The Improbability of

PROBABILISTIC ERROR ANALYSES

for Numerical Computations

by Prof. W. Kahan
E.E. & Computer Science, & Mathematics Depts.
Univ. of Calif. @ Berkeley

prepared for the

UCB STATISTICS COLLOQUIUM
4 - 5 pm., Wed. 28 Feb. 1996
1011 Evans Hall

and subsequently revised.

This is an extended version of transparencies titled
"The Improbability of Probabilistic Error Analysis"
first presented in Hamburg at the
third ICIAM Congress, 3-7 July 1995

PostScript file: http://http.cs.berkeley.edu/~wkahan/improber.ps

ABSTRACT:

" Everything I know nothing about is equally improbable."

exemplifies the illogic that plagues attempts, by statistical means, to assess error and risk in floating-point computation.

In the early 1980s, the French Academy of Sciences awarded a prize for a patented method that automated the estimation of roundoff's effect upon any computation and dispensed with error analysis; two decades earlier essentially the same method had been tried and abandoned as altogether too risky.

In 1994-5 the manufacturer of a very popular micro-processor acceded to a recall because its floating-point division was defective although the defect was believed to pose less of a hazard than cosmic rays do, and was expected to inflict a perceptible error upon its user perhaps once every millennium by the manufacturer's estimate, once every three weeks by a competitor's. The division algorithm had been " proved " correct, and had passed a few billion random tests; afterwards, however, a short test program exhibited over 2000 failures in the first million divisions tested.

Perhaps the hazards posed by numerical computation (and by earthquakes and by carcinogens) would be misjudged less often if first courses on statistics spent a little more time on the probabilistics of extremely improbable events. Anyway, we need help from Statisticians to avoid these abuses of Statistics.

Prudence demands Respect for Probability and Statistics.

Abuse of Statistics undermines that Respect.

. . . . . .

Wishful Thinking often intrudes into probabilistic assessments of hazards posed by errors in numerical computations. Those numerical errors fall into three classes:

Intentional:

Idealized models, truncated series, discretization, ...

Unavoidable:

Rounding errors.

Unintentional:

Bugs, blunders in software and hardware

Compared with idealizations,
roundoff and bugs are unlikely menaces.

How unlikely ?

Roundoff in Floating-Point Arithmetic:

Suppose the program asks the computer to calculate

$$W := X \cdot Y + Z ;$$

what the computer actually calculates is

$$w = \left((x \cdot y) \cdot (1 + ß) + z \right) \cdot (1 + \mu)$$

in which ß and μ stand for rounding errors, tiny values for which we know *a priori* bounds like, say,

$$|ß| < 2^{-53} , \quad |\mu| < 2^{-53} ; \qquad 2^{-53} \approx 10^{-16} .$$

(These bounds suit Double Precision (REAL*8) on most computers nowadays.)

The simplest model of roundoff assumes that nothing more can be known about ß and μ .

The simplest probabilistic model of roundoff assumes that ß and μ are independent random variates distributed Uniformly between their bounds $\pm 2^{-53}$.

Both models merely approximate the truth.

What is an Error Analysis ?

It is a process and its product; it is an estimate of the error in a computation, and a proof of the estimate's validity.

If the error were known, it would normally be subtracted off, so error analysis is usually expected only to bound the error. The best kind of bound is a proved over-estimate that is not too much too big. For example, on most computers nowadays,

" Y = ATAN(X) "

has been proved by its implementor (and tested) to return

y = arctan(x) ± (less than 0.7 ulp.)

where an ulp is one Unit in the Last Place (last sig. bit or dec. digit) returned. Any bound below 1 ulp is acceptable here; the bound always exceeds 0.5 ulp.

There are also "Backward" error analyses. For instance, given matrices A and B , a good program to solve $A \cdot X = B$ for X by Gaussian Elimination actually computes a solution $X+dX$ of a perturbed equation $(A+dA)\cdot(X+dX) = B+dB$ in which dA and dB are bounded in norm by tiny quantities that depend upon the norms of A and B respectively, their dimensions, and the precision of the computer's arithmetic. These "tiny quantities" are inferred by an error analysis.

Error analyses start from models of errors, like ß and μ in

$$w = \left((x\cdot y)\cdot(1 + ß) + z \right)\cdot(1 + μ) ,$$

taking more or less of their properties into account, to infer estimates of their subsequent effects. Some analyses obscure their domains of validity by ignoring nonlinear terms like $ß^2$, $ß\cdot μ$ and $μ^2$. Anyway, inferences entail tedious manipulations of numerous inequalities, only partly mechanizable.

Probabilistic error analyses estimate errors' means and standard deviations instead of upper bounds for errors.

Two statistical strategies: Theoretical and Experimental.

1. Theoretical: Probabilistic Error-Analyses

... are based upon attempts to approximate each rounding error by a random variate of tiny amplitude, and then estimate how lots of them will propagate and accumulate in the final computed results.

Such analyses produce means and standard deviations for computed results as if these were random samples drawn from a population of similar computations, each in error by roughly a linear combination (to first order) of the individual rounding errors. Probabilistic error-analyses are not much easier than non-probabilistic.

2. Experimental: Randomized Error-Sampling

... attempts to assay the impact of roundoff upon any computation by treating that computation as one sample drawn from a population of similar randomized computations differing only

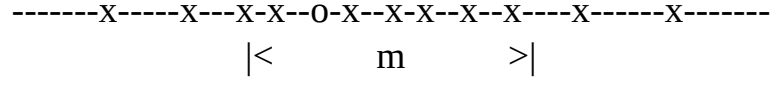
either ...

in the data, which are randomly perturbed slightly from the given data (cf. F. Chatelin and V. Frayssé),

or ...

in arithmetic operations, which are randomly perturbed slightly (cf. J. Vignes *et al.*, CESTAC, CADNA; he has patented CESTAC, and also received a prize for it from the French Academy of Sciences.).

The idea is to re-run the computation a few times, each time with a different randomized perturbation, and treat the set of results as a sample whose mean and standard deviation estimate the intended result and the error in that estimate:



Randomized Error-Sampling appears to obviate Error Analysis !

What Good are

Probabilistic Error-Analyses
and
Randomized Error-Sampling

?

They afford some

Corroboration of what should already be known

by Numerical Analysts and Developers of Numerical Software.

To Scientists, Engineers and other Users of Numerical Software,

Probabilistic Estimates of Error are Probably Useless or Worse.

Why ?

Probabilistic Error-Analyses
and
Randomized Error-Sampling

tend to provide

unsatisfactory answers

for two crucial questions :

1. Insurance Premiums :

   How much should a prudent corporation put into reserve to cover the expected cost of extraordinarily big numerical errors detected too late ? The answer matters to people who have to measure calamities on a monetary scale.

2. Unreliability :

   How likely is an extraordinarily big numerical error, if one occurs, to be detected too late despite probabilistic analysis and/or randomized error-sampling ? The answer matters to an individual who has his own threshold for intolerable risk, regardless of monetary compensation.

