Package 'fromo'

August 29, 2016

Type Package

Maintainer Steven E. Pav <shabbychef@gmail.com>

Version 0.1.3

Date 2016-04-04

License LGPL-3

Title Fast Robust Moments

BugReports https://github.com/shabbychef/fromo/issues

Description Fast computation of moments via 'Rcpp'. Supports computation on vectors and matrices, and Monoidal append of moments.

Imports Rcpp (>= 0.12.3), methods

LinkingTo Rcpp

Suggests testthat, moments, PDQutils, microbenchmark

RoxygenNote 5.0.1

URL https://github.com/shabbychef/fromo

Collate 'fromo.r' 'RcppExports.R' 'zzz_centsums.R'

NeedsCompilation yes

Author Steven E. Pav [aut, cre]

Repository CRAN

Date/Publication 2016-04-05 00:08:03

R topics documented:

`fromo-package` *fast robust moments*

Description

Fast Robust Moments.

Robust Moments

Welford described a method for 'robust' one-pass computation of the standard deviation. By 'robust', we mean robust to round-off caused by a large shift in the mean. This method was generalized by Terriberry, and Bennett *et. al.* to the case of higher-order moments. This package provides those algorithms for computing moments.

Generally we should find that the stock implementations of `sd`, `skewness` and so on are *already* robust and likely using these algorithms under the hood. This package was written for a few reasons:

1. As an exercise to learn Rcpp.
2. Often I found I needed the first k moments. For example, when computing the Z-score, the standard deviation and mean must be computed separately, which is inefficient. Similarly Merten's correction for the standard error of the Sharpe ratio uses the first four moments. These are all computed as a side effect of computation of the kurtosis, but discarded by the standard methods.

Legal Mumbo Jumbo

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

This package was developed as an exercise in learning Rcpp.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

---

`accessor` *Accessor methods.*

Description

Access slot data from a `centsums` object.

Usage

```
sums(x)

## S4 method for signature 'centsums'
sums(x)

moments(x, type = c("central", "raw", "standardized"))

## S4 method for signature 'centsums'
moments(x, type = c("central", "raw", "standardized"))
```

Arguments

`x`	a `centsums` object.
`type`	the type of moment to compute.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

`as.centcosums` *Coerce to a centcosums object.*

Description

Convert data to a `centcosums` object.

Usage

```
as.centcosums(x, order=2, na.omit=TRUE)

## Default S3 method:
as.centcosums(x, order = 2, na.omit = TRUE)
```

Arguments

`x`	a matrix.
`order`	the order, defaulting to `2`.
`na.omit`	whether to remove rows with `NA`.

Details

Computes the raw cosums on data, and stuffs the results into a `centcosums` object.

Value

A centcosums object.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

Examples

```
set.seed(123)
x <- matrix(rnorm(100*3),ncol=3)
cs <- as.centcosums(x, order=2)
```

`as.centsums` *Coerce to a centsums object.*

Description

Convert data to a `centsums` object.

Usage

```
as.centsums(x, order=3, na.rm=TRUE)

## Default S3 method:
as.centsums(x, order = 3, na.rm = TRUE)
```

Arguments

`x`	a numeric, array, or matrix.
`order`	the order, defaulting to `length(sums)+1`.
`na.rm`	whether to remove `NA`.

Details

Computes the raw sums on data, and stuffs the results into a `centsums` object.

Value

A centsums object.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

Examples

```
set.seed(123)
x <- rnorm(1000)
cs <- as.centsums(x, order=5)
```

c.centcosums *concatenate centcosums objects.*

Description

Concatenate centcosums objects.

Usage

```
\method{c}{centcosums}(...)
```

Arguments

...	centcosums objects

See Also

join_cent_cosums

c.centsums *concatenate centsums objects.*

Description

Concatenate centsums objects.

Usage

```
\method{c}{centsums}(...)
```

Arguments

...	centsums objects

See Also

join_cent_sums

cent2raw *Convert between different types of moments, raw, central, standardized.*

Description

Given raw or central or standardized moments, convert to another type.

