WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
فارسی
Türkçe
Edit links

Algorithms for calculating variance

From Wikipedia, the free encyclopedia

It has been suggested that *Standard deviation#Rapid calculation methods* be merged into this article. (Discuss) *Proposed since February 2013.*

Algorithms for calculating variance play a major role in computational statistics. A key problem in the design of good algorithms for this problem is that formulas for the variance may involve sums of squares, which can lead to numerical instability as well as to arithmetic overflow when dealing with large values.

Contents [hide]

Naïve algorithm [edit]

A formula for calculating the variance of an entire population of size *N* is:

$$\sigma^2 = \bar{(x^2)} - \bar{x}^2 = \frac{\sum_{i=1}^N x_i^2 - (\sum_{i=1}^N x_i)^2/N}{N}.$$

Using Bessel's correction to calculate an unbiased estimate of the population variance from a finite sample of *n* observations, the formula is:

$$s^2 = \frac{\sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2/n}{n-1}.$$

Therefore, a naive algorithm to calculate the estimated variance is given by the following:

```
• Let $n \leftarrow 0, \mathrm{Sum} \leftarrow 0, \mathrm{SumSq} \leftarrow 0$
• For each datum $x$:
    • $n \leftarrow n + 1$
    • $\mathrm{Sum} \leftarrow \mathrm{Sum} + x$
    • $\mathrm{SumSq} \leftarrow \mathrm{SumSq} + x \times x$
• $\mathrm{Var} = (\mathrm{SumSq} - (\mathrm{Sum} \times \mathrm{Sum}) / n) / (n - 1)$
```

This algorithm can easily be adapted to compute the variance of a finite population: simply divide by *N* instead of *n* − 1 on the last line.

Because SumSq and $(\mathrm{Sum}\times\mathrm{Sum})/n$ can be very similar numbers, cancellation can lead to the precision of the result to be much less than the inherent precision of the floating-point arithmetic used to perform the computation. Thus this algorithm should not be used in practice.[1][2] This is particularly bad if the standard deviation is small relative to the mean. However, the algorithm can be improved by adopting the method of the assumed mean.

Computing shifted data [edit]

We can use a property of the variance to avoid the catastrophic cancellation in this formula, namely the variance is invariant with respect to changes in a location parameter

$$\mathrm{Var}(X - k) = \mathrm{Var}(X).$$

with k any constant, which leads to the new formula

$$s^2 = \frac{\sum_{i=1}^n (x_i - K)^2 - (\sum_{i=1}^n (x_i - K))^2/n}{n-1}.$$

the closer K is to the mean value the more accurate the result will be, but just choosing a value inside the samples range will guarantee the desired stability. If the values $(x_i - K)$ are small then there are no problems with the sum of its squares, on the contrary, if they are large it necessarily means that the variance is large as well. In any case the second term in the formula is always smaller than the first one therefore no cancellation may occur.[2]

If we take just the first sample as K the algorithm can be written in Python programming language as

```
def shifted_data_variance(data):
   if len(data) == 0:
      return 0
   K = data[0]
   n = 0
   Sum = 0
   Sum_sqr = 0
   for x in data:
      n = n + 1
      Sum += x - K
      Sum_sqr += (x - K) * (x - K)
   variance = (Sum_sqr - (Sum * Sum)/n)/(n - 1)
   # use n instead of (n-1) if want to compute the exact variance of the given data
   # use (n-1) if data are samples of a larger population
   return variance
```

this formula facilitates as well the incremental computation, that can be expressed as

```
K = 0
n = 0
Ex = 0
Ex2 = 0

def add_variable(x):
    if (n == 0):
      K = x
    n = n + 1
    Ex += x - K
    Ex2 += (x - K) * (x - K)

def remove_variable(x):
    n = n - 1
    Ex -= (x - K)
    Ex2 -= (x - K) * (x - K)

def get_meanvalue():
   return K + Ex / n

def get_variance():
    return (Ex2 - (Ex*Ex)/n) / (n-1)
```

Two-pass algorithm [edit]