Let’s see what goes wrong ...

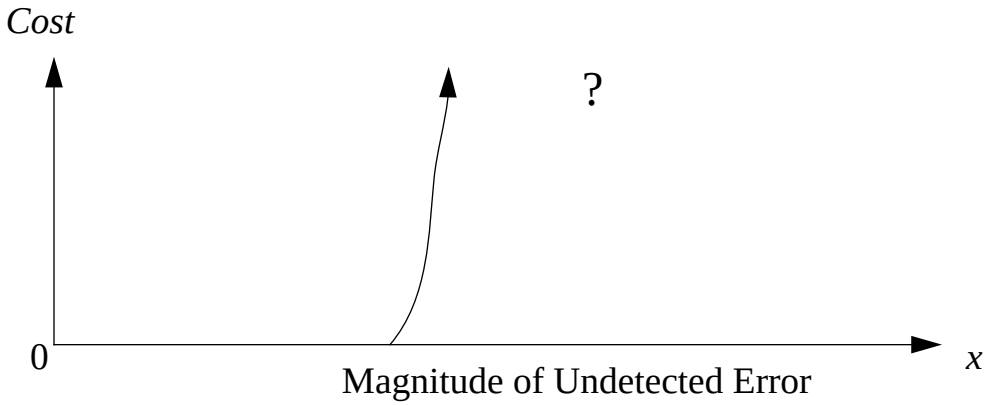
1. Insurance Premiums :

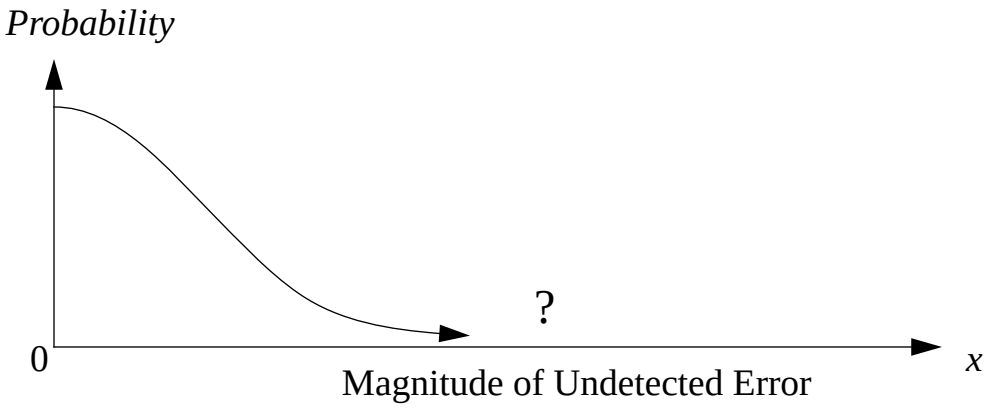
$$Premium = Overhead + (1 + Profit) \cdot \int_0^{\infty} (Cost(x) \cdot Probability(x))\, dx$$

Cost(*x*) = cost induced by an error as big as *x* detected too late.

Probability(*x*) = probability density for errors as big as *x* detected too late.

What can we know about *Cost* and *Probability* ?

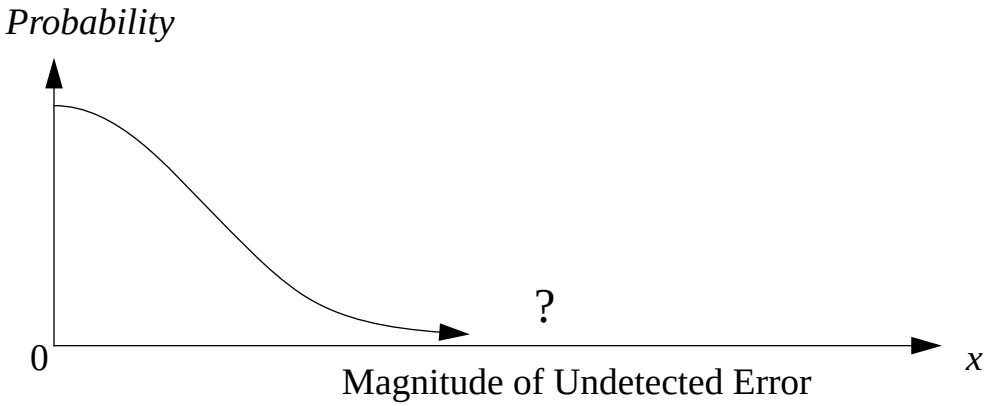




∫ (Inestimable *Cost*)·(Unknown *Probability*) --> Unknowable *Premium* .

Why is the *Probability* unknown ?

2. Unreliability :



Why is the *Probability* unknown ?

There are three reasons, all of which call into question the application of the

Central Limit Theorem

to justify approximating *Probability* via a *Normal* or X^2 distribution.

Three reasons:

i. The *Central Limit Theorem* is generally cited to justify approximating *Probability* via a *Normal* or X^2 distribution. But such approximations converge very slowly along the tails of the distribution. Therefore, where *Probability* is tiny (and *Cost* may be huge), the approximation can be extremely tiny and yet wrong by orders of magnitude.

ii. To justify invoking the *Central Limit Theorem*, rounding errors are generally presumed to be ...

random,
weakly correlated, and
distributed continuously over a tiny interval.

Actually, they are ...

not random,
often correlated (perhaps intentionally), and
often behave more like discrete than continuous variables.

iii. Further to undermine the applicability of the *Central Limit Theorem*, often only a few (as few as two or three) rounding errors are the dominant contributors to the final error, especially when it is extraordinarily big because some nearby *Singularity* amplified them.

Reasons ii and iii are little known, so they will be expounded here.

To find evidence inconsistent with the presumption of uncorrelatedly random behavior of rounding errors, we need merely plot the errors that arise from an innocuous formula:

Insert here Figures 1 to 4 from MathCAD NonRandomRoundoff.

Figure 1 of 4 on Non-Random Roundoff Using MathCAD 3.1 on a Mac.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ W. Kahan 28 July 1995

Two of the infinitely many ways to express a rational function are as a Compact Formula

