# Error-Free Statistics in Forth

*Leonard F. Zettel*
*Research Staff, Ford Motor Company*
*Dearborn, Michigan, USA*

## *Abstract*

Methods for the one-pass, error-free calculation of the mean and standard deviation of a set of numbers are presented along with code for their implementation in Forth and an accuracy analysis.

## *Introduction*

The mean and standard deviation of a set of numbers are probably the most commonly used statistics. Calculating them as rapidly and accurately as possible is not a trivial problem. The methods presented here were selected after a review of the literature. Rodden [11] in particular warned against the temptations to bad practice presented by the uncritical use of floating point calculations invited by most FORTRAN compilers. Subsequent experience has confirmed his fears. The routines presented here are designed for the case (believed by the author to be relatively common) where the numbers to be summarized may be represented as suitably scaled single precision integers without loss of accuracy. The imputation of an implicit binary point will not affect the results as long as its location is the same for all numbers used in the computations. This will be the case for enumeration data or data captured with a fixed level of accuracy (2 millivolts, 1 minute of arc, etc.). We agree with Rodden that, wherever possible, calculations should be done in ways that involve no loss of information. The principles outlined here can often be profitably used where high accuracy statistics must be calculated in floating point, although in that case there will inevitably be some rounding error and therefore information loss.

## *Algorithms*

In this work we follow the lead of [11] and for a series of integer values $x_1, x_2 \ldots x_n$ compute the following statistics:

$n$, the number of values ($x$'s)
$c$, an integer estimate of the mean (initially zero)
$s$, the value $\sum(x-c)$
$t$, the value of $\sum(x-c)^2$

$n, s,$ and $t$ are updated each time a new observation is added to the data set. In case of overflow $c, s,$ and $t$ are recomputed as follows:

$$d = \lfloor s/n \rfloor \tag{1}$$
$$c' = c + d \tag{2}$$
$$s' = s \bmod n \tag{3}$$
$$t' = t - d(s + s') \tag{4}$$

This set of recomputations, a bit different from those recommended in [11], has been chosen to take advantage of some of the operators of Forth while keeping the desirable information properties of Rodden's recommendations. **M/MOD** lets us get $d$ and $s'$ in one calculation, and **+!** is handy for $c'$ and $t'$. As we shall show below, at all times $d, s$ and $t$ are exact, having no truncation or rounding error.

Finally, when they are needed, the mean $\mu$ and variance $v$ may be computed to any desired degree of accuracy using the formulas

$$\mu = s/n + c \tag{5}$$

and

$$v = ( t - s^2/n )/( n - 1 ) \tag{6}$$

with suitable scaling and operations on remainders.

The use of the offset $c$ (a statistician would call it a coding constant) allows us to accumulate much larger sums without overflow than we could without it. This saves storage space, since $c$ is effectively the high-order bits of several quantities simultaneously. We also avoid time consuming multiple precision arithmetic on the individual updates.

## *Mathematics*

Suitably recasting the work of Rodden [11], we note that, for arbitrary $c$ and $c'$, with summations running from one to $n$,

$$\sum (x - c') = \sum (x - c) - n(c' - c) \tag{7}$$

$$\sum (x - c')^2 = \sum (x - c)^2 - 2(c' - c) \sum (x - c) + n\,(c' - c)^2 \tag{8}$$

also,

$$\sum (x - \bar{x})^2 = \sum (x - c)^2 - (1/n) \left(\sum (x - c)\right)^2 \tag{9}$$

where $\bar{x}$ is the mean, $\left(\sum x\right)/n$ .

In the notation of the previous section, (7) is

$$s' = s - n( c' - c )$$

and from (1) and (2)

$$s' = s - n \lfloor s/n \rfloor = s \bmod n$$

Also, with a little rearrangement, (8) gives us

$$t' = t - d(2s - nd) = t - d(s' + s),$$

so we see that the truncation of the floored division will not affect the accuracy of the accumulated totals.

