rigid pendulum's total energy $ph^2 + 4\cdot\sin^2(th/2)$ ? Not exactly. It neither accretes nor decays but fluctuates periodically within bounds separated by an amount that shrinks like $\Delta\tau^{Order}$ provided $\Delta\tau$ is not too big. This conforms to a theorem published several years ago by Prof. J. Marsden and a collaborator at Cal. Tech.:

> A scheme that predicts approximately the behavior of a friction-free mechanical system after any given elapsed time, and that also conserves both total energy and all momenta, must be exact, not approximate. [Exact schemes are uncommon in numerical work.]

Neither do up2 and up4 conserve the relations $cs = \cos(th)$ and $sn = \sin(th)$ exactly; they too are conserved approximately. To some extent th is redundant, since it can be recovered after many updates from a formula like $th \approx 2\cdot\arctan((sn/(1 + |cs|))^{sign(cs)})$, in which case the two lines " th := th + ph·Δτ/2 ; " could be deleted from up2 to save a little time. And then, if an approximation for total energy is needed after updates, it too could be approximated by

$$ph^2 + 4\cdot\sin^2(th/2) \;\approx\; e := ph^2 + 2\cdot sn\cdot(sn/(1 + |cs|))^{sign(cs)} .$$

Numerical Results from a MATLAB 5.2 version of $prd2(\Theta) \approx 2 \cdot P(\Theta)$ on a Mac Quadra 950:

$prd2(\pi/3) = 13.486003 \pm 0.0000013$ in 156 calls upon up4 ; e fluctuated by $\pm 0.0000006$
$= 13.48600284 \pm 10^{-8}$ in 296 calls upon up4 ; e fluctuated by $\pm 3.7 \cdot 10^{-8}$
$= 13.486002838501 \pm 1.5 \cdot 10^{-12}$ in 2226 calls upon up4 ; e fluctuated by $\pm 10^{-11}$

$prd2(\pi/2) = 14.832597 \pm 2.2 \cdot 10^{-6}$ in 167 calls upon up4 ; e fluctuated by $\pm 1.2 \cdot 10^{-6}$
$= 14.83259741841 \pm 2.4 \cdot 10^{-11}$ in 1233 calls upon up4 ; e fluctuated by $\pm 3 \cdot 10^{-10}$

$prd2(3) = 32.311079 \pm 5 \cdot 10^{-6}$ in 343 calls upon up4 ; e fluctuated by $\pm 2.5 \cdot 10^{-6}$
$= 32.311078744787 \pm 7 \cdot 10^{-11}$ in 2655 calls upon up4 ; e fluctuated by $6 \cdot 10^{-10}$

The extravagantly accurate values are exhibited here only to corroborate the ±error estimates of the less accurate values. Further corroboration comes from the calculator's integrations exhibited four pages ago. Compare the ±error estimate of prd2 after our 156 steps with the interval estimate of ±0.000001 the book did not mention after 11520 steps. Our ±error estimates were derived using the evil Calculus, using ideas that were misrepresented along with the misquoted formula from the book's p. 198. We addressed that on p.22 above. What matters here is less that our method can reproduce the book's at a far lower computational cost, more that …

Gustafson's method is incorrigibly unrealistic.

**If Energy is NOT Conserved:** Gustafson intentionally disregards friction; instead he says

> "(We can ignore air resistance and assume someone is giving the child a slight push at the end of each cycle to keep the amplitude at ±60°.)" p. 275

Why disregard friction? How slight a push suffices to keep a young girl happy and unafraid? (A typical young boy may ask for a harder push.)

Without conservation of energy, time $\tau$ is not an integrable explicit elementary function of $\theta$. His time-bounds can no longer all be computed in parallel because friction shortens successive swings, so the range of values to be taken by $\theta$ is unknown until the pendulum's motion has been simulated. This simulation must compute $\theta$ as a function of $\tau$, instead of Gustafson's $\tau$ as a function of $\theta$. His disparagement on p. 277 of such sequential simulations resembles his sneer at backward error-analysis. We could paraphrase it thus:

> You asked **"*Where* will it be *then* ?"**
> I cannot give you the answer you requested, and it is all your fault, because you *should* have asked a slightly different question. I gave you a perfect answer to the question you should have asked: **"*When* will it be *there* ?"** *Cf.* his box from p. 76.

But what if (as Gertrude Stein said of Oakland CA) there is no ***there*** there?

