Miscalculating Area and Angles of a Needle-like Triangle

(from Lecture Notes for Introductory Numerical Analysis Classes)

Prof. W. Kahan

§0. Abstract:

This case study drawn from an elementary numerical analysis course is aimed at computer language designers and implementors who took no competent course on the subject or forgot it and consequently subscribe to principles of language design inimical to the best interests of writers and users of software with a little floating–point arithmetic in it. Though triangles rarely matter to computer language designers and implementors, recollections of their high–school trigonometry and calculus will enable them to follow the discussion, which is intended primarily to expose and correct common misconceptions. The first of these is that classical trigonometric formulas taught in schools and found in handbooks and software must have passed the Test of Time. Actually they have withstood it; they are unnecessarily inaccurate, sometimes badly, for some data some of which is unexceptionable. Better formulas are supplied here. Even if these are impeccably accurate and cost no more than the classical formulas they should supplant, the expectation that the better formulas will supplant the worse is based upon misconceptions too (see pp. 6, 10, 17 and 22). Other misconceptions addressed herein (on indicated pages) are …

- That subtractive cancellation always causes numerical inaccuracy, or is its only cause. (3, 6)
- That a singularity always degrades accuracy when data approach it. (4, 6, 8, 9, 11, 13-17)
- That algorithms known to be numerically unstable should never be used. (11)
- That arithmetic much more precise than the data it operates upon is pointless. (8, 10-13, 15, 18-22)
- That modern " Backward Error–Analysis " explains everything, or excuses it. (13, 15-18)
- That bad results are due to bad data or bad programmers, never to a programming language. (10-13, 19-21)

Misconceptions like these hinder programming languages from conveying to owners of by far the majority of computers now on desk–tops the benefits of their hardware's superior floating–point semantics. (10-12, 20-21)

Contents:

Miscalculating Area and Angles of a Needle-like Triangle

(from Lecture Notes for Introductory Numerical Analysis Classes)

Prof. W. Kahan
Math. Dept., and E. E. and Computer Sci. Dept.
University of California, Berkeley CA 94720-1776

Readers who do not enjoy reading mathematical formulas may skip to §10 on page 10 after reading this page.

§1. Classical Formulas for Area Δ and Angle C :

Given the side–lengths a, b, c of a triangle, classical trigonometric formulas determine its area

$$\Delta(a, b, c) := \sqrt{\big(s\,(s-a)\,(s-b)\,(s-c)\big)}\,, \qquad \text{where} \quad s := (a+b+c)/2\,,$$

(this formula goes back two millennia to Heron of Alexandria) and determine the angle

$$C(a, b, c) := \arccos\big(\,(a^2 + b^2 - c^2)/(2ab)\,\big) \;=\; 2\arctan\Big(\sqrt{\big(\,(s-a)\,(s-b)/(s\,(s-c))\,\big)}\,\Big)$$

opposite side c . However, rounding errors can undermine the relative accuracy of formulas like these so badly as to invalidate them computationally whenever the triangle is too nearly like a needle,— that is, whenever some two sides add up to scarcely more than the third. For instance, when c is very tiny compared with a and b , which must then be very nearly equal, roundoff in s can be almost as big as c , and then (s-a) and (s-b) can be left relatively inaccurate after cancellation; then too, the argument of arccos(...) can be so near 1 that its roundoff causes a relatively big error in its tiny computed value C . Examples appear below.

Another defect in the formulas above is their occasional failure to warn when data, for example

$$a = -3\,, \quad b = 4 \quad \text{and} \quad c = 2\,,$$

violate the constraints that ought to be satisfied by the sides of a real triangle, namely

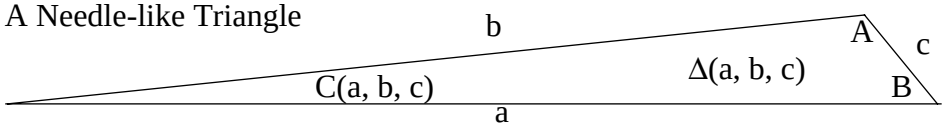
$$0 \le a \le b+c \quad \text{and} \quad 0 \le b \le c+a \quad \text{and} \quad 0 \le c \le a+b\ .$$

Such a defect seems easy to cure, but roundoff can obscure borderline cases if it is ignored.

One purpose of this article is to exhibit better formulas that warn when the data cannot be side–lengths of a real triangle and whose results otherwise are correct to almost as many significant figures as the computation carries regardless of the triangle's shape. These formulas work well on almost all computers and calculators, including all IBM mainframes, all North American personal computers and workstations, and all HP calculators. (CRAY's X-MPs to J90s are the exceptions for lack of a guard digit during subtraction.) But our better formulas get little use partly because they contradict a few common misconceptions about roundoff in floating-point: one is that cancellation is *always* bad news; another is that a singularity, near which small changes in input data can drastically change a desired result, *always* degrades accuracy.

Besides exhibiting better formulas, this article explains why they are correct despite roundoff, and displays numerical results obtained from programs on an HP-15C shirtpocket calculator.

A Needle-like Triangle



§2. How to compute Δ :

First sort a, b, c so that $a \geq b \geq c$; this can be done at the cost of at most three comparisons. If $c-(a-b) < 0$ then the data are not side-lengths of a real triangle; otherwise compute its area

$$\Delta(a, b, c) := \tfrac{1}{4}\sqrt{\big((a+(b+c))\,(c-(a-b))\,(c+(a-b))\,(a+(b-c)) \big)} .$$

Do not remove parentheses from this formula! It cannot give rise to $\sqrt{(< 0)}$.

§3. How to compute C :

If necessary, swap a and b so that $a \geq b$. Then perform two more comparisons to decide whether and how to compute an intermediate quantity μ thus:

If $b \geq c \geq 0$ then $\mu(a, b, c) := c-(a-b)$
elseif $c > b \geq 0$ then $\mu(a, b, c) := b-(a-c)$
else the data are not side–lengths of a real triangle.

If μ has been computed, attempt to compute angle

$$C(a, b, c) := 2 \arctan\Big(\sqrt{\big(((a-b)+c)\,\mu(a, b, c) / ((a+(b+c))((a-c)+b)) \big)} \Big) .$$

Once again, do not remove parentheses. Now $\sqrt{(< 0)}$ will be encountered if and only if the side–lengths do not belong to a real triangle. Division by zero may be encountered; if so, arctan(positive/0) = arctan($+\infty$) = **{** $+\pi/2$ or 90° **}** is well–determined but arctan(0/0) is *NaN* (*N*ot-*a*-*N*umber) and deservedly deemed undeterminable for a degenarate triangle.

§4. Why Cancellation Cannot Hurt:

It is not hard to prove that if p and q are two of a computer's floating–point numbers, and if $1/2 \leq p/q \leq 2$, then p-q is a floating-point number too, representable exactly in the computer, unless it underflows. But we shall ignore spurious over/underflow phenomena since we can evade them by scaling the data in all but the most extreme cases. And we shall assume that when p-q is exactly representable as a floating-point number then subtraction will deliver that value exactly, as it does on all but a few aberrant machines like CRAY X-MPs. Therefore cancellation in p-q introduces no new error that was not already present in p and q . We shall find that no error is already present when cancellation occurs in our formulas for Δ and C , so cancellation cannot hurt their accuracy.

Δ and C depend upon four factors each of the form $x \pm (y \pm z)$; if we prove that each factor is accurate to within a unit or two in its last significant digit carried, then we shall conclude easily that Δ and C must be accurate to within a few units in their last digits. The factors fall into two classes: Some are sums of two positive quantities each accurate to within a unit in its last digit; each such sum must obviously be accurate to within a unit or two in its last digit. The other factors are differences between two positive quantities each of which we shall show to be exact, so these factors will turn out to be accurate to within a unit in the last digit too. And then the factors must all be non–negative whenever the data satisfy the constraints that ought to be satisfied by the side-lengths of a triangle, as we shall henceforth take for granted for the sake of argument.

Let us consider the factor μ := {c-(a-b) or b-(a-c) according as $b \geq c$ or not} . If $b \geq c$ then $c \leq b \leq a \leq b+c \leq 2b$, so (a-b) must be exact. If $b < c$ then either $a < c$, in which case μ is an accurate sum of two positive quantities, or else $a \geq c$ and then $b < c \leq a \leq b+c < 2c$, in which case (a-c) must be exact. In all these cases, μ must be accurate within a unit or two in its last digit, as claimed. Similar reasoning copes with the factor (a-c)+b in the formula for C , and with the factor c-(a-b) in the formula for Δ . All the other factors are sums of positive quantities.

When the data are not side-lengths of a real triangle, attempts to calculate Δ or C encounter a negative factor that can be proved correctly negative despite roundoff even if it is inaccurate.

