THE TABLE-MAKERS' DILEMMA

W. Kahan
Computer Science Department
University of California at Berkeley

August 1971

Abstract

Accuracy in numerical computation is not an end in itself, but a means to other ends. Only when those ends are not clearly in view is accuracy an attractive expedient, if it is achievable at all.

CR Categories: 5.11 and 5.12

Key phrases: Error Analysis, Accuracy Limitations

This work was supported in part by the grant ONR Contract No. N00014-69-A0200-1017 from the Office of Naval Research.

The Table-Makers' Dilemma

"It is confidently believed that the cases where the error exceeds ±0.51 units of the last decimal could be counted on the fingers of one hand; those that are known to exist form an uncomfortable trap for any would-be plagiarist."

Chambers's Shorter Six-Figure Mathematical Tables (1959)
L.J. Comrie

The ideal computation would produce correctly rounded numerical results in error by at most ±0.5 ulps (Units of the Last decimal Place cited). But achieving this ideal can be more than expensive; it may be impossible. Consider, for example, the computation to 13 significant decimals of

$$1/3^3 + 1/5^3 + 1/7^3 + \cdots + 1/(2n+1)^3 + \cdots = (7/8)\zeta(3) - 1$$

where $\zeta(s) = \Sigma_1^\infty n^{-s}$ is Riemann's Zeta-function. Working to 15 significant decimals yields a value

$$0.0\,51799\ 79026\ 464\ 50\ldots ,$$

uncertain by 1 in the last decimal cited. Should the 13th significant figure be rounded up or down? Rather than guess we recompute working to 18 significant decimals, and get

$$0.0\,51799\ 79026\ 464\ 49999\ldots ,$$

uncertain by 2 in the last decimal cited. Should we try again, carrying more figures, or should we conjecture that the true value is precisely

$$0.0\,51799\ 79026\ 464\ 5\ ?$$

Can this conjecture be decided more easily than another more famous conjecture[*] about the Zeta-function?

This dilemma, familiar to the table-makers of old, now afflicts their descendants who write numerical subroutines intended to serve other users of the electronic computer.

The dilemma persists when "correctly rounded" is replaced by the ostensibly less demanding phrase "correctly chopped". For example, the value[†] of

$$\exp(\pi\sqrt{163}) = 262\ 53741\ 26407\ 68743.99999\ 99999\ 99250\ 072597\ldots$$

cannot be correctly chopped to 18 significant decimals until 31 are known; anybody who computed just the first 20 digits, then the first 25, then the first 30, might think he had discovered a new integral value $\exp(\pi\sqrt{n})$ to join with $\exp(\pi\sqrt{0}) = 1$ and $\exp(\pi\sqrt{-1}) = -1$.

The dilemma is latent in any demand that the computed approximation a to a number x be obtainable from some rule, $a = r(x)$, which maps the continuum onto a discrete set, as do the rules for correct chopping and correct rounding (see Knuth (1969) p. 197 for one definition of the function $r(x) = \text{round}(x,p) \equiv$ "x rounded to p digits"). Such a mapping r effects a partition of the continuum into non-overlapping intervals with whose boundary points x must be compared in order to determine $r(x)$, and that comparison may defy effective computation (cf. also Aberth (1971) and references cited there). However, while it would be impractical to insist upon none but disjoint partitions r, it would be unwise to relinquish them entirely.

[*] *A propos* of Riemann's Hypothesis see Rosser, Yohe and Schoenfeld (1969). As for $(7/8)\zeta(3)-1$, the value given by H.T. Davis in *Tables of the higher mathematical functions*, vol. II (1935) is $0.0\ 51799\ 79026\ 464\ 499972\ldots$

[†] This value is given by D.H. Lehmer in M.T.A.C. 1 (1943) p. 31.

Why should we not, when specifying how accurate some computer subroutine ought to be, merely relax the error tolerance "±0.5 ulps" to, say, "±0.51 ulps"? Because doing so would, for example, permit the 6-decimal approximations $\cos(0.358) \doteq 0.936600$ and $\cos(-0.358) \doteq 0.936599$ to be supplied in place of $\cos(\pm 0.358) = 0.936599\,499881\ldots$, thereby abandoning the sign-symmetry of the cosine function. For another example, imagine a function $\phi(x)$ known by everyone who needs it to be strictly monotonic, and consider this 6-decimal tabulation

x	$\phi(x)$ to 6 dec.	True $\phi(x)$
0.999998	0.700002	0.700002 35043...
0.999999	0.700003	0.700002 49002...
1.00000	0.700002	0.700002 50997...
1.00001	0.700022	0.700022 45007...

which forsakes ϕ's monotonicity. Deviations like these from familiar relationships such as sign-symmetry, monotonicity, periodicity (e.g. of $f(x) = \sin \pi x$), integer values (e.g. $\log 1 = 0$), inequalities (e.g. $\cos x \le 1$, and $\cosh x \ge \exp x \ge \sinh x \ge x$ when $x \ge 0$) and certain identities (e.g. $\sqrt{x^2} = |x|$, and $\log_{10}(10^n) = n$ for integers $n \ge 0$) can create confusion, especially during program de-bugging, out of all proportion to the small end-figure errors that cause the deviations. A departure, from those relationships which persist after elementary functions are rounded correctly, cannot be excused by observing that some other relationships (e.g. $(\sqrt{x})^2 = x$, $\cos^2 x + \sin^2 x = 1$, $\tan x = \sin x/\cos x$) can only be approximated to within one or two ulps when each elementary function is correctly rounded; such an excuse could be promoted to *carte blanche* by the further observation that other simpler relationships like $\log(\exp x) = x$ for tiny x, or $\exp(\log x) = x$ for huge x, must be approximated poorly.