## *Forth*

Screens 90 to 94 implement these ideas in Forth. To get a mean and standard deviation, three things have to be done: 1) Allocate storage for the totals; 2) Accumulate the statistics; 3) Calculate the mean and standard deviation from the accumulated totals. Since an application may need to do statistics on more than one variable, we introduce a defining word that does 1) in the course of creating a word that does 2). Screen 92 defines a defining word, **COLLECT**, that allocates sufficient storage to carry $c$, $n$, and the coded minimum and maximum observed values in single precision and the sum and sum of squares in double precision. $C, n, s$ and $t$ are initially zero, while the minimum is given the largest possible value and the maximum the smallest, insuring that the first actual observation will set them to their proper values. Thus the phrase **COLLECT VOLTAGES** would allocate and initialize the space in the Forth dictionary necessary for accumulating information about the statistics of voltages.

When **VOLTAGES** is subsequently executed the **DOES>** part of **COLLECT** executes code for updating the running totals with the number on the top of the stack (presumably a voltage). This gives the routines the most general applicability, since in a particular application it may be neither possible nor desirable to save the individual numbers either in computer memory or mass storage. This could be the case in an elaborate simulation or in real time collection of large amounts of data. For cases where the data are in memory, users can define a word to walk through the data, put each value on the stack, and invoke the appropriate word defined by **COLLECT**, or they can incorporate the principles outlined here in definitions tailored to fit their special circumstances. The last line of the **COLLECT DOES>** checks for overflow of the sum of squares and if necessary recalculates the appropriate quantities by invoking the word **RECODE** (shown in screen 91). Screen 90 defines the actions of the component parts of the **DOES>** of **COLLECT**.

Screens 93 and 94 are concerned with the output of statistics from information accumulated by a word defined by **COLLECT**. **AVERAGE.OF VOLTAGES** will leave the floored integer average voltage on the stack. **SD.OF MAX.OF** and **MIN.OF** work similarly for the standard deviation, maximum and minimum. **AVERAGE**, **VARIANCE**, and **SD** have been factored out and set up to use only stack values in their calculations. This allows their use in situations where the quantities have been accumulated by a method other than a word defined by **COLLECT**. Words using **/MOD** and ***/** that would compute the quantities to more significant figures should be fairly straightforward and are left to users and their particular applications.

**CLEAR** is useful in applications like simulation, where one wants to clear out statistics gathered during a starting transient. Retaining the current value of *c* here delays the time when recoding becomes necessary.

## *Analysis and Limits*

Individual numbers on which statistics are gathered are limited to the range -32768 to 32767 both before and after coding, so besides being in the range above, no number can differ from the average of the set by more than 32,767. The memory locations reserved for *n*, *s* and *t* can all potentially overflow, so statistics can be collected for a maximum of 32,767 entries. The buckets for *s* and *t* are the same size, 32 bits. Since any given addition to *t* must be greater than the corresponding addition to *s* unless they are both one or zero, *t* will overflow first if overflow occurs with less than 32,768 entries. Also, since *t* is a sum of squares, all entries and therefore the total will always be nonnegative. The largest possible single addition to *t* is $2^{30}$ ( hex 4000 0000 decimal 1073741824). Since overflow occurs for sums greater than $2^{31} - 1$ (7FFF FFFF or 2147483647), overflow will always be signaled by a negative value for $t'$ because the sum after overflow will have to lie between $2^{32}$ and $2^{31}$.

If the variance of the numbers is big enough, *t* may overflow even after recoding. This condition is not checked for and could lead to erroneous results. [11] sketches methods of overcoming this problem that would require somewhat more elaborate algorithms than those given here.

## *Instrumentation and Software*

The screens shown here were coded in F83 [6] and compiled and run on an IBM PS2 model 80-111. Words not part of the Forth-83 standard and their meanings are as follows:

| | | |
|---|---|---|
| *D | ( n1 n2 - d1 ) | n1 and n2 are multiplied, giving a signed double precision result. [6] |
| .ID | ( addr - ) | The contents of name field address addr are displayed. [6] |
| .R | ( n1 n2 - ) | n1 is converted using BASE and displayed in a field n2 characters wide. Forth-83 controlled reference word [1]. |
| 2+! | ( d addr - ) | Add signed double number d to the 32 bit memory contents starting at addr. [6] |
| D* | ( d1 d2 - q1 ) | q1 is the signed 64 bit product of signed double numbers d1 and d2. |
| DN* | ( d1 n1 - d2 ) | d2 is the signed product of signed double number d1 and signed number n1. |
| ERASE | ( addr u - ) | U bytes of memory beginning at addr are set to zero. Forth-83 controlled reference word [1]. |

