

# **NOTE O: AVERAGES, STANDARD DEVIATIONS AND PRE-CENTERING**

## **AVERAGE FUNCTION, EXCEL 2000 AND 2003**

Tests and comments all affirm that the AVERAGE function calculation is accurate. One commentator did say that it is only by taking the list for which an average is to be calculated, by first sorting from smallest to largest, the summation properly increases, allowing the smaller values to properly be included in the sum. My tests have shown that this method does not really improve the accuracy over the standard method for typical data lists. This is supported by Higman (1984)

## **STDEV FUNCTION, EXCEL 2000**

The Excel standard deviation function is a main element of most statistical functions and routines. It is as the commentators say, an inaccurate algorithm that should be corrected. The Excel algorithm is shown in Microsoft (1992a) for the STDEV function and in the Help screens. The algorithm was extensively used in the days when the only calculations were made with mechanical desk calculators. It is an obsolete form, and should not be used in modern day computers. It is not robust against number size given the limitations of IEEE 64 bit floating point arithmetic

The error in the standard deviation calculation depends on the number of significant figures in the data set, the mean of the data set, the magnitude of the variation of data values and the number of data elements. A relative measure of the variation is the coefficient of variation ( $COV = \text{standard deviation}/\text{mean}$ ) for data sets having one sign.

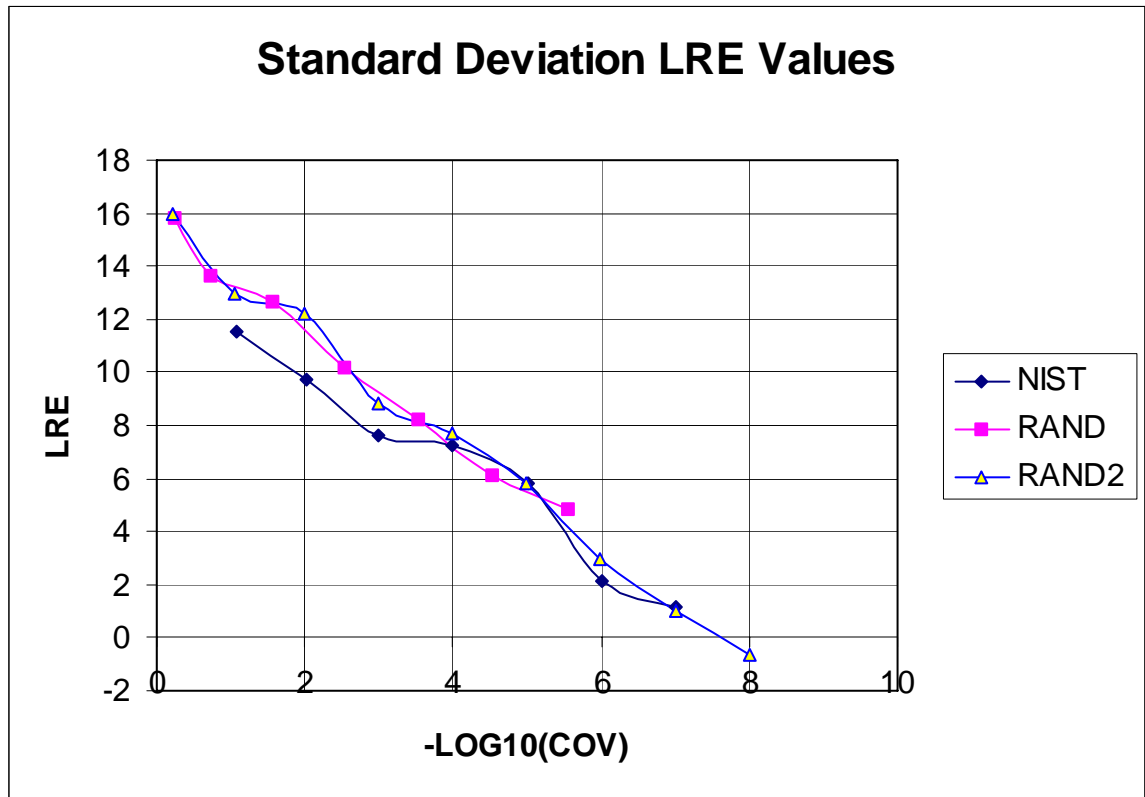
For each data set a COV value can be calculated, but it is generally a very small value. However by taking the log to the base 10 of the COV and changing the sign, a number is obtained that is a more acceptable measure of variability. The symbol L10COV is used for this value.