§5. Examples:

Table 1 below exhibits, for each set of data a, b, c, the values of Δ and C calculated on an HP-15C into which was programmed (with full respect for parentheses) the formulas

$\Delta' := \sqrt{(\ s\ (s-a)\ (s-b)\ (s-c)\ )}$ after $s := ((a+b)+c)/2$,

$\Delta := \frac{1}{4}\sqrt{(\ (a+(b+c))\ (c-(a-b))\ (c+(a-b))\ (a+(b-c))\ )}$ after sorting $a \geq b \geq c$ etc.,

$C'' := \arccos(\ ((a^2 + b^2) - c^2)/(2ab)\ )$,

$C' := 2\arctan(\ \sqrt{((s-a)\ (s-b)/(s\ (s-c)))}\ )$ after $s := ((a+b)+c)/2$, and

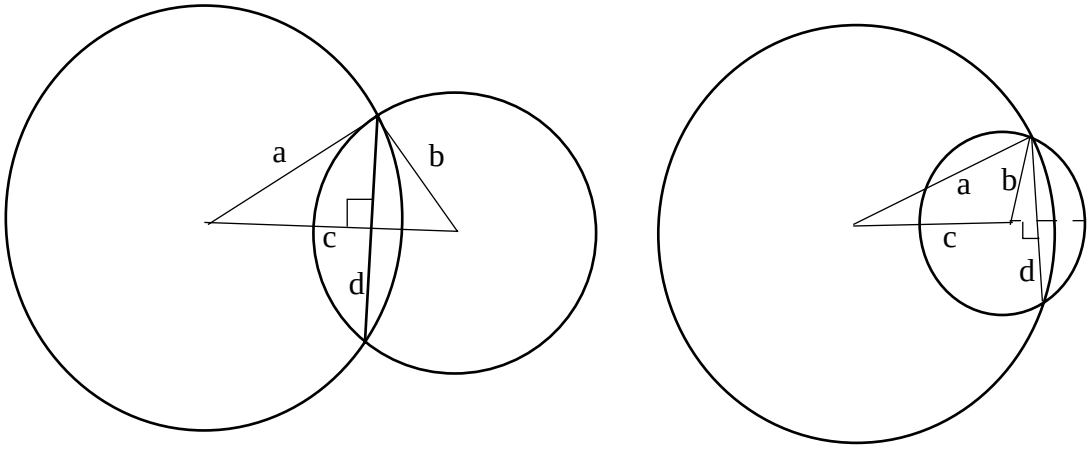
$C := 2\arctan(\ \sqrt{(\ (c+(a-b))\ \mu/((a+(b+c))((a-c)+b))\ )}\ )$ after $\mu := c-a+b$ etc.

Table 1: Area Δ and Angle C

a	b	c	Heron's Δ'	Accurate Δ	C" °	C' °	Accurate C °
10	10	10	43.30127019	43.3012702**0**	60	60	60
-3	4	2	**2.905**	Error	**151.045**	**151.045**	Error
100000	99999.99979	0.00029	**17.6**	9.999999990	**0**	**2.02**E-7	1.145915591E-7
100000	100000	1.00005	500**10.0**	50002.50003	**0**	5.730**72**E-4	5.72986443E-4
99999.99996	99999.99994	0.00003	**Error**	1.118033988	**0**	**Error**	1.281172578E-8
99999.99996	0.00003	99999.99994	**Error**	1.118033988	48.18968509	**Error**	48.18968510
10000	5000.000001	15000	**0**	612.372435**8**	180.00**0**	180.00**0**	179.9985965
99999.99999	99999.99999	200000	**0**	Error	**180**	**180**	Error
5278.64055	94721.35941	99999.99996	**Error**	0	**Error**	**Error**	180
100002	100002	200004	0	0	**Error**	180	180
31622.77662	0.000023	31622.77661	0**.447**	0.327490458	**90**	7**0.5**	64.22853824
31622.77662	0.0155555	31622.77661	246**.18**	245.9540000	90.0**0**	89.9631**87**	89.9631527**6**
a	b	c	Heron's Δ'	Accurate Δ	C" °	C' °	Accurate C °

Digits known to be wrong are displayed **bold**. The HP-15C carries ten significant decimal digits. The formulas for Δ and C were also programmed in BASIC into an HP-71B , which carries twelve significant digits, to confirm that the HP-15C calculated its values of Δ and C (in degrees) correctly to at least nine significant digits.

Example: Two Circles' Intersections



Let d be the distance between the two intersections of two circles, one of radius a and another of radius b , with centers separated by a distance $c \le a + b$.

$$\text{Then} \quad d = 4\cdot\Delta(a, b, c)/c \ .$$

As the distance c between centers increases from $|a - b|$ to $a + b$, the distance d between intersections rises from 0 to its maximum $2\cdot\min\{a, b\}$ and falls back to 0 . That maximum is reached when $c = \sqrt{(\ |\ a - b|\cdot(a + b)\ )}$; then if radii a and b are nearly equal the triangle with edge-lengths a, b, c becomes very narrow though they determine d relatively accurately.

When the circles are specified by their equations in Cartesian coordinates, equations like

$$x^2 + y^2 - 2X_a\cdot x - 2Y_a\cdot y \ - A = 0 \ ,$$

their centers and radii must be inferred from the equations' coefficients. For example, this circle's center is at (X_a, Y_a) , and its radius is $a := \sqrt{(\ A + X_a^{\,2} + Y_a^{\,2}\ )}$. The circles' centers are separated by $c := \sqrt{(\ (X_a - X_b)^2 + (Y_a - Y_b)^2\ )}$. Even if the accurate formula for $\Delta(a, b, c)$ is used, roundoff that contaminates the computed lengths a, b, c can degrade the accuracy of d when the circles are almost tangent, in which case the edge-lengths a, b, c belong to an almost degenerate triangle. Only if d is relatively tiny is degradation severe. It is unavoidable if the equations' coefficients come contaminated by roundoff already. If not, the simplest expedient is to perform all computations in arithmetic at least twice as precise as holds the coefficients. If arithmetic that precise is unavailable, some differences that would mostly cancel may be worth computing in tricky ways. For example,

$$a - b = \big(\ (A - B) + (X_a - X_b)(X_a + X_b) + (Y_a - Y_b)(Y_a + Y_b)\ \big)/(a + b)$$

is worth trying when a and b are both much bigger than c .

Problem 23 on pp. 152-3 of *Floating-Point Computation* by Pat H. Sterbenz (1974, Prentice-Hall, New Jersey) concerns Cartesian coordinates of two circles' intersections computed using the accurate formula for $\Delta(a, b, c)$, but not the foregoing tricky difference.

§6. Opposite Side c and Adjacent Angle B :

Two more formulas for other elements of a triangle are discussed below. One formula computes the length of side c opposite angle C . The other formula computes angle B given two sides a and b and non–included angle A . The formulas usually printed in texts and handbooks (like pp. 341-2 & 354 of *Math. Tables from Handbook of Chem. and Phys.* 11th ed. (1959) Chem. Rubber Publ., Cleveland) become numerically inaccurate under certain circumstances. Better formulas are presented below.

§7. How to compute c :

If c is too much smaller than a and b , roundoff spoils the usual textbook formula for side–length $c := \sqrt{(a^2 + b^2 - 2\,a\,b \cos C)}$. A small change turns it into an always accurate formula

$$c := \sqrt{\left((a-b)^2 + 4\,a\,b \sin^2(C/2) \right)} .$$

§8. How to compute B :

Given two sides a and b and a non–included angle A between 0 and 180˚ , texts usually invoke the " Law of Sines," sin(B)/b = sin(A)/a , to obtain the other non–included angle

$$B := \arcsin((b/a) \sin A) .$$

But this is sometimes ambiguous; when $A < 90^\circ$ and a < b , this other angle could as easily be 180˚-B as B . And no other non–included angle B exists at all when either

$$a < b \sin A \quad \text{and} \quad A \le 90^\circ , \quad \text{or} \quad a < b \quad \text{and} \quad A \ge 90^\circ ;$$

in the latter case the foregoing formula for B can produce a plausible but spurious result. Since no subtractive cancellation can occur in it, the formula's ability to lose accuracy may come as a surprise. It can lose up to half the figures carried when B is too close to 90˚ . Some of that loss is avoidable when A is close to 90˚ too, but only by the use of a better procedure. The idea behind a better procedure is to select, in a data–dependent way, the least inaccurate from a list of algebraically equivalent but numerically non–fungible formulas like these:

$B_S(a,A,b) :=$ arc**s**in((b/a) sin A) ;

$B_T(a,A,b) :=$ A + 2 arc**t**an((b-a)/**(** (a/tan A) + $\sqrt{\left((a/\tan A)^2 - (b-a)(b+a) \right)}$ **)**) ,

$:= 0$ if A = 0 , regardless of a ;

$B_C(a,A,b) :=$ arc**c**os($\sqrt{((b \cos A)^2 + (a-b)(a+b))}$/a) ;

A selection is effected by the following conditional assignments:

$B_2(A,X) :=$ if $A > 90^\circ$ then NaN else the *pair* of angles **{** X , 180˚ - X **}** ;

$B_b(a,A,b) :=$ if b-a < a then $B_T(a,A,b)$ else $B_S(a,A,b)$;

$B_A(a,A,b) :=$ if ($A \ge 90^\circ$ and $(b/a) \sin A \ge 0.756$) then $B_C(a,A,b)$

else if ($A < 90^\circ$ and $(b/a) \ge 0.756$) then $B_T(a,A,b)$ else $B_S(a,A,b)$;

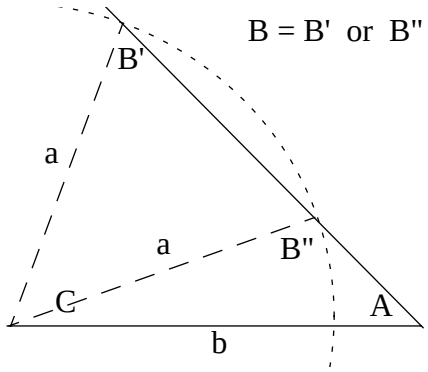
$B(a,A,b) :=$ if a < b then $B_2(A, B_b(a,A,b))$ else $B_A(a,A,b)$.