An alternative approach, using a different formula for the variance, first computes the sample mean,

$$\bar{x} = \frac{\sum_{j=1}^{n} x_j}{n},$$

and then computes the sum of the squares of the differences from the mean,

$$\mathrm{variance} = s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1},$$

where s is the standard deviation. This is given by the following pseudocode:

```
def two_pass_variance(data):
    n    = 0
    sum1 = 0
    sum2 = 0

    for x in data:
        n    = n + 1
        sum1 = sum1 + x

    mean = sum1 / n

    for x in data:
        sum2 = sum2 + (x - mean)*(x - mean)

    variance = sum2 / (n - 1)
    return variance
```

This algorithm is numerically stable if n is small.[1][3] However, the results of both of these simple algorithms (I and II) can depend inordinately on the ordering of the data and can give poor results for very large data sets due to repeated roundoff error in the accumulation of the sums. Techniques such as compensated summation can be used to combat this error to a degree.

Compensated variant [edit]

The compensated-summation version of the algorithm above reads:[4]

```
def compensated_variance(data):
    n = 0
    sum1 = 0
    for x in data:
        n = n + 1
        sum1 = sum1 + x
    mean = sum1/n

    sum2 = 0
    sum3 = 0
    for x in data:
        sum2 = sum2 + (x - mean)**2
        sum3 = sum3 + (x - mean)
    variance = (sum2 - sum3**2/n)/(n - 1)
    return variance
```

Online algorithm [edit]

It is often useful to be able to compute the variance in a single pass, inspecting each value x_i only once; for example, when the data are being collected without enough storage to keep all the values, or when costs of memory access dominate those of computation. For such an online algorithm, a recurrence relation is required between quantities from which the required statistics can be calculated in a numerically stable fashion.

The following formulas can be used to update the mean and (estimated) variance of the sequence, for an additional element x_{new}. Here, $\bar{x}_n$ denotes the sample mean of the first n samples (x_1, ..., x_n), s^2_n their sample variance, and σ^2_n their population variance.

$$\bar{x}_n = \frac{(n-1)\,\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$s^2_n = \frac{(n-2)}{(n-1)}\,s^2_{n-1} + \frac{(x_n - \bar{x}_{n-1})^2}{n}, \quad n>1$$

$$\sigma^2_n = \frac{(n-1)\,\sigma^2_{n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n}.$$

These formulas suffer from numerical instability. A better quantity for updating is the sum of squares of differences from the (current) mean, $\sum_{i=1}^n (x_i - \bar{x}_n)^2$, here denoted $M_{2,n}$:

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

$$s^2_n = \frac{M_{2,n}}{n-1}$$

$$\sigma^2_n = \frac{M_{2,n}}{n}$$

A numerically stable algorithm for the sample variance is given below. It also computes the mean. This algorithm is due to Knuth,[5] who cites Welford,[6] and it has been thoroughly analyzed.[7][8] It is also common to denote $M_k = \bar{x}_k$ and $S_k = M_{2,k}$.[9]

```
def online_variance(data):
    n = 0
    mean = 0.0
    M2 = 0.0

    for x in data:
        n = n + 1
        delta = x - mean
        mean = mean + delta/n
        M2 = M2 + delta*(x - mean)

    if n < 2:
        return float('nan');
    else:
        return M2 / (n - 1)
```

This algorithm is much less prone to loss of precision due to massive cancellation, but might not be as efficient because of the division operation inside the loop. For a particularly robust two-pass algorithm for computing the variance, first compute and subtract an estimate of the mean, and then use this algorithm on the residuals.

The parallel algorithm below illustrates how to merge multiple sets of statistics calculated online.