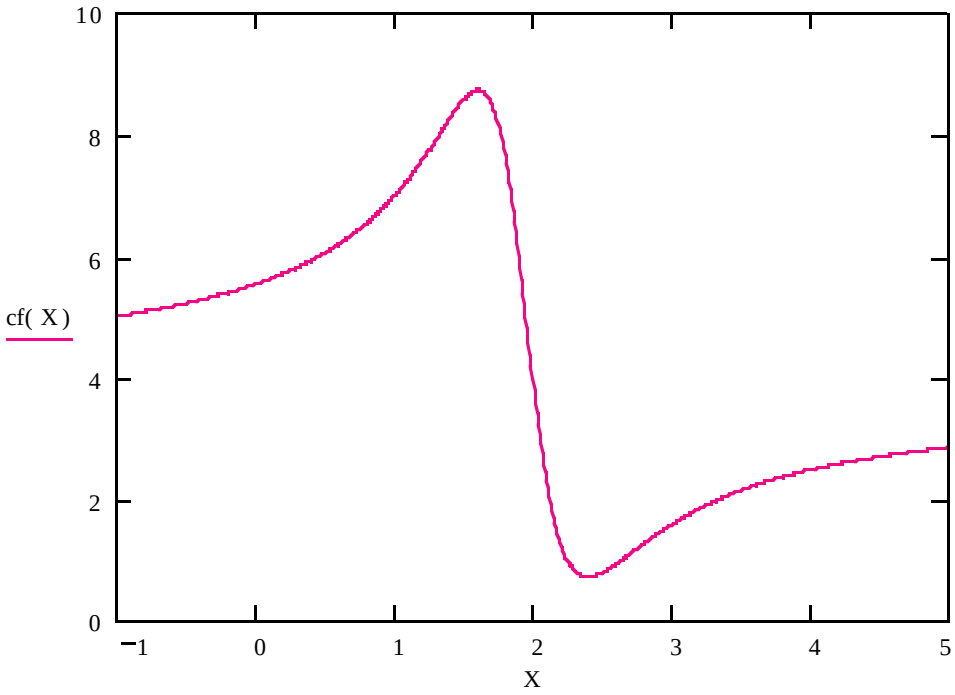
$$cf(x) := 4 - \frac{3\cdot(x-2)\cdot((x-5)^2+4)}{(x+(x-2)^2\cdot((x-5)^2+3))}$$

and as a Ratio of Polynomials in the customary form

$$rp(x) := \frac{622 - x\cdot(751 - x\cdot(324 - x\cdot(59 - 4\cdot x)))}{(112 - x\cdot(151 - x\cdot(72 - x\cdot(14 - x))))} \quad .$$

Algebraic manipulation (MAPLE V) simplifies cf(x) - rp(x) to 0 by way of confirmation.

Here is their graph over a range X : $X := -1, -1 + 0.01 .. 5$

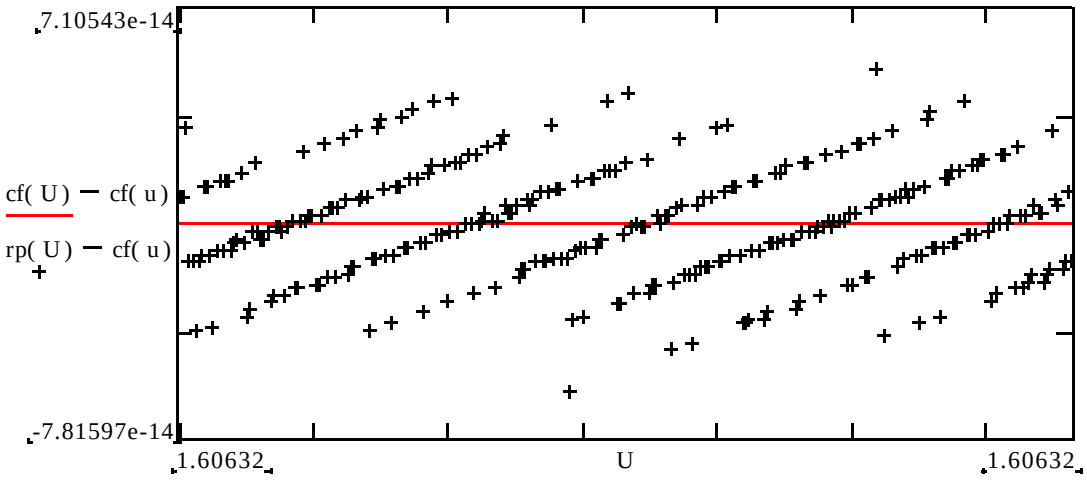


This graph will be expanded to explore roundoff in the neighborhoods of its extrema.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Fig. 2 of 4 on Non-Random Roundoff

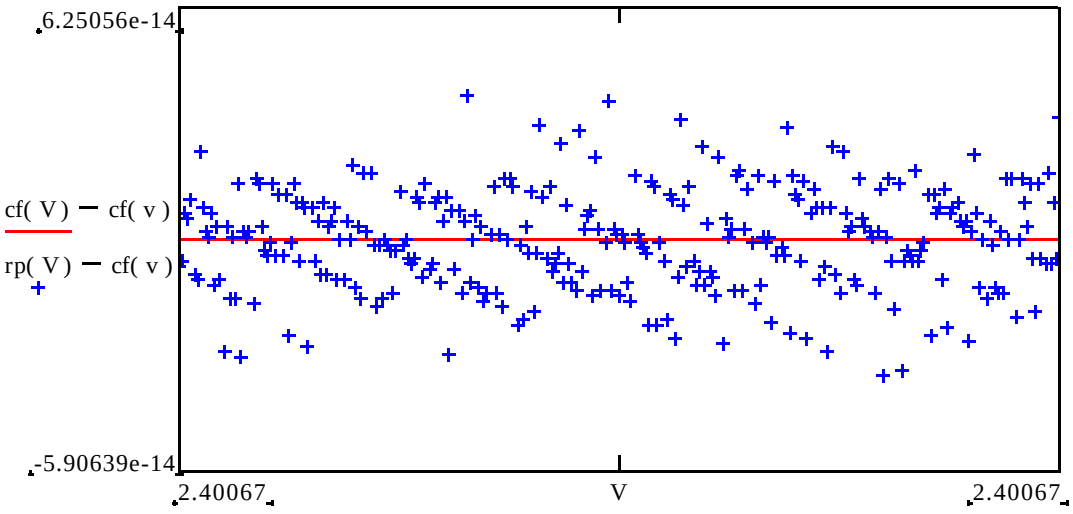
The next picture shows how much more sensitive rp(x) is to roundoff than is cf(x) . Both expressions are plotted at 301 consecutive floating-point numbers in the range U defined below. The nearly smooth graph belongs to cf(x) , the scattered + signs to rp(x) .

$eps := 0.5^{52}$ $u := 1.60631924$ $U := u, u + eps .. u + 300 \cdot eps$



Roundoff in rp(x) does not lookentirely random; could this be a fluke? Look elsewhere:

$v := 2.40066610208539$ $V := v, v + 2 \cdot eps .. v + 600 \cdot eps$



These patterns should undermine confidence in the randomness of roundoff.

