A Wordset for Error-Propagation Arithmetic

Ferren MacIntyre

Center for Atmospheric-Chemistry Studies
Graduate School of Oceanography
University of Rhode Island
Narragansett, Rhode Island 02882-1197

Abstract

Experimental and observational errors propagate as data is transformed or mathematically manipulated to produce quantities of interest. The mathematics of this propagation are simple, well known, but tedious to apply. Forth proves to be an ideal language for error analysis. Presented here is a short wordset which reduces the mathematical portion of the process (in contrast to its philosophical aspects) to routine coding, supplying the minimum propagated error of any data reduction nearly automatically. Additional words can be built upon the models supplied. Floating-point capability is assumed.

The Need for Error Analysis

Because no experiment or observation is free of errors, all of the data of science are flawed. The problem is not the presence of errors, but the subtler question of how far one can trust the conclusions based upon such suspect data. As I write, the excitement over whether Eötvös' *et al.* (1922) data on gravitational acceleration support classical theory (as has always been assumed), or reveal a dependence upon composition (Fischbach *et al.*, 1986), rests entirely upon error analysis.

The foundations of error analysis—an art with a long and honorable history—are well set out in Beers (1953).

The basis of the theory of errors is that if one knows the input errors, their propagation can be tracked as they move through mathematical calculations, and a meaningful error estimate derived for the final result. The arithmetic of error propagation is trivial but tedious, and is precisely the sort of thing that computers are better at than people. Accordingly, I present a wordset which will take care of this problem, based upon one written—with much effort—in MAD (Michigan Algorithm Decoder, an ALGOL-58 relative) for the room-sized mainframes of the IBM 704-7090 series (MacIntyre 1964).

Certainly nothing prevents a FORTRAN or BASIC programmer from writing functions to accomplish the same end, but I have never seen it done. I once wrote a set for HP calculators, but the HP library was baffled by the concept. Forth, because it makes no distinction between ordinary arithmetic operators and complicated functions, is a natural language in which to automate error analysis.

The Arithmetic of Error Analysis

We begin by supposing that each experimental value is composed of two parts, a mean μ and a standard deviation σ. Thus the initial assumption is that the observation/experiment has been replicated often enough so that these concepts are meaningful, or that an alternate method of assigning a standard deviation exists (as it frequently does). For a given variable A we write $\mu \pm \sigma = A \pm a$, and note that the format with an absolute error, as here, and the relative form $A(1 \pm a/A)$, are both useful. One must, of course, decide which way his data will be stored, but the wordset in the Appendix provides `A-R` and `R-A` to convert from one to the other.

Our strategy is simply to put A and a on the stack in that order, and manipulate each number appropriately, without programmer attention, just as in complex arithmetic. For example, consider a binary function $f(A \pm a, B \pm b)$. The error term can be estimated directly from its differential

$$df(A,B) = \{\partial f(A,B)/\partial A\}dA + \{\partial f(A,B)/\partial B\}dB$$
$$= af_A + bf_B,$$

where the subscripts denote differentiation with respect to the subscripted variable. But direct addition of errors is appropriate if and only if the errors are dependent (e.g., they arise from an instrumental bias), for it overestimates the propagation of independent errors, which are randomly distributed. To account for the possibility of partial cancelation of random errors, one calculates the standard deviation of $f(A,B)$, and, as Beers shows (1953, p 27 ff), the result is that scalar addition is replaced by orthogonal vector addition, as in Fig. 1.

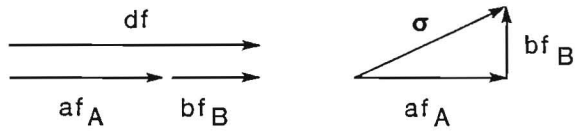

Fig. 1. Dependent errors add as scalars; independent as orthogonal vectors. (After Beers 1953).

There are some cases in which dependent propagation is mandated, of which the most obvious is when squaring an experimental number. In using the Forth words given here, one *must* express this dependency by using a power operation for the squaring, since multiplying the number by itself calls an operator which assumes independent errors.

Dependent errors propagate faster, and have a larger effect upon results, than independent errors. Distinguishing error dependencies is a major aspect of error analysis which the wordset below will do nothing to simplify, except to make it easier for people to take the first steps. As one example, the casual reader might understandably think that errors in R, A, H, and K were independent in the oceanographic example below, whereas because of the manner in which they are determined, they are to some extent dependent.

