Package 'matrixStats'

March 13, 2020

Version 0.56.0

Depends R (>= 2.12.0)

Suggests base64enc, ggplot2, knitr, microbenchmark, R.devices, R.rsp

VignetteBuilder R.rsp

Title Functions that Apply to Rows and Columns of Matrices (and to Vectors)

Author Henrik Bengtsson [aut, cre, cph],
Hector Corrada Bravo [ctb],
Robert Gentleman [ctb],
Ola Hossjer [ctb],
Harris Jaffee [ctb],
Dongcan Jiang [ctb],
Peter Langfelder [ctb],
Peter Hickey [ctb],
Brian Montgomery [ctb]

Maintainer Henrik Bengtsson <henrikb@braju.com>

Description High-performing functions operating on rows and columns of matrices, e.g. col / rowMedians(), col / rowRanks(), and col / rowSds(). Functions optimized per data type and for subsetted calculations such that both memory usage and processing time is minimized. There are also optimized vector-based methods, e.g. binMeans(), madDiff() and weightedMedian().

License Artistic-2.0

LazyLoad TRUE

NeedsCompilation yes

ByteCompile TRUE

URL https://github.com/HenrikBengtsson/matrixStats

BugReports https://github.com/HenrikBengtsson/matrixStats/issues

RoxygenNote 7.1.0

Repository CRAN

Date/Publication 2020-03-13 10:00:15 UTC

R topics documented:

`matrixStats-package` *Package matrixStats*

Description

High-performing functions operating on rows and columns of matrices, e.g. col / rowMedians(), col / rowRanks(), and col / rowSds(). Functions optimized per data type and for subsetted calculations such that both memory usage and processing time is minimized. There are also optimized vector-based methods, e.g. binMeans(), madDiff() and weightedMedian().

How to cite this package

Henrik Bengtsson (2017). matrixStats: Functions that Apply to Rows and Columns of Matrices (and to Vectors). R package version 0.52.2. https://github.com/HenrikBengtsson/matrixStats

Author(s)

Henrik Bengtsson, Hector Corrada Bravo, Robert Gentleman, Ola Hossjer, Harris Jaffee, Dongcan Jiang, Peter Langfelder

---

anyMissing *Checks if there are any missing values in an object or not*

Description

Checks if there are any missing values in an object or not. *Please use* `base::anyNA()` *instead of* `anyMissing()`, `colAnyNAs()` *instead of* `colAnyMissings()`, *and* `rowAnyNAs()` *instead of* `rowAnyMissings()`.

Usage

```
anyMissing(x, idxs = NULL, ...)

colAnyMissings(x, rows = NULL, cols = NULL, ...)

rowAnyMissings(x, rows = NULL, cols = NULL, ...)

colAnyNAs(x, rows = NULL, cols = NULL, ...)

rowAnyNAs(x, rows = NULL, cols = NULL, ...)
```

Arguments

`x`	A `vector`, a `list`, a `matrix`, a `data.frame`, or `NULL`.
`idxs, rows, cols`	A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done.
`...`	Not used.

Details

The implementation of this method is optimized for both speed and memory. The method will return `TRUE` as soon as a missing value is detected.

Value

Returns `TRUE` if a missing value was detected, otherwise `FALSE`.

Author(s)

Henrik Bengtsson

See Also

Starting with R v3.1.0, there is `anyNA()` in the **base**, which provides the same functionality as `anyMissing()`.

Examples

```
x <- rnorm(n = 1000)
x[seq(300, length(x), by = 100)] <- NA
stopifnot(anyMissing(x) == any(is.na(x)))
```

---

binCounts *Fast element counting in non-overlapping bins*

Description

Counts the number of elements in non-overlapping bins

Usage

```
binCounts(x, idxs = NULL, bx, right = FALSE, ...)
```

Arguments

`x`	A `numeric vector` of K positions for to be binned and counted.
`idxs`	A `vector` indicating subset of elements to operate over. If `NULL`, no subsetting is done.
`bx`	A `numeric vector` of B + 1 ordered positions specifying the B > 0 bins `[bx[1],bx[2])`, `[bx[2],bx[3])`, ..., `[bx[B],bx[B + 1])`.
`right`	If `TRUE`, the bins are right-closed (left open), otherwise left-closed (right open).
`...`	Not used.

Details

`binCounts(x,bx,right = TRUE)` gives equivalent results as `rev(binCounts(-x,bx = rev(-bx),right = FALSE))`, but is faster and more memory efficient.

Value

Returns an `integer vector` of length B with non-negative integers.

Missing and non-finite values

Missing values in x are ignored/dropped. Missing values in bx are not allowed and gives an error.

Author(s)

Henrik Bengtsson

See Also

An alternative for counting occurrences within bins is hist, e.g. hist(x,breaks=bx,plot=FALSE)$counts. That approach is ~30-60% slower than binCounts(...,right=TRUE).

To count occurrences of indices x (positive integers) in [1,B], use tabulate(x,nbins=B), where x does *not* have to be sorted first. For details, see tabulate().

To average values within bins, see binMeans().

---

binMeans *Fast mean calculations in non-overlapping bins*

Description

Computes the sample means in non-overlapping bins

Usage

```
binMeans(y, x, idxs = NULL, bx, na.rm = TRUE, count = TRUE, right = FALSE, ...)
```

Arguments

y	A numeric or logical vector of K values to calculate means on.
x	A numeric vector of K positions for to be binned.
idxs	A vector indicating subset of elements to operate over. If NULL, no subsetting is done.
bx	A numeric vector of B + 1 ordered positions specifying the B > 0 bins [bx[1],bx[2]), [bx[2],bx[3]), ..., [bx[B],bx[B + 1]).
na.rm	If TRUE, missing values in y are dropped before calculating the mean, otherwise not.
count	If TRUE, the number of data points in each bins is returned as attribute count, which is an integer vector of length B.
right	If TRUE, the bins are right-closed (left open), otherwise left-closed (right open).
...	Not used.

Details

binMeans(x,bx,right=TRUE) gives equivalent results as rev(binMeans(-x,bx=sort(-bx),right=FALSE)), but is faster.

Value

Returns a numeric vector of length B.