Our numerical method does not assume Conservation of Energy, and can account for friction as Gustafson's method cannot. The pendulum's differential equation $d^2\theta/d\tau^2 = -\sin(\theta)$ changes to

$$d^2\theta/d\tau^2 = -\sin(\theta) - \rho \cdot (d\theta/d\tau) \cdot |d\theta/d\tau| \quad \text{for some small } \rho > 0$$

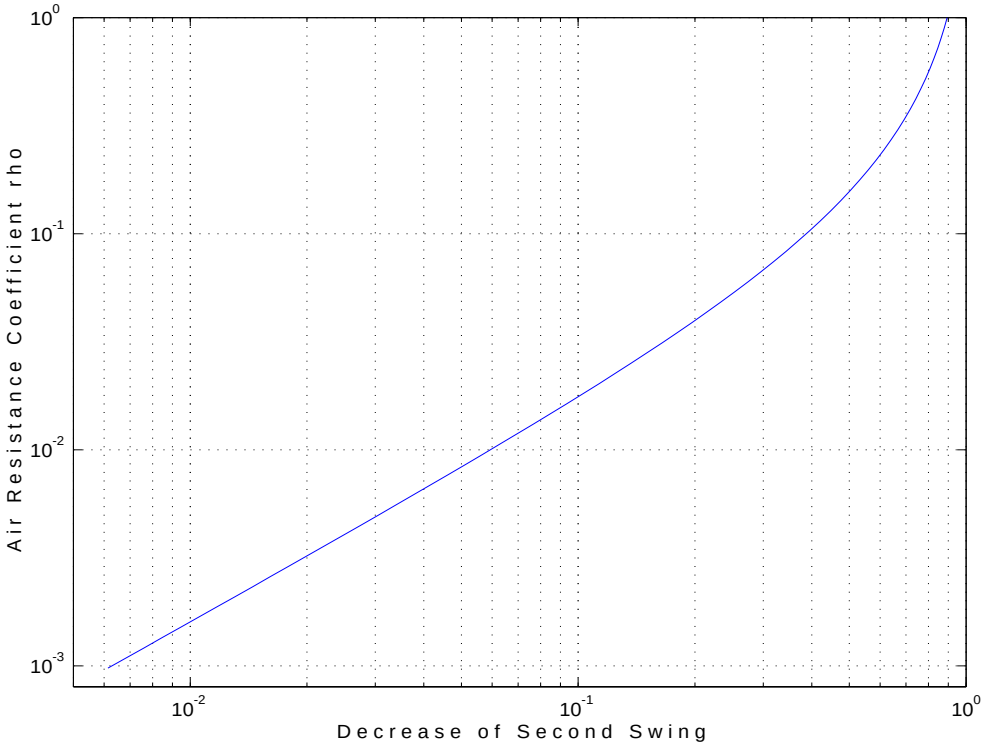
at small air velocities $|d\theta/d\tau|$. The constant $\rho$ depends in a complex way upon the girl's shape and size and the air's density. Ideally $\rho$ could be determined by experiments in a wind tunnel.

Rather than put a little girl in a wind tunnel, we can estimate $\rho$ by observing how the amplitudes of the swings decrease from one swing to the next, assuming the swing's pivot is lubricated well enough to render friction there negligible compared with air resistance. Then it changes both appearances above of " ph := ph – sn·Δτ/2 ; " in procedure up2 to

$$ph := ph - (sn + \rho \cdot ph \cdot |ph|) \cdot (\Delta\tau/2) / (1 + \rho \cdot |ph| \cdot \Delta\tau/2) ;$$

which is only approximately Anadromic with a negligible departure of the order of $\rho \cdot ph^2 \cdot sn \cdot \Delta\tau$ just when ph reverses sign from a value of the order of $\Delta\tau$. Also changed is prd2 ; for each value of $\rho$, prd2 must stop to deliver $th \approx \theta(\tau)$ when $\tau > 0$ and $d\theta/d\tau = \phi \approx ph = 0 > th$ for the second time after starting from, say, $ph = -\Theta = -\pi/3 = 1.047197551\ldots$ .