Those who make careers of adding another significant figure to handbook values of physical constants such as the speed of light spend a great deal of time examining both the experimental procedures and the train of interrelations connecting other "constants" used in the original data reduction, and they consistently arrive at larger error estimates than those reported by the experimenters (Dumond 1959, Cohen and Dumond 1965, Cohen 1980). This is in part simple optimism on the part of researchers, but in part arises from the chain of subtle dependencies in the "constants". One hopeful feature is that when dependent errors are identified and separated, they are then independent from the remaining errors, and all can thereafter be treated uniformly.

Leaving aside these non-arithmetic considerations, and considering the propagation of independent errors, we have from Fig. 1

$$\sigma_f = [(af_A)^2 + (bf_B)^2]^{1/2}$$

A little experimentation shows that whenever two orthogonal vectors are added, the result is again a single vector. The next orthogonal addition does not care whether its predecessor was the result of a measurement or a calculation. Accordingly, we can chain this result to itself as often as we like, which allows us to calculate the error of the most complicated data reduction in simple binary steps. In other words, as long as each elementary operation takes care of its own error propagation, the end result will take care of itself.

Specifically, the elementary operations become:

$A + B = (A+B) \pm (a^2+b^2)^{1/2}$

$A - B = (A-B) \pm (a^2+b^2)^{1/2}$

$A * B = (A*B)[1 \pm \{(a/A)^2+(b/B)^2\}^{1/2}]$

$A / B = (A/B)[1 \pm [(a/A)^2+(b/B)^2]^{1/2}]$

where we have written the algebraically simpler of absolute or relative formats.

For unary functions the possibility of cancellation does not arise, and we have

$df(A \pm a) = (df/dA)dA = af_A$

so that, for instance,

$sin(A \pm a) = sin\ A \pm a*cos\ A$

(which is, incidentally, identical to the expected $sin(A+B) = sinAcosB + cosAsinB$ if $cosB=1$ and $sinB=B$, as is true if B is small enough). This procedure may be used for any function whose operation on a sum is easily obtained, and the reader might like to verify that it works for other functions in the Appendix.

Because I don't see how to go about such operations in integer arithmetic, or why one would want to (even in Forth), I follow the notation introduced in MacIntyre and Dowling (1986) for floating-point and complex arithmetic. The complex-arithmetic data-types and the associated fetch-store-locate, stack-manipulation, and keyboard-input words used therein are perfectly suited for error-propagation calculations. Only the operators are different.

The Appendix gives high-level Forth words for a number of error-propagation operations. In time-consuming applications, of course, one will want these written in appropriate assembler code. `>R` doesn't work for floating-point, but the unfamiliar words `F>T` and `F<T` from MacIntyre and Dowling (1986) allow one to move an IEEE-754 80-bit temporary real from the floating-point stack to the parameter stack and back, and these are used in the essential operation `ESWAP`.

Words like `EROT` and `ERT2` (`= EROT EROT`) can also be implemented in assembler, but working as I do on the 8087, whose stack is only 4 double-words deep, I have not found these particularly useful. Nor, for that matter, can the 8087 stack always be trusted to hold even four words: a number of functions (trigonometric, logarithmic, etc.) are likely to use some stack locations for internal scratch work, as does `HYPOT` in the Appendix, which is called by all of the elementary arithmetic operators.

Prudence suggests not trusting the 8087 stack to hold intermediate results. In time-critical applications where assembler speed is needed and one wants above all to avoid 8-byte memory operations, the stack will need close attention.

The word `.E` borrows the `G.` format from MacIntyre and Dowling (1986) to print the mean and standard deviation in optional floating-point or exponential format (machine choice, based upon available space), separated by a ± sign.

Examples

A couple of examples of what error-propagation arithmetic is all about would seem to be in order. The first we draw from the field of nuclear analytical chemistry, in which an unknown sample is irradiated in the neutron flux inside a nuclear reactor, so that most of its atomic constituents become briefly, and characteristically, radioactive. Gamma rays of the energy appropriate to a given consituent are then counted for a known period, and with certain additional information, the number of original atoms of that constituent can be estimated. However, each energy has its own background rate (from cosmic rays, brehmstrahlung from higher energy gammas, etc.). The background count

must be subtracted from all other counts. To convert count-rate to concentration, standards made up in a similar matrix (to keep background, shielding, and self-absorption similar) are irradiated simultaneously, and the corrected count rates divided. A number of other corrections may also be needed, but we end up with the following as the simplest equation for a concentration:

$$C \pm c = [(A \pm a) - (B \pm b)] / [(S \pm s) - (B \pm b)]$$

where A is the raw count, B the background count, and S the standard. (Note that the dimensions of the numerator are unity (counts), while those of the denominator are counts/mass, so that the answer is not a simple ratio, as might appear, but a mass.)