Missing and non-finite values

Data points where either of y and x is missing are dropped (and therefore are also not counted). Non-finite values in y are not allowed and gives an error. Missing values in bx are not allowed and gives an error.

Author(s)

Henrik Bengtsson with initial code contributions by Martin Morgan [1].

References

[1] R-devel thread *Fastest non-overlapping binning mean function out there?* on Oct 3, 2012

See Also

binCounts(). aggregate and mean().

Examples

```
x <- 1:200
mu <- double(length(x))
mu[1:50] <- 5
mu[101:150] <- -5
y <- mu + rnorm(length(x))

# Binning
bx <- c(0, 50, 100, 150, 200) + 0.5
y_s <- binMeans(y, x = x, bx = bx)

plot(x, y)
for (kk in seq_along(y_s)) {
  lines(bx[c(kk, kk + 1)], y_s[c(kk, kk)], col = "blue", lwd = 2)
}
```

---

indexByRow *Translates matrix indices by rows into indices by columns*

Description

Translates matrix indices by rows into indices by columns.

Usage

```
indexByRow(dim, idxs = NULL, ...)
```

Arguments

dim	A numeric vector of length two specifying the length of the "template" matrix.
idxs	A vector of indices. If NULL, all indices are returned.
...	Not use.

Value

Returns an integer vector of indices.

Known limitations

The current implementation does not support long-vector indices, because both input and output indices are of type integers. This means that the indices in argument idxs can only be in range [1,2^31-1]. Using a greater value will be coerced to NA_integer_. Moreover, returned indices can only be in the same range [1,2^31-1].

Author(s)

Henrik Bengtsson

Examples

```
dim <- c(5, 4)
X <- matrix(NA_integer_, nrow = dim[1], ncol = dim[2])
Y <- t(X)
idxs <- seq_along(X)

# Assign by columns
X[idxs] <- idxs
print(X)

# Assign by rows
Y[indexByRow(dim(Y), idxs)] <- idxs
print(Y)

stopifnot(X == t(Y))
```

---

logSumExp *Accurately computes the logarithm of the sum of exponentials*

Description

Accurately computes the logarithm of the sum of exponentials, that is, $log(sum(exp(lx)))$. If $lx = log(x)$, then this is equivalently to calculating $log(sum(x))$.

Usage

```
logSumExp(lx, idxs = NULL, na.rm = FALSE, ...)
```

Arguments

lx	A numeric vector. Typically lx are $log(x)$ values.
idxs	A vector indicating subset of elements to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, any missing values are ignored, otherwise not.
...	Not used.

Details

This function, which avoid numerical underflow, is often used when computing the logarithm of the sum of small numbers ($|x| << 1$) such as probabilities.

This is function is more accurate than `log(sum(exp(lx)))` when the values of $x = exp(lx)$ are $|x| << 1$. The implementation of this function is based on the observation that

$$log(a+b) = [la = log(a), lb = log(b)] = log(exp(la) + exp(lb)) = la + log(1 + exp(lb - la))$$

Assuming $la > lb$, then $|lb - la| < |lb|$, and it is less likely that the computation of $1 + exp(lb - la)$ will not underflow/overflow numerically. Because of this, the overall result from this function should be more accurate. Analogously to this, the implementation of this function finds the maximum value of lx and subtracts it from the remaining values in lx.

Value

Returns a numeric scalar.

Benchmarking

This method is optimized for correctness, that avoiding underflowing. It is implemented in native code that is optimized for speed and memory.

Author(s)

Henrik Bengtsson

References

[1] R Core Team, *Writing R Extensions*, v3.0.0, April 2013.
[2] Laurent El Ghaoui, *Hyper-Textbook: Optimization Models and Applications*, University of California at Berkeley, August 2012. (Chapter 'Log-Sum-Exp (LSE) Function and Properties')
[3] R-help thread *logsumexp function in R*, 2011-02-17. https://stat.ethz.ch/pipermail/r-help/2011-February/269205.html

See Also

To compute this function on rows or columns of a matrix, see rowLogSumExps().

For adding *two* double values in native code, R provides the C function `logspace_add()` [1]. For properties of the log-sum-exponential function, see [2].

Examples

```
## EXAMPLE #1
lx <- c(1000.01, 1000.02)
y0 <- log(sum(exp(lx)))
print(y0) ## Inf

y1 <- logSumExp(lx)
print(y1) ## 1000.708


## EXAMPLE #2
lx <- c(-1000.01, -1000.02)
y0 <- log(sum(exp(lx)))
print(y0) ## -Inf

y1 <- logSumExp(lx)
print(y1) ## -999.3218


## EXAMPLE #3
## R-help thread 'Beyond double-precision?' on May 9, 2009.

set.seed(1)
x <- runif(50)

## The logarithm of the harmonic mean
y0 <- log(1 / mean(1 / x))
print(y0)  ## -1.600885

lx <- log(x)
y1 <- log(length(x)) - logSumExp(-lx)
print(y1)  ## [1] -1.600885

# Sanity check
stopifnot(all.equal(y1, y0))
```

`product` *Calculates the product for each row (column) in a matrix*

Description

Calculates the product for each row (column) in a matrix.

Usage

```
product(x, idxs = NULL, na.rm = FALSE, ...)

rowProds(
  x,
```

```
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  method = c("direct", "expSumLog"),
  ...
)

colProds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  method = c("direct", "expSumLog"),
  ...
)
```

Arguments

x	A numeric NxK matrix.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, missing values are ignored, otherwise not.
...	Not used.
method	A character string specifying how each product is calculated.

Details

If method = "expSumLog", then then product() function is used, which calculates the produce via the logarithmic transform (treating negative values specially). This improves the precision and lowers the risk for numeric overflow. If method = "direct", the direct product is calculated via the prod() function.

Value

Returns a numeric vector of length N (K).

Missing values

Note, if method = "expSumLog", na.rm = FALSE, and x contains missing values (NA or NaN), then the calculated value is also missing value. Note that it depends on platform whether NaN or NA is returned when an NaN exists, cf. is.nan().

Author(s)

Henrik Bengtsson

rowAlls *Checks if a value exists / does not exist in each row (column) of a matrix*

Description

Checks if a value exists / does not exist in each row (column) of a matrix.