Here is $\rho$ plotted *vs*. the difference between this second extreme $\theta < 0$ and initial $-\Theta = -\pi/3$ :




The foregoing computation can exploit parallelism if a separate thread is allocated to each value of $\rho$ though each thread must compute its $\theta(\tau)$ sequentially because its terminal values of $\tau$ and $\theta$ are unknown initially. *Ditto* for interval arithmetic's Self-Validating Computation. The graph cannot be obtained from Gustafson's methods; they rely upon conservation of energy and, in a later chapter, momenta, to reduce the dimensions or order of a differential equation. His scheme is inapplicable when he does not know what is conserved, with or without the evil Calculus.

By ignoring friction he laboriously computes rigorous bounds for unrealistic physical behavior.

Not yet is it "time to overthrow a century of numerical analysis" in favor of "Mindless, brute-force application of large-scale parallel computing." Neither are Gustafson's choices of crude methods to solve unrealistic physical problems relevant to a fair appraisal of Unum Computation.

Why, then, does his book denigrate so many other things he appears to misunderstand?

• • • • • • •

# §9: Puffery instead of Percipience

*THE END OF ERROR* goes far beyond a text about *Unum Computation*. The book also attempts to sell Unum Computation to gullible readers. Like late-night TV commercials for diet pills, unregulated herbal remedies and magnetic bracelets, the book takes full advantage of the *First Amendment*'s freedom of speech to tell ignorant readers what they wish to believe — that their ignorance is no impediment to reliable computation. TV commercials usually append a *caveat*:

"This product is not intended to prevent, diagnose, treat or cure any disease. Results may vary.".

Gustafson has left out the fine print. His worked numerical examples serve the same rôle as do testimonials from "satisfied customers" voiced on TV by paid actors. He vents misguided opinions regardless of facts, and assert bold generalities uninhibited by inconvenient details.

P. 316 has an instance of one of Gustafson's bold generalities:

"**Every physical effect can be modeled without rounding error or sampling error if the model is discrete.**" [His **boldface**]

What he says here differs from what he means. Because he accepts the usual definitions of energy and momentum in terms of derivatives, and accepts their conservation by the usual differential equations of friction-free motion, his models of physics are the usual models. His numerical treatments of these models using unum/interval arithmetic, when they work, take roundoff into account and render it negligible by extending precision at run-time. His "discrete" means only that his intervals encompass "sampling" (discretization) errors too. He doesn't say how much more computation his chosen numerical methods will cost to produce better than poor accuracy.

None of his physical models are actually *discrete*.

" 'When I make a word do a lot of work like that,' said Humpty Dumpty, 'I always pay it extra.' " Ch. VI of *Through the Looking-Glass* by Lewis Carroll

We have experienced discretized Physics before. What Gustafson imagines he has done was actually done in the 1960s by Donald Greenspan at the Univ. of Wisconsin at Madison. He advocated discrete mechanics to do away with calculus and derivatives by redefining energy and momentum in terms of discrete divided differences. For example, he replaced velocity $dx(\tau)/d\tau$ by $(x(\tau+\Delta\tau/2) - x(\tau-\Delta\tau/2))/\Delta\tau$ . Then he redeveloped Newtonian-like laws of motion that do conserve discretized energy and momenta. He wrote a book about them. It never caught on for reasons now familiar. To obtain adequate accuracy from his discrete models required time-steps $\Delta\tau$ so small and consequently computations so time-consuming that they could not compete with numerical methods of higher than 2nd order derived for differential equations by using calculus.

Gustafson's methods for evaluating integrals and solving differential equations have 2nd order at best. Like Greenspan's methods, they are uneconomical for better than poor accuracy.

• • • • • • •

In a free country, we all have the right to be wrong. Gustafson exercises his rights to assert misguided opinions about floating-point arithmetic conforming to IEEE Standard 754. He does not understand why each of the standard's flags and arithmetic exceptions is needed, so he scoffs at them all, saying on p. 30 …

"*No one ever looks at these flags.*" [His *italics*.]

"Computer languages provide no way to view the flags, …
… computer users find them useless at best, and annoying at worst."
[ If flags cannot be viewed by a program(mer), how could they be useful, or annoying?]

He goes on to say …

> A fundamental mistake in the IEEE design is putting the "inexact" description in three processor flags that are very difficult for a computer user to access. The right place is *in the number itself*, with a ubit at the end of the fraction. Putting just that single bit in the number *eliminates the need for overflow, underflow, and rounding flags*.

His opinions are shared widely, but not by the committees responsible for revised standards for C and Fortran. Their recent language standards provide for IEEE 754's flags. What do they see that Gustafson doesn't?