Why the final selection B(…) is better than the first for calculating B takes a lot of explaining of which only a little will be done here. First switch from degrees to radians (π radians = 180˚) to make derivatives simpler; only the constants 30˚ , 90˚ and 180˚ have to be replaced by $\pi/6$, $\pi/2$ and π above. Then let æ stand for any tiny positive quantity not much bigger than a rounding error in a number near $\pi/2$. On the HP-15C , which rounds to 10 sig. dec., æ is a modest multiple of $5/10^{10}$. On most computers and calculators a number roughly like æ can be obtained from the expression |3(4/3 - 1) - 1| evaluated after " 4/3 " has been rounded to the working precision of the machine's floating–point arithmetic. We assume initially that æ is so tiny that approximations like $\partial f := f(x+\partial x)-f(x) \approx f'(x)\,\partial x$ obtained through the Differential Calculus are satisfactory provided both ∂f and ∂x are of the order of æ . Later this proviso will be relaxed somewhat to accommodate perturbations ∂x of the order of $\sqrt{æ}$.

Because of roundoff, we expect the computed value of (b/a) sin A to be $(1 \pm æ)((b/a) \sin A)$ in formula B_S ; then this formula would deliver $B + \partial B :=$ arcsin((1±æ) sin B) instead of B , with error $\partial B \approx \pm æ \tan B$ except for terms that are negligible so long as $æ \tan B << \sqrt{æ}$. On the other hand, when a > b formula B_C delivers $B+\partial B :=$ arccos((1±æ) cos B) , whence $\partial B \approx \pm æ \cot B$; this explains why formula B_C is better when B is close to $\pi/2$ (or 90˚), and B_S is better when B is close to 0 . But internal cancellation degrades the accuracy of B_C when a < b , in which case its error turns out to be roughly $\partial B \approx \pm æ \cot^2(A) \tan(B)$, which is usually worse than the error in another formula B_T .

When $a \le b$ and $A < \pi/2$ (or 90˚), the error that formula B_T suffers turns out to be roughly $\partial B \approx \pm æ \sin(B-A) \cos(A)/\cos(B)$. This is much smaller than the uncertainty in B_S when A is close to B or close to $\pi/2$. When $a \ge b$ and $A < \pi/2$ the error that formula B_T suffers is about $\pm æ \sin(A-B)$, which is smaller than the uncertainty in B_S provided B is not much smaller than A . Because B_T costs more to compute, it is used only when it is substantially less uncertain than B_S .

When a < b sin A within a few rounding errors, arcsin(> 1) occurs in B_S and $\sqrt{(< 0)}$ in B_T to signal that no angle B exists. Finally, $B_2(…)$ copes with cases when B(…) can take two values B' and B" , or none. (Recall that " *NaN* " stands for " *N*ot *a N*umber.")



§9. How Accurate is B ?

Table 2 compares results from B(a,A,b) and the classical formula B_S(a,A,b) , obtained on an HP-15C carrying 10 sig. dec., with correct values of B computed carrying at least 30 sig. dec. When B takes two values only the smaller is displayed. Data {a,A,b} has been selected to show how narrow is the range of data at which the formulas in question lose accuracy more than slightly. Digits known to be wrong are displayed **bold**; *italicized* values are anomalous.

Table 2: Angle B , New Formula B(…) vs. Classical B_S(…)

a	b	A°	True B°	B(a,A,b)°	B_S(a,A,b)°
3	17	10	79.73894317	79.7389432**6**	79.7389432**6**
3	17	10.16424862	89.99895044	89.999**2**	89.999**2**
0.9999999998	2	28	69.87481897	69.8748189**4**	69.8748189**4**
0.9999999998	2	29.9916	88.70873768	88.708737**3**	88.708737**3**
0.9999999998	2	29.99999999	89.99918045	89.99**84**	89.99**84**
0.9999999998	2	30	NaN	**90**	**90**
1	2	30	90	90	90
0.9999999999	1.999999999	30.00000001	89.99919909	89.998**7**	89.9991**97**
0.9999999999	1.999999999	30	89.99837943	89.998**7**	89.998**2**
0.9999999999	1.999999999	29.9916	88.70873615	88.7087360**0**	88.7087360**4**
0.9999999999	1.999999999	28	69.87481888	69.8748189**0**	69.8748188**6**
18817	21728	59.9	87.42569088	87.4256908**4**	87.425691**3**
18817	21728	60	89.99695511	89.997**1**	89.997**2**
38620	38673	86.9	89.21861758	89.21861758	89.218617**7**
38620	38673	87	89.99935642	89.999355**9**	90.00**0**
4.999999999	5.000000001	88.3	88.30000077	88.30000077	*88.29999**996***
4.999999999	5.000000001	89.99827	89.99939448	89.99939448	*89.99**819***
4.999999999	5.000000001	89.999007	NaN	NaN	**89.9992**
5	5	89.999007	89.999007	89.999007	*89.999**2***
8.000000001	7.999999999	89.999007	89.99837906	89.99837906	89.998**6**
8.000000001	7.999999999	90.000993	89.99837906	89.99837906	89.998**6**
8.000000001	7.999999999	91	88.99999918	88.99999918	88.999999**5**
8.000000001	7.999999999	5.739170509	5.739170508	5.739170508	*5.739170512*
a	b	A°	True B°	B(a,A,b)°	B_S(a,A,b)°

Though more accurate than B_S(…) , apparently B(…) can lose about half the figures carried. This is the worst that can happen, and it happens only when A is noticeably less than 90° and B is much nearer 90° , very nearly a double root of the equation sin(B)/b = sin(A)/a that B must satisfy. This deterioration illustrates an important general principle:

> Nearly double roots are much more sensitive than well isolated simple roots to perturbations like those due to roundoff.

When an equation $f(z) = 0$ has a simple root z (which means that the derivative $f'(z) \neq 0$), changing f slightly to $f+\partial f$ changes z to a root $z+\partial z$ satisfying $(f+\partial f)(z+\partial z) = 0$, whence it follows that $\partial z \approx -\partial f(z)/f'(z)$ if ∂f is tiny enough that terms of order $(\partial f)^2$ can be ignored. But when z is a double root (when $f(z) = f'(z) = 0 \neq f''(z)$) then, depending upon the sign of ∂f , equation $(f+\partial f)(z+\partial z) = 0$ has either no real root or two with $\partial z \approx \pm\sqrt{(2|\partial f(z)/f''(z)|)}$. If z is almost a double root, $\min\{ |\partial f/f'(z)|, \sqrt{(2|\partial f/f''(z)|)} \}$ turns out to be an approximation of $|\partial z|$ adequate for our purposes.

Suppose ∂f is due to roundoff, and therefore proportional to quantities like æ . Then the error ∂z induced in a nearly double root cannot be bigger than something proportional to $\sqrt{æ}$, which explains why at most about half the figures carried to compute z can be lost. (Similarly a nearly triple root could lose at most about two thirds the figures carried, and so on.)

Induced error ∂z can be diminished in only two ways. One carries more figures to reduce æ . The other replaces " $f(z) = 0$ " by an equivalent equation with the same root(s) z and smaller rounding errors relative to its first and second derivatives. For instance, a double root of " $f(z) = 0$ " is a simple root of " $f'(z) = 0$ " ; therefore a nearly double root z of "$f(z) = 0$ " must make $f'(z)$ small. Whenever f satisfies a known differential equation it can be used to replace " $f(z) = 0$ " by an equivalent equation " $f'(z) = \ldots$ " with the same root z but with possibly smaller rounding errors. Otherwise a better replacement for f may be hard to find.

Such a replacement turns $\sin(B) = (b/a)\sin(A)$ into $\cos(B) = \sqrt{((a-b)(a+b) + (b\cos(A))^2)}/a$ advantageously when a/b exceeds 1 slightly and A is near 90˚ . This yields $B_C(a,A,b)$ which, as we have seen, is then accurate in all but the last digit carried under circumstances when the original equation's $B_S(a,A,b)$ could lose up to half the figures carried.