Usage

```
rowAlls(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

colAlls(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

allValue(x, idxs = NULL, value = TRUE, na.rm = FALSE, ...)

rowAnys(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

colAnys(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
```

```
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

anyValue(x, idxs = NULL, value = TRUE, na.rm = FALSE, ...)
```

Arguments

x	An NxK matrix or an N * K vector.
value	A value to search for.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.

Details

These functions takes either a matrix or a vector as input. If a vector, then argument dim. must be specified and fulfill prod(dim.) == length(x). The result will be identical to the results obtained when passing matrix(x,nrow = dim.[1L],ncol = dim.[2L]), but avoids having to temporarily create/allocate a matrix, if only such is needed only for these calculations.

Value

rowAlls() (colAlls()) returns an logical vector of length N (K). Analogously for rowAnys() (rowAlls()).

Logical value

When value is logical, the result is as if the function is applied on as.logical(x). More specifically, if x is numeric, then all zeros are treated as FALSE, non-zero values as TRUE, and all missing values as NA.

Author(s)

Henrik Bengtsson

See Also

rowCounts

Examples

```
x <- matrix(FALSE, nrow = 10, ncol = 5)
x[3:7, c(2, 4)] <- TRUE
x[2:4, ] <- TRUE
x[, 1] <- TRUE
x[5, ] <- FALSE
x[, 5] <- FALSE
print(x)

print(rowCounts(x))       # 1 4 4 4 0 3 3 1 1 1
print(colCounts(x))       # 9 5 3 5 0

print(rowAnys(x))
print(which(rowAnys(x)))  # 1  2  3  4  6  7  8  9 10
print(colAnys(x))
print(which(colAnys(x)))  # 1 2 3 4
```

`rowCollapse` *Extracts one cell per row (column) from a matrix*

Description

Extracts one cell per row (column) from a matrix. The implementation is optimized for memory and speed.

Usage

```
rowCollapse(x, idxs, rows = NULL, dim. = dim(x), ...)

colCollapse(x, idxs, cols = NULL, dim. = dim(x), ...)
```

Arguments

`x`	An NxK `matrix`.
`idxs`	An index `vector` of (maximum) length N (K) specifying the columns (rows) to be extracted.
`rows, cols`	A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done.
`dim.`	An `integer vector` of length two specifying the dimension of `x`, also when not a `matrix`.
`...`	Not used.

Value

Returns a `vector` of length N (K).

Author(s)

Henrik Bengtsson

See Also

Matrix indexing to index elements in matrices and arrays, cf. [().

Examples

```
x <- matrix(1:27, ncol = 3)

y <- rowCollapse(x, 1)
stopifnot(identical(y, x[, 1]))

y <- rowCollapse(x, 2)
stopifnot(identical(y, x[, 2]))

y <- rowCollapse(x, c(1, 1, 1, 1, 1, 3, 3, 3, 3))
stopifnot(identical(y, c(x[1:5, 1], x[6:9, 3])))

y <- rowCollapse(x, 1:3)
print(y)
y_truth <- c(x[1, 1], x[2, 2], x[3, 3], x[4, 1], x[5, 2],
             x[6, 3], x[7, 1], x[8, 2], x[9, 3])
stopifnot(identical(y, y_truth))
```

---

rowCounts *Counts the number of occurrences of a specific value*

Description

The row- and column-wise functions take either a matrix or a vector as input. If a vector, then argument `dim.` must be specified and fulfill `prod(dim.) == length(x)`. The result will be identical to the results obtained when passing `matrix(x,nrow = dim.[1L],ncol = dim.[2L])`, but avoids having to temporarily create/allocate a matrix, if only such is needed only for these calculations.

Usage

```
rowCounts(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)
```

```
colCounts(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

count(x, idxs = NULL, value = TRUE, na.rm = FALSE, ...)
```

Arguments

x	An NxK matrix or an N * K vector.
value	A value to search for.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.

Value

rowCounts() (colCounts()) returns an integer vector of length N (K). count() returns a scalar of type integer if the count is less than 2^31-1 (= .Machine$integer.max) otherwise a scalar of type double.

Author(s)

Henrik Bengtsson

See Also

rowAlls

Examples

```
x <- matrix(0:11, nrow = 4, ncol = 3)
x[2:3, 2:3] <- 2:5
x[3, 3] <- NA_integer_
print(x)

print(rowCounts(x, value = 2))
## [1]  0  1 NA  0
print(colCounts(x, value = 2))
## [1]  1  1 NA
```

```
print(colCounts(x, value = NA_integer_))
## [1] 0 0 1

print(rowCounts(x, value = 2, na.rm = TRUE))
## [1] 0 1 1 0
print(colCounts(x, value = 2, na.rm = TRUE))
## [1] 1 1 0

print(rowAnys(x, value = 2))
## [1] FALSE  TRUE  TRUE FALSE
print(rowAnys(x, value = NA_integer_))
## [1] FALSE FALSE  TRUE FALSE

print(colAnys(x, value = 2))
## [1] TRUE TRUE   NA
print(colAnys(x, value = 2, na.rm = TRUE))
## [1]  TRUE  TRUE FALSE

print(colAlls(x, value = 2))
## [1] FALSE FALSE FALSE
```

`rowCumsums` *Cumulative sums, products, minima and maxima for each row (column) in a matrix*

Description

Cumulative sums, products, minima and maxima for each row (column) in a matrix.

Usage

```
rowCumsums(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

colCumsums(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

rowCumprods(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

colCumprods(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

rowCummins(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

colCummins(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

rowCummaxs(x, rows = NULL, cols = NULL, dim. = dim(x), ...)

colCummaxs(x, rows = NULL, cols = NULL, dim. = dim(x), ...)
```

Arguments

x	An NxK matrix.
rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Value

Returns a numeric NxK matrix of the same mode as x, except when x is of mode logical, then the return type is integer.

Author(s)

Henrik Bengtsson

See Also

See cumsum(), cumprod(), cummin(), and cummax().

Examples

```
x <- matrix(1:12, nrow = 4, ncol = 3)
print(x)

yr <- rowCumsums(x)
print(yr)

yc <- colCumsums(x)
print(yc)

yr <- rowCumprods(x)
print(yr)

yc <- colCumprods(x)
print(yc)

yr <- rowCummaxs(x)
print(yr)

yc <- colCummaxs(x)
print(yc)

yr <- rowCummins(x)
print(yr)

yc <- colCummins(x)
print(yc)
```

rowDiffs *Calculates difference for each row (column) in a matrix*

Description

Calculates difference for each row (column) in a matrix.