Weighted incremental algorithm [edit]

The algorithm can be extended to handle unequal sample weights, replacing the simple counter n with the sum of weights seen so far. West (1979)[10] suggests this incremental algorithm:

```
def weighted_incremental_variance(dataWeightPairs):
    sumweight = 0
    mean = 0
    M2 = 0
    for x, weight in dataWeightPairs:  # Alternatively "for x, weight in zip(data, weights):"
        temp = weight + sumweight
        delta = x - mean
        R = delta * weight / temp
        mean = mean + R
        M2 = M2 + sumweight * delta * R  # Alternatively, "M2 = M2 + weight * delta * (x-mean)"
        sumweight = temp

    variance_n = M2/sumweight
    variance = variance_n * len(dataWeightPairs)/(len(dataWeightPairs) - 1)
```

Parallel algorithm [edit]

Chan et al.[4] note that the above online algorithm III is a special case of an algorithm that works for any partition of the sample X into sets X_A, X_B:

$$\delta = \bar x_B - \bar x_A$$
$$\bar x_X = \bar x_A + \delta\cdot\frac{n_B}{n_X}$$
$$M_{2,X} = M_{2,A} + M_{2,B} + \delta^2\cdot\frac{n_A n_B}{n_X}.$$

This may be useful when, for example, multiple processing units may be assigned to discrete parts of the input.

Chan's method for estimating the mean is numerically unstable when $n_A \approx n_B$ and both are large, because the numerical error in $\bar x_B - \bar x_A$ is not scaled down in the way that it is in the $n_B = 1$ case. In such cases, prefer $\bar x_X = \frac{n_A \bar x_A + n_B \bar x_B}{n_A + n_B}$.

Example [edit]

Assume that all floating point operations use the standard IEEE 754 double-precision arithmetic. Consider the sample (4, 7, 13, 16) from an infinite population. Based on this sample, the estimated population mean is 10, and the unbiased estimate of population variance is 30. Both Algorithm I and Algorithm II compute these values correctly. Next consider the sample (10^8 + 4, 10^8 + 7, 10^8 + 13, 10^8 + 16), which gives rise to the same estimated variance as the first sample. Algorithm II computes this variance estimate correctly, but Algorithm I returns 29.333333333333332 instead of 30. While this loss of precision may be tolerable and viewed as a minor flaw of Algorithm I, it is easy to find data that reveal a major flaw in the naive algorithm: Take the sample to be (10^9 + 4, 10^9 + 7, 10^9 + 13, 10^9 + 16). Again the estimated population variance of 30 is computed correctly by Algorithm II, but the naive algorithm now computes it as −170.66666666666666. This is a serious problem with Algorithm I and is due to catastrophic cancellation in the subtraction of two similar numbers at the final stage of the algorithm.

Higher-order statistics [edit]

Terriberry[11] extends Chan's formulae to calculating the third and fourth central moments, needed for example when estimating skewness and kurtosis:

$$M_{3,X} = M_{3,A} + M_{3,B} + \delta^3\frac{n_A n_B (n_A - n_B)}{n_X^2} + 3\delta\frac{n_A M_{2,B} - n_B M_{2,A}}{n_X}$$
$$M_{4,X} = M_{4,A} + M_{4,B} + \delta^4\frac{n_A n_B\left(n_A^2 - n_A n_B + n_B^2\right)}{n_X^3}$$
$$+ 6\delta^2\frac{n_A^2 M_{2,B} + n_B^2 M_{2,A}}{n_X^2} + 4\delta\frac{n_A M_{3,B} - n_B M_{3,A}}{n_X}$$

Here the M_k are again the sums of powers of differences from the mean $\Sigma(x - \bar x)^k$, giving

skewness: $g_1 = \frac{\sqrt{n} M_3}{M_2^{3/2}},$

kurtosis: $g_2 = \frac{n M_4}{M_2^2} - 3.$

For the incremental case (i.e., $B = \{x\}$), this simplifies to:

$$\delta = x - m$$
$$m' = m + \frac{\delta}{n}$$
$$M_2' = M_2 + \delta^2\frac{n-1}{n}$$
$$M_3' = M_3 + \delta^3\frac{(n-1)(n-2)}{n^2} - \frac{3\delta M_2}{n}$$
$$M_4' = M_4 + \frac{\delta^4(n-1)(n^2-3n+3)}{n^3} + \frac{6\delta^2 M_2}{n^2} - \frac{4\delta M_3}{n}$$

By preserving the value δ/n, only one division operation is needed and the higher-order statistics can thus be calculated for little incremental cost.