Figure 3 of 4 on Non-Random Roundoff

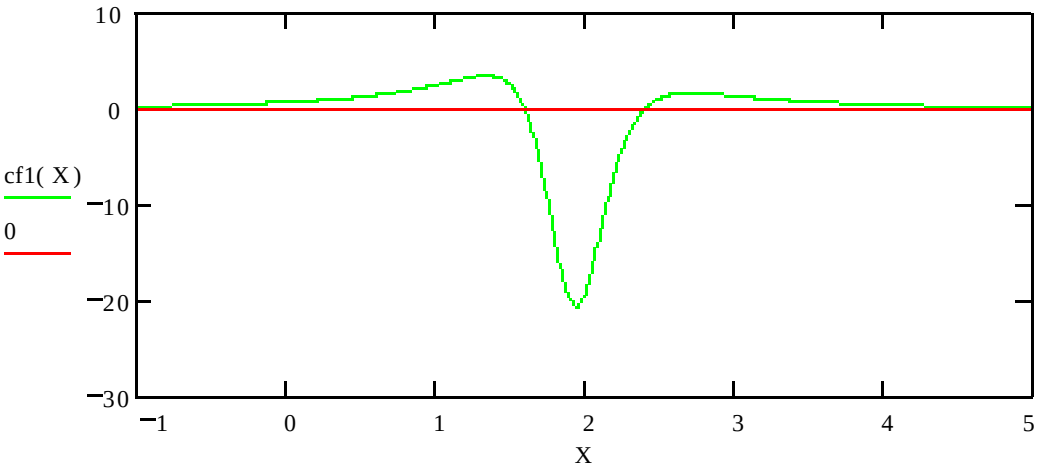
The derivative of rp(x) is

$$rp1(x) := 3\cdot\frac{(x^6 - 24\cdot x^5 + 243\cdot x^4 - 1302\cdot x^3 + 3816\cdot x^2 - 5664\cdot x + 3270)}{(x^4 - 14\cdot x^3 + 72\cdot x^2 - 151\cdot x + 112)^2}$$

The derivative of cf(x) is cf1(x) and is expressed in a compact form thus:

$$f1(z) := \frac{((((z - 12)\cdot z + 63)\cdot z - 158)\cdot z + 156)\cdot z + 24)\cdot z - 26}{(2 + z + ((z - 3)^2 + 3)\cdot z^2)^2}\cdot 3$$

$$cf1(x) := f1(x - 2)$$



Now the effect of roundoff upon rp and cf can be compared over any sufficiently narrow range of values X including a value y by plotting the near-constant expressions

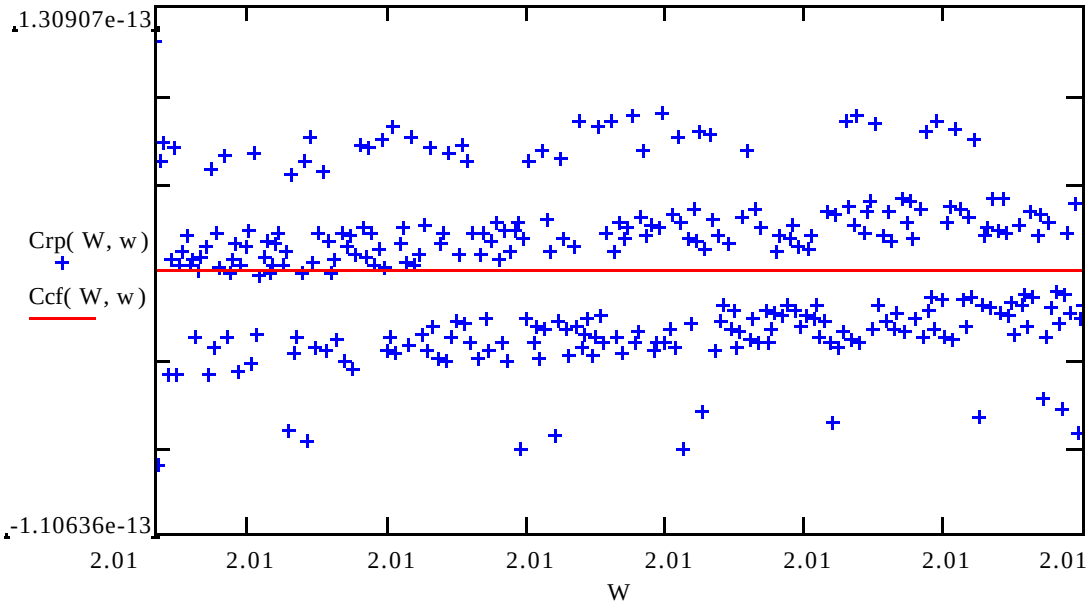
$$Ccf(X, y) := (cf(X) - cf(y)) - (X - y)\cdot cf1(y)$$

$$Crp(X, y) := (rp(X) - cf(y)) - (X - y)\cdot cf1(y)$$

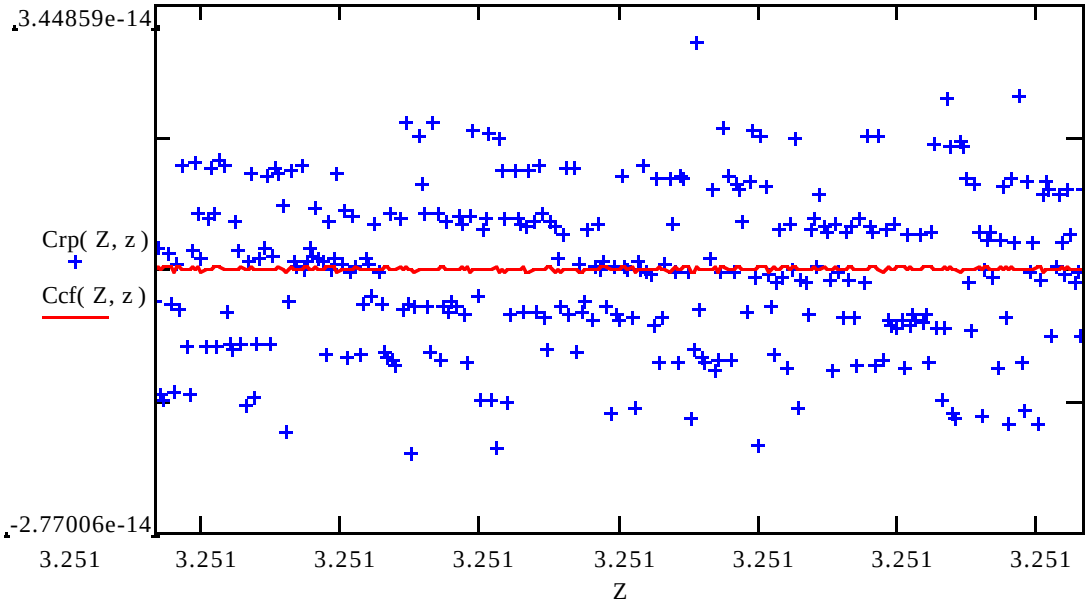
Any scatter in their graphs may be attributed to roundoff. For example, ...