Fortunately, most mathematical relationships which survive the 0.5 ulp error in correct rounding will also survive errors larger than 0.5 ulp but smaller than 1 ulp, and those few relationships which might not survive the larger errors can be saved with a little extra care in critical parts of a program. This extra care is worthwhile if it costs less time than will the confusion and recriminations it saves. While programming elementary functions for the IBM 7094-II at the University of Toronto between 1962 and 1965 (see my 1968 notes) I found that extra care cost less than one month's extra work per program, and slowed the program by less than 10% if at all. Moreover, much of the extra work was devoted to proving, mathematically first and then by running tests on data, that the program performed as well as I claimed. The kinds of claims I made are illustrated by the programs

COS , COSPI , SIN , SINPI

which were written to compute trigonometric functions:

$$\begin{aligned} \mathrm{COSPI}(Y) &= (1+\varepsilon)\cos(\pi Y) \quad , \\ \mathrm{SINPI}(Y) &= (1+\varepsilon)\sin(\pi Y) \quad , \\ \mathrm{COS}(X) &= \mathrm{COSPI}((1+\delta)X/\pi) \quad , \\ \mathrm{SIN}(X) &= \mathrm{SINPI}((1+\delta)X/\pi) \quad , \end{aligned}$$

where ε represents an error smaller than 1 ulp of the computed value, upon which alone ε depends, and δ represents an error smaller than 4 ulps in double-precision of the argument X, upon which alone δ depends.

Specifically, $|\varepsilon| < 1.2 \times 10^{-8}$ and $|\delta| < 10^{-15}$ on the 7094. The argument-perturbation δ is an unavoidable consequence of the reduction of X modulo 2π; the value of π is represented only to double precision. δ can be ignored when $|X| < 2\pi$ or when every zero of $\cos x$ or $\sin x$ lies farther from X than 1 ulp in single precision. And when X is very large δ can be ignored unless X is represented precisely by its stored value because, when $|X| > 10^8$, changing X by one ulp may change sin X and cos X utterly.

The computed values satisfy familiar inequalities

$$\mathrm{ABS(COS(X))} \le 1.0, \quad \mathrm{ABS(SIN(X))} \le 1.0, \quad \mathrm{ABS(SIN(X))} \le \mathrm{ABS(X)}$$

for all X. Sign-symmetry is preserved. SINPI and COSPI are precisely periodic and appropriately monotonic; e.g. if X and V differ by one ulp then COSPI(X) - COSP(V) has the correct sign or else vanishes. COS(X) and SIN(X) are similarly monotonic for $|X| < 2\pi$. The identity $\sin^2 x + \cos^2 x = 1$ is approximated to within two ulps by an inequality

$$\mathrm{ABS(1.D0 - DBLE(COS(X))**2 - DBLE(SIN(X))**2)} < 2 \times 10^{-8}$$

valid for all X. Consequently no diagnostic is issued when $|X|$ is huge, contrary to a practice common with programs that violate the last inequality.

As of May 1966 these single-precision SIN and COS programs were the most accurate available for the IBM 7094-II and 6 to 10% faster than any other programs claiming accuracy within 2 ulps.

Perfection vs. Practicality

"The maxim
'Nothing avails but Perfection'
may be spelt shorter,
'Paralysis'."

"*The Hinge of Fate*", p. 793,
Winston S. Churchill

Born in the shelter of the groves of Academe, my programs do not exemplify current commercial practice. That practice tends to be less meticulous, according to Cody (1970), with one outstanding exception; the elementary function subroutines distributed with the FORTRAN library for IBM's 7090-7094 and system/360 machines since 1965 have been nearly unimprovable. These programs, documented in

IBM 7090/7094 *IBSYS v.13 IBJOB Processor Manual*, appendix H; Form C28-6389

IBM System/360 *FORTRAN IV Library Subprograms*; Form C28-6596,

are the work mostly of Hirondo Kuki at the University of Chicago. The excellence of his work is the more noteworthy for having been accomplished despite the nastiness of System/360's hexadecimal arithmetic. The software industry owes him a debt that could best be repaid by imitating his style, which is characterized by Quintilian's maxim "Ars est celare artem".