When the counting time is short compared to the half-life (which is the usual case), the absolute error of radioactive-decay counts is equal to the square root of the number of counts (Friedlander *et al.* 1964, p. 175), so that in this case an analysis of the counting error (but not of handling errors or sample variability) is possible without additional counting.

To minimize the propagation of errors in this calculation it is customary to count the background for long periods. Thus if the background is counted 10 times as long as the sample, we might have $A \pm a = 1678 \pm 40.96$, $B \pm b = (232 \pm 15.23)/10 = 23.2 \pm 1.52$, and $S \pm s = 8923 \pm 94.46$. Then writing

```
: C±c  A B E-  S B E-  E/  ;
```

we find `C±c` $= 0.186 \pm 0.005$. Whether or not such a result is meaningful is outside of the present discussion: what *is* germane is that the question of meaning cannot arise until the magnitude of the possible error is calculated. In any case, one could not call this result different from 0.181 or 0.191, and it may be the same as 0.176 (2σ) or, not impossibly 0.171 (3σ).

We draw a second example from oceanography, where the availability of carbonate ion $C = [CO_3^{=}]$ to organisms (for building shells) is a matter of some concern. (The brackets denote concentration, as in moles/ton-seawater, about equivalent to millimoles/liter.) No direct analytical method can separate carbonate ion from bicarbonate ion $B = [HCO_3^{-}]$. The simplest indirect method is to obtain the concentration of hydrogen ion $H = [H^{+}]$ by measuring the pH $= -\log [H^{+}]$ and the alkalinity $A = B + 2C + R$ by titration. Oceanographers customarily assign R, which measures a contribution from boric acid, a constant value proportional to salinity. B and C are related through a stoichiometric equation for their chemical reaction and an arithmetical equation for their equilibrium constant K,

$$HCO_3^{-} \rightleftarrows H^{+} + CO_3^{=}$$

$$K = [H^{+}][CO_3^{=}]/[HCO_3^{-}] = H^{*}C/B$$

so we have only to solve the pair of simultaneous equations

$$B = H^{*}C/K$$

$$A = B + 2C + R$$

for C. Inserting the first into the second and rearranging gives

$$C = (A-R)/[H/K + 2],$$

or, after inclusion of the error terms,

$$C(1 \pm c/C) = [A(1 \pm a/A) - R(1 \pm r/R)] / [H(1 \pm h/H)/K(1 \pm k/K) + 2]$$

where we have written relative errors because these can be estimated, after a fashion, while the absolute errors are harder to come by. To convert this total concentration to the "activity" $[CO_3^{=}]$, or biologically available concentration, we multiply by the carbonate-ion activity coefficient γ. (One might loosely describe the difference between total and available concentrations by saying that 98%

of the carbonate ions are tied up as ion pairs with magnesium and calcium, forming free-floating one-molecule bits of solid chalk which organisms cannot use.)

If we arbitrarily but not unrealistically suppose that the errors in A, H, and R are 5%, but 10% in γ and K (which are notoriously difficult to measure), we have:

$$C(1 \pm c/C) = [[A(1 \pm 0.05) - R(1 \pm 0.05]/[H(1 \pm 0.05)/K(1 \pm 0.10) + 2]]\gamma(1 \pm 0.10)$$

Taking A = 2.234, R = 0.051, H = 6.61E-9, K = 5.62E-10, and γ = 0.021, we write:

```
: {CO3}  A R E-  H K E/  2, Ø, E+  E/  γ E* ; .
```

Floating-point constants are here indicated by commas (a European decimal point). Note that pure E-mode words require that we assign a zero error term to constants. Evaluation yields $[CO_3^{=}] = (3.33 \pm 0.49)E-3$, a 15% error despite our assumption that our measurements were good to 5%.

Caveats and Conclusions

The approach taken here is not appropriate to curve fitting, which requires more subtle and more extensive analysis. The standard least-squares approach found in an HP calculator will provide estimates for a and b in

$$y = (A \pm a)x + (B \pm b)$$

but for any more complicated equation you are on your own, and difficulties can arise. For example, there does not seem to be a satisfactory approach to error estimates for the exponential equation

$$y = (A \pm a)exp\{(B \pm b)x\} .$$

Linearization by transforming variables is not reliable (i.e.: A and B are wrong) because the least-squares fit assumes that the errors are Gaussian around the transformed line, whereas in practice they are probably Gaussian around the original exponential data. Non-linear (trial-and-error) fitting provides a feel for the sensitivity of A and B, but no firm estimates of a and b.

Despite the practical importance of understanding error propagation, and the years of attention devoted to it by statisticians, one can safely conclude that it is an underutilized tool in the hands of most experimenters. Perhaps the simplicity of using it in Forth will encourage more attention to the problem.