Usage

```
cent2raw(input)
```

Arguments

input	a vector of the count, then the mean, then the 2 through k raw or central moments.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

centcosums-accessor *Accessor methods.*

Description

Access slot data from a centcosums object.

Usage

```
cosums(x)

## S4 method for signature 'centcosums'
cosums(x)

comoments(x, type = c("central", "raw"))

## S4 method for signature 'centcosums'
comoments(x, type = c("central", "raw"))
```

Arguments

x	a centcosums object.
type	the type of moment to compute.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

---

centcosums-class *centcosums Class.*

Description

An S4 class to store (centered) cosums of data, and to support operations on the same.

Usage

```
## S4 method for signature 'centcosums'
initialize(.Object, cosums, order = NA_real_)

centcosums(cosums, order = NULL)
```

Arguments

.Object	a centcosums object, or proto-object.
cosums	the output of cent_cosums, say.
order	the order, defaulting to 2.

Details

A centcosums object contains a multidimensional array (now only 2-diemnsional), as output by cent_cosums.

Value

An object of class centcosums.

Slots

cosums a multidimensional array of the cosums.

order the maximum order. ignored for now.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

See Also

cent_cosums

Examples

```
obj <- new("centcosums",cosums=cent_cosums(matrix(rnorm(100*3),ncol=3),max_order=2),order=2)
```

---

centsums-class *centsums Class.*

Description

An S4 class to store (centered) sums of data, and to support operations on the same.

Usage

```
## S4 method for signature 'centsums'
initialize(.Object, sums, order = NA_real_)

centsums(sums, order = NULL)
```

Arguments

.Object	a centsums object, or proto-object.
sums	a numeric vector.
order	the order, defaulting to length(sums)+1.

Details

A centsums object contains a vector value of the data count, the mean, and the kth centered sum, for k up to some maximum order.

Value

An object of class centsums.

Slots

sums a numeric vector of the sums.

order the maximum order.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

Examples

```
obj <- new("centsums",sums=c(1000,1.234,0.235),order=2)
```

cent_cosums *Multivariate centered sums; join and unjoined.*

Description

Compute, join, or unjoin multivariate centered (co-) sums.

Usage

```
cent_cosums(v, max_order = 2L, na_omit = FALSE)

cent_comoments(v, max_order = 2L, used_df = 0L, na_omit = FALSE)

join_cent_cosums(ret1, ret2)

unjoin_cent_cosums(ret3, ret2)
```

Arguments

v	an m by n matrix, each row an independent observation of some n variate variable.
max_order	the maximum order of cosum to compute. For now this can only be 2; in the future higher order cosums should be possible.
na_omit	a boolean; if TRUE, then only rows of v with complete observations will be used.
used_df	the number of degrees of freedom consumed, used in the denominator of the centered moments computation. These are subtracted from the number of observations.
ret1	a multdimensional array as output by cent_cosums.
ret2	a multdimensional array as output by cent_cosums.
ret3	a multdimensional array as output by cent_cosums.

Value

a multidimensional arry of dimension max_order, each side of length $1 + n$. For the case currently implemented where max_order must be 2, the output is a symmetric matrix, where the element in the 1,1 position is the count of complete) rows of v, the 2:(n+1),1 column is the mean, and the 2:(n+1),2:(n+1) is the co *sums* matrix, which is the covariance up to scaling by the count. cent_comoments performs this normalization for you.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

See Also

cent_sums

Examples

```
set.seed(1234)
x1 <- matrix(rnorm(1e3*5,mean=1),ncol=5)
x2 <- matrix(rnorm(1e3*5,mean=1),ncol=5)
max_ord <- 2L
rs1 <- cent_cosums(x1,max_ord)
rs2 <- cent_cosums(x2,max_ord)
rs3 <- cent_cosums(rbind(x1,x2),max_ord)
rs3alt <- join_cent_cosums(rs1,rs2)
stopifnot(max(abs(rs3 - rs3alt)) < 1e-7)
rs1alt <- unjoin_cent_cosums(rs3,rs2)
rs2alt <- unjoin_cent_cosums(rs3,rs1)
stopifnot(max(abs(rs1 - rs1alt)) < 1e-7)
stopifnot(max(abs(rs2 - rs2alt)) < 1e-7)
```

---

cent_sums *Centered sums; join and unjoined.*

Description

Compute, join, or unjoin centered sums.