Usage

```
rowDiffs(
  x,
  rows = NULL,
  cols = NULL,
  lag = 1L,
  differences = 1L,
  dim. = dim(x),
  ...
)

colDiffs(
  x,
  rows = NULL,
  cols = NULL,
  lag = 1L,
  differences = 1L,
  dim. = dim(x),
  ...
)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
lag	An integer specifying the lag.
differences	An integer specifying the order of difference.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Value

Returns a numeric Nx(K-1) or (N-1)xK matrix.

Author(s)

Henrik Bengtsson

See Also

See also diff2().

Examples

```
x <- matrix(1:27, ncol = 3)

d1 <- rowDiffs(x)
print(d1)

d2 <- t(colDiffs(t(x)))
stopifnot(all.equal(d2, d1))
```

---

rowIQRs *Estimates of the interquartile range for each row (column) in a matrix*

Description

Estimates of the interquartile range for each row (column) in a matrix.

Usage

```
rowIQRs(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)

colIQRs(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)

iqr(x, idxs = NULL, na.rm = FALSE, ...)
```

Arguments

x	A numeric NxK matrix.
na.rm	If TRUE, missing values are dropped first, otherwise not.
...	Additional arguments passed to rowQuantiles() (colQuantiles()).
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.

Value

Returns a numeric vector of length N (K).

Missing values

Contrary to IQR, which gives an error if there are missing values and na.rm = FALSE, iqr() and its corresponding row and column-specific functions return NA_real_.

Author(s)

Henrik Bengtsson

See Also

See IQR. See rowSds().

Examples

```
set.seed(1)

x <- matrix(rnorm(50 * 40), nrow = 50, ncol = 40)
str(x)

# Row IQRs
q <- rowIQRs(x)
print(q)
q0 <- apply(x, MARGIN = 1, FUN = IQR)
stopifnot(all.equal(q0, q))

# Column IQRs
q <- colIQRs(x)
print(q)
q0 <- apply(x, MARGIN = 2, FUN = IQR)
stopifnot(all.equal(q0, q))
```

---

rowLogSumExps *Accurately computes the logarithm of the sum of exponentials across rows or columns*

Description

Accurately computes the logarithm of the sum of exponentials across rows or columns.

Usage

```
rowLogSumExps(lx, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(lx), ...)

colLogSumExps(lx, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(lx), ...)
```

Arguments

lx	A numeric NxK matrix. Typically lx are $log(x)$ values.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, any missing values are ignored, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Value

A numeric vector of length N (K).

Benchmarking

These methods are implemented in native code and have been optimized for speed and memory.

Author(s)

Native implementation by Henrik Bengtsson. Original R code by Nakayama ??? (Japan).

See Also

To calculate the same on vectors, logSumExp().

---

rowMads *Standard deviation estimates for each row (column) in a matrix*

Description

Standard deviation estimates for each row (column) in a matrix.

Usage

```
rowMads(
  x,
  rows = NULL,
  cols = NULL,
  center = NULL,
  constant = 1.4826,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

colMads(
  x,
  rows = NULL,
  cols = NULL,
  center = NULL,
  constant = 1.4826,
  na.rm = FALSE,
  dim. = dim(x),
  ...
)

rowSds(
```

```
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  dim. = dim(x),
  ...
)

colSds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  dim. = dim(x),
  ...
)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
center	(optional) The center, defaults to the row means for the SD estimators and row medians for the MAD estimators.
constant	A scale factor. See mad for details.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Additional arguments passed to rowMeans() and rowSums().

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

See Also

sd, mad and var. rowIQRs().

rowMeans2 *Calculates the mean for each row (column) in a matrix*

Description

Calculates the mean for each row (column) in a matrix.

Usage

```
rowMeans2(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colMeans2(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)
```

Arguments

x	A numeric or a logical NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Details

The implementation of rowMeans2() and colMeans2() is optimized for both speed and memory.

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

rowMedians *Calculates the median for each row (column) in a matrix*

Description

Calculates the median for each row (column) in a matrix.

Usage

```
rowMedians(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colMedians(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Details

The implementation of rowMedians() and colMedians() is optimized for both speed and memory. To avoid coercing to doubles (and hence memory allocation), there is a special implementation for integer matrices. That is, if x is an integer matrix, then rowMedians(as.double(x)) (rowMedians(as.double(x))) would require three times the memory of rowMedians(x) (colMedians(x)), but all this is avoided.

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson, Harris Jaffee

See Also

See rowWeightedMedians() and colWeightedMedians() for weighted medians. For mean estimates, see rowMeans2() and rowMeans().

rowOrderStats *Gets an order statistic for each row (column) in a matrix*

Description

Gets an order statistic for each row (column) in a matrix.

Usage

```
rowOrderStats(x, rows = NULL, cols = NULL, which, dim. = dim(x), ...)

colOrderStats(x, rows = NULL, cols = NULL, which, dim. = dim(x), ...)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
which	An integer index in [1,K] ([1,N]) indicating which order statistic to be returned.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Details

The implementation of rowOrderStats() is optimized for both speed and memory. To avoid coercing to doubles (and hence memory allocation), there is a unique implementation for integer matrices.

Value

Returns a numeric vector of length N (K).

Missing values

This method does *not* handle missing values, that is, the result corresponds to having na.rm = FALSE (if such an argument would be available).

Author(s)

The native implementation of rowOrderStats() was adopted by Henrik Bengtsson from Robert Gentleman's rowQ() in the **Biobase** package.

See Also

See rowMeans() in colSums().

rowQuantiles *Estimates quantiles for each row (column) in a matrix*

Description

Estimates quantiles for each row (column) in a matrix.

