

# Package ‘matrixStats’

May 6, 2012

**Version** 0.5.0

**Date** 2012-04-16

**Title** Methods that apply to rows and columns of a matrix

**Author** Henrik Bengtsson, Hector Corrada Bravo, Robert Gentleman, Harris Jaffee

**Maintainer** Henrik Bengtsson <henrikb@braju.com>

**Depends** R (>= 2.9.0), methods

**Imports** R.methodsS3 (>= 1.2.2)

**Description** This packages provides methods operating on rows and columns of matrices. The objective is to have all methods being optimized for speed and memory consumption.

**License** Artistic-2.0

**LazyLoad** TRUE

**biocViews** Infrastructure, Statistics

**Repository** CRAN

**Repository/R-Forge/Project** matrixstats

**Repository/R-Forge/Revision** 74

**Date/Publication** 2012-05-06 06:57:56

## R topics documented:

matrixStats-package . . . . .	2
anyMissing . . . . .	4
indexByRow . . . . .	4
rowCollapse . . . . .	5
rowCounts . . . . .	6
rowDiffs . . . . .	7

rowIQRs . . . . .	8
rowMedians . . . . .	9
rowOrderStats . . . . .	10
rowProds . . . . .	11
rowQuantiles . . . . .	11
rowRanges . . . . .	12
rowRanks . . . . .	13
rowSds . . . . .	14
rowTabulates . . . . .	15
rowVars . . . . .	16
rowWeightedMeans.matrix . . . . .	17
rowWeightedMedians.matrix . . . . .	19
varDiff . . . . .	20
weightedMad . . . . .	21
weightedMedian . . . . .	22

<b>Index</b>	<b>26</b>
--------------	-----------

---

matrixStats-package	<i>Package matrixStats</i>
---------------------	----------------------------

---

## Description

This packages provides methods operating on rows and columns of matrices. The objective is to have all methods being optimized for speed and memory consumption.. This package went public (on CRAN) in June 2009. It is currently in a beta version