Usage

```
cent_sums(v, max_order = 5L, na_rm = FALSE)

join_cent_sums(ret1, ret2)

unjoin_cent_sums(ret3, ret2)
```

Arguments

v	a vector
max_order	the maximum order of the centered moment to be computed.
na_rm	whether to remove NA, false by default.
ret1	an $ord + 1$ vector as output by cent_sums consisting of the count, the mean, then the k through ordth centered sum of some observations.
ret2	an $ord + 1$ vector as output by cent_sums consisting of the count, the mean, then the k through ordth centered sum of some observations.
ret3	an $ord + 1$ vector as output by cent_sums consisting of the count, the mean, then the k through ordth centered sum of some observations.

Value

a vector the same size as the input consisting of the adjusted version of the input. When there are not sufficient (non-nan) elements for the computation, NaN are returned.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

Examples

```
set.seed(1234)
x1 <- rnorm(1e3,mean=1)
x2 <- rnorm(1e3,mean=1)
max_ord <- 6L
rs1 <- cent_sums(x1,max_ord)
rs2 <- cent_sums(x2,max_ord)
rs3 <- cent_sums(c(x1,x2),max_ord)
rs3alt <- join_cent_sums(rs1,rs2)
stopifnot(max(abs(rs3 - rs3alt)) < 1e-7)
rs1alt <- unjoin_cent_sums(rs3,rs2)
rs2alt <- unjoin_cent_sums(rs3,rs1)
stopifnot(max(abs(rs1 - rs1alt)) < 1e-7)
stopifnot(max(abs(rs2 - rs2alt)) < 1e-7)
```

---

fromo-NEWS *News for package 'fromo':*

Description

News for package 'fromo'

Version 0.1.3 (2016-04-04)

- submit to CRAN

Initial Version 0.1.0 (2016-03-25)

- start work

---

running_apx_quantiles *Compute approximate quantiles over a sliding window*

Description

Computes cumulants up to some given order, then employs the Cornish-Fisher approximation to compute approximate quantiles using a Gaussian basis.

Usage

```
running_apx_quantiles(v, p, window = NULL, max_order = 5L, na_rm = FALSE,
  min_df = 0L, used_df = 0L, restart_period = 100L)

running_apx_median(v, window = NULL, max_order = 5L, na_rm = FALSE,
  min_df = 0L, used_df = 0L, restart_period = 100L)
```

Arguments

v	a vector
p	the probability points at which to compute the quantiles. Should be in the range (0,1).
window	the window size. if given as finite integer or double, passed through. If NULL, NA_integer_, NA_real_ or Inf are given, equivalent to an infinite window size. If negative, an error will be thrown.
max_order	the maximum order of the centered moment to be computed.
na_rm	whether to remove NA, false by default.
min_df	the minimum df to return a value, otherwise NaN is returned. This can be used to prevent moments from being computed on too few observations. Defaults to zero, meaning no restriction.
used_df	the number of degrees of freedom consumed, used in the denominator of the centered moments computation. These are subtracted from the number of observations.
restart_period	the recompute period. because subtraction of elements can cause loss of precision, the computation of moments is restarted periodically based on this parameter. Larger values mean fewer restarts and faster, though less accurate results. Note that the code checks for negative second and fourth moments and recomputes when needed.

Details

Computes the cumulants, then approximates quantiles using AS269 of Lee & Lin.

Value

A matrix, with one row for each element of x, and one column for each element of q.