One example, extended precision division, will illustrate Kuki's style. To compute a correctly rounded or chopped quotient requires either an integer-arithmetic division-with-remainder or an explicit calculation of approximately twice as many digits as are going to be kept; for example

$$0.99998/0.99999 = 0.99998\ 99998\ 99998.....$$

can be chopped correctly to five digits only after the tenth is proved not to be a 9. Is doing so worthwhile? This question confronted Kuki and Ascoly (1971, pp. 45-6) when they began to construct a program which provides correctly chopped quotients of extended precision (28 significant hexadecimal digit) floating point numbers on IBM System/360 machines. Their program first approximates the quotient $q = a/b$ by $\hat{q}$ which, though in error by less than one ulp, occasionally exceeds the correctly chopped quotient $\breve{q}$ by one ulp. Next $\hat{q}$'s remainder $r = a - b\hat{q}$ is computed precisely to determine whether $\hat{q}$ should be diminished by one ulp to get $\breve{q}$. That r takes nearly as long to compute as $\hat{q}$ is a price Kuki and Ascoly decided to pay for a correctly chopped quotient $\breve{q}$.

A different decision was made by Anderson *et al.* (1967) when they optimized the speed of hardware division in the IBM/360 model 91 by wiring in a quadratically convergent iteration which is terminated as soon as the computed (6- or 14-hexadecimal digit) quotient $\hat{q}$ is correct to better than one ulp. In some cases $\hat{q}$ computed on the model 91 will exceed by one ulp the corresponding quotient $\breve{q}$ computed on all other IBM/360 models, which correctly chop their quotients. For example, the assignment

Q = (1.0 - 0.5**23)/(1.0 - 0.5**24)

stores in Q a value $\hat{q}$ = 1.0 - 0.5**24 on the model 91 and a value $\breve{q}$ = 1.0 - 0.5**23 on the other models, which is analogous to approximating $q = 0.99998/0.99999$ by $\hat{q} = 0.99999$ instead of the correctly chopped but ostensibly less accurate $\breve{q} = 0.99998$. In general, when $\hat{q} \neq \breve{q}$ then $|\hat{q} - q| < |\breve{q} - q|$.

Who made the better decision, Kuki and Ascoly or Anderson *et al.*? At first sight, the latter decision seems better because it is more accurate, but this is not so. The correctly chopped quotient $\breve{q}$, by confining q to an interval narrower than one ulp, gives more information about q than a value $\hat{q}$ which may be almost one ulp smaller than q or else a fraction of an ulp bigger. Anyway this difference in accuracy is practically inconsequential; if all IBM/360 models divided the same way as the model 91 few people would complain. Unfortunately, a programmer who exploits, perhaps unconsciously, the characteristics of chopped quotients may find that his program, which he has "proved" to function correctly on all other IBM/360 models, occasionally fails mysteriously on the model 91. Is the failure due to the different division algorithm? Probably not, but how long will he take to find out?

The program by Kuki and Ascoly provides no surprises for anyone accustomed to System/360 arithmetic, and it offers anyone to whom faster division is important an opportunity to nearly halve the division time by accepting $\hat{q}$ instead of $\breve{q}$ provided he is confident that doing so will not have other less pleasant consequences.

It is important to realize that Accuracy, unlike Virtue, is not its own reward, but a means to another end. That end is achieved, in numerical computation, through the conservation of mathematical relationships that vary widely from one application to another. When we do not know which mathematical relationships must be preserved and which can be abandoned in any particular application, we try to preserve as many relationships as possible; this is what accuracy is good for. When we do know which relationships are important we can occasionally accept less than perfect accuracy, thereby avoiding the table-makers' dilemma, provided we adhere to one fundamental principle (Knuth (1969) p.204):

"Numerical subroutines should deliver results which satisfy simple, useful mathematical laws whenever possible."

References

Aberth, Oliver (1971) "The Concept of Effective Method Applied to Computational Problems of Linear Algebra", *J. Computer & System Sci.* 5, 17-25.

Anderson, S.F.; Goldschmidt, R.E.; Earle, J.G. and Powers, D.M. (1967) "Floating-Point Execution Unit", *IBM J. Res. and Dev.* 11, 34-53.

Cody, W.J. Jr. (1970) "Software for the Elementary Functions", presented at the Mathematical Software Symposium at Purdue University in April 1970 (proceedings to appear).

Kahan, W.M. (1968) "Error in Numerical Computation", part of the notes for summer course #6818 "Numerical Analysis", University of Michigan Engineering Summer Conferences, Ann Arbor, Mich.

Knuth, Donald E. (1969) *The Art of Computer Programming*, vol. 2, Semi-Numerical Algorithms. Addison-Wesley, Massachusetts.

Kuki, Hirondo and Ascoly, Joseph (1971) "FORTRAN extended-precision library" *IBM Syst. J.* 10 39-61.

Rosser, J. Barkley; Yohe, J.M and Schoenfeld, Lowell (1969) "Rigorous computation and the zeros of the Riemann Zeta-function" pp.70-76 in *Information Processing 1968*, Proceedings of IFIP Congress 1968, vol. 1, ed. by A.J.H. Morrell; North-Holland, Amsterdam.