Formula B_T is harder to explain. It was found by a search for a formula that roundoff does not prevent from honoring a well-known geometrical theorem: $\text{sign}(B-A) = \text{sign}(b-a)$. Whenever b = a , such a formula must deliver B = A exactly despite roundoff. Neither B_S nor B_C can do this for every A < 90˚ ; four instances where B_S violates the theorem appear in *italics* in the last column of Table 2. The simplest (not the first) formula found to fit the bill is B_T . Though complicated, it solves an equation $\tan((B-A)/2) = (b-a)/\ldots$ whose rounding errors are attenuated by a factor roughly proportional to $|B-A|$ when it is small regardless of whether B is nearly a double root, as it is when A is near 90˚ too. A double root's proximity spoils B_T far less than it spoils our other two formulas when A < 90˚ and a/b is a little less than 1 .

Our three formulas' uncertainties due to roundoff are summarized in Table 3 below, in which æ stands for an unknown quantity not much (not five times) bigger than a rounding error in numbers near $\pi/2$. (To specify æ more precisely would require tedious attention to details.) Combining the table with a little additional analysis leads to the following conclusions:

Classical formula $B_S(a,A,b)$ can be in error by

$$\pm(180^\circ/\pi)\ \min\{\ æ\tan(B)\ ,\ \sqrt{(2æ)}\ \}\ .$$

Better formula B(a,A,b) is in error only in its last digit when $a \geq b$, and otherwise by

$$\pm(180^\circ/\pi)\cos(A)\ \min\{\ æ\sin(B-A)/\cos(B)\ ,\ \sqrt{æ}\ \}\ .$$

Table 3: Uncertainty ∂B due to Roundoff æ

Formula	Conditions	±(π/180˚) (∂B/æ)
$B = B_S(a,A,b)$	—	min{tan(B) , √(2/æ)} (not used if B/A is near 1)
$B = B_T(a,A,b)$	$a \le b$ & $A < 90^\circ$	min{sin(B-A)/cos(B) , 1/√æ} cos(A)
	$a \ge b$ & $A < 90^\circ$	sin(A-B) (not used if B/A is too small)
$B = B_C(a,A,b)$	$a < b$	min{$\cot^2$(A) tan(B) , ...} (never used)
	$a \ge b$	cot(B) (not used if $A < 90^\circ$ or $B < 49^\circ$)

§10. Must the Classical Formulas be Amended ?

Despite their occasional vulnerability to roundoff, the classical trigonometric formulas for Δ, C, c and B have served mankind for millennia without complaint. They are propagated by popular software like MathCAD® from MathSoft, Cambridge Mass. No text recommends their replacement nor mentions the better formulas presented here. Why fix what ain't broken?

Actually the classical formulas are broken, but so slightly as to escape notice in most instances. These formulas' defects afflict only extreme configurations:

- The triangle is too nearly degenerate — too needle-like — or
- The computed angle B is too nearly a right angle.

Such configurations arise infrequently, and when they do arise the formulas' errors may still go uncorrected for any of a number of reasons:

- Erroneous results are difficult to notice without correct results for comparison.
- Errors may be truly negligible if enough extra figures were carried during computation.
- Errors may be deemed " No worse than the data deserve " because of a mistaken doctrine.

The last two reasons require explanation.

The accuracy of computation depends upon (and is usually less than) its " precision," which is the number of digits carried when arithmetic operations get rounded off. Nowadays floating–point numbers are stored in the memories of computers and calculators with a fixed number of " significant digits " that depends upon the machine and, to a lesser extent, upon the kind of software in use. For instance, the HP-15C calculator stores 10 sig. dec. Most computers' hardware supports two or three of the standard binary floating–point formats named in Table 4 along with their wordsizes and precisions. The third format, unsupported by some computers, may be inaccessible on others for lack of support by a programming language or its compiler.

Table 4: Formats of IEEE Standard 754 for Binary Floating-Point Arithmetic

Format	Name in *C*	Name in Fortran	Wordsize	Sig. Bits	Sig. Dec.	≈ æ ≈
Single Precision:	`float`	REAL*4	4 Bytes	24	6 - 9	$1.2/10^7$
Double Precision:	`double`	REAL*8	8 Bytes	53	15 - 17	$2.2/10^{16}$
Double-Extended:	`long double`	REAL*10 +	≥ 10 Bytes	≥ 64	≥ 18 - 21	$\le 1.1/10^{19}$

Today most programming languages match the precision to which arithmetic is rounded with the precision to which arithmetic operands and results are stored in memory. This is not a Law of Nature; it is a Rule of Thumb inherited from the era of slide rules and promoted to a Law of Language by mistake. In the early 1960s, when compilers had to be simple to fit into the small memories computers had then, compiler writers rationalized this over–simplified way of evaluating expressions by misapplying pejorative terms like " numerical instability " and " ill–condition " that were coming into vogue then. Rather than blame results spoiled by roundoff upon the way compilers evaluated expressions, we were to blame bad results upon bad data or bad algorithms. An algorithm stood convicted of numerical instability if it could be replaced by a new algorithm at least about as fast and accurate as the old for all data, and good for all data for which the old algorithm was bad. By this criterion, the classical formulas for Δ, C and c are unstable. Were no better algorithm known, the data could be convicted of ill–condition if end–figure perturbations in the data changed at least the last several digits of *correct* results. By this criterion, needle–like triangles are mostly ill–conditioned data for Δ, C and c; and for B the ill–conditioned triangles are those that have too nearly a right angle at B.

How far can we trust a chain of reasoning that tells us to condemn right–angled triangles?

The chain has three weak links. First is the assumption that arithmetic precision should match the precision of operands in memory. Actually arithmetic precision is a means to an end and therefore should ideally be chosen with a view to its consequences: Choose enough precision, from what is available with adequate speed, to obtain results of adequate accuracy from the algorithm selected. Occasionally, as we shall see, the required arithmetic precision will exceed the precision of operands in memory. When the required precision is too difficult to ascertain, a prudent policy is to use the widest arithmetic precision available without much loss of speed.

The second weak link is the assumption that unstable algorithms should never be used. There is some truth to this; many an algorithm is useless because it is too inaccurate for practically all data. The classical formulas for Δ, C, c and B are different; they are excessively inaccurate for a fraction of their data so tiny that most computer users will never encounter it and most of the rest will never notice. It's not an unusual situation. Gaussian Elimination without pivotal exchanges is like that; it solves systems of linear equations, solving some quite inaccurately though they are otherwise innocuous. This notoriously unstable algorithm is used anyway (and not just to solve diagonally dominant systems known to be safe) whenever the computational costs of pivotal exchanges would be too burdensome. Intolerable inaccuracy, detectable from dissatisfied equations, occurs rarely because elimination is performed carrying rather more precision than would suffice if pivotal exchanges were used; every additional decimal digit of precision carried reduces the incidence of intolerable inaccuracy by a factor near 1/10 .

Similarly, the tiny fractions of their data for which the classical formulas for Δ, C, c and B are intolerably inaccurate can be attenuated by carrying more digits during arithmetic operations. The attenuation factor is typically 1/10 for every additional decimal digit of precision carried. This factor depends upon three assumptions. One is that the data is distributed randomly and not too nonuniformly. Second, the data's distribution does not change when precision changes. Third, inaccuracy arises from a cancellation–like singularity of the simplest and most common kind but still too complicated to describe fully here. This third assumption is invalid for B .

§11. What Extra Precision Does for B :

Provided the data {a,A,b} is distributed in advance and not too nonuniformly, the fraction of data for which the classical formula $B_S(a,A,b)$ is intolerably inaccurate shrinks by a factor near 1/100 (not 1/10) for every additional decimal digit carried during computation. Carrying three extra decimal digits attenuates the fraction by a factor near $1/10^6$, and so on, until half the precision carried exceeds both the data's precision and the accuracy desired in the result. Beyond that doubled precision, $B_S(a,A,b)$ delivers the desired accuracy for almost all data, all except data less than a rounding error away from the boundary beyond which no real B exists. The better formula B(a,A,b) enjoys the same rapid attenuation starting from a smaller fraction of data. Since this rapid attenuation may come as a surprise, it deserves to be explained.

Revert again to radians instead of degrees, and suppose we wish to compute B with a tiny error smaller than ß using classical formula $B_S(a,A,b)$. Unless $\sqrt{(2æ)} < ß$ already, the error æ tan(B) is too big just when $\tan(B) \geq ß/æ$. This is tantamount to $\csc^2(B) \leq 1 + (æ/ß)^2$ and, because $\sin(B) = (b/a)\sin(A) \leq 1$, places the data {a,A,b} into a region where

$$\sin(A) \leq a/b \leq \sin(A)\sqrt{(1 + (æ/ß)^2)}$$

must be satisfied. This region's width, and therefore its volume, approaches zero like $æ^2$, which explains the rapidity of attenuation. (A slightly messier argument leads to the same conclusion for B(a,A,b) .) When half the precision of the arithmetic exceeds the precision of the data and the desired accuracy, the error in $B_S(a,A,b)$ cannot be worse than $\sqrt{(2æ)} < ß$, which is small enough, unless the computed value $(1 \pm æ)(b/a)\sin(A) > 1 \geq (b/a)\sin(A)$ in which case the result will be an error message instead of B . This can occur only if one end–figure perturbation in the stored data could render B nonexistent. (In some applications this boundary case deserves to be detected and assigned the result $B := \pi/2$ (or 90°) by fiat.)