Note

The current implementation is not as space-efficient as it could be, as it first computes the cumulants for each row, then performs the Cornish-Fisher approximation on a row-by-row basis. In the future, this computation may be moved earlier into the pipeline to be more space efficient. File an issue if the memory footprint is an issue for you.

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Lee, Y-S., and Lin, T-K. "Algorithm AS269: High Order Cornish Fisher Expansion." Appl. Stat. 41, no. 1 (1992): 233-240. http://www.jstor.org/stable/2347649

Lee, Y-S., and Lin, T-K. "Correction to Algorithm AS269: High Order Cornish Fisher Expansion." Appl. Stat. 42, no. 1 (1993): 268-269. http://www.jstor.org/stable/2347433

AS 269. https://web.archive.org/web/20110103030111/http://lib.stat.cmu.edu/apstat/269

Jaschke, Stefan R. "The Cornish-Fisher-expansion in the context of Delta-Gamma-normal approximations." No. 2001, 54. Discussion Papers, Interdisciplinary Research Project 373: Quantification and Simulation of Economic Processes, 2001. http://www.jaschke-net.de/papers/CoFi.pdf

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb6395

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

See Also

running_cumulants, PDQutils::qapx_cf, PDQutils::AS269.

Examples

```
x <- rnorm(1e5)
xq <- running_apx_quantiles(x,c(0.1,0.25,0.5,0.75,0.9))
xm <- running_apx_median(x)
```

---

running_centered *Compare data to moments computed over a sliding window.*

Description

Computes moments over a sliding window, then adjusts the data accordingly, centering, or scaling, or z-scoring, and so on.

Usage

```
running_centered(v, window = NULL, na_rm = FALSE, min_df = 0L,
  lookahead = 0L, restart_period = 100L)

running_scaled(v, window = NULL, na_rm = FALSE, min_df = 0L,
  lookahead = 0L, restart_period = 100L)

running_zscored(v, window = NULL, na_rm = FALSE, min_df = 0L,
  lookahead = 0L, restart_period = 100L)

running_sharpe(v, window = NULL, na_rm = FALSE, compute_se = FALSE,
  min_df = 0L, restart_period = 100L)

running_tstat(v, window = NULL, na_rm = FALSE, min_df = 0L,
  restart_period = 100L)
```

Arguments

v	a vector
window	the window size. if given as finite integer or double, passed through. If NULL, NA_integer_, NA_real_ or Inf are given, equivalent to an infinite window size. If negative, an error will be thrown.
na_rm	whether to remove NA, false by default.
min_df	the minimum df to return a value, otherwise NaN is returned. This can be used to prevent *e.g.* Z-scores from being computed on only 3 observations. Defaults to zero, meaning no restriction, which can result in infinite Z-scores during the burn-in period.
lookahead	for some of the operations, the value is compared to mean and standard deviation possibly using 'future' or 'past' information by means of a non-zero lookahead. Positive values mean data are taken from the future.
restart_period	the recompute period. because subtraction of elements can cause loss of precision, the computation of moments is restarted periodically based on this parameter. Larger values mean fewer restarts and faster, though less accurate results. Note that the code checks for negative second and fourth moments and recomputes when needed.
compute_se	for running_sharpe, return an extra column of the standard error, as computed by Mertens' correction.

Details

Given the length n vector x, for a given index i, define $x^{(i)}$ as the vector of $x_{i-window+1}, x_{i-window+2}, ..., x_i$, where we do not run over the 'edge' of the vector. In code, this is essentially `x[(max(1,i-window+1)):i]`. Then define μ_i, σ_i and n_i as, respectively, the sample mean, standard deviation and number of non-NA elements in $x^{(i)}$.

We compute output vector m the same size as x. For the 'centered' version of x, we have $m_i = x_i - \mu_i$. For the 'scaled' version of x, we have $m_i = x_i/\sigma_i$. For the 'z-scored' version of x, we have $m_i = (x_i - \mu_i)/\sigma_i$. For the 't-scored' version of x, we have $m_i = \sqrt{n_i}\mu_i/\sigma_i$.