For data sets having both + and – values, the calculated standard deviation will be more accurate than one sign data sets.

Using L10COV as a measure of the relative variation, we can compare the LRE of the Excel STDEV function on different data sets having different sizes and characteristics. This effect is shown in tables 3 and 10 of the main article

From table 3 in the main article, we can focus on the larger data sets, since error increases as the data set size increases. The NIST NumAcc data sets all have alternating values of X.3 and X.1 with one X.2 added to give a true standard deviation of 0.1 and a true mean of X.2, where X varies from 1 to 10 million. Figure 1 is a plot of the LRE of the STDEV function versus L10COV values for data sets with 1001 points. The lines refer to the three variations put on the data as described for tables 6, 7 and 8 in the main article. The type of variation does have an effect.

Figure I-1: LRE Values for Different Data Sets With Different COV Values (As LOG10COV)



1

Given the general inaccuracy of the STDEV function, it is of interest to show what improvements can be made in the standard deviation calculation.

The first change or workaround is to center the data and then use the Excel functions. This is shown below in table I-1.

Table I-1: Univariate Analysis Results with StRD data sets

Sequence	Source	Data Set	Category	Difficulty	Size	Number of Significant Figures	1/(Coefficient of Variation)
1	NIST	PiDigits	Univariate	1	5000	1	1.58
2	NIST	Lottery	Univariate	1	218	3	1.78
3	NIST	Lew	Univariate	1	200	3	0.64
4	NIST	Maveo	Univariate	1	50	5	4,665
5	NIST	Michelso	Univariate	1	100	5	3,795
6	NIST	NumAcc1	Univariate	1	3	8	10,000,002
7	NIST	NumAcc2	Univariate	2	1001	2	120
8	NIST	NumAcc3	Univariate	2	1001	8	10,000,002
9	NIST	NumAcc4	Univariate	3	1001	9	100,000,002

**Table I-1: Continued**

Sequence	Excel With Centered data			Excel, Welford's Algorithm		
	Mean	Standard Deviation	First Order Correlation Coefficient	Mean	Standard Deviation	First Order Correlation Coefficient
1	15.7	15.2		14.6	15.0	13.8
2	15.2	15.7		15.4	15.7	15.2
3	15.7	15.4		15.8	15.4	14.9
4	15.7	13.1		15.4	12.1	11.8
5	15.7	13.9		15.7	12.4	12.1
6	15.7	15.7		16.0	15.7	15.7
7	15.7	15.7		14.0	15.3	13.7
8	15.7	9.5	12.2	16.0	9.5	11.5
9	15.7	3.3	11.3	15.7	8.3	10.7

In all cases, the accuracy of the calculation is improved by centering the data about its mean or near mean before doing the calculation. The sums of the values here become zero, and the errors from differences of large values in the Microsoft algorithm disappears.

The next change is to use a different function that contains a different algorithm. Knuth (1998) Vol. 2, page 232 recommends Welford's algorithm (Welford 1962). Results of computations using the algorithm also are shown in table 1. Welford's algorithm however could not be directly adapted to the specific autocorrelation function given by NIST, and a mixed approach was taken. Welford's algorithm is an exact solution, not an approximation. One form of the algorithm is given below: It is Knuth's modification. Both the mean and standard deviation are output values.

```

Data X(1 to N)
A = X(1)
B = 0
FOR K = 2 to N
    U = X(K) - A
    A = A + U / CDBL(K)
    B = B + U * (X(K) - A)
NEXT K
AVERAGE = A
STDEV = SQR( B / CDBL(N - 1) )

```

Table I-2 shows the improvements that can be made in the accuracies by the above changes. There are 1001 values in each data set. The first row represents the size of the additive by the number of zeros after a one. The number 3 represents 1,000. The values in the other cells are LRE values.