The phenomenon just explained, attenuation of risk by 1/100 per extra decimal digit, occurs more often than is generally appreciated, usually associated with nearly double roots and with some optimization problems. Every extra decimal digit attenuates risk by 1/10 for most other problems. Were such risk reduction appreciated more widely, naive numerical computation might be less hazardous than it is now under current programming language standards.

For instance take the computer language ***C*** . In the 1970s this language evaluated all floating–point expressions in `double` regardless of the formats, `float` or `double`, of arithmetic operands. Doing so was easier and ran faster on the computer, a DEC PDP-11, on which ***C*** was first developed. Programmers whose data and results were preponderantly `floats` were mostly unaware of the extent to which ***C***'s floating–point semantics all but guaranteed the reliability of their computations. By the mid 1980s the dismal politics of computer language standardization had undermined that reliability by allowing " new " semantics that forced the precisions of arithmetic and operands to match. A few computers ran faster this way. Most computers installed today, AMD–Cyrix–Intel–based PCs and Power-PC/Power-Macs and old 680x0-based Macs, run faster the old way and get better results than when they are forced to abide by the newer semantics. Numerical software, compiled to take full advantage of their floating–point hardware, would run more reliably on these than on other machines. That may happen some day if a proposed revision, C9X, to the ANSI C standard is ever adopted.

§12. How Much Accuracy do Data Deserve ?

We can eliminate the risk of inaccurate results from floating–point computation for a price:— time. Time spent in thought, or time spent computing, or both. Before paying that price we ought to compare it with the cost of inaccuracy. But the comparison, if nontrivial, tends to be imponderable; whenever the the costs of inaccuracy and its elimination are both substantial they can hardly ever be ascertained until after they have been incurred. Rather than embark upon tedious computations or error analyses likely to cost more than their worth, we often fall back upon ancient Rules of Thumb derived from simple answers to simple questions, perhaps too simple. An example is …

" Inaccurate data deserve appropriately inaccurate results."

It looks simple, but it's not. Inattention to its subtleties causes egregious errors to be deemed " no worse than the data deserve " and innocuous data to be condemned as " ill–conditioned." Logical and doctrinal mistakes like these have persuaded too many of us to acquiesce to ill–advised decisions by designers of computer languages like *Java*. Hoping to help correct those decisions, let us expose the mistakes that contributed to them. Our exposé begins with a typical " Backward Error Analysis."

We have seen that the classical formula $B_S(a,A,b) := \arcsin((b/a)\sin A)$ can lose up to about half the digits carried. How bad is that compared to the error B inherits from end–figure errors in its data? The first line in Table 3 came from an assertion that the computed value $B + \partial B$ satisfies $\sin(B+\partial B) = (1 \pm æ)\sin(B)$ in which æ is due to roundoff. To be precise, æ comes from four rounding errors: one in the quotient b/a , one in sin(A) , one in the multiplication of (b/a) sin(A) , and one in arcsin(…) . These errors can be treated as if they belonged to the data by rewriting $B + \partial B = B_S(a+\partial a, A+\partial A, b+\partial b)$ computed *exactly* from perturbed data

$b+\partial b := b\sqrt{(1\pm æ)}$, so $\partial b/b \approx (\pm æ/2)$,
$a+\partial a := a/\sqrt{(1\pm æ)}$, so $\partial a/a \approx -\partial b/b$, and
$A+\partial A := A$, so $\partial A = 0$.

> The error due to roundoff in the classical formula $B_S(a,A,b)$ is no worse than if it had been computed exactly from data {a,A,b} wrong in only their last digit stored. *Since data more accurate than that are too good to expect, the improved formula* B(a,A,b) *is overkill; its better accuracy is better than the data deserve, and all the more so if extra digits are carried during its computation .*

The last sentence, the one in *italics*, is mistaken. The sentence preceding it is correct, though.

One might argue that the sentence in italics is mistaken because the given data could be exact, or because its errors could be correlated in a way that cancels them off instead of being anti–correlated like $\partial a/a$ and $\partial b/b$ above. Correlated errors are crucial to some other calculations, but not to B here. The mistake here was committed by stopping the error analysis too soon, just at the point where an explanation of the error in $B_S(a,A,b)$ was turned into an excuse for accepting it without complaint. Let's pursue the analysis beyond that point.

What purpose does a computation of B serve? It has something to do with a triangle.

§13. A Triangle is an Object:

Not knowing its dimensions precisely, we cannot know precisely which triangle it is though we know it is a triangle. As a software object a triangle responds to inquiries about its constituents, its sides and angles and area, which cannot be arbitrary numbers but should satisfy relations like $A+B+C = 180^\circ$. How consistent with such relations and with each other will responses to inquiries be? Four *italicized* entries in the last column of Table 2 reveal that the classical formula $B_S(a,A,b)$ (but not better formula $B(a,A,b)$) can violate the well-known relation $\text{sign}(B-A) = \text{sign}(b-a)$ slightly. How badly may other inconsistencies sully these formulas?

Consider two ways to compute side–length c. One computes c directly from the same data $\{a,A,b\}$ as were used to determine B; here are conscientious formulas:

$$\begin{aligned} c_b(a,A,b) &:= (a-b)(a+b)/\left(\sqrt{(b\cos A)^2 + (a-b)(a+b)} - b\cos A\right) \quad \text{if } A > 90^\circ\text{; otherwise} \\ &:= b\cos A + \sqrt{(b\cos A)^2 + (a-b)(a+b)} \quad \text{if } 90^\circ \ge A \ge 53^\circ \text{ or } b \le a\text{; otherwise} \\ &:= b\cos A + \sqrt{(a - b\sin A)(a + b\sin A)} \quad \text{if } A < 53^\circ \text{ and } b > a\,. \end{aligned}$$

When c can take two values the larger is c_b. (In the first formula replace $0/0$ by 0. The third can lose up to half the sig. digits carried.) A second way to compute c is from data $\{a,A,B\}$; here are accurate formulas:

$$\begin{aligned} c_B(a,A,B) &:= a\cdot\sin(A + B)/\sin(A) && \text{if } A + B \le 90^\circ\,, \\ &:= a\cdot\sin((90^\circ\text{-}A) + (90^\circ\text{-}B))/\sin(A) && \text{if } A \le 90^\circ < A+B \text{ and } B \le 90^\circ\,, \\ &:= a\cdot\sin((180^\circ\text{-}A) - B)/\sin(A) && \text{if } B \le 90^\circ < A\,, \\ &:= a\cdot\sin((180^\circ\text{-}B) - A)/\sin(A) && \text{if } A \le 90^\circ < B\,. \quad (\text{Unneeded here.}) \end{aligned}$$

In any case a negative or complex c must be replaced by NaN to signal an improper triangle.

Table 5 exhibits the true value of c alongside values computed for $c_b(a,A,b)$ and $c_B(a,A,B)$, with both the better $B = B(a,A,b)$ and the classical $B = B_S(a,A,b)$, on an HP-15C carrying 10 sig. dec. The data $\{a,A,b\}$ were chosen to show how much diversity roundoff can generate regardless of whether the triangle is needle–like. Digits known to be wrong are displayed **bold**.

Table 5: Side c, $c_b(a,A,b)$ vs. $c_B(a,A,B(a,A,b))$

a	b	A°	True c	$c_b(a,A,b)$	$c_B(a,A,B(a,A,b))$	$c_B(a,A,B_S(a,A,b))$
3	17	10.16424862	16.73325549	16.7332**8**	16.7332**4**	16.7332**4**
0.9999999999	1.999999999	30	1.732079091	1.732079091	1.73207**3**	1.73208**2**
18817	21728	60	10865.00000	10865.00000	10864.9**5**	10864.**92**
38620	38673	87	2024.422239	2024.42**3**	2024.422**6**	2023.**99**
49999.99999	50000.00001	88.3	2966.623735	2966.62373**6**	2966.62373**6**	2966.624**4**
49999.99999	50000.00001	89.99827	2.038126012	2.038126011	2.038125**7**	**3.**
49999.99999	50000.00001	89.999007	NaN	NaN	NaN	**1.6**
50000	50000	89.999007	1.733111947	1.733111947	1.733111947	1**.6**
800000000.1	799999999.9	89.999007	36497.51207	36497.5120**6**	36497.4**9**	3**3461.**
800000000.1	799999999.9	90.000993	8767.720918	8767.720920	8767.6**96**	**5731.**
800000000.1	799999999.9	91	11.45973300	11.45973300	11.45**1**	**7.**

Consistency is better with $B(\ldots)$ than with the classical $B_S(\ldots)$, often far better, but flawed.

Inconsistencies arise because the two computed values $c_b(a,A,b) = c(a+da, A+dA, b+db)$ and $c_B(a,A,B(a,A,b)) = c(a+\partial a, A+\partial A, b+\partial b)$ are values of c obtainable exactly from data with *different* roundoff–induced end–figure perturbations {da,dA,db} and {∂a,∂A,∂b} , neither worse than {±æa, ±æA, ±æb} . Although both perturbations appear almost negligible compared with the data, they can change half the digits of c , and change them *differently*, whenever data lies in a narrow boundary–layer of {a,A,b}–space containing two kinds of triangles, those with too nearly a right angle at B and those with c too tiny compared with a and b . Call such data " ill–conditioned " if doing so makes you feel better, but it won't change the facts:

> An otherwise unexceptionable triangle can respond to two inquiries, before and after an inquiry about B , by returning grossly inconsistent values of c . Inconsistency is attenuated, not always eliminated, by using improved instead of classical formulas.