We also allow a 'lookahead' for some of these operations. If positive, the moments are computed using data from larger indices; if negative, from smaller indices. Letting $j = i+lookahead$: For the 'centered' version of x, we have $m_i = x_i - \mu_j$. For the 'scaled' version of x, we have $m_i = x_i/\sigma_j$. For the 'z-scored' version of x, we have $m_i = (x_i - \mu_j)/\sigma_j$.

Value

a vector the same size as the input consisting of the adjusted version of the input. When there are not sufficient (non-nan) elements for the computation, NaN are returned.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

See Also

scale

Examples

```
if (require(moments)) {
    set.seed(123)
    x <- rnorm(5e1)
    window <- 10L
    rm1 <- t(sapply(seq_len(length(x)),function(iii) {
                  xrang <- x[max(1,iii-window+1):iii]
                  c(sd(xrang),mean(xrang),length(xrang)) },
                  simplify=TRUE))
    rcent <- running_centered(x,window=window)
    rscal <- running_scaled(x,window=window)
    rzsco <- running_zscored(x,window=window)
```

```
    rshrp <- running_sharpe(x,window=window)
    rtsco <- running_tstat(x,window=window)
    rsrse <- running_sharpe(x,window=window,compute_se=TRUE)
    stopifnot(max(abs(rcent - (x - rm1[,2])),na.rm=TRUE) < 1e-12)
    stopifnot(max(abs(rscal - (x / rm1[,1])),na.rm=TRUE) < 1e-12)
    stopifnot(max(abs(rzsco - ((x - rm1[,2]) / rm1[,1])),na.rm=TRUE) < 1e-12)
    stopifnot(max(abs(rshrp - (rm1[,2] / rm1[,1])),na.rm=TRUE) < 1e-12)
    stopifnot(max(abs(rtsco - ((sqrt(rm1[,3]) * rm1[,2]) / rm1[,1])),na.rm=TRUE) < 1e-12)
    stopifnot(max(abs(rsrse[,1] - rshrp),na.rm=TRUE) < 1e-12)

    rm2 <- t(sapply(seq_len(length(x)),function(iii) {
                  xrang <- x[max(1,iii-window+1):iii]
                  c(kurtosis(xrang)-3.0,skewness(xrang)) },
                  simplify=TRUE))
    mertens_se <- sqrt((1 + ((2 + rm2[,1])/4) * rshrp^2 - rm2[,2]*rshrp) / rm1[,3])
    stopifnot(max(abs(rsrse[,2] - mertens_se),na.rm=TRUE) < 1e-12)
}
```

`running_sd3` *Compute first K moments over a sliding window*

Description

Compute the (standardized) 2nd through kth moments, the mean, and the number of elements over an infinite or finite sliding window, returning a matrix.

Usage

```
running_sd3(v, window = NULL, na_rm = FALSE, min_df = 0L, used_df = 1L,
  restart_period = 100L)

running_skew4(v, window = NULL, na_rm = FALSE, min_df = 0L,
  used_df = 1L, restart_period = 100L)

running_kurt5(v, window = NULL, na_rm = FALSE, min_df = 0L,
  used_df = 1L, restart_period = 100L)

running_cent_moments(v, window = NULL, max_order = 5L, na_rm = FALSE,
  max_order_only = FALSE, min_df = 0L, used_df = 0L,
  restart_period = 100L)

running_std_moments(v, window = NULL, max_order = 5L, na_rm = FALSE,
  min_df = 0L, used_df = 0L, restart_period = 100L)

running_cumulants(v, window = NULL, max_order = 5L, na_rm = FALSE,
  min_df = 0L, used_df = 0L, restart_period = 100L)
```