An example of the online algorithm for kurtosis implemented as described is:

```
def online_kurtosis(data):
    n = 0
    mean = 0
    M2 = 0
    M3 = 0
    M4 = 0

    for x in data:
        n1 = n
        n = n + 1
        delta = x - mean
        delta_n = delta / n
        delta_n2 = delta_n * delta_n
        term1 = delta * delta_n * n1
        mean = mean + delta_n
        M4 = M4 + term1 * delta_n2 * (n*n - 3*n + 3) + 6 * delta_n2 * M2 - 4 * delta_n * M3
        M3 = M3 + term1 * delta_n * (n - 2) - 3 * delta_n * M2
        M2 = M2 + term1

    kurtosis = (n*M4) / (M2*M2) - 3
    return kurtosis
```

Pébay[12] further extends these results to arbitrary-order central moments, for the incremental and the pairwise cases. One can also find there similar formulas for covariance.

Choi and Sweetman [13] offer two alternative methods to compute the skewness and kurtosis, each of which can save substantial computer memory requirements and CPU time in certain applications. The first approach is to compute the statistical moments by separating the data into bins and then computing the moments from the geometry of the resulting histogram, which effectively becomes a one-pass algorithm for higher moments. One benefit is that the statistical moment calculations can be carried out to arbitrary accuracy such that the computations can be tuned to the precision of, e.g., the data storage format or the original measurement hardware. A relative histogram of a random variable can be constructed in the conventional way: the range of potential values is divided into bins and the number of occurrences within each bin are counted and plotted such that the area of each rectangle equals the portion of the sample values within that bin:

$$H(x_k) = \frac{h(x_k)}{A}$$

where $h(x_k)$ and $H(x_k)$ represent the frequency and the relative frequency at bin x_k and $A = \sum_{k=1}^{K} h(x_k)\,\Delta x_k$ is the total area of the histogram. After this normalization, the n raw moments and central moments of $x(t)$ can be calculated from the relative histogram:

$$m_n^{(h)} = \sum_{k=1}^{K} x_k^n\, H(x_k)\Delta x_k = \frac{1}{A}\sum_{k=1}^{K} x_k^n\, h(x_k)\Delta x_k$$

$$\theta_n^{(h)} = \sum_{k=1}^{K} \left(x_k - m_1^{(h)}\right)^n H(x_k)\Delta x_k = \frac{1}{A}\sum_{k=1}^{K} \left(x_k - m_1^{(h)}\right)^n h(x_k)\Delta x_k$$

where the superscript $^{(h)}$ indicates the moments are calculated from the histogram. For constant bin width $\Delta x_k = \Delta x$ these two expressions can be simplified using $I = A/\Delta x$:

$$m_n^{(h)} = \frac{1}{I}\sum_{k=1}^{K} x_k^n\, h(x_k)$$

$$\theta_n^{(h)} = \frac{1}{I}\sum_{k=1}^{K} \left(x_k - m_1^{(h)}\right)^n h(x_k)$$

The second approach from Choi and Sweetman [13] is an analytical methodology to combine statistical moments from individual segments of a time-history such that the resulting overall moments are those of the complete time-history. This methodology could be used for parallel computation of statistical moments with subsequent combination of those moments, or for combination of statistical moments computed at sequential times.

If Q sets of statistical moments are known: $(\gamma_{0,q}, \mu_q, \sigma_q^2, \alpha_{3,q}, \alpha_{4,q})$ for $q = 1, 2, \ldots, Q$, then each γ_n can be expressed in terms of the equivalent n raw moments:

$$\gamma_{n,q} = m_{n,q}\gamma_{0,q} \qquad \text{for} \quad n = 1,2,3,4 \quad \text{and} \quad q = 1,2,\dots,Q$$

where $\gamma_{0,q}$ is generally taken to be the duration of the q^{th} time-history, or the number of points if Δt is constant.