| | | |
|---|---|---|
| `M/MOD` | ( d1 n1 - n2 n3 ) | n2 is the signed single remainder and n3 is the signed single quotient when signed double number d1 is divided by signed single number n1. [6] |
| `MU/MOD` | ( d1 n1 - n2 d2 ) | n2 is the signed single remainder and d2 is the signed double quotient of signed double dividend d1 and signed single divisor n1. [6] |
| `QS/` | ( q1 n - q2) | q2 is the 64 bit signed quotient of 64 bit signed dividend q1 and signed single number n1. |
| `S>D` | ( n - d) | Convert signed single precision number to signed double precision. [6] |
| `SQRT` | ( d1 - n1) | n1 is the single number square root of double number d1. [13] |

## *References*

[1] Chan, Tony F. and John Gregg Lewis, "Computing Standard Deviations: Accuracy," *Comm ACM*, Vol 22, #9, September 1979, pp 526-31.

[2] Chan, Tony F., Gene H. Golub and Randall J. LeVeque, "Algorithms for Computing the Sample Variance: Analysis and Recommendations," *Amer. Statistician,* Vol 37, #3, August 1983, pp 242-7.

[3] Cloutier, Mark J. and Matthew J. Friedman, "Precision Averaging for Real-Time Analysis," *Comm ACM*, Vol 26, #7, July 1983, pp 525-29.

[4] Deegan, John Jr., "A Test of the Numerical Accuracy of Some Matrix Inversion Algorithms Commonly Used in Least Squares Programs," *J. Statist. Comput. Simul.,* 1976, Vol 4, #4, pp 269-78.

[5] Forth Standards Team, *Forth-83 Standard*, Mountain View Press, Mountain View CA 1983.

[6] Laxen, Henry and Michael Perry, *F83*. Available from Offete Enterprises, Inc., 1306 South B St., San Mateo CA 94402.

[7] Ling, Robert F., "Comparison of Several Algorithms for Computing Sample Means and Variances," *J. Am. Statistical Assn*, Vol 69, #348, December 1974, PP 859-66.

[8] Marasinghe, Mervyn G., "A Note on Methods for Computing Tukey's and Mandel's Interaction Sums of Squares," *Comm, Statist.-Simula.,* Vol 15, #3, 1986, pp 649-54.

[9] Neely, Peter M., "On the Use of Integer Arithmetic to Achieve Confirmably Correct Computation," *Software-Practice and Experience*, Vol 7, #2, 1977, pp 159-63.

[10] Neely, Peter M., "Comparison of Several Algorithms for Computation of Means, Standard Deviations and Correlation Coefficients," *Comm ACM,* Vol 9, #7, July 1966, pp 496-9.

[11] Rodden, B. E., "Error-Free Methods for Statistical Computations," *Comm ACM*, Vol 10, #3, March 1967, pp 179-80.

[12] Ryder, K., "Calculators – How to Get the Answers Right," *Expl Agric*, Vol 19, #1, 1983, pp 15-21.

[13] Suralis, Klaxon, "Square root routines in Forth," *Forth Dimensions*, Vol. IV, #1, pp. 9-11.

[14] Thisted, Ronald A., *Elements of Statistical Computing*, Chapman and Hall, 29 West 35th Street, New York NY 10001, 1988

[15] Welford, B. P., "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, Vol 4, #3, August 1962, pp 419-20.

[16] West, D. H. D., "Updating Mean and Variance Estimates: An Improved Method," *Comm ACM*, Vol 22, #9, September 1979, pp 532-5.