Nobody can say in advance how badly such inconsistencies might disrupt a program's logic. Prudence requires their extirpation, and that is straightforward if the specification of a triangle in object–oriented software is designed by a programmer aware of the problems with classical formulas and acquainted with reliable numerical algorithms to solve some of those problems. A programmer unaware of the problems is unlikely to be enlightened by casual or random testing; those problems are confined to " ill–conditioned " triangles in so tiny a sliver of the space of all triangles that its location, unknown to the unaware programmer, is likely to stay unknown.

Fortunately, the same phenomenon as tends to keep the unaware programmer unenlightened tends to protect his unwitting clients from harm. As arithmetic precision increases beyond what the data " deserve," the sliver of triangles treated too inaccurately by the program shrinks, the likelihood that the programmer will notice it shrinks, and the risk of harm to his clients shrinks, and usually shrinks quickly. As algebra has shown above and a picture will show below, …

> Every three additional decimal digits of arithmetic precision carried, in excess of the precision to which data and results are stored, reduces the incidence of intolerably " ill–conditioned " triangles by factors …
> - near 1/1,000,000 for triangles too nearly right–angled,
> - less than 1/1,000 for triangles too nearly needle–like.
>
> Carrying arithmetic precision somewhat more than twice the precision to which data and results are stored practically eliminates intolerably " ill–conditioned " triangles.

Wider is better. Plus large, c'est mieux. Breiter ist besser. …

§14. Three *Caveats* and a Picture:

This digression is intended to forestall unwarranted generalizations.

First, the incidence in practice of intolerably ill-conditioned data can hardly ever be predicted reliably. Predictions, based upon simulations that generate data randomly distributed uniformly over all possible data, fail typically by orders of magnitude. Why they fail is a story for another day. Still, extra precision generally does reduce that incidence to the extent claimed above.

Second, more precision is no panacea. A few numerical algorithms are so virulently unstable on almost all data that no practicable precision can rehabilitate them. No algorithm mentioned here is that bad, however.

Third, unlike $B_S(\dots)$, some algorithms are degraded by roundoff in ways inexplicable by a satisfactory backward error–analysis. Heron's classical formula Δ' for a triangle's area Δ is one of these. To understand why requires lengthy analysis. Start by deriving expressions like

$$\partial\Delta/\partial a \;=\; \tfrac{1}{8}(b^2 + c^2 - a^2)\; a/\Delta \;=\; \tfrac{1}{4}\, a\, b\, c\, \cos(A)/\Delta \;=\; \tfrac{1}{2}\, a\, \cot(A)$$

for all the first partial derivatives of Δ with respect to its data $\{a, b, c\}$. Perturbing that data infinitesimally to $\{a+da, b+db, c+dc\}$ changes Δ infinitesimally to $\Delta + d\Delta$ wherein

$$d\Delta \;=\; (\partial\Delta/\partial a)\, da \;+\; (\partial\Delta/\partial b)\, db \;+\; (\partial\Delta/\partial c)\, dc \;.$$

Suppose now that $|da|/a \le æ$, $|db|/b \le æ$ and $|dc|/c \le æ$ for any sufficiently tiny positive $æ$; this amounts to allowing arbitrary end–figure perturbations in the data. They lead to

$$|d\Delta|/\Delta \;\le\; æ\,(\, a\,|\partial\Delta/\partial a| \;+\; b\,|\partial\Delta/\partial b| \;+\; c\,|\partial\Delta/\partial c|\,)/\Delta \;.$$

Now two alternatives arise. One is that the triangle's angles are all acute, none bigger than 90°; in this case the last inequality's right–hand side simplifies to $2æ$, which is small (very well–conditioned). The other alternative is that one angle, say A, exceeds 90°; in this case the last inequality's right–hand side turns into $2æ\,\cot(B)\,\cot(C)$, which gets huge (very ill–conditioned) as B or C or both get tiny. Thus we conclude that …

> When tiny end–figure perturbations change side–lengths $\{a, b, c\}$ to $\{(1\pm æ)a, (1\pm æ)b, (1\pm æ)c\}$ they change the triangle's area Δ to very nearly …
> … $(1 \pm 2æ)\,\Delta$ if the triangle's angles are all acute (none exceeds 90°) ,
> … $(1 \pm 2æ\,\cot(B)\,\cot(C))\,\Delta$ if one of the triangle's angles $A > 90^\circ$.
> Only in the latter case can Δ be an ill-conditioned function of the data $\{a, b, c\}$.

This conclusion contrasts with line 4 in Table 1. Perturbations in the tenth sig. dec. of that data cannot possibly corrupt the fourth sig. dec. of the area Δ the way roundoff has corrupted Δ' obtained from Heron's formula. That isosceles triangle is not ill–conditioned at all; the only thing wrong with it is that Heron's formula Δ' doesn't like its needle–like shape.

The picture on the next page will help to elucidate the situation. Let us treat a triangle's side–lengths $\{a, b, c\}$ as *Barycentric Coordinates* of a point in the (x, y)–plane by setting

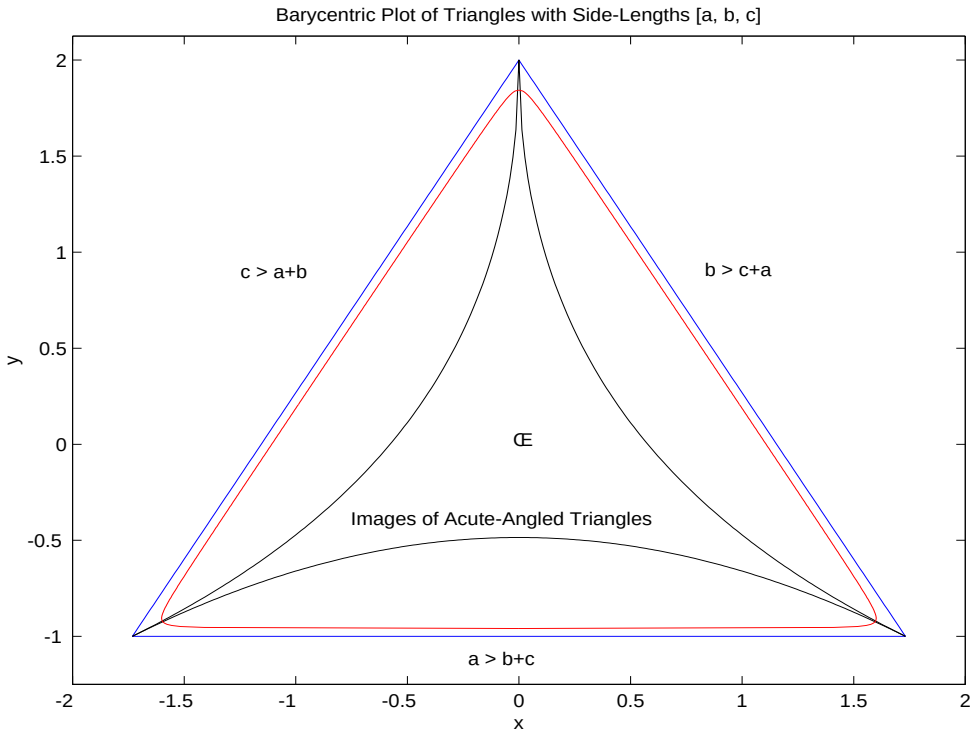
$$x = (b-c)\sqrt{12}/(a+b+c) \quad \text{and} \quad y = 2(b+c-2a)/(a+b+c) \;.$$

Doing so maps all triples $\{a, b, c\}$ of triangles' side-lengths to an equilateral triangle $Æ$ in the (x, y)–plane with vertices at $(0, 2)$ and $(\pm\sqrt{3}, -1)$. Every point (x, y) in $Æ$ is the image of a family of *Similar* triangles with edge-lengths proportional to $\{a, b, c\}$. Equilateral triangles $\{a, a, a\}$ map to the center $(0, 0)$ of $Æ$. Each isosceles triangle (with two equal edges) maps to three points on the medians of $Æ$ corresponding to $\{a, b, b\}$, $\{b, a, b\}$ and $\{b, b, a\}$. Each scalene triangle (with no two edges equal) maps to six points in $Æ$ corresponding to the permutations of $\{a, b, c\}$. The edges of $Æ$ are images of collapsed triangles:

$$\{b+c, b, c\} \to\; y = -1 \;; \quad \{a, b, a+b\} \to\; y = 2 + x\sqrt{3} \;; \quad \{a, a+c, b\} \to\; y = 2 - x\sqrt{3} \;.$$

The images of triangles whose angles are all acute sweep out the inside of a curvilinear triangle $Œ$ inside $Æ$; the cusped vertices of $Œ$ are vertices of $Æ$ too, but $Œ$ is bounded by three hyperbolic arcs that are the images of all right-angled triangles:

The Triangle Æ of Similar Triangles



The points (x, y) in Œ satisfy

$$8 - \sqrt{(3x^2 + 72)} \le y \le 8(1 - |x|\sqrt{3})/(4 + |x|\sqrt{3} + \sqrt{(3x^2 + 24|x|\sqrt{3})})$$

corresponding respectively to inequalities $b^2+c^2 \ge a^2 \ge |b^2-c^2|$ satisfied by the edge-lengths {a, b, c} of all acute-angled triangles. All their areas Δ are very well-conditioned functions of their edge-lengths. Outside Œ a triangle's area becomes ever worse-conditioned as its image approaches an edge of Æ, on which areas vanish and are infinitely ill-conditioned. Regardless of their ill-condition our formula Δ (the one that first sorts the data) computes all their areas accurately despite roundoff. However our formula Δ is not so well known as Heron's.