Usage

```
rowQuantiles(
  x,
  rows = NULL,
  cols = NULL,
  probs = seq(from = 0, to = 1, by = 0.25),
  na.rm = FALSE,
  type = 7L,
  ...,
  drop = TRUE
)

colQuantiles(
  x,
  rows = NULL,
  cols = NULL,
  probs = seq(from = 0, to = 1, by = 0.25),
  na.rm = FALSE,
  type = 7L,
  ...,
  drop = TRUE
)
```

Arguments

x	An integer, numeric or logical NxK matrix with N >= 0.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
probs	A numeric vector of J probabilities in [0, 1].
na.rm	If TRUE, NAs are excluded first, otherwise not.
type	An integer specify the type of estimator. See quantile for more details.
...	Additional arguments passed to quantile.
drop	If TRUE, singleton dimensions in the result are dropped, otherwise not.

Value

Returns a NxJ (KxJ) matrix, where N (K) is the number of rows (columns) for which the J quantiles are calculated. The return type is either integer or numeric depending on type.

Author(s)

Henrik Bengtsson

See Also

quantile.

Examples

```
set.seed(1)

x <- matrix(rnorm(50 * 40), nrow = 50, ncol = 40)
str(x)

probs <- c(0.25, 0.5, 0.75)

# Row quantiles
q <- rowQuantiles(x, probs = probs)
print(q)
q_0 <- apply(x, MARGIN = 1, FUN = quantile, probs = probs)
stopifnot(all.equal(q_0, t(q)))

# Column IQRs
q <- colQuantiles(x, probs = probs)
print(q)
q_0 <- apply(x, MARGIN = 2, FUN = quantile, probs = probs)
stopifnot(all.equal(q_0, t(q)))
```

---

rowRanges *Gets the range of values in each row (column) of a matrix*

Description

Gets the range of values in each row (column) of a matrix.

Usage

```
rowRanges(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

rowMins(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

rowMaxs(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colRanges(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colMins(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colMaxs(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Value

rowRanges() (colRanges()) returns a numeric Nx2 (Kx2) matrix, where N (K) is the number of rows (columns) for which the ranges are calculated.

rowMins()/rowMaxs() (colMins()/colMaxs()) returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

See Also

rowOrderStats() and pmin.int().

---

rowRanks *Gets the rank of the elements in each row (column) of a matrix*

Description

Gets the rank of the elements in each row (column) of a matrix.

Usage

```
rowRanks(
  x,
  rows = NULL,
  cols = NULL,
 ties.method = c("max", "average", "first", "last", "random", "max", "min", "dense"),
  dim. = dim(x),
  ...
)

colRanks(
  x,
  rows = NULL,
  cols = NULL,
```

```
  ties.method = c("max", "average", "first", "last", "random", "max", "min", "dense"),
  dim. = dim(x),
  preserveShape = FALSE,
  ...
)
```

Arguments

x	A numeric or integer NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
ties.method	A character string specifying how ties are treated. For details, see below.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.
preserveShape	A logical specifying whether the matrix returned should preserve the input shape of x, or not.

Details

These functions rank values and treats missing values the same way as rank(). For equal values ("ties"), argument ties.method determines how these are ranked among each other. More precisely, for the following values of ties.method, each index set of ties consists of:

- "first" - increasing values that are all unique
- "last" - decreasing values that are all unique
- "min" - identical values equaling the minimum of their original ranks
- "max" - identical values equaling the maximum of their original ranks
- "average" - identical values that equal the sample mean of their original ranks. Because the average is calculated, the returned ranks may be non-integer values
- "random" - randomly shuffled values of their original ranks.
- "dense" - increasing values that are all unique and, contrary to "first", never contain any gaps

For more information on ties.method = "dense", see frank() of the **data.table** package. For more information on the other alternatives, see rank().

Note that, due to different randomization strategies, the shuffling order produced by these functions when using ties.method = "random" does not reproduce that of rank().

WARNING: For backward-compatibility reasons, the default is ties.method = "max"*, which differs from* rank*() which uses* ties.method = "average" *by default. Since we plan to change the default behavior in a future version, we recommend to explicitly specify the intended value of argument* ties.method.

Value

A matrix of type integer is returned, unless ties.method = "average" when it is of type numeric.

The rowRanks() function always returns an NxK matrix, where N (K) is the number of rows (columns) whose ranks are calculated.

The colRanks() function returns an NxK matrix, if preserveShape = TRUE, otherwise a KxN matrix.

Any names of x are ignored and absent in the result.

Missing values

Missing values are ranked as NA_integer_, as with na.last = "keep" in the rank() function.

Performance

The implementation is optimized for both speed and memory. To avoid coercing to doubles (and hence memory allocation), there is a unique implementation for integer matrices. Furthermore, it is more memory efficient to do colRanks(x,preserveShape = TRUE) than t(colRanks(x,preserveShape = FALSE)).

Author(s)

Hector Corrada Bravo and Harris Jaffee. Peter Langfelder for adding 'ties.method' support. Brian Montgomery for adding more 'ties.method's. Henrik Bengtsson adapted the original native implementation of rowRanks() from Robert Gentleman's rowQ() in the **Biobase** package.

See Also

For developers, see also Section Utility functions' in 'Writing R Extensions manual', particularly the native functions R_qsort_I() and R_qsort_int_I().

---

rowSums2 *Calculates the sum for each row (column) in a matrix*

Description

Calculates the sum for each row (column) in a matrix.

Usage

```
rowSums2(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)

colSums2(x, rows = NULL, cols = NULL, na.rm = FALSE, dim. = dim(x), ...)
```

Arguments

x	A numeric or a logical NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, NAs are excluded first, otherwise not.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Not used.

Details

The implementation of rowSums2() and colSums2() is optimized for both speed and memory.

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

---

rowTabulates *Tabulates the values in a matrix by row (column).*

Description

Tabulates the values in a matrix by row (column).

Usage

```
rowTabulates(x, rows = NULL, cols = NULL, values = NULL, ...)

colTabulates(x, rows = NULL, cols = NULL, values = NULL, ...)
```

Arguments

x	An integer, a logical, or a raw NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
values	An vector of J values of count. If NULL, all (unique) values are counted.
...	Not used.

Details

An alternative to these functions, is to use `table(x,row(x))` and `table(x,col(x))`, with the exception that the latter do not support the `raw` data type. When there are no missing values in `x`, we have that `all(rowTabulates(x) == t(table(x,row(x))))` and `all(colTabulates(x) == t(table(x,col(x))))`. When there are missing values, we have that `all(rowTabulates(x) == t(table(x,row(x),useNA = "always")[,seq_len(nrow(x))]))` and `all(colTabulates(x) == t(table(x,col(x),useNA = "always")[,seq_len(ncol(x))]))`.

Value

Returns a NxJ (KxJ) `matrix` where N (K) is the number of row (column) `vector`s tabulated and J is the number of values counted.

Author(s)

Henrik Bengtsson

Examples