Arguments

v	a vector
window	the window size. if given as finite integer or double, passed through. If NULL, NA_integer_, NA_real_ or Inf are given, equivalent to an infinite window size. If negative, an error will be thrown.
na_rm	whether to remove NA, false by default.
min_df	the minimum df to return a value, otherwise NaN is returned. This can be used to prevent moments from being computed on too few observations. Defaults to zero, meaning no restriction.
used_df	the number of degrees of freedom consumed, used in the denominator of the centered moments computation. These are subtracted from the number of observations.
restart_period	the recompute period. because subtraction of elements can cause loss of precision, the computation of moments is restarted periodically based on this parameter. Larger values mean fewer restarts and faster, though less accurate results. Note that the code checks for negative second and fourth moments and recomputes when needed.
max_order	the maximum order of the centered moment to be computed.
max_order_only	for running_cent_moments, if this flag is set, only compute the maximum order centered moment, and return in a vector.

Details

Computes the number of elements, the mean, and the 2nd through kth centered (and typically standardized) moments, for $k = 2, 3, 4$. These are computed via the numerically robust one-pass method of Bennett *et. al.*

Given the length n vector x, we output matrix M where $M_{i,j}$ is the $order-j+1$ moment (*i.e.* excess kurtosis, skewness, standard deviation, mean or number of elements) of $x_{i-window+1}, x_{i-window+2}, ..., x_i$. Barring NA or NaN, this is over a window of size window. During the 'burn-in' phase, we take fewer elements.

Value

Typically a matrix, where the first columns are the kth, k-1th through 2nd standardized, centered moments, then a column of the mean, then a column of the number of (non-nan) elements in the input, with the following exceptions:

running_cent_moments Computes arbitrary order centered moments. When max_order_only is set, only a column of the maximum order centered moment is returned.

running_std_moments Computes arbitrary order standardized moments, then the standard deviation, the mean, and the count. There is not yet an option for max_order_only, but probably should be.

running_cumulants Computes arbitrary order cumulants, and returns the kth, k-1th, through the second (which is the variance) cumulant, then the mean, and the count.

Note

the kurtosis is *excess kurtosis*, with a 3 subtracted, and should be nearly zero for Gaussian input.

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

Examples

```
x <- rnorm(1e5)
xs3 <- running_sd3(x,10)
xs4 <- running_skew4(x,10)

if (require(moments)) {
    set.seed(123)
    x <- rnorm(5e1)
    window <- 10L
    kt5 <- running_kurt5(x,window=window)
    rm1 <- t(sapply(seq_len(length(x)),function(iii) {
                xrang <- x[max(1,iii-window+1):iii]
                c(moments::kurtosis(xrang)-3.0,moments::skewness(xrang),
                sd(xrang),mean(xrang),length(xrang)) },
             simplify=TRUE))
    stopifnot(max(abs(kt5 - rm1),na.rm=TRUE) < 1e-12)
}

xc6 <- running_cent_moments(x,window=100L,max_order=6L)
```

sd3 *Compute first K moments*

Description

Compute the (standardized) 2nd through kth moments, the mean, and the number of elements.

Usage

```
sd3(v, na_rm = FALSE)

skew4(v, na_rm = FALSE)

kurt5(v, na_rm = FALSE)

cent_moments(v, max_order = 5L, used_df = 0L, na_rm = FALSE)

std_moments(v, max_order = 5L, used_df = 0L, na_rm = FALSE)

cent_cumulants(v, max_order = 5L, used_df = 0L, na_rm = FALSE)
```

Arguments

v	a vector
na_rm	whether to remove NA, false by default.
max_order	the maximum order of the centered moment to be computed.
used_df	the number of degrees of freedom consumed, used in the denominator of the centered moments computation. These are subtracted from the number of observations.

Details

Computes the number of elements, the mean, and the 2nd through kth centered standardized moment, for $k = 2, 3, 4$. These are computed via the numerically robust one-pass method of Bennett *et. al.* In general they will *not* match exactly with the 'standard' implementations, due to differences in roundoff.

These methods are reasonably fast, on par with the 'standard' implementations. However, they will usually be faster than calling the various standard implementations more than once.