References

[1] Beers, Y. 1953. *Introduction to the theory of error.* (Addison-Wesley, Reading, MA) 65 pp.

[2] Cohen, E.R., 1980. Determining the best numerical values of the fundamental physical constants. *Proc. Internat. Sch. Phys. "Enrico Fermi" 1976*:581-622.

[3] Cohen, E.R., and J.W.M. DuMond, 1965. Knowledge of the fundamental constants of physics and chemistry in '65. *Rev. Mod. Phys. 37*:537-594.

[4] DuMond, J.W.M., 1959. Status of knowledge of the fundamental constants of physics and chemistry as of Jan. '59. *Ann. Phys. 7*:365-403.

[5] Eötvös, R.v., D. Pekar and E. Fekety, 1922. Inertia and gravity. *Ann. Phys. 68*:11-60.

[6] Fischbach, E., D. Sudarsky, A. Szafer, C. Talmadge and S.H. Aranson, 1968. Reanalysis of the Eötvös experiment. *Phys. Rev. Lett. 56*:3-6.

[7] Friedlander, G., J.W. Kennedy and J.M. Miller, 1964. *Nuclear and Radiochemistry.* (John Wiley & Son, NY).

[8] MacIntyre, F. 1964. FUZZ: An error-propagation mode for MAD. MIT Computer Center Memo.

[9] MacIntyre, F. and T. Dowling, 1968. User-oriented suggestions for floating-point and complex-arithmetic Forth Standard extensions. *J. Forth Applic. Res.*: this issue.

[10] Miller, A.R. 1979→. *MMSFORTH User's Manual.* 5th ed. 1985 (61 Lakeshore Road, Natick, MA)

Manuscript received August 1985.

Appendix: Error-Propagation Words

This appendix assumes that at least portions of the FP/CP wordsets of MacIntyre and Dowling (1986) have been loaded. In the stack notation, all numbers are on the 8087 floating-point stack in 80-bit-wide normalized form except the format-width n and flag ? , which are on the parameter stack. e is an appropriate error term. Stores and fetches are as for complex-arithmetic data types.

Stack manipulation, often identical to the related complex-arithmetic words.

```
: EDROP ( A a ->) FDROP FDROP ;
: EDUP  ( A a -> A a A a) FOVER FOVER ;
: ESWAP ( A a B b -> B b A a) F>T FRT2 F<T FRT2 ;
: MU    ( A a -> A )        FDROP ;
: SIGMA ( A a -> a ) FSWAP FDROP ;

: VARIANCE ( A a -> a^2 ) SIGMA FDUP F* ;
```

Comparisons between means and deviations.

```
: MCOMP ( A a B b -: ?) FDROP FSWAP FDROP FCOMP ;
: SCOMP ( A a B b -: ?) FSWAP FDROP FCOMP FDROP ;
```

Utility routines.

```
: SHUF  ( x y z -> yz x z ) FSWAP FOVER F* FRT2 ;
: HYPOT ( A a B b -> {a^2+b^2}^1/2 A B ) FROT FDUP F* FSWAP FDUP F*
                        F+ FSQRT FRT2 ;
: A-R   ( A a -> A a/A) FOVER F/ ;
: R-A   ( A a/A -> A a) FOVER F* ;
```

Elementary arithmetic operators

```
: E+  ( A a B b -> A+B e)              HYPOT F+ FSWAP ;
: E-  ( A a B b -> A-B e)              HYPOT F- FSWAP ;
: E*  ( A a B b -> A*B e) A-R ESWAP A-R HYPOT F* FSWAP R-A ;
: E/  ( A a B b -> A/B e) A-R ESWAP A-R HYPOT F\ FSWAP R-A ;

: 1/E ( A a -> 1/A e)     A-R FSWAP          1/X FSWAP R-A ;
: E*F ( A a f -> fA fa)  FROT FOVER F* FRT2 F* ; ( = C*F)
```

Unary Functions

```
: E^    ( A a f -> A^f afA^f-1) SHUF FOVER FSWAP 1, F- F^ SHUF F* ;
: ESIN  ( A a -> sinA a*cosA) FOVER COS F* FSWAP SIN FSWAP ;
: ECOS  ( A a -> cosA a*sinA) FOVER SIN F* FSWAP COS FSWAP ;
: EEXP  ( A a -> expA a*expA) FSWAP EXP FSWAP R-A ;
: ELN   ( A a -> lnA a/A)     A-R FSWAP LN FSWAP ;
```

I/O. See also complex-arithmetic input words in MacIntyre and Dowling (1986). `241 EMIT` prints a ± sign.

```
: .E    ( n ->) DUP FSWAP G. SPACE 241 EMIT G. ;
```