```
x <- matrix(1:5, nrow = 10, ncol = 5)
print(x)
print(rowTabulates(x))
print(colTabulates(x))
# Count only certain values
print(rowTabulates(x, values = 1:3))


y <- as.raw(x)
dim(y) <- dim(x)
print(y)
print(rowTabulates(y))
print(colTabulates(y))
```

---

rowVars *Variance estimates for each row (column) in a matrix*

Description

Variance estimates for each row (column) in a matrix.

Usage

```
rowVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
```

```
  dim. = dim(x),
  ...
)

colVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  dim. = dim(x),
  ...
)
```

Arguments

x	A numeric NxK matrix.
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, missing values are excluded first, otherwise not.
center	(optional) The center, defaults to the row means.
dim.	An integer vector of length two specifying the dimension of x, also when not a matrix.
...	Additional arguments passed to rowMeans() and rowSums().

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

See Also

See rowMeans() and rowSums() in colSums().

Examples

```
set.seed(1)

x <- matrix(rnorm(20), nrow = 5, ncol = 4)
print(x)

# Row averages
print(rowMeans(x))
print(rowMedians(x))

# Column averages
```

```
print(colMeans(x))
print(colMedians(x))


# Row variabilities
print(rowVars(x))
print(rowSds(x))
print(rowMads(x))
print(rowIQRs(x))

# Column variabilities
print(rowVars(x))
print(colSds(x))
print(colMads(x))
print(colIQRs(x))


# Row ranges
print(rowRanges(x))
print(cbind(rowMins(x), rowMaxs(x)))
print(cbind(rowOrderStats(x, which = 1), rowOrderStats(x, which = ncol(x))))

# Column ranges
print(colRanges(x))
print(cbind(colMins(x), colMaxs(x)))
print(cbind(colOrderStats(x, which = 1), colOrderStats(x, which = nrow(x))))


x <- matrix(rnorm(2400), nrow = 50, ncol = 40)

# Row standard deviations
d <- rowDiffs(x)
s1 <- rowSds(d) / sqrt(2)
s2 <- rowSds(x)
print(summary(s1 - s2))

# Column standard deviations
d <- colDiffs(x)
s1 <- colSds(d) / sqrt(2)
s2 <- colSds(x)
print(summary(s1 - s2))
```

`rowWeightedMeans` *Calculates the weighted means for each row (column) in a matrix*

Description

Calculates the weighted means for each row (column) in a matrix.

Usage

```
rowWeightedMeans(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)

colWeightedMeans(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```

Arguments

x	A numeric NxK matrix.
w	A numeric vector of length K (N).
rows, cols	A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, missing values are excluded from the calculation, otherwise not.
...	Not used.

Details

The implementations of these methods are optimized for both speed and memory. If no weights are given, the corresponding rowMeans()/colMeans() is used.

Value

Returns a numeric vector of length N (K).

Author(s)

Henrik Bengtsson

See Also

See rowMeans() and colMeans() in colSums() for non-weighted means. See also weighted.mean.

Examples

```
x <- matrix(rnorm(20), nrow = 5, ncol = 4)
print(x)

# Non-weighted row averages
mu_0 <- rowMeans(x)
mu <- rowWeightedMeans(x)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (uniform weights)
w <- rep(2.5, times = ncol(x))
mu <- rowWeightedMeans(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (excluding some columns)
w <- c(1, 1, 0, 1)
mu_0 <- rowMeans(x[, (w == 1), drop = FALSE])
```

```
mu <- rowWeightedMeans(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (excluding some columns)
w <- c(0, 1, 0, 0)
mu_0 <- rowMeans(x[, (w == 1), drop = FALSE])
mu <- rowWeightedMeans(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted averages by rows and columns
w <- 1:4
mu_1 <- rowWeightedMeans(x, w = w)
mu_2 <- colWeightedMeans(t(x), w = w)
stopifnot(all.equal(mu_2, mu_1))
```

`rowWeightedMedians` *Calculates the weighted medians for each row (column) in a matrix*

Description

Calculates the weighted medians for each row (column) in a matrix.

Usage

```
rowWeightedMedians(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)

colWeightedMedians(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```

Arguments

`x`	A `numeric` NxK `matrix`.
`w`	A `numeric` `vector` of length K (N).
`rows, cols`	A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done.
`na.rm`	If `TRUE`, missing values are excluded from the calculation, otherwise not.
`...`	Additional arguments passed to `weightedMedian`().

Details

The implementations of these methods are optimized for both speed and memory. If no weights are given, the corresponding `rowMedians`()/`colMedians()` is used.

Value

Returns a `numeric` `vector` of length N (K).

Author(s)

Henrik Bengtsson

See Also

Internally, weightedMedian() is used. See rowMedians() and colMedians() for non-weighted medians.

Examples

```
x <- matrix(rnorm(20), nrow = 5, ncol = 4)
print(x)

# Non-weighted row averages
mu_0 <- rowMedians(x)
mu <- rowWeightedMedians(x)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (uniform weights)
w <- rep(2.5, times = ncol(x))
mu <- rowWeightedMedians(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (excluding some columns)
w <- c(1, 1, 0, 1)
mu_0 <- rowMedians(x[, (w == 1), drop = FALSE])
mu <- rowWeightedMedians(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted row averages (excluding some columns)
w <- c(0, 1, 0, 0)
mu_0 <- rowMedians(x[, (w == 1), drop = FALSE])
mu <- rowWeightedMedians(x, w = w)
stopifnot(all.equal(mu, mu_0))

# Weighted averages by rows and columns
w <- 1:4
mu_1 <- rowWeightedMedians(x, w = w)
mu_2 <- colWeightedMedians(t(x), w = w)
stopifnot(all.equal(mu_2, mu_1))
```

---

varDiff *Estimation of scale based on sequential-order differences*

Description

Estimation of scale based on sequential-order differences, corresponding to the scale estimates provided by var, sd, mad and IQR.

Usage