**Table I-2: A Consolidation of the Effects of Different Calculation Methods and Different Distributions on STDEV and Welford's Algorithm. LRE Values.**

Distribution	Additive, Number of zero's	0	1	2	3	4	5	6	7	8
Uniform	L10COV	0.23	1.55	2.53	3.53	4.53	5.53	6.53	7.53	8.53
	STDEV	14.49	11.86	10.77	8.32	5.76	4.18	2.89	0.44	0.00
	STDEV Centered	15.24	15.72	15.42	14.52	13.31	13.11	11.23	10.46	10.42
	Welford's	16.00	15.02	14.20	12.97	12.11	11.23	10.11	9.43	8.36
	Welford's with Kahan's	16.00	16.00	15.42	14.20	13.13	12.22	11.79	10.13	10.56
NumAcc	L10COV	0.30	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00
	STDEV	13.54	9.76	7.64	7.22	5.82	2.10	1.14	0.00	0.00
	STDEV Centered	13.54	13.54	13.64	13.39	12.50	11.26	10.84	9.46	8.25
	Welford's	16.00	16.00	14.23	13.85	12.46	11.26	10.84	9.45	8.25
Normal	L10COV	0.00	1.03	2.03	3.03	4.03	5.03	6.03	7.03	8.03
	STDEV	16.00	13.10	11.55	9.37	7.03	5.01	3.01	2.06	0.00
	STDEV Centered	15.45	15.15	15.23	14.63	14.10	12.89	11.94	10.86	10.04
	Welford's	15.23	16.00	15.32	13.82	12.55	12.06	10.59	11.19	8.97

The standard STDEV performs poorly as the LOG10COV value increases. In a general sense, only 3 accurate figures can be obtained for data sets with LOG10COV values of 6. It decreases to no accurate figures for LOG10COV values above 7. This loss in accuracy is due to summation errors.

When the number of digits is increased from the Excel standard 15 to 30 (in xnumbers see Note aa) for the NIST NumAcc4 data set results in 0.1000000000000000000000000024682259. NIST states that the exact value is 0.1

Centering in normal Excel calculations has a slight advantage over Welford's algorithm. However as a fast algorithm, Welford's is an acceptable method for calculating standard deviations. The loss in LRE values follows the loss in the size of the significant digits, as the additive increases. By making summation error corrections

Welford's algorithm is consistent in that the sum of the L10COV value and the LRE value run consistently from 15 to 17. An estimation of the accuracy of Welford's algorithm can simply be based on the L10COV value of the data set.

An improvement in Welford's algorithm would be to include Kahan's algorithm to correct for summation errors. This would add to the code and slightly reduce the speed of

execution. The improvement in accuracy is shown above for the uniform distribution case.

## **VAR AND STDEV FUNCTIONS, EXCEL 2003**

### **BASIC CHANGES MADE**

Microsoft made extensive changes to the VAR function with impacts (and corrects) a lot of other functions that used VAR, including STDEV. The changes are described in KBA 826112 and KBA 826349. They refer to the new version as “the two pass” method, the first pass calculates the average, and the second pass calculates the deviations from the average, squares the deviation and sums the squares.

In Excel 2003, all the following functions were changed.

VAR  
VARA  
VARP  
VARPA  
STDEV  
STDEVA  
STDEVP  
STDEVPA  
DVAR  
DVARP  
DSTDEV  
DSTDEVP

### **ARE THESE FUNCTIONS FIXED?**

For about all of the general uses of these functions, they will now return correct values.

However there still exists a fundamental error that Microsoft entirely overlooked.

The change for Excel 2003 was probably a quick fix without any real search in the computer sciences literature. The Microsoft fix has an old known problem, which is easily shown. Work on correcting this problem was done back in the 1970's.

Enter three identical values, 1E+30, 1E+30 and 1E+30 into cells and do a STDEV function on this range. The result is 1.72368E+14, not zero as expected. Also do a VAR on this range, and 2.97106E+28 will appear.

This problem appeared in the R (language) archives several years ago, and is discussed below. The correct value is a zero standard deviation, but Excel returns 1.72368E+14.

The correct fix would have been to first center the data, and then use the old calculator formula, which corrects for the fact that the sum of the centered data does not come out to a true zero. The old calculator formula had a correction for a non-zero mean.