Most exceptions occur very rarely in debugged programs. Testing every vulnerable floating-point operation to detect whether it was exceptional is intolerably expensive. Testing a big array's every element to detect whether it is wrong because of an exception can be expensive too if the elements were computed in parallel each from a short formula. There are too many ways for an element to inherit a misleading unexceptional value from prior exceptions like over/underflow often harmless but otherwise tedious to detect without a flag. Rather than waste time testing every element for troubles that almost never happen, a program can run faster on average by testing one or two flags after the array has been computed and, if a flag was raised, recompute the array some better way.

Most people ignore the Inexact flag. It matters to someone generating data to test the Math. library's accuracy, where the slightest error in test data can overwhelm the accuracy of a carefully crafted function. For example, see "Accuracy Tests for Polynomials' Zero-Finders" posted at www.eecs.berkeley.edu/~wkahan/Math128/Fibs_2_6.pdf . Another use for that flag would arise among programming languages that do not distinguish big integer variables from floating-point variables when computing *Greatest Common Divisors* and *Least Common Multiples*, which roundoff ruins with no other warning. It has happened in MATLAB, which lacks access to flags.

• • • • • • •

Relatively few computer practitioners have tried to debug immense floating-point programs used by scientists and engineers, especially programs thought already debugged. Gustafson must lack that experience. Perhaps that is why he scoffs at IEEE 754's flags and won't "waste bit patterns on a plethora of NaNs" (p. 49). On p. 24 he dismisses peremptorily their use for debugging.

Ideally, in a software development ambience devised to help debug those immense floating-point programs, flags and NaNs would do double-duty as pointers (indirectly) to places in programs where a flag was first raised or a NaN first created. The rules for propagation of NaNs through arithmetic operations reveal something about the origin of a final result that is NaN. If it was caused by a reference to an uninitialized variable, that would be revealed after a program's work-space had been initialized to NaNs that point to their variables' names. Names of such variables rarely outnumber the capacity of a NaN as a pointer; and the sites in a program at which a flag can first be raised or a NaN created rarely outnumber their pointing capacities. The first creation is the one most worth recording; the consequent cascade of exceptions matters less to debugging.

What alternative are there to these debugging aids? Must a program crash at its first encounter with an arithmetic exception deemed an error? Whose error? The programmer's or the user's?

Crashing a program at the first exception deemed an error seems a reasonable policy while the author of a short program is debugging it, but to enforce that policy universally is a bad policy. That policy would preclude interwoven speculative execution of two processes of which one will succeed when the other doesn't. That policy would prematurely abort searches that jumped out beyond the domain of a subprogram; the correct response is to shrink the jump back inside the domain and resume the search. That bad policy would cause software embedded in a controller to abandon the process being controlled, rather than detect and diagnose the exception to maintain control of a nuclear power plant, an aircraft, an automobile, or a medical life-support system.

Crashing is a bad policy to enforce universally. Continued execution oblivious to every arithmetic exception is a bad policy to enforce universally, as JAVA does. Flags help programs detect exceptions economically, diagnose them and compensate for them. NaNs convey information helpful to programmers and users of programs who must debug them. The computing industry has been very slow to recognize and support these debugging capabilities of IEEE 754. That does not excuse Gustafson for scoffing at what he does not appreciate.

• • • • • • •

"**The set of ubounds is closed under addition, subtraction, multiplication, and division.**" This assertion on p. 63 is misleading. The book defines division by 0 to be NaN (p. 137), and defines division by a ubound (interval) containing 0 to produce NaN instead of an interval that contains ∞, though infinite unums are defined on p. 29. What are these good for?

Unum Computation's ubounds differ from the usual interval arithmetic's intervals in this one way:

A ubound may include or exclude either or both endpoints.

This detail matters rarely, and then it can matter greatly if 0 is an endpoint of an interval divisor:

$2/(1 + \mathbf{[}0, 3\mathbf{]}/\mathbf{[}0, 1\mathbf{]}) = 2/(1 + \mathbf{NaN}) = \mathbf{NaN}$ because 0/0 is NaN; however

$2/(1 + \mathbf{[}0, 3\mathbf{]}/\mathbf{(}0, 1\mathbf{]}) = 2/(1 + \mathbf{[}0, \infty\mathbf{)}) = 2/\mathbf{[}1, \infty\mathbf{)} = \mathbf{(}0, 2\mathbf{]}$ .