Figure 4 of 4 on Non-Random Roundoff

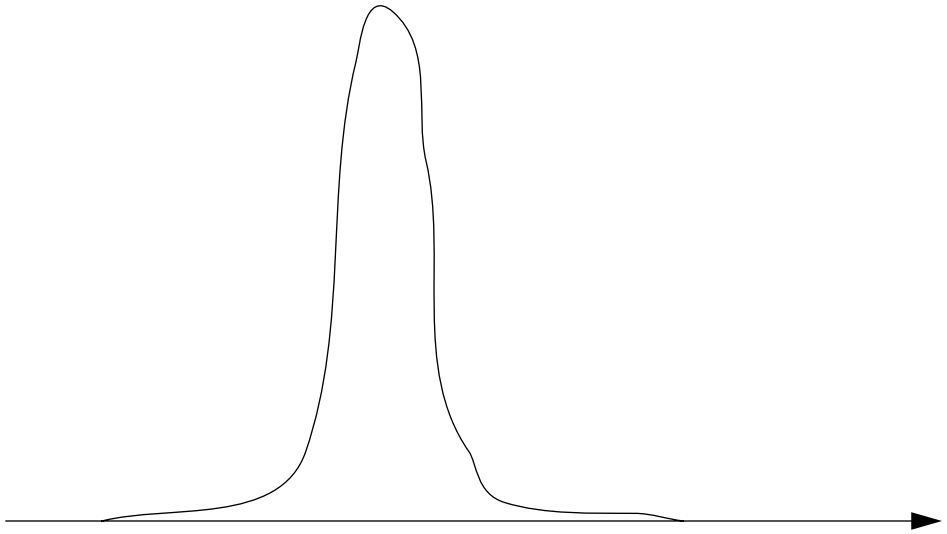
$w := 2.01$ $W := w - 600 \cdot eps, w - 598 \cdot eps .. w$



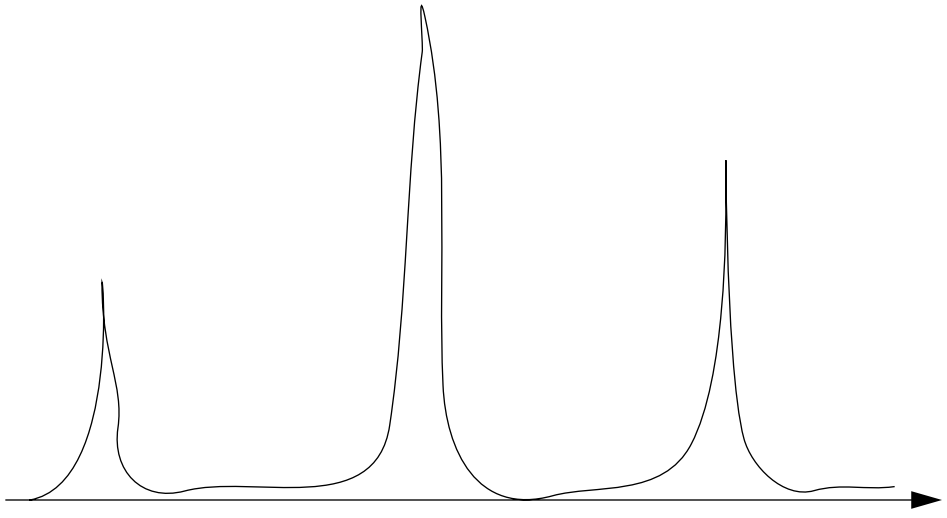
$z := 3.251$ $Z := z - 300 \cdot eps, z - 298 \cdot eps .. z + 300 \cdot eps$



Note how computed values in some sub-ranges can fall into clusters each narrower than the values' error over that sub-range.



Presumed and Expected Continuous Distribution of Errors



Actual Near-Discrete Distribution of Errors

If relatively few samples are drawn they can, with non-negligible probability, all fall under the same outlying peak, thus masquerading as closely spaced (and hence allegedly "accurate") estimates of what is actually an inaccurate result.

Of course, the fact that rounding errors are neither random nor uncorrelated will not in itself preclude the possibility of modelling them usefully by uncorrelated random variables. What will jeopardize the utility of such models is their failure to mimic important properties that actual rounding errors possess. Two of those properties, chosen for their brevity, are spelled out in the following *Theorems* :

The Exact Cancellation Theorem :

> If p and q are floating-point variables of the same *type* or *format* (both *Single-Precision*, or both *Double-Precision*, on the same hardware), and if $1/2 \leq p/q \leq 2$, then $p - q$ is computable exactly (without any rounding error) in that format unless it Underflows (which it cannot if the hardware conforms to IEEE Standard 754 or 854 for floating-point arithmetic, as do practically all commercially significant North American computers) or the computer is a CRAY X-MP/Y-MP/C90/J90 .

This theorem figures crucially in some algorithms that compute eigensystems, in others that solve differential equations, in some that compute transcendental functions, and in others that extend the hardware's precision.

A Theorem about Five Tame Rounding Errors :

> In the expression $\arccos(x / \sqrt{(x^2 + y^2)})$, the argument of arccos is always valid (between -1 and +1 inclusive) despite five rounding errors (unless it suffers Division-by-Zero or Over/Underflow) on all Hewlett-Packard calculators and on all commercially significant North American computers except CRAY X-MP/Y-MP/C90/J90.

This theorem is non-trivial first because its proof is different for every different family of computers, and second because it fails on CRAYs. That failure struck down a program written by Prof. J. Sethian, and it cost him two weeks to locate it. Randomly perturbed rounding errors would violate the theorem too.

By disregarding theorems like these, random models of roundoff can grossly exaggerate the effects of roundoff.

Recall ...

iii. Further to undermine the applicability of the *Central Limit Theorem*,

often only a few (as few as two or three) rounding errors are the dominant contributors to the final error,

especially when it is extraordinarily big because some nearby *Singularity* amplified them.

Example: The first subtractions performed in the numerator and denominator of rp(x) above contribute two rounding errors that dominate all the rest.

Example: Gaussian Elimination to solve $A \cdot x = b$.

Often, when A is nearly singular for systematic reasons, it agrees with a matrix of rank 1 in its first few sig. dec. Then the first pass of Gaussian Elimination, which subtracts multiples of one row from all others to put zeros into their first column, inflicts substantial cancellation upon all other rows. As Gaussian Elimination continues to act upon the diminished array, subsequent rounding errors contribute negligibly compared with those in the first pass. The final pivot, which can be quite small when A is nearly singular, will then be affected far more by the earliest two rounding errors than by all that follow. During back-substitution, the first element of the solution x to be computed is obtained by division by that pivot, and then dominates the computation of all subsequent elements. Therefore the computed solution x is in error by preponderantly the contributions of three rounding errors.

When this phenomenon occurs, it especially undermines the validity of Randomized Perturbation of arithmetic operations, because then only a few of those perturbations can affect the final result significantly, and therefore its perturbed distribution becomes more nearly discrete than smoothly continuous.

How can Rounding Errors be "Randomly Perturbed" ?

Worst way (1959): IBM 7030 "Stretch"

Setting a control-bit to 1 turns on "Noisy Mode" which forces a 1 onto the last bit of every floating-point operation's result; toggle that control bit at random.

Worse way (late 1970s): J. Vignes' patented "CESTAC"

Randomly add ±1 or 0 to the last bit of every floating-point operation's result.