In the presence of roundoff Heron's formula Δ' is ill suited to computing, say, ratios of areas or reflectivities of needle–like triangles, all of whose images lie near Æ's edges. Along these edges runs a narrow ribbon, portrayed with exaggerated width in the picture above, containing the images of all triangles for which the relative uncertainty due to roundoff in Δ' is intolerably big. A tiny fraction of them, those nearly isosceles triangles with images very near the vertices of Œ, having well-conditioned areas, suffice to condemn Heron's formula Δ' as numerically unstable for delivering far less accuracy than the data deserve in certain instances. Were these instances' existence unknown, trying to find them by sampling random data distributed roughly uniformly over Æ would be a futile quest. This is why most programmers hardly ever discover whatever numerical instability may afflict their programs, much less debug it.

What is the incidence of intolerable inaccuracy when Heron's formula Δ' is used? If roundoff is roughly as random as it usually seems, that incidence is roughly proportional to the incidence of data {a, b, c} whose images fall into the narrow ribbon portrayed above. Its width is easily proved proportional to (roundoff threshold æ)/(biggest tolerable relative uncertainty in Δ') .

Though this does not reveal the incidence of intolerable inaccuracy it does explain how carrying extra precision beyond the data's during arithmetic operations reduces that incidence: Every ten additional bits or three additional decimal digits of precision reduces the roundoff threshold æ , and with it the ribbon's width and area, and with that the proportion of data whose images lie in the ribbon, and therefore the incidence of intolerable inaccuracy by a factor of roughly 1/1000 until a law of diminishing returns sets in.

(Returns diminish after the ribbon becomes so narrow that hardly any images of data are left strictly inside it, as must happen ultimately if the data {a, b, c} reside among a finite set of floating-point numbers whose precision was fixed before the arithmetic's precision was increased. Returns from Heron's formula Δ' begin to diminish as the arithmetic's precision surpasses twice the data's precision, beyond which only isosceles triangles with one extremely tiny side benefit from increased arithmetic precision.)

An error-analysis like the one just performed upon Δ' can also be performed upon the formulas for angle C as a function of side-lengths {a, b, c} . Though the picture and many details must change, what happens to C resembles what happened to Δ above:

> When tiny end–figure perturbations change side–lengths {a, b, c} to
> {(1±æ)a, (1±æ)b, (1±æ)c} they change the triangle's angle C to very nearly …
> … C ± (360˚/π) æ sin(C)/(sin(A) sin(B)) if angles A and B are both acute,
> … C ± (360˚/π) æ cot(B) if A > 90˚ .

Like Δ , a tiny value of C is determined well by the data for a needle–like nearly isosceles triangle though both classical formulas C' and C" dislike it. Roundoff impairs these formulas' accuracy when used to compute parallax at a far distant vertex, so experienced astronomers, navigators and surveyors avoid them. Accurate formula C , the one that first sorts the data, is accurate for all triangles but unlikely to be selected by a programmer who sees only the classical formulas C' and C" in his textbook. Each of C' and C" is intolerably inaccurate for data {a, b, c} whose images lie in a narrow ribbon along Æ's boundary. Although the ribbon's width varies in a way that depends upon which formula was selected, its width and area and therefore the incidence of intolerable inaccuracy shrink at least as fast as the roundoff threshold æ does when arithmetic precision is increased.

As before, roundoff in the classical formulas can inflict inconsistencies upon object–oriented software. For instance, classical formulas can violate the equation a b sin(C)/Δ = 2 severely. It and the accurate formulas are consistent in all but perhaps the last digit stored except when C is too close to 180˚, in which case an error in C's last digit can spoil sin(C) utterly as almost occurs in line 7 of Table 1.

What if the software you use was programmed by someone ignorant of the accurate formulas? Then you have to hope that it was compiled to carry sufficiently more arithmetic precision than your data occupy in memory. Every extra three sig. dec. or ten sig. bits carried reduces by typically 99.9% the population of triangles that classical formulas dislike enough to hurt you.

§15. Proper Precision Management:

" The purpose of computing is insight, not numbers." (R.W. Hamming)
" The purpose of computing numbers in not yet in sight." (G.E. Forsythe)

Numerical computation is a kind of simulation performed to explain or predict. Its accuracy is not an end in itself but need only be adequate to support reliable explanations and predictions.

To that end, the arithmetic's precision should be determined from outside in rather than from inside out, by the uses to which results will be put rather than by operands' precisions. When, as happens often, the necessary precision is so difficult to ascertain in advance that it is not known, the widest precision available that does not sacrifice too much speed is the right choice. It is the easiest way we know to achieve the accuracy we seek, sought not for its own sake as if, like Virtue, it were its own reward, but sought because we know no better way to secure as many mathematical relationships as we can afford in the hope that they include the ones that matter to an impending computation to which we do not yet know which ones matter. When, rarely, the chosen precision turns out to be inadequate, we have to find something else to do.

If that does not sound like Science, compare it with the assignment of precision according to traditional naive rules. Some rules followed by applications programmers are rules of thumb, venerable but no less fallible than ancient trigonometric formulas. Other rules are enforced by computer languages designed by other programmers who have reasons for their rules but not reasons informed by modern error–analysis. The linguistic tradition that assigns to each arithmetic operation the same precision as is occupied in memory by its operands and result has never been and never will be justified numerically. When, rarely, software written according to these naive rules malfunctions because of rounding errors, users and perhaps the programmer of the software (but why not the language designer?) have to find something else to do.

How different are the rates of malfunction due to roundoff under the two regimes of precision assignment just described? Alas, the hits, runs and errors recorded so sedulously for baseball are not recorded also for numerical computation. Lacking good records, we have only crude guesses based upon unreliable anecdotal accounts of the few malfunctions that have been caught.

Back when computation ran at leisurely kiloflops in one big room instead of gigaflops on many desktops, I used to look over my colleagues' shoulders out of curiosity to see what they were computing. What I observed made me as welcome as any other bringer of bad tidings:

- About a third of the results I found interesting were far more in error than had been thought.
- When serious errors were discovered, their causes were misdiagnosed more often than not.
- About a quarter of those errors arose from failures to appreciate a compiler's " features."
- Most numerical results were discarded unused, often before anyone had looked at them.

The last observation helps explain why calamitous miscomputations were not celebrated daily. The other observations accord with and motivate the examples and analyses presented in this article, and suggest that the malfunction rate is not negligible though nobody knows what it is.

Whatever the rate of malfunction due to roundoff, it would be orders of magnitude smaller if compilers made better use of the precision built into the overwhelming majority of computers now installed on desktops, precision their owners have paid for but are not getting.

§16. Different Floating–Point Semantics:

For the purposes of this article just three floating–point hardware designs will be distinguished according to how fast operands and results of different widths circulate among their fast on–chip cache memories, their floating–point registers, and their pipelines where the arithmetic gets done. All computers move numbers to and from main memory (DRAM) and disk faster if the numbers occupy a narrower format, so we can assume for all computers that large volumes of data and results will be stored in the narrowest format of Table 4 adequate to hold them. Each of the three floating–point architectures has its own way to handle scratch variables in registers and intermediate variables already resident in the cache.

Today's less common architecture gets `float` results from `float` operands significantly faster than it gets `double` from `double`s, and cannot mix a `double` with a `float` without first promoting the `float` to `double`. This architecture, which we call " Orthogonal," appears to match a linguistic tradition that assigns to each arithmetic operation the same precision as its operands and result occupy in memory. We call that tradition's arithmetic semantics " Strict." We shall see that it is not the best numerical semantics to use with this architecture.

A now more common architecture holds all operands in `double` registers regardless of whether they came from memory as `float`s, and obtains `double` results about as fast as `float` if not faster. The IBM RS/6000, Power-PC and Apple Power-Mac, and DEC Alpha have this architecture. It matches the semantics of old–fashioned Kernighan-Ritchie ***C***, which treated all constants and subexpressions as `double`s and rounded to `float` only values then assigned to variables the programmer had declared `float`. We call the architecture and the semantics " Old***C***." They suit each other well. Languages and compilers like *Java*, that enforce Strict semantics upon an Old***C*** architecture, are squandering its precision for no good reason.