The benefit of expressing the statistical moments in terms of γ is that the Q sets can be combined by addition, and there is no upper limit on the value of Q.

$$\gamma_{n,c} = \sum_{q=1}^{Q} \gamma_{n,q} \qquad \text{for} \quad n = 0,1,2,3,4$$

where the subscript $_c$ represents the concatenated time-history or combined γ. These combined values of γ can then be inversely transformed into raw moments representing the complete concatenated time-history

$$m_{n,c} = \frac{\gamma_{n,c}}{\gamma_{0,c}} \quad \text{for} \quad n = 1,2,3,4$$

Known relationships between the raw moments (m_n) and the central moments ($\theta_n = E[(x-\mu)^n]$) are then used to compute the central moments of the concatenated time-history. Finally, the statistical moments of the concatenated history are computed from the central moments:

$$\mu_c = m_{1,c} \qquad \sigma^2_c = \theta_{2,c} \qquad \alpha_{3,c} = \frac{\theta_{3,c}}{\sigma^3_c} \qquad \alpha_{4,c} = \frac{\theta_{4,c}}{\sigma^4_c} - 3$$

Covariance [edit]

Very similar algorithms can be used to compute the covariance. The naive algorithm is:

$$\operatorname{Cov}(X,Y) = \frac{\sum_{i=1}^n x_i y_i - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)/n}{n}.$$

For the algorithm above, one could use the following pseudocode:

```
def naive_covariance(data1, data2):
    n = len(data1)
    sum12 = 0
    sum1 = sum(data1)
    sum2 = sum(data2)

    for i in range(n):
        sum12 += data1[i]*data2[i]

    covariance = (sum12 - sum1*sum2 / n) / n
    return covariance
```

As for the variance, the covariance of two random variables is also shift-invariant, so given that K_x and K_y are whatever two constant values it can be written:

$$\operatorname{Cov}(X,Y) = \operatorname{Cov}(X-k_x, Y-k_y) = \frac{\sum_{i=1}^n (x_i - K_x)(y_i - K_y) - (\sum_{i=1}^n (x_i - K_x))(\sum_{i=1}^n (y_i - K_y))/n}{n}.$$

and again choosing a value inside the range of values will stabilize the formula against catastrophic cancellation as well as make it more robust against big sums. Taking the first value of each data set, the algorithm can be written as:

```
def shifted_data_covariance(dataX, dataY):
   n = len(dataX)
   if (n < 2):
     return 0
   Kx = dataX[0]
   Ky = dataY[0]
   Ex = 0
   Ey = 0
   Exy = 0
   for i in range(n):
      Ex += dataX[i] - Kx
      Ey += dataY[i] - Ky
      Exy += (dataX[i] - Kx) * (dataY[i] - Ky)
   return (Exy - Ex * Ey / n) / n
```

The two-pass algorithm first computes the sample means, and then the covariance:

$$\bar{x} = \sum_{i=1}^n x_i/n$$

$$\bar{y} = \sum_{i=1}^n y_i/n$$

$$\operatorname{Cov}(X,Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n}.$$

The two-pass algorithm may be written as:

```
def two_pass_covariance(data1, data2):
    n = len(data1)

    mean1 = sum(data1) / n
    mean2 = sum(data2) / n

    covariance = 0

    for i in range(n):
        a = data1[i] - mean1
        b = data2[i] - mean2
        covariance += a*b / n
    return covariance
```

A slightly more accurate compensated version performs the full naive algorithm on the residuals. The final sums $\sum x_i$ and $\sum y_i$ *should* be zero, but the second pass compensates for any small error.

A slight modification of the online algorithm for computing the variance yields an online algorithm for the covariance:

```
def online_covariance(data1, data2):
    mean1 = mean2 = 0.
    M12 = 0.
    n = len(ls1)
    for i in range(n):
        delta1 = (data1[i] - mean1) / (i + 1)
        mean1 += delta1
        delta2 = (data2[i] - mean2) / (i + 1)
        mean2 += delta2
        M12 += i * delta1 * delta2 - M12 / (i + 1)
    return n / (n - 1.) * M12
```

A stable one-pass algorithm exists, similar to the one above, that computes co-moment $C_n = \sum_{i=1}^{n}(x_i - \bar{x}_n)(y_i - \bar{y}_n)$:

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$
$$\bar{y}_n = \bar{y}_{n-1} + \frac{y_n - \bar{y}_{n-1}}{n}$$
$$C_n = C_{n-1} + (x_n - \bar{x}_n)(y_n - \bar{y}_{n-1}) = C_{n-1} + (y_n - \bar{y}_n)(x_n - \bar{x}_{n-1})$$

The apparent asymmetry in that last equation is due to the fact that $(x_n - \bar{x}_n) = \frac{n-1}{n}(x_n - \bar{x}_{n-1})$, so both update terms are equal to $\frac{n-1}{n}(x_n - \bar{x}_{n-1})(y_n - \bar{y}_{n-1})$. Even greater accuracy can be achieved by first computing the means, then using the stable one-pass algorithm on the residuals.