The foregoing schemes violate both Theorems on a previous page.
The following scheme conserves the Exact Cancellation Theorem.

Bad way (1980s): Toggle IEEE 754/854's Directed Roundings

Control bits can be set (from some compilers) to round every floating-point operation towards $+\infty$, or towards $-\infty$, instead of the default "to nearest". Toggle those control bits at random. (This method circumvents Vignes' patent.)

All of these schemes can spoil algorithms that would have worked well without random perturbation. None of these schemes escapes from the discrete behavior that often befalls roundoff. Consequently, repeated recalculation with randomly perturbed roundoff generates a population of randomized results of which neither the desired (infinitely precise) result nor the result from ordinary floating-point computation need be typical members.

Here follow some simple examples showing how CESTAC can fail:

PROSOLVEUR's Welcoming Screen:

```
==========================================================================

      (c) Copyright 1987 - LA COMMANDE ELECTRONIQUE - Tous droits réservés

                              PROSOLVEUR

            »  ProSolveur Version 1.1  par Stephan G. POPOVITCH  «

                        Frappez une touche pour continuer

==========================================================================
```

This software tries to solve small systems of equations on an IBM PC, and uses Vignes' CESTAC scheme, i.e. Randomized Recomputation, to assay the accuracies of results. ProSolveur then displays only those figures it "knows" to be correct. Of the many ways ProSolveur can go astray, only those we believe characteristic of CESTAC are exposed by the examples exhibited here.

ProSolveur displays its results and the user's data and equations in two panels under headings of which only the following need be explained:

st	Status of an entry:	p means parameter (constant); i means "inconnu" (unknown).
entrée	Initially, user's guess; afterwards, ProSolveur's résultat.	
±(%)	Percentage uncertainty ProSolveur attributes to entrée or résultat.	
unité	Unit ($, Km, Kg, sec., ...) if one is chosen by the user.	
id	Line number for an equation, or for a comment starting with * .	
fichier	Name of the disk(ette)-file containing this line.	

The command line beneath the panels displays the id numbers of equations that ProSolveur has been asked to solve, and also its warning messages.

2x2 **Problem** submitted thrice to ProSolveur :

```
====================== variables ====================================
st   entrée          ± (%)       nom          unité       résultat
p      4194304.000               A
i                                x
p      4194303.000               B
i                                y
p      4194302.000               C
p            3.000               p
i                                X
i                                Y
i                                µ
i                                ß
====================== équations ====================================
id    équation                                              fichier
(1)                                                         << ? >>
(2)   A*x + B*y = 0                                         2X2
(3)   B*x + C*y = p                                         2X2
(4)                A*X + B*Y = 0                            2X2
(5)                B*X + C*Y = p                            2X2
(6)                              A*µ + B*ß = 0              2X2
(7)                              B*µ + C*ß = p              2X2
=====================================================================
no des équations du système à résoudre : 2:7
```

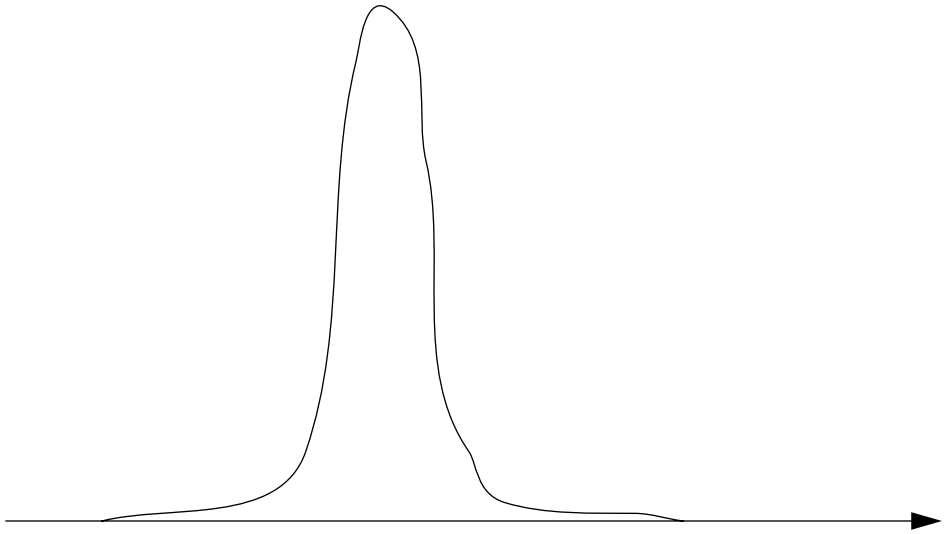
Results delivered by ProSolveur :

```
====================== variables ====================================
st   entrée          ± (%)       nom          unité       résultat
p      4194304.000               A
i         1.3E+007   1           x                        1.3E+7
p      4194303.000               B
i        -1.3E+007   1           y                       -1.3E+7
p      4194302.000               C
p            3.000               p
i          1.2E+007   1          X                        1.2E+7
i         -1.2E+007   1          Y                       -1.2E+7
i     12509610.504               µ                        1.2509611E+7
i    -12509613.487               ß                       -1.2509613E+7
=====================================================================
```

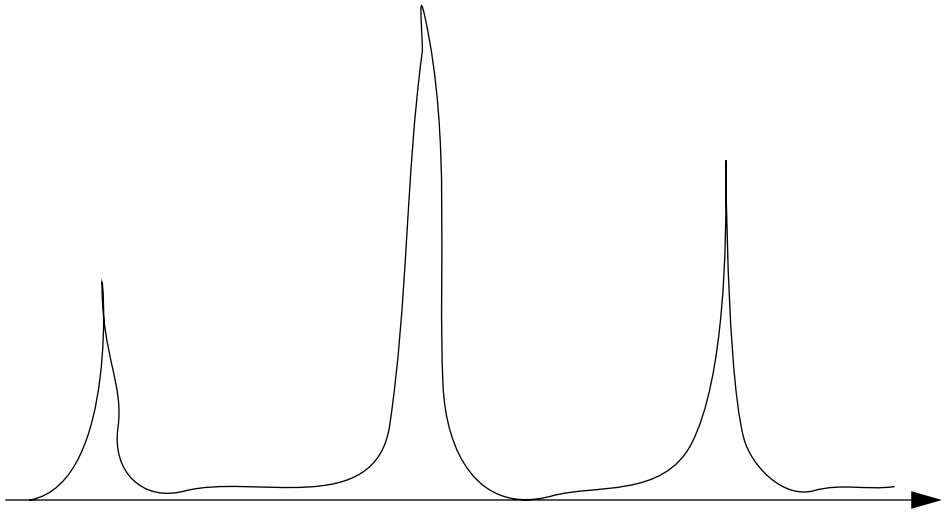
Comment: Since the determinant of the equations is A*C-B*B = -1 , the correct results for this ill-conditioned system should be

```
       x = X = µ = 3B = 12582909   and   y = Y = ß = -3A = -12582912 .
                          ~~~~~                                ~~~~~
```

ProSolveur's extravagantly optimistic claims for the accuracies of µ and ß indicate that the three "random" samples drawn by CESTAC are far too few because they are drawn from a nearly *discrete* rather than continuously distributed population.