Today's most common architecture, found in about 90% of computers installed on desktops, holds all operands in `long double` registers regardless of the formats in which they came from memory, and obtains `long double` results about as fast as `double` or `float` if not faster. This is the architecture of the Motorola 680x0 in old Apple Macintoshes and older Sun IIIs, the Motorola 88110 (few exist), the Intel 80960 KB (in embedded systems like PostScript printers and in some military computers), and the ubiquitous Intel x86/Pentium cloned by AMD and Cyrix. We call this architecture " Extended." It includes a control register in which two bits can be set to abbreviate the arithmetic so that it will mimic the roundoff behavior (but perhaps not the over/underflow thresholds) of the previous two architectures. Therefore the Extended architecture can match the Strict and Old***C*** semantics practically perfectly in case software recompiled from the other machines has to be run the way they run it. However, the semantics natural for the Extended architecture is to evaluate all constants and subexpressions in `long double` and round to narrower formats only values then assigned to variables declared narrower by the programmer; we call this semantics " Extended."

On most Extended machines the library of `long double` elementary transcendental functions (log, exp, cos, arctan, …) is supported extensively by hardware. Therefore the programmer is best advised to declare almost all local floating–point variables `long double` and enjoy near obliviousness to obscure rounding errors. Most of these (all that occur in subexpressions) will occur more than three decimal digits to the right of the rightmost digit stored with most data and

results, so roundoff will cause trouble far less frequently than on non-Extended machines. But few compilers afford programmers this pleasure. Among them are the compilers endorsed by Apple for old 680x0-based Macintoshes; a good example is the Fortner (formerly LSI) Fortran compiler. Another example is Borland's C compiler for Intel–based PCs and their clones. I know of no other compilers for PCs that support Extended semantics fully if at all.

There are historical reasons for the dearth of linguistic support for Extended semantics, and some of those reasons made sense once if not now. For two decades up to about 1985, the big number–crunchers had Orthogonal architectures. Among them were the IBM /370 and 3090, CDC 6600/7600, CRAYs, and ultimately the DEC VAX. IBM ended up with hardware for three floating–point formats, DEC for four, of which the widest (also called Extended) was 16 bytes wide and too slow for most applications. The illusion of Compatibility served to excuse Strict semantics, and numerical experts tolerated it because we had bigger fish to fry. Besides, we were proud of our ability to program the behemoths successfully by assiduous attention to details that sane people find too annoying, and we were paid well to do it.

Now the evolution of computer languages has reached a state in which language designers feel uncomfortable with any semantics other than Strict, and doubly uncomfortable at the thought of entertaining two or three floating–point semantics. This is a pity because Strict semantics was at best a convenient expedient to simplify compilers, never as good numerically as the other semantics described here, and rather worse for numerically inexpert programmers.

(Let's digress to consider a semantics that serves programmers of Orthogonal architectures better than Strict semantics. We call it "Scan for Widest." It affects the overloading of infix operators like +, -, *, / and := amidst an expression with operands of different precisions; the idea is to perform all arithmetic in that expression to the widest of those precisions. Unlike Strict semantics, which overloads an infix operator according to the syntactic types exclusively of its operands, Scan for Widest has to take account also of the expected type of the result when that is decidable. The bottom line is that Scan for Widest does the same as Strict when precisions are not mixed, and avoids the need for Strict's explicit widenings (which are often omitted by mistake) when precisions are mixed for the usual reasons. These are that more than the data's precision will be needed by intermediate variables like S := (a+b+c)/2 in Heron's formula, T := a/tan(A) in $B_T(\dots)$, U := b·cos(A) in $c_b(\dots)$, etc. to get consistent results; and we expect to compute $\sqrt{(T\cdot T - (b-a)\cdot(b+a))}$ accurately without first explicitly coercing b to the same higher precision as T. Similar scanning is needed to facilitate correct mixtures of a multiplicity of precisions, and mixtures of Interval Arithmetic with non-Interval data, and to permit the language of coordinate–free Linear Algebra to work upon many geometrical objects each specified in its own coordinate system. Of course, Scan for Widest eases the lives of programmers at the cost of complicating life for language designers and implementors who have to cope with a host of details that are easy to botch. But that is a story for another day.)

The Strict semantics specified by *Java* is at odds with its promotion as the one language for Everyman to program everything everywhere. Old***C*** semantics would be safer and Extended semantics more so for the overwhelming majority of computers in service, and probably also for almost all programmers. *Java*'s designers should not be surprised if they are suspected of denying advantages to the many in order to secure advantages for a few.

§17. Conclusion:

The computing community has always regarded floating–point computation as a black art. It's not really. It's very mathematical, too mathematical to suit most programmers attracted by the opportunities to sell software into a mass market. Maybe they should study the subject a little, perhaps read an article or two like this, before they write floating–point software. Such advice is Counsel of Perfection, recommended but not obligatory for programmers who have spent their time refining different skills. Therefore programmers remain mostly unaware of numerical hazards lurking in classical formulas copied from texts and handbooks, and unaware of better methods that attenuate or eliminate those hazards. Yet we all depend unwittingly upon their programs, which expose us all to those hazards.

Computational facilities, hard– and software, intended for the masses can do better than blindly amplify intelligence or the lack of it. Just as seat–belts, gas bags, ABS brakes and impact absorption are designed into automobiles to enhance their safety without excessive detriment to performance or price, computational capabilities ought to be designed with the attenuation of computational hazards in mind, always subject to considerations of performance, price and time to market. We know a lot about the programs programmers are inclined to write, and that knowledge has influenced the design of computer hardware and optimizing compilers to run those programs faster. Running them more reliably too is a worthwhile goal for hardware and software engineers, especially if the added cost of enhanced reliability is ultimately negligible.

By far the easiest way to attenuate and sometimes extinguish the risks from roundoff in otherwise correct computation is to carry more precision than data or results " deserve." Hardware to do so is in place. Now the computer language community must do its part.

Think not of Duty nor Indulgence; think about Self–Defense.

§18. Acknowledgments and Bibliography:

This note is derived from material I have long taught to introductory classes in Numerical Analysis. The material is very old; I cannot recall from whom (perhaps astronomers) some of it came in the 1950s, but I do recall that many programmers knew such things in the early 1960s. Exact cancellation of floating–point numbers was exploited in my letter "Further Remarks on Reducing Truncation Errors" in *Comm. Assoc. Comput. Mach.* **8** (1965) p. 40, and in R.H. Møller's "Quasi Double–Precision in Floating–Point Addition" in *BIT* **5** (1965) pp. 37-50. The accurate formula for Δ(a,b,c) appeared in my "Mathematics Written in Sand" in the Statistical Computing Section of the 1983 *Proceedings of the American Statistical Association*; that work had been supported by a grant from the U.S. Office of Naval Research, contract # N 00014-76-C-0013. A formula similar to Δ appears in problem 23 on pp. 152–3 of *Floating–Point Computation* by P.H. Sterbenz (1974) Prentice-Hall, New Jersey. An earlier version of the accurate formula for C(a,b,c) appeared in pp. 194–200 of the *Hewlett–Packard HP-15C Advanced Functions Handbook* (1982). The selected formulas for B and c have appeared nowhere else that I know.

A little extra precision can enhance accuracy in industrial–strength computations dramatically, as it does in " Roundoff Degrades an Idealized Cantilever " by Melody Y. Ivory and myself; see http://http.cs.berkeley.edu/~wkahan/Cantilever.ps . The rate at which extra precision reduces the incidence of intolerably inaccurate results is a subtlety discussed in papers like J.W. Demmel's "The Probability that a Numerical Analysis Problem is Difficult" *Math. of Computation* **50** (1988) pp. 449-481. N.J. Higham's *Accuracy and Stability of Numerical Algorithms* (1996) Soc. Indust. & Appl. Math., Philadelphia, is a 700 page encyclopedia.

The uneasy relationship between compilers and floating–point is the subject of C. Farnum's "Compiler Support for Floating–Point Computation" *Software Practices and Experience* **18** #7 (1988) pp. 701-9, and of D. Goldberg's "What every computer scientist should know about floating–point arithmetic" *ACM Computing Surveys* **23** #1 (1991) pp. 5-48., and some of their citations. An updated and emended version of the latter is bundled in Sun's documentation and can be downloaded as a postscript file from http://www.validgh.com. For old anecdotes about numerical miscomputation see my "A Survey of Error Analysis" pp. 1214-1239 in *Information Processing 71* (1972) North Holland, Amsterdam. For IEEE Standard 754 for Binary Floating–Point Arithmetic see http://http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps and its citations. For a critique of *Java*'s floating-point see …/~wkahan/JAVAhurt.pdf .

§19. Footnote about Examples computed long ago:

Why were my examples run on old calculators and not a current computer? Hexadecimal digits are so much harder than decimal digits for humans to appraise that all data and results had to be printed in decimal; and then Decimal ‹—› Binary conversions had to be avoided because they could change the printed data slightly before it went into the computer's memory. Conversions varied among computers and compilers, as did trigonometric functions; such variations were distractions best avoided no matter how small. Angles computed in degrees instead of radians avoided variations on π . HP-15Cs and 11Cs were abundant, and they performed arithmetic identically and well, so the results presented here were easy for others to reproduce back then.