```
varDiff(x, idxs = NULL, na.rm = FALSE, diff = 1L, trim = 0, ...)

sdDiff(x, idxs = NULL, na.rm = FALSE, diff = 1L, trim = 0, ...)

madDiff(
  x,
  idxs = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  constant = 1.4826,
  ...
)

iqrDiff(x, idxs = NULL, na.rm = FALSE, diff = 1L, trim = 0, ...)

rowVarDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

colVarDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

rowSdDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

colSdDiffs(
```

```
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

rowMadDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

colMadDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

rowIQRDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)

colIQRDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  ...
)
```

Arguments

x	A numeric vector of length N or a numeric NxK matrix.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	If TRUE, NAs are excluded, otherwise not.
diff	The positional distance of elements for which the difference should be calculated.
trim	A double in [0,1/2] specifying the fraction of observations to be trimmed from each end of (sorted) x before estimation.
...	Not used.
constant	A scale factor adjusting for asymptotically normal consistency.

Details

Note that n-order difference MAD estimates, just like the ordinary MAD estimate by mad, apply a correction factor such that the estimates are consistent with the standard deviation under Gaussian distributions.

The interquartile range (IQR) estimates does *not* apply such a correction factor. If asymptotically normal consistency is wanted, the correction factor for IQR estimate is `1 / (2 * qnorm(3/4))`, which is half of that used for MAD estimates, which is `1 / qnorm(3/4)`. This correction factor needs to be applied manually, i.e. there is no `constant` argument for the IQR functions.

Value

Returns a numeric vector of length 1, length N, or length K.

Author(s)

Henrik Bengtsson

References

[1] J. von Neumann et al., *The mean square successive difference*. Annals of Mathematical Statistics, 1941, 12, 153-162.

See Also

For the corresponding non-differentiated estimates, see var, sd, mad and IQR. Internally, diff2() is used which is a faster version of diff().

weightedMad *Weighted Median Absolute Deviation (MAD)*

Description

Computes a weighted MAD of a numeric vector.

Usage

```
weightedMad(
  x,
  w = NULL,
  idxs = NULL,
  na.rm = FALSE,
  constant = 1.4826,
  center = NULL,
  ...
)

rowWeightedMads(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  constant = 1.4826,
  center = NULL,
  ...
)

colWeightedMads(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  constant = 1.4826,
  center = NULL,
  ...
)
```

Arguments

x	a numeric vector containing the values whose weighted MAD is to be computed.

w	a vector of weights the same length as x giving the weights to use for each element of x. Negative weights are treated as zero weights. Default value is equal weight to all values.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	a logical value indicating whether NA values in x should be stripped before the computation proceeds, or not. If NA, no check at all for NAs is done. Default value is NA (for efficiency).
constant	A numeric scale factor, cf. mad.
center	Optional numeric scalar specifying the center location of the data. If NULL, it is estimated from data.
...	Not used.

Value

Returns a numeric scalar.

Missing values

Missing values are dropped at the very beginning, if argument na.rm is TRUE, otherwise not.

Author(s)

Henrik Bengtsson

See Also

For the non-weighted MAD, see mad. Internally weightedMedian() is used to calculate the weighted median.

Examples

```
x <- 1:10
n <- length(x)

m1 <- mad(x)
m2 <- weightedMad(x)
stopifnot(identical(m1, m2))

w <- rep(1, times = n)
m1 <- weightedMad(x, w)
stopifnot(identical(m1, m2))

# All weight on the first value
w[1] <- Inf
m <- weightedMad(x, w)
stopifnot(m  == 0)

# All weight on the first two values
```

```
w[1:2] <- Inf
m1 <- mad(x[1:2])
m2 <- weightedMad(x, w)
stopifnot(identical(m1, m2))

# All weights set to zero
w <- rep(0, times = n)
m <- weightedMad(x, w)
stopifnot(is.na(m))
```

`weightedMean` *Weighted Arithmetic Mean*

Description

Computes the weighted sample mean of a numeric vector.

Usage

```
weightedMean(x, w = NULL, idxs = NULL, na.rm = FALSE, refine = FALSE, ...)
```

Arguments

`x`	a `numeric vector` containing the values whose weighted mean is to be computed.
`w`	a vector of weights the same length as `x` giving the weights to use for each element of `x`. Negative weights are treated as zero weights. Default value is equal weight to all values. If a missing-value weight exists, the result is always a missing value.
`idxs`	A `vector` indicating subset of elements to operate over. If `NULL`, no subsetting is done.
`na.rm`	a logical value indicating whether `NA` values in `x` should be stripped before the computation proceeds, or not. If `NA`, no check at all for `NA`s is done. Default value is `NA` (for efficiency).
`refine`	If `TRUE` and `x` is `numeric`, then extra effort is used to calculate the average with greater numerical precision, otherwise not.
`...`	Not used.

Value

Returns a `numeric` scalar. If `x` is of zero length, then `NaN` is returned, which is consistent with `mean`().

Missing values

This function handles missing values consistently with weighted.mean. More precisely, if na.rm = FALSE, then any missing values in either x or w will give result NA_real_. If na.rm = TRUE, then all (x,w) data points for which x is missing are skipped. Note that if both x and w are missing for a data points, then it is also skipped (by the same rule). However, if only w is missing, then the final results will always be NA_real_ regardless of na.rm.

Author(s)

Henrik Bengtsson

See Also

mean() and weighted.mean.

Examples

```
x <- 1:10
n <- length(x)

w <- rep(1, times = n)
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))

# Pull the mean towards zero
w[1] <- 5
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))

# Put even more weight on the zero
w[1] <- 8.5
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))

# All weight on the first value
w[1] <- Inf
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))

# All weight on the last value
w[1] <- 1
w[n] <- Inf
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))

# All weights set to zero
```

```
w <- rep(0, times = n)
m0 <- weighted.mean(x, w)
m1 <- weightedMean(x, w)
stopifnot(identical(m1, m0))
```

`weightedMedian` *Weighted Median Value*

Description

Computes a weighted median of a numeric vector.

Usage