Presumed and Expected Continuous Distribution of Errors



Actual Near-Discrete Distribution of Errors

Relatively few samples were drawn, so they succumbed to the non-negligible risk that all may fall under the same outlying peak, thus posing as closely spaced (and hence allegedly "accurate") estimates of an actually inaccurate result.

ProSolveur **miscalculates** the ratio of areas of two narrow triangles:

```
====================== variables ======================================
st   entrée          ± (%)       nom          unité       résultat
i    1.              10          r                        1.
p    1.234567E+6                 x
p    1.234567E+6                 y
p    1.043E-8                    z
====================== équations =====================================
id    équation                                                 fichier
(1)                                                            << ? >>
                  (x+y+z)*(z-(x-y))*(z+(x-y))*(x+(y-z))
(2)   r = sqrt( ------------------------------------- )    RATAREA
                (x+y+2z)*(2z-(x-y))*(2z+(x-y))*(x+(y-2z))
======================================================================
```

Comment: Equation (2) defines

$$r = \text{Area\_of\_Triangle}(x, y, z) / \text{Area\_of\_Triangle}(x, y, 2z) .$$

The triangles exist provided sorted side-lengths x, y, 2z, z satisfy

$$x \geq y \geq 2z \geq 2(x-y) ,$$

and then roundoff cannot significantly degrade this formula's accuracy on any commercially significant North American computer except most CRAYs. Moreover, then $0 \leq r \leq \sqrt{(5/16)} = 0.5590169944$. But ProSolveur alleges that $r = 1 \pm 0.1$ when the correct result above should be $r = 0.5000$.

ProSolveur's troubles may be attributable to CESTAC's disregard for the Exact Cancellation Theorem, not to mention parentheses. In some extreme cases, ProSolveur may even declare data to be invalid when it isn't:

```
====================== variables ======================================
st   entrée          ± (%)       nom          unité       résultat
i    1.75657E-5                  r
p    1.234568E+6                 x
p    1.234567E+6                 y
p    1.00000000023               z

====================== équations =====================================
              . . .
======================================================================
Nombre indéfini => ( racine d'un nombre négatif )
======================================================================
```

The correct result should be $r = 1.238278374E\text{-}5$ without any $\sqrt{(\text{negative})}$.

Wrong by several Orders of Magnitude:

```
====================== variables ====================================
st   entrée          ± (%)       nom          unité       résultat
i    3E+8                        t                        3.0000000E+8
i    3E+8                        L                        3.0000001E+8
i    3.3E+9                      M                        3.3000000E+9
i    9E+9                        N                        9.0000000E+9
i    5.590164        1E-5        z                        5.590164
i    61/11                       y                        61/11
i    11/2                        x                        11/2
p    6                           b
p    5                           c
p    3E+8                        a
====================== équations ====================================
id    équation                                                 fichier
                        (M-N/(L-(M-N/z)/(L-(M-N/y)/z)))
(2)   t = L -  ---------------------------------------------    J-MM
               (L-(M-N/(L-(M-N/y)/z))/(L-(M-N/z)/(L-(M-N/y)/z)))
(3)   z = L - (M - N/x)/y                                      J-MM
(4)   y = (b*b+c*c)/(b+c)                                      J-MM
(5)   x = (b+c)/2                                              J-MM
(6)   L = a+b+c                                                J-MM
(7)   M = a*(b+c)+b*c                                          J-MM
(8)   N = a*b*c                                                J-MM
======================================================================
```

This example is adapted from one constructed by Jean-Michel Muller. The correct value for t is not 3.0000000 E+8 as claimed so confidently, but

t = (b^7 + c^7)/(b^6 + c^6) = 358061/62281 = 5.7491209197 .

This was confirmed by using DERIVE on a PC.

Why does ProSolveur get t so wrong, yet claim it is so accurate? The same thing happens with every computation that is virulently unstable but in a way that almost always diverges to the same wrong destination, so randomized recomputation almost always yields the same wrong result. A similar phenomenon is found among certain unstable differential equations.

(This example is really a numerically unstable recurrence

x[n+1] = L - (M - N/x[n-1])/x[n] for n = 2, 3, 4, ...

whose particular solution is

x[n] = (b^(n+1) + c^(n+1))/(b^n + c^n) --> b ,

but whose perturbed solution is

x[n] = (∂*a^(n+1) + b^(n+1) + c^(n+1))/(∂*a^n + b^n + c^n)
--> a as n --> +∞ .

Here ∂ is like roundoff, about 1E-16 .)

Our examples must weigh heavily upon the mind of anyone who attempts to assess how accurate a computation is merely by repeating it on different machines or on the same machine with different precisions or with different rounding procedures. The examples confirm a truism:

Nobody can know even roughly how wrong a computation is without knowing at least roughly what would have been right.

Analogous difficulties afflict recomputation from randomly perturbed input data, though perhaps less severely. The worst risk is that a program will wrongly be declared numerically stable only because the data that would upset it has yet to be discovered. A less dangerous risk is that data perturbations will destroy data correlations that would, if preserved, allow a program to get correct results.

There is no substitute for mathematically correct error-analysis.

Approximate arithmetic cannot cohabit safely with approximate logic; on the contrary, as an old Jewish saying warns,

"Almost All True" = "Entirely a Lie."

Why then do the advocates of Randomized Recomputation push it so zealously ?

It usually works; and when they see it fail they study and revise the computation until they get their scheme to work. Still, their scheme is liable to fail in just those rare situations when we most need to be warned that computation has gone astray; how then can we know that their scheme has failed to give a warning?

Something ostensibly similar could be said about Interval Arithmetic, which always provides pessimistic error bounds; when those bounds are outrageously pessimistic, zealots study and revise the computation until the pessimism abates to a tolerable level. However,

Interval Arithmetic never lulls its users into a false sense of security.

Now for something entirely different:

Bugs

How likely is one to bite you ?

Example:

Bug in MicroSoft's spreadsheet EXCEL
versions 3.0 to 7.0
for Macintosh, Windows, OS/2.

Type in number 1.40737488355328 ; it turns into 0.64 .

Similar troubles afflict any number with the same 15 digits:

14.0737488355328
140.737488355328
1407.37488355328
. . .
14073748835532.8
140737488355328

and similarly 281474976710656 and 562949953421312 when entered or resulting from certain computations.

How hazardous is this bug ?

Does your assessment of risk change when you learn that

$$140737488355328 = 2^{47}$$
$$281474976710656 = 2^{48}$$
$$562949953421312 = 2^{49} \quad ?$$

(These are the only digit strings believed to pose a hazard; and patches are available from MicroSoft to correct this bug in Excel versions 5.0c and later.)