Thus does infinity go away after it serves as a divisor. It can do the same if it is inside an interval, though the idea of an interval containing infinity seems strange at first. Such intervals (ubounds) would exist if the set of ubounds really were closed under all the rational arithmetic operations.

What good are intervals containing infinity?

They are needed to evaluate continued fractions. This ratio of Bessel functions is one of them:

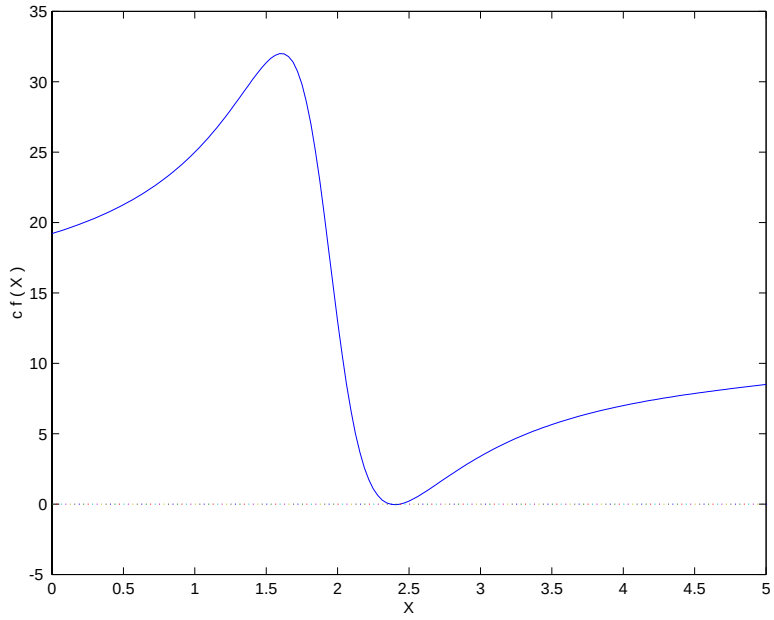
$$J_n(2\sqrt{x})/J_{n-1}(2\sqrt{x}) = \cfrac{\sqrt{x}}{n - \cfrac{x}{n + 1 - \cfrac{x}{n + 2 - \cfrac{x}{n + 3 - \dots}}}} \ .$$

Many higher transcendental functions have continued fraction expansions. These often converge over a wider domain in the complex plane than do power series. Rather than get involved in that topic, let's consider a simpler didactic example restricted to a real variable:

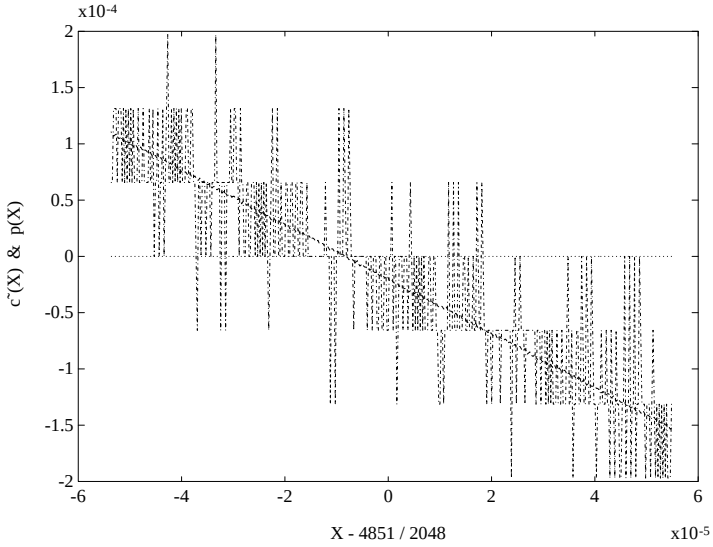
$$cf(x) := 13 - \cfrac{12}{x - 2 - \cfrac{1}{x - 7 + \cfrac{10}{x - 2 - \cfrac{2}{x - 3}}}} = p(x) := \frac{2152 - x \cdot (2551 - x \cdot (1000 - x \cdot (194 - 13 \cdot x)))}{112 - x \cdot (151 - x \cdot (72 - x \cdot (14 - x)))} .$$

Of these two *expressions* for the same rational *function*, p(x) is by far the more vulnerable to over/underflow and to roundoff, though the function is otherwise well-behaved at every real x :

Ideally, *cf*(x) = p(x) :




*cf*(x) and p(x) computed in 4-byte *float* arithmetic (24 sig.bits):




cf(x) and p(x) are here computed at about 500 consecutive *float* values of x , each in 24 sig.bits.