```
weightedMedian(
  x,
  w = NULL,
  idxs = NULL,
  na.rm = FALSE,
  interpolate = is.null(ties),
  ties = NULL,
  ...
)
```

Arguments

`x`	a `numeric vector` containing the values whose weighted median is to be computed.
`w`	a vector of weights the same length as `x` giving the weights to use for each element of `x`. Negative weights are treated as zero weights. Default value is equal weight to all values.
`idxs`	A `vector` indicating subset of elements to operate over. If `NULL`, no subsetting is done.
`na.rm`	a logical value indicating whether `NA` values in `x` should be stripped before the computation proceeds, or not. If `NA`, no check at all for `NA`s is done. Default value is `NA` (for efficiency).
`interpolate`	If `TRUE`, linear interpolation is used to get a consistent estimate of the weighted median.
`ties`	If `interpolate == FALSE`, a character string specifying how to solve ties between two `x`'s that are satisfying the weighted median criteria. Note that at most two values can satisfy the criteria. When `ties` is `"min"` ("lower weighted median"), the smaller value of the two is returned and when it is `"max"` ("upper weighted median"), the larger value is returned. If `ties` is `"mean"`, the mean of the two values is returned. Finally, if `ties` is `"weighted"` (or `NULL`) a weighted average of the two are returned, where the weights are weights of all values `x[i] <= x[k]` and `x[i] >= x[k]`, respectively.
`...`	Not used.

Value

Returns a numeric scalar.

For the n elements x = c(x[1],x[2],...,x[n]) with positive weights w = c(w[1],w[2],...,w[n]) such that sum(w) = S, the *weighted median* is defined as the element x[k] for which the total weight of all elements x[i] < x[k] is less or equal to S/2 and for which the total weight of all elements x[i] > x[k] is less or equal to S/2 (c.f. [1]).

When using linear interpolation, the weighted mean of x[k-1] and x[k] with weights S[k-1] and S[k] corresponding to the cumulative weights of those two elements is used as an estimate.

If w is missing then all elements of x are given the same positive weight. If all weights are zero, NA_real_ is returned.

If one or more weights are Inf, it is the same as these weights have the same weight and the others have zero. This makes things easier for cases where the weights are result of a division with zero.

If there are missing values in w that are part of the calculation (after subsetting and dropping missing values in x), then the final result is always NA of the same type as x.

The weighted median solves the following optimization problem:

$$\alpha^* = \arg_\alpha \min \sum_{i=1}^{n} w_i |x_i - \alpha|$$

where $x = (x_1, x_2, \ldots, x_n)$ are scalars and $w = (w_1, w_2, \ldots, w_n)$ are the corresponding "weights" for each individual x value.

Author(s)

Henrik Bengtsson and Ola Hossjer, Centre for Mathematical Sciences, Lund University. Thanks to Roger Koenker, Econometrics, University of Illinois, for the initial ideas.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, Massachusetts Institute of Technology, 1989.

See Also

median, mean() and weightedMean().

Examples

```
x <- 1:10
n <- length(x)

m1 <- median(x)                          # 5.5
m2 <- weightedMedian(x)                  # 5.5
stopifnot(identical(m1, m2))

w <- rep(1, times = n)
m1 <- weightedMedian(x, w)               # 5.5 (default)
```

```
m2 <- weightedMedian(x, ties = "weighted")  # 5.5 (default)
m3 <- weightedMedian(x, ties = "min")       # 5
m4 <- weightedMedian(x, ties = "max")       # 6
stopifnot(identical(m1, m2))

# Pull the median towards zero
w[1] <- 5
m1 <- weightedMedian(x, w)                  # 3.5
y <- c(rep(0, times = w[1]), x[-1])         # Only possible for integer weights
m2 <- median(y)                             # 3.5
stopifnot(identical(m1, m2))

# Put even more weight on the zero
w[1] <- 8.5
weightedMedian(x, w)                # 2

# All weight on the first value
w[1] <- Inf
weightedMedian(x, w)                # 1

# All weight on the last value
w[1] <- 1
w[n] <- Inf
weightedMedian(x, w)                # 10

# All weights set to zero
w <- rep(0, times = n)
weightedMedian(x, w)                # NA

# Simple benchmarking
bench <- function(N = 1e5, K = 10) {
  x <- rnorm(N)
  gc()
  t <- c()
  t[1] <- system.time(for (k in 1:K) median(x))[3]
  t[2] <- system.time(for (k in 1:K) weightedMedian(x))[3]
  t <- t / t[1]
  names(t) <- c("median", "weightedMedian")
  t
}

print(bench(N =     5, K = 100))
print(bench(N =    50, K = 100))
print(bench(N =   200, K = 100))
print(bench(N =  1000, K = 100))
print(bench(N =  10e3, K =  20))
print(bench(N = 100e3, K =  20))
```

---

weightedVar *Weighted variance and weighted standard deviation*

Description

Computes a weighted variance / standard deviation of a numeric vector or across rows or columns of a matrix.

Usage

```
weightedVar(x, w = NULL, idxs = NULL, na.rm = FALSE, center = NULL, ...)

weightedSd(...)

rowWeightedVars(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)

colWeightedVars(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)

rowWeightedSds(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)

colWeightedSds(x, w = NULL, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```

Arguments

x	a numeric vector containing the values whose weighted variance is to be computed.
w	a vector of weights the same length as x giving the weights to use for each element of x. Negative weights are treated as zero weights. Default value is equal weight to all values.
idxs, rows, cols	A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done.
na.rm	a logical value indicating whether NA values in x should be stripped before the computation proceeds, or not. If NA, no check at all for NAs is done. Default value is NA (for efficiency).
center	Optional numeric scalar specifying the center location of the data. If NULL, it is estimated from data.
...	Not used.

Details

The estimator used here is the same as the one used by the "unbiased" estimator of the **Hmisc** package. More specifically, `weightedVar(x,w = w) == Hmisc::wtd.var(x,weights = w)`,

Value

Returns a numeric scalar.

Missing values

This function handles missing values consistently with weightedMean(). More precisely, if na.rm = FALSE, then any missing values in either x or w will give result NA_real_. If na.rm = TRUE, then all (x,w) data points for which x is missing are skipped. Note that if both x and w are missing for a data points, then it is also skipped (by the same rule). However, if only w is missing, then the final results will always be NA_real_ regardless of na.rm.

Author(s)

Henrik Bengtsson

See Also

For the non-weighted variance, see var.

Index