The following correspondence from the R Language Archive describes the problem, since it is identical to the Excel situation. The same IEEE standards are used and the same algorithm is used in both statistical programs.

## DIALOGUE

```
Not a bug, round off:  
> (1e30+1e30+1e30)/3-1e30  
[1] 1.407375e+14  
> 3*((1e30+1e30+1e30)/3-1e30)^2/2  
[1] 2.971056e+28
```

Relative errors of the order 1e-16 are completely expectable in floating point arithmetic.

For illustration, look at it in octal, after removing a bunch of trailing zeroes:

```
> x<-1e30/(8^16)  
> structure(x, class="octmode")  
[1] "144762623464043165"  
> structure(x+x+x, class="octmode")  
[1] "456730272634151540"
```

If you look carefully, you will see that a bit is lost in the process since 5+5+5 is 17 octal.

```
---- Peter Dalgaard      Blegdamsvej 3  
--- Dept. of Biostatistics 2200 Cph. N  
-- University of Copenhagen Denmark Ph: (+45) 35327918  
~~~~~ - (p.dalgaard@biostat.ku.dk)      FAX: (+45) 35327907
```

Essentially our current algorithm in cov.c is

```
xm<- sum(x)/n  
v <- 1/(n-1)*sum((x-xm)^2)
```

"It" being the behavior reported in bug report PR#1228 --- [too bad you didn't use the whole string "PR#1228" in your subject; if you had, no new report would have been created, and things would have gone to the proper place in the R-bugs repository...]

Namely the fact that

```
var(rep(1e30, n))  
does not always (for all n) give 0. With the following function  
tst.var <- function(x,nset= 2:100) {  
  for(n in nset) {  
    v <- var(rep(x,n))  
    if(v != 0) cat(sprintf("%4d: %20.12g\n", n,v))  
  }  
}
```

Actually, on my current desktop (AMD Athlon, Linux, R 1.9.0beta) the computations seem to be more often exact than they used to: Even `tst.var(1e30, 2:5000)` doesn't produce any output nor do a few other 'x' arguments I tried --- but the fundamental criticism is still correct, and even on the Athlon, it's easy to find cases where `tst.var()` \*does\* produce output.

daheiser> It is an excellent example of how errors and  
daheiser> faults occur when the programmer follows the  
daheiser> mathematical formula exactly. Welford's algorithm  
daheiser> does not produce this error. It gives correct  
daheiser> standard deviation and variance values.

Actually, after reading

```
@article{ChaTL97,  
  author = {Tony F. Chan and John Gregg Lewis},  
  title = {Computing standard deviations: accuracy},  
  journal = {Commun. ACM},  
  volume = 22,  
  number = 9,  
  year = 1979,  
  issn = {0001-0782},  
  pages = {526--531},  
  doi = {http://doi.acm.org/10.1145/359146.359152},  
  publisher = {ACM Press},  
}
```

```
@article{WesD97,  
  author = {D. H. D. West},  
  title = {Updating mean and variance estimates: an improved method},  
  journal = {Commun. ACM},  
  volume = 22,  
  number = 9,  
  year = 1979,  
  issn = {0001-0782},  
  pages = {532--535},  
  doi = {http://doi.acm.org/10.1145/359146.359153},  
  publisher = {ACM Press},  
}
```

I'd conclude (from page 531) that Welford's algorithm is a bit  
less accurate than the (very similar) "West" version,  
and we (the R developers) should rather implement the latter.

Maybe for R 1.9.1 -- or even later {there are even more  
important numerical accuracy problems I know about!}

Martin Maechler <maechler@stat.math.ethz.ch>

<http://stat.ethz.ch/~maechler/>

Seminar fuer Statistik, ETH-Zentrum LEO C16 Leonhardstr. 27  
ETH (Federal Inst. Technology) 8092 Zurich SWITZERLAND  
phone: x-41-1-632-3408 fax: ...-1228 <>

## HIGH ACCURACY VARIANCE AND STANDARD DEVIATION FUNCTIONS

A substantial improvement in the accuracy of the variance and standard deviation values can be attained by using the XNUMBERS add-in capabilities. This add-in is described in Note AA. Xnumbers are actually character strings made up of the 10 numerical digits. The default length is 30 decimal digits, but can be extended to 200 decimal digits..

Welford's basic algorithm is an accurate computation of variance/standard deviation, but as discussed above, loses accuracy as the L10COV value gets large. This is the basic accuracy-in-summations problem. By using the Xnumber approach, high accuracy can be readily attained for high L10COV data sets. Welford's algorithm programmed as a VBA function using xnumber functions would be as follows:

```
Public Function xWelford(ar) As Variant
'xnumber version of Welford's algorithm for standard deviation
'ar is the range of the data, in cell range format (A1:A1000) for example

Dim u As Range
Dim got As Boolean
Dim nxrows, nxcols, fcoln, frown As Integer
Dim xM0, xS0, xM1, xS1, xc As String
Dim xt1, xt2, xt3, xt4 As String

'Stop
With Application.ActiveSheet
'Stop
  nxrows = ar.Rows.count 'number of rows
  nxcols = ar.Columns.count 'number of columns
  fcoln = ar.Column 'column number first column
  frown = ar.Row 'row number first row
'Stop
  xM0 = "0"
  xS0 = "0"
  xc = "0"
'Stop
  If nxcols = 1 Then 'data in a column
    For i = 1 To nxrows
      ri = i + frown - 1 'cell row location of input '.cells(ri,fcoln)
      If WorksheetFunction.IsText(.Cells(ri, fcoln).Value) Then
        If IsNumeric(.Cells(ri, fcoln).Value) Then
          xx1 = .Cells(ri, fcoln).Value
          got = True 'x-number string like "1.2834...", etc
        Else
          got = False 'string like "xyx..", "a", etc..
        End If
      End If
    Stop '
  End If

  If WorksheetFunction.IsLogical(.Cells(ri, fcoln)) Then
    got = False
    Stop
  Else
    If WorksheetFunction.IsText(.Cells(ri, fcoln).Value) Then
      xx1 = .Cells(ri, fcoln).Value
      got = True
    Stop
  End If
End With
End Function
```

```

ElseIf WorksheetFunction.IsNumber(.Cells(ri, fcoln).Value) Then
    xx1 = str$(.Cells(ri, fcoln).Value)
    got = True
    'Stop
Else
    got = False
    Stop
End If
End If
If got Then
    'as xNumbers
    xc = xadd(xc, "1", 30)
    xt1 = xsub(xx1, xM0, 30)
    xt2 = xdiv(xt1, xc, 30)
    xM1 = xadd(xM0, xt2, 30)
    xt3 = xsub(xx1, xM1, 30)
    xt4 = xmult(xt1, xt3, 30)
    xS1 = xadd(xS0, xt4, 30)
    xM0 = xM1
    xS0 = xS1
    'Stop
End If
Next i
'Stop
ElseIf nxrows = 1 Then 'data in a row
    For i = 1 To nxcols
        ri = i + fcoln - 1 'cell column location of input
        '.cells(frown,ri)
        If WorksheetFunction.IsLogical(.Cells(frown, ri)) Then
            got = False
            Stop
        Else
            If WorksheetFunction.IsText(.Cells(frown, ri).Value) Then
                xx1 = .Cells(frown, ri).Value
                got = True
                Stop
            ElseIf WorksheetFunction.IsNumber(.Cells(frown, ri).Value) Then
                xx1 = str$(.Cells(frown, ri).Value)
                got = True
                'Stop
            Else
                got = False
                Stop
            End If
            If got Then
                'as xNumbers
                xc = xadd(xc, "1", 30)
                xt1 = xsub(xx1, xM0, 30)
                xt2 = xdiv(xt1, xc, 30)
                xM1 = xadd(xM0, xt2, 30)
                xt3 = xsub(xx1, xM1, 30)
                xt4 = xmult(xt1, xt3, 30)
                xS1 = xadd(xS0, xt4, 30)
                xM0 = xM1
                xS0 = xS1
            End If
        End If
    Next i
Else
    Stop
End If
xt1 = xsub(xc, "1", 30) 'this is the count of the number of numbers added
xt2 = xdiv(xS1, xt1, 30) 'this is the basic variance
xWelford = xsqr(xt2, 30) 'this is the standard deviation

```

End With  
End Function