Value

a vector, filled out as follows:

sd3 A vector of the (sample) standard devation, mean, and number of elements.

skew4 A vector of the (sample) skewness, standard devation, mean, and number of elements.

kurt5 A vector of the (sample) excess kurtosis, skewness, standard devation, mean, and number of elements.

cent_moments A vector of the (sample) kth centered moment, then $k-1$th centered moment, ..., then the *variance*, the mean, and number of elements.

std_moments A vector of the (sample) kth standardized (and centered) moment, then $k-1$th, ..., then standard devation, mean, and number of elements.

cent_cumulants A vector of the (sample) kth (centered, but this is redundant) cumulant, then the $k-1$th, ..., then the *variance* (which is the second cumulant), the mean, and number of elements.

Note

The first centered (and standardized) moment is often defined to be identically 0. Instead `cent_moments` and `std_moments` returns the mean. Similarly, the second standardized moments defined to be identically 1; `std_moments` instead returns the standard deviation. The reason is that a user can always decide to ignore the results and fill in a 0 or 1 as they need, but could not efficiently compute the mean and standard deviation from scratch if we discard it.

The last minus two element of the output of `cent_moments` and `cent_cumulants` is the *variance*, not the standard deviation. All other code return the standard deviation in that place.

The kurtosis is *excess kurtosis*, with a 3 subtracted, and should be nearly zero for Gaussian input.

'centered cumulants' is redundant. The intent was to avoid collision with existing code named 'cumulants'.

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

References

Terriberry, T. "Computing Higher-Order Moments Online." http://people.xiph.org/~tterribe/notes/homs.html

J. Bennett, et. al., "Numerically Stable, Single-Pass, Parallel Statistics Algorithms," Proceedings of IEEE International Conference on Cluster Computing, 2009. https://www.semanticscholar.org/paper/Numerically-stable-single-pass-parallel-statistics-Bennett-Grout/a83ed72a5ba86622d5eb639

Cook, J. D. "Accurately computing running variance." http://www.johndcook.com/standard_deviation.html

Cook, J. D. "Comparing three methods of computing standard deviation." http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation

Examples

```
x <- rnorm(1e5)
sd3(x)[1] - sd(x)
skew4(x)[4] - length(x)
skew4(x)[3] - mean(x)
skew4(x)[2] - sd(x)
if (require(moments)) {
  skew4(x)[1] - skewness(x)
}
# check 'robustness'; only the mean should change:
kurt5(x + 1e12) - kurt5(x)
# check speed
if (require(microbenchmark) && require(moments)) {
  dumbk <- function(x) { c(kurtosis(x) - 3.0,skewness(x),sd(x),mean(x),length(x)) }
  set.seed(1234)
  x <- rnorm(1e6)
  print(kurt5(x) - dumbk(x))
  microbenchmark(dumbk(x),kurt5(x),times=10L)
}
y <- std_moments(x,6)
cml <- cent_cumulants(x,6)
```

---

show *Show a centsums object.*

Description

Displays the centsums object.

Usage

```
show(object)

## S4 method for signature 'centsums'
show(object)
```

Arguments

object	a centsums object.

Note

The moment computations provided by fromo are numerically robust, but will often *not* provide the same results as the 'standard' implementations, due to differences in roundoff. We make every attempt to balance speed and robustness. User assumes all risk from using the fromo package.

Author(s)

Steven E. Pav <shabbychef@gmail.com>

Examples

```
set.seed(123)
x <- rnorm(1000)
obj <- as.centsums(x, order=5)
obj
```

%-%,centcosums,centcosums-method *unconcatenate centcosums objects.*

Description

Unconcatenate centcosums objects.

Usage

```
## S4 method for signature 'centcosums,centcosums'
x %-% y
```

Arguments

x	a centcosums objects
y	a centcosums objects

See Also

unjoin_cent_cosums

%-% *unconcatenate centsums objects.*

Description

Unconcatenate centsums objects.

Usage

```
x %-% y

## S4 method for signature 'centsums,centsums'
x %-% y
```

Arguments

x	a centsums objects
y	a centsums objects

See Also

unjoin_cent_sums

Index