Subjective Probabilities:

Some people behave as if they believed that the probability of encountering any preassigned noninteger real number in the course of a computation is practically zero, and the probability of encountering any nonzero integer N is slightly smaller than some Constant / N . Therefore, they take the Excel bug much more seriously after they discover that those 15-digit numbers can be generated from such simple expressions.

FIST bugs:

(Floating-point --> Integer convert and STore)

Symptoms: My program misbehaved mysteriously, but in the same way, on two different machines, ...

Intel 386 with Cyrix 83D87 numeric coprocessor designed in 1984 near Dallas TX .

Intel Pentium with on-chip floating-point designed in 1988 near San Francisco.

Except for last-(64th-)bit differences in transcendental functions, their arithmetics should be identical.

Inference: My program's logic is defective ?

But the defect eluded capture.

It turned out that both machines disliked numbers 65535.xxxx ; the Cyrix chip disliked -65535.xxxx too; and Pentium disliked 0.0625 , 0.125 and 0.1875 .

(See my "Beastly Numbers" in http://http.cs.berkeley.edu/~wkahan/tests/numbeast.ps etc.)

(These bugs are absent from current production.)

FIST bugs, continued.

Acting on a tip, I investigated the following program:

INTEGER*2 J ... 16-bit integer: $-2^{15} \le J < 2^{15}$
REAL*8 X
. . .
J = ceil(X) ... = least integer no less than X .
. . .
Overflow occurs if $X \le -2^{15} - 1$ or $X > 2^{15} - 1$.
Intel's Convention: Overflow to $J = -2^{15}$.

Bug: J = 0 instead of -2^{15} when ...
on both machines, $2^{16} - 1 < X < 2^{16}$.
on Cyrix 83D87, $-2^{16} < X < 1 - 2^{16}$.
on Pentium, X = 1/16 , 1/8 or 3/16 .

How likely are two machines,
designed so far apart in time and space,
both to dislike 65535.xxxx ?

These bugs bit some purchasers of the chips.
How likely is such a bug to bite you ?

If your job is to find bugs on such chips before they go into production, what are your chances of finding bugs like these ?
... by random testing ?

What can be done to enhance the likelihood that such bugs will be revealed by tests before production begins?

Validation of Designs and their Implementations

A problem at Many Levels.

Certainty is unachievable; Proofs and Tests merely Corroborate each other.

Proof: A computation, vulnerable to error, therefore incapable of conveying certainty, but designed to enhance confidence and understanding, if it is short.

Test: A computation, vulnerable to error, intended to expose errors.

Where should tests be concentrated to enhance their yield of design errors ?

At and *Near* Singularities and Interfaces .

(Some singularities can be reduced or hidden; cf. Meromorphic functions.)

=> Only implementations designed with testing in mind can be tested economically; they have fewer singularities.

=> Random testing is unavoidable lest some singularities be overlooked.

=> Personnel policies determine the return on validation:

- Designers <—> Testers.
- Know history lest it repeat.

Ideally, **all** engineers should be rotated through
Design, Testing, Production, Customer support, Marketing support.
At the same time, someone has to take responsibility for a valuable accumulation of experience derived from tests and blunders, and its continual refinement into something that aids design rather than inhibits it. Someone special.

Danger: Long exposure to self-serving " Realities of the Marketplace " breeds Cynicism and Shortsightedness.

Man’s instinctive urge to cover up his mistakes is as strong as a dog’s instinctive urge to bury its bone.

=> When a mistake is discovered, engineers come under intense pressure to "Assess our Exposure" and hence to invent probabilities.

=> Excuses are found to postpone correcting mistakes;

if you wait long enough, they may become "Features" .

=> The longer you postpone corrections,
the more they cost,
probably someone else.

. . . . . .

The best defence against these temptations is strength of character.

Example: Pentium's FDIV Bug of 1994-5 : (Floating-point DIVision)

A new (for Intel) and faster division algorithm was devised.

Its correctness was "Proved" by a slightly defective proof.

A transcription error introduced a flaw in the implementation.

This flaw was hidden by the proof's defect.

Nobody thought to create new tests appropriate for the new division algorithm.

Because of measures taken to speed up the division process, the flaw would affect roughly one division in eight or nine billion divisions generated at random; it could not be expected to be revealed by the few billion random and other tests that the old division algorithm had passed. The new division passed too.

The flaw went unnoticed for about a year; it was revealed first during Intel's tests of a successor chip's division compared with Pentium's. The flaw's "probability" of less than 10^{-10} was interpreted as "inconsequential" so it was kept secret.

The flaw was rediscovered a few months later by a customer, Prof. T.M. Nicely, and made its way to the Wall St. Journal.

Later, a test program SRTEST, focussing its search at and near known singularities for "SRT" division algorithms of the kind used in Pentiums, found the FDIV flaw after the 356th division tested, and found it 2294 times in the first million divisions tested. Was this just luck ? (For Intel it would have been $450,000,000 luckier if I had written it 3 years earlier.)

The inner loop of the test consists of about a dozen arithmetic operations, half of them devoted to deciding whether a quotient is correct. Therefore, each test datum { Numer., Denom.} costs only about six arithmetic operations to generate from what has to be a comparatively simple formula. Evidently ...

Algebraic sets may look complicated until we discover how they were generated.

=> The probability of an unwelcome occurrence is best not estimated out of ignorance.

Postscript:
So far as I know, nobody besides Prof. Nicely has reported a "Natural" occurrence of the flaw although there are still over a million defective Pentiums out there. Intel now offers to replace defective Pentiums, and publishes known flaws.

Epilogue: Has any real calamity ever been traced to an under-estimated rounding error?

The decaying accuracy of Patriot anti-missile batteries, and their consequent failure to intercept Scud missiles after more than several hours in action during the Gulf War of 1995, could be attributed to accumulating roundoff, though not in floating-point.

More typical of losses to roundoff is the experience of the Vancouver BC stock exchange about two decades ago. Their published stock index, intended to reflect the average of current mining stock prices on that exchange, was falling ever farther below the actual average. The trouble was traced to chopped floating-point arithmetic on an IBM /370 ; every transaction updated the index by a small amount, but every such update was chopped to the machine's REAL*4 format, thus incurring a loss on average of 0.5 ulp per transaction. The updates are now rounded instead of chopped; a slight bias will cause the published index to exceed the actual average by an amount that grows very slowly over time. I expect that bias to persist until the computer is replaced by one that rounds updates without bias according to IEEE Standard 754 for Binary Floating-Point Arithmetic.

Competent engineers rightly distrust all numerical computations and seek corroboration from alternative numerical methods, from scale models, from prototypes, from experience, ...

The most likely hazard from roundoff to Scientific and Engineering work arises during simulations intended to predict how a proposed device will behave. If roundoff changes a prediction from "success" to "failure" then the proposal may well be abandoned before the error is discovered. Then the error may stay hidden until later someone else succeeds with a device similar to the abandoned proposal and the question "How did they do that?" arises.

See my paper "A Survey of Error Analysis" (1972) in *Information Processing '71* (Elsevier).