Thus we can compute the covariance as

$$\begin{aligned}\operatorname{Cov}_N(X,Y) = \frac{C_N}{N} &= \frac{\operatorname{Cov}_{N-1}(X,Y)\cdot(N-1) + (x_n - \bar{x}_n)(y_n - \bar{y}_{n-1})}{N}\\ &= \frac{\operatorname{Cov}_{N-1}(X,Y)\cdot(N-1) + (y_n - \bar{y}_n)(x_n - \bar{x}_{n-1})}{N}\\ &= \frac{\operatorname{Cov}_{N-1}(X,Y)\cdot(N-1) + \frac{N-1}{N}(x_n - \bar{x}_{n-1})(y_n - \bar{y}_{n-1})}{N}.\end{aligned}$$

Likewise, there is a formula for combining the covariances of two sets that can be used to parallelize the computation:

$$C_X = C_A + C_B + (\bar{x}_A - \bar{x}_B)(\bar{y}_A - \bar{y}_B)\cdot\frac{n_A n_B}{n_X}.$$

See also [edit]

- Algebraic formula for the variance
- Kahan summation algorithm

References [edit]

1. ^ a b Bo Einarsson (1 August 2005). *Accuracy and Reliability in Scientific Computing*. SIAM. p. 47. ISBN 978-0-89871-584-2. Retrieved 17 February 2013.
2. ^ a b T.F.Chan, G.H. Golub and R.J. LeVeque (1983). ""Algorithms for computing the sample variance: Analysis and recommendations", The American Statistician, 37" (PDF). pp. 242–247.
3. ^ Higham, Nicholas (2002). *Accuracy and Stability of Numerical Algorithms (2 ed) (Problem 1.10)*. SIAM.
4. ^ a b Chan, Tony F.; Golub, Gene H.; LeVeque, Randall J. (1979), "Updating Formulae and a Pairwise Algorithm for Computing Sample Variances." (PDF), *Technical Report STAN-CS-79-773*, Department of Computer Science, Stanford University.
5. ^ Donald E. Knuth (1998). *The Art of Computer Programming*, volume 2: *Seminumerical Algorithms*, 3rd edn., p. 232. Boston: Addison-Wesley.
6. ^ B. P. Welford (1962)."Note on a method for calculating corrected sums of squares and products". *Technometrics* 4(3):419–420.
7. ^ Chan, Tony F.; Golub, Gene H.; LeVeque, Randall J. (1983). Algorithms for Computing the Sample Variance: Analysis and Recommendations. The American Statistician 37, 242-247. http://www.jstor.org/stable/2683386
8. ^ Ling, Robert F. (1974). Comparison of Several Algorithms for Computing Sample Means and Variances. Journal of the American Statistical Association, Vol. 69, No. 348, 859-866. doi:10.2307/2286154
9. ^ http://www.johndcook.com/standard_deviation.html
10. ^ D. H. D. West (1979). *Communications of the ACM*, 22, 9, 532-535: *Updating Mean and Variance Estimates: An Improved Method*
11. ^ Terriberry, Timothy B. (2007), *Computing Higher-Order Moments Online*
12. ^ Pébay, Philippe (2008), "Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments" (PDF), *Technical Report SAND2008-6212*, Sandia National Laboratories
13. ^ a b Choi, Muenkeun; Sweetman, Bert (2010), *Efficient Calculation of Statistical Moments for Structural Health Monitoring* (PDF)

External links [edit]

- Weisstein, Eric W., "Sample Variance Computation", *MathWorld*.

Categories: Statistical algorithms | Statistical deviation and dispersion