[17] Wilkinson, Leland and Gerard E. Dallal, "Accuracy of Sample Moments Calculations Among Widely Used Statistical Programs," *American Statistician*, Vol 31, #3, August 1977, pp 128-31.

```
Scr # 90
\ ADJUST INCREMENT SUM SUM.SQUARES MINIMUM MAXIMUM     LFZ11Jul86
\ Words to update a statistics array.

: ADJUST ( n1 addr -- n2 ) @ - ;

: INCREMENT ( addr -- ) \ Add 1 to the count of entries.
  1 SWAP +! ;

: SUM ( n1 addr -- ) SWAP S>D ROT 2+! ; \ Update sum(x-c).

: SUM.SQUARES ( n1 addr -- )            \ Update sum(x-c)**2.
  SWAP DUP *D ROT 2+! ;

: MINIMUM ( n1 addr -- )                \ Update minimum.
  SWAP OVER @ MIN SWAP ! ;
: MAXIMUM ( n1 addr -- ) SWAP OVER @ MAX SWAP ! ;
Scr # 91
\ RECODE                                                LFZ13Jul86

: RECODE ( addr -- ) \ Calculate a new offset for the statistics
                \ array at addr & adjust the entries accordingly.
  DUP >R 4 + 2@                        \ sum
  2DUP R@ 2+ @                         \ number entries
  M/MOD DUP R@ +!                      \ new offset.
  OVER 0 R@ 4 + 2!                     \ new sum.
  DUP NEGATE DUP R@ 12 + +!            \ new minimum.
                 R@ 14 + +!            \ new maximum.
  >R 0 D+ R> DN* DNEGATE R> 8 + 2+! ;  \ new sum of squares.





Scr # 92
\ COLLECT CLEAR                                         LFZ
: COLLECT CREATE HERE 16 ALLOT DUP 12 ERASE
                 32767 OVER 12 +  !      \ minimum.
                -32768 SWAP 14 +  !      \ maximum.
( n -- )  DOES>  SWAP OVER     ADJUST
                 SWAP DUP   2+  INCREMENT
                 2DUP       4 + SUM
                 2DUP       8 + SUM.SQUARES
                 2DUP      12 + MINIMUM
                 SWAP OVER 14 + MAXIMUM
                 DUP 8 + @ 0< IF RECODE ELSE DROP THEN ;

: CLEAR (  -- <name> )      \ Reset the named statistics
  ' >BODY >R R@ 2+ 10 ERASE \ Clear count, sum, sum of squares.
   32767 R@ 12 + !          \ New minimum.
  -32768 R> 14 + ! ;        \ New maximum.
```

```
Scr # 93
\ AVERAGE VARIANCE SD          ZETTEL                   LFZ24Jul86
: AVERAGE ( n1 d1 n2 -- n3 )   \ N3 = The average of the sum sum d1
                               \      count n2 and offset n1.
  M/MOD SWAP DROP + ;

: VARIANCE ( d1 d2 n1 -- d3 )  \ D3 = the variance of sum of
 \ squares d1, sum d2 and count n1. Return -1 if count too low
  DUP 2 < IF DROP 2DROP 2DROP -1
          ELSE >R DABS 2DUP D* R@ QS/ 2DROP D-  R> 1- MU/MOD
               ROT DROP
          THEN ;

: SD ( d1 d2 n1 -- n2 )  \ n2 is the standard deviation of the
                         \ sum of squares d1 sum d2 and count n1.
                         \ Return -1 if count too low.
  VARIANCE DUP 0< IF ELSE  SQRT THEN ;
Scr # 94
\ AVERAGE.OF SD.OF MIN.OF MAX.OF                        LFZLFZ
: OFFSET.SUM.COUNT.OF ( -- <name> n1 d1 n3 )
  ' >BODY DUP >R @ R@ 4 + 2@ R> 2+ @ ;
: AVERAGE.OF ( -- <name> n1 )    \ N1 = average of the COLLECT
  OFFSET.SUM.COUNT.OF AVERAGE ; \ statistics at addr.

: SD.OF ( -- <name> n1 )         \ n1 = standard deviation of the
                                 \ COLLECT statistics at addr.
  ' >BODY >R R@ 8 + 2@ R@ 4 + 2@ R> 2+ @ SD ;

: MIN.OF ( -- <name> n1 )        \ n1 = minimun of the COLLECT
  ' >BODY DUP @ SWAP 12 + @ + ; \ statistics at addr.

: MAX.OF ( -- <name> n1 )        \ n1 = The maximum of the COLLECT
  ' >BODY DUP @ SWAP 14 + @ + ; \ statistics at addr.

```

*Mr. Zettel has a B.S. in Chemical Engineering. He received an M.B.A. from Canisius College in 1975. He has been actively involved with computers since 1962, and is currently a researcher at the Ford Motor Company.*