The graph of cf is the comparatively smooth line sloping from upper left to lower right thus: \

The graph of p is the ragged oscillatory graph. It shows that p suffers from roundoff by orders of magnitude more than cf does.

But there is a catch:

The computation of cf(x) encounters divisions by zero at x = 1, 2, 3 and 4.

Instead of rounding, unum/interval arithmetic widens intervals, and widens them vastly more for p than for cf . For instance,

$cf([303/128,\ 304/128]) \subset [-0.158,\ 0.146]$ ; $p([303/128,\ 304/128]) \subset [-5.02,\ 5.001]$ .
And ordinary unum/interval arithmetic produces NaN instead of $cf(X)$ at any interval X that contains 1, 2, 3 or 4, and also produces NaN for $p(X)$ if X is too wide; for instance …
$p(X)$ is NaN for X := $[-0.75,\ 0.75]$, $[0.8,\ 1.2]$, $[1.99,\ 2.01]$, $[2.95,\ 3.05]$, $[3.9,\ 4.1]$ and $[4.9,\ 5.1]$ .

Far fewer NaNs are produced when intervals containing $\infty$ are allowed during the computation of $cf$. I called such intervals "Exterior Intervals" when I advocated them in 1968 during a Summer Course on Numerical Analysis, #6818 at the University of Michigan at Ann Arbor. An ordinary "Interior" interval $[\alpha, \beta]$ has $\alpha \le \beta$; an Exterior Interval $X = [\alpha, \beta]$ has $\alpha > \beta$ and includes $\infty$ and all real numbers $\xi \ge \alpha$ or $\xi \le \beta$. For instance, $[6,\ 8]/[-1,\ 2] = [3,\ -6]$ .

Thus do exterior intervals close the set of all intervals under addition, subtraction, multiplication *and* division. Exterior intervals occur rarely and typically exist briefly, disappearing after being used as divisors, or else generating NaNs as does $[3,\ -6]\cdot[-1,\ 2]$ because it includes $\infty\cdot 0$ . At all the intervals X above at which $p(X)$ is NaN in ordinary interval arithmetic, $cf(X)$ delivers finite intervals when a transient exterior interval is allowed to occur. Here they are:

| X | $cf(X) \subset$ | X | $cf(X) \subset$ | X | $cf(X) \subset$ |
|---|---|---|---|---|---|
| $[-0.75,\ 0.75]$ | $[17.4,\ 23.4]$ | $[1.99,\ 2.01]$ | $[12.0,\ 14.0]$ | $[3.9,\ 4.1]$ | $[6.58,\ 7.36]$ |
| $[0.8,\ 1.2]$ | $[22.7,\ 28.5]$ | $[2.95,\ 3.05]$ | $[2.85,\ 3.91]$ | $[4.9,\ 5.1]$ | $[8.22,\ 8.74]$ |

Exterior intervals complicate interval arithmetic severely. In 1968 core memory cost too much to waste on complexities rarely needed, so exterior intervals were hardly ever implemented. Now unum arithmetic is so complicated that a little more complexity would barely be noticed. The alternative dumps complexity onto the user of Unum Computation, who must rewrite $cf$ as …

$$cf(x) = g(x) := 13 - 12\cdot(x-2)\cdot((x-5)^2 + 4)/\big(\, x + ((x-5)^2 + 3)\cdot(x-2)^2 \,\big)$$

to obtain intervals from $g(X)$ not much wider and occasionally narrower than from $cf(X)$. Still, $g([+100,\ -100])$ cannot be computed so directly as can $cf([+100,\ -100]) = [12.877,\ 13.118]$ .
And the challenge of figuring out whence $g(x)$ came is left to the diligent reader.

Unum Computation could have been closed under *all* rational operations, but it wasn't.

• • • • • • •

A fair appraisal of *Unum Computation* will ignore Gustafson's misleading assertions intended to sell it as *THE END OF ERROR*.

Whatever the cost of Unum Computation, it would be worth its price
IF
it *always* delivered all of the benefits Gustafson's book claims for it.

But it doesn't,
not even *usually*.

# §10: A Curate's Egg

The curate, Mr. Jones, attends a luncheon hosted by his superior, the Bishop:

Bishop: *"I'm afraid you've got a bad egg, Mr. Jones".*

Curate: *"Oh, no, my Lord, I assure you that parts of it are excellent!"*

From "True Humility", a cartoon by George du Maurier, originally published in *Punch*, 9 November 1895.

Dictionary Definition:

A "curate's egg" describes something at least mostly bad, with perhaps some good bits.

• • • • • • •

Compared with variable precision interval arithmetic implemented in floating-point for a widely used imperative language like C++ or Fortran, *Unum Computation* has its benefits and its costs. The benefits come more from how it manages roundoff, less from how Gustafson uses it to manage discretization errors and uncertain data.

Unum Computation shares benefits and limitations with an interval arithmetic whose precision can vary at run-time. Roughly, if computed interval results' widths are too big by some factor less than $\kappa$, say, *and if those widths are due almost entirely to roundoff,* then recomputation with unum or interval arithmetic of a wider precision adequate to reduce rounding errors by a factor smaller than $1/\kappa$ almost always produces acceptably accurate results. And if not, doubling arithmetic's precision almost always works. And if not, an error-analyst is needed to recast the problem, if possible, perhaps as a Self-Validating Computation mentioned in §6 p. 18 above.

Rigorous unum/interval bounds for discretization errors can be costly in two ways. First, they often require mathematically deft (re)formulations of a problem to render its discretization errors subject to simple bounds, like those for $\int_a^b f(x)\cdot dx$ in §7 p.23 above, or else eligible for Self-Validation. For example, here is how an initial-value problem "$d\mathbf{y}/d\tau = \boldsymbol{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}^\circ$" would be reformulated:

Recasting this differential equation as a Volterra integral equation "$\mathbf{y}(\tau) = \mathbf{y}^\circ + \int_0^\tau \boldsymbol{f}(\mathbf{y}(\xi))\cdot d\xi$" turns it into a fixed-point problem for a contractive map of a function-space. Now fixed-point $\mathbf{y}(\tau)$ becomes the limit of a convergent iteration that illustrates the second way that rigorous interval bounds can be costly: They converge too slowly to be economical for high accuracy.

Rigorous interval bounds for discretization errors usually come from processes that converge slower, as discretizations are refined, than do processes that yield error estimates (not rigorous bounds) based upon asymptotic behaviors, like the $\pm$ estimates offered in §7 p. 23-4 above for Takahashi-Mori quadrature. Some discretization errors are bounded rigorously by error-analyses based upon differential inequalities or monotonicity theorems that cost far less to apply than does interval arithmetic. Still, when really needed, perhaps for a certificate of correctness acceptable as evidence in a lawsuit, rigorous interval bounds can be worth their cost in effort and time.

Some of the unacknowledged run-time costs of address computations during Unum Computation were discussed above in §5, p. 15. Further run-time costs include increased latency incurred by additional pipeline stages needed to cope with the greater complexity of unum arithmetic, if we

assume as Gustafson does that the CPU performs as much of unum arithmetic as possible on-chip. The complexity worsens if both ends of ubounds must be computed in parallel to save a factor greater than 2 in run-time throughput. These further costs seem similar to the run-time costs of software support for variable-precision floating-point interval arithmetic at roughly the same or better speed. However, the far greater complexity of on-chip circuitry needed to handle tightly packed variable-width unums will likely increase substantially the price of the CPU chip. Until one be built, cost estimates remain speculative.

Comparing costs with the value of benefits remains speculative too. Almost every computation costs far less than an appraisal of its correctness would add to its cost. Over the past few decades, as the cost of computation has dwindled, so has the value of an ever increasing fraction of what is being computed. Not everyone is willing to pay extra for an appraisal that costs more than the computation is worth, especially if this appraisal merely confirms what might otherwise be taken for granted. Consequently there is little demand for error-analyses, and less for error-analysts, though the demand is intense when and where it exists. The demand for error-bounds evaporates when they cost more time, hardware or human labor to compute than they are worth.

As a scientist or engineer, I wish not to know how big rounding and discretization errors in my results aren't. I need to know only that they are negligible — so tiny that I need not know them. I would rather increase arithmetic's precision and refine discretization if necessary to render their errors almost surely negligible. And I am unwilling to pay much for what I wish not to know.

*Uncertainty Quantification* is different: How much uncertainty have my results inherited from uncertain data? I wish dearly to know that; but Unum Computation and interval arithmetic too often overestimate uncertainty grotesquely *without* an intense error-analysis, and *with* one they play a minor rôle. For example, take the topic of the book's ch. 21.4, pp. 321-6, structural analysis. The chapter begins with photos before and after the collapse of North Sea oil platform Sleipner A in Aug. 1991. The engineer in charge had miscalculated concrete pillars' strengths.

The book does not mention that Unum Computation would not have prevented that mistake.

The book treats a cantilevered structure with one strut and two cables, reducing its load to two equations in two unknown forces rather than the more complicated equations for the structure's deflection under load. The chapter ends with a computer-generated picture of a structure with hundreds of members. The thousands of equations that would have to be solved for the structure's deflections under loads have not been programmed. Instead Gustafson concludes that Unum Computation of C-Solutions will provide "*provable bounds on the results*." [His *italics*] And so they will; but the bounds may be millions of times too big. It happens because the equations are much more sensitive than the structure to ill-oriented perturbations, as was illustrated by an example of Failure Mode IV on p. 20 above. Interval arithmetic used naively does no better. Realistic sensitivity analyses of elastic structures continue to "require some artistic choices" of variations in parameters notwithstanding his assurance on p. 324 that instead "With uboxes, all the possible parameter sensitivity is already in the c-solution, for all to see." Exaggerated.

• • • • • • •

What about BIG DATA? Much of it requires only one or two bytes per datum. Would the cost of conversion to unums be repaid by better computations of uncertainty inherited from the data?

Statistical methods, like regression and estimates of standard deviations, compute the uncertainty inherited from randomly drawn samples of noisy data. Unum/interval arithmetic can do that, as can ordinary floating-point, provided their precisions exceed twice the data's and the desired result's. Either way, ubounds or roundoff become negligible compared with standard deviations. (Carrying less precision increases the likelihood that ubounds or roundoff will obscure results excessively when the data are nearly redundant.) Error-analysis has obviated interval arithmetic.

Long experience and some error-analyses support a rough rule-of-thumb that renders roundoff extremely unlikely to causes embarrassment if *all* intermediate floating-point computations are performed carrying a little more than twice the precision trusted in data and desired in results. This rule has survived the test of time in statistics, optimization, root-finding, geometry, structural analysis and differential equations. Of course exceptions exist, but they are so rare as to have given rise to a wry joke among numerical analysts:

Nobody unlucky enough to have been betrayed by that rule of
thumb need concern us; he has already been run over by a truck.

Gustafson chooses unum/interval computation to insure against that betrayal. Doing so exposes him instead to the risk of betrayal by the failure modes exposed in §6, pp. 16-20 above.

Memory movements so dominate the costs of parallel computation with big data that numerous *Communications-Avoiding* algorithms have been developed to lessen costs, sometimes by orders of magnitude. Some of these algorithms demand considerably more arithmetic precision than the data's during certain intermediate computations. For an example see

www.eecs.berkeley.edu/~aditya/caml-talk .

How would Unum Computation know which intermediate computations need higher precision and how much? Or would end-figure uncertainties in the data be misinterpreted as justifying degraded accuracy? Who decides? (A similar question was raised on p. 8, §2 above.)

• • • • • • •

Spectral analysis uses a fast Fourier transform to split a signal into its constituent frequencies. The transform preserves in the spectrum the Root-Mean-Square of the signal's noise, which is usually unknown at the outset. The computation's arithmetic should carry enough precision that ubounds or roundoff will augment that noise negligibly. This can be assured if, for a signal of length $2^N$, the arithmetic's precision exceeds the data's by at least $N-1$ sig.bits. For a signal stored in IEEE 754 (2008) half-precision numbers (2 bytes wide, 11 sig.bits), IEEE 754 single-precision arithmetic (4 bytes wide, 24 sig.bits) can handle signals of lengths up to $2^{13} = 8192$ at least. Whether a shorter signal of length, say, $2^{10} = 1024$ would benefit enough from a Unum Computation, with slightly less precision, to offset the cost of its implementation seems unlikely when the extra costs of storing diverse length unums (§5, p. 15 above) are taken into account.

• • • • • • •

Perhaps some uncertainty still beclouds the costs and benefits of Unum Computation. Anyway,

it is certainly **NOT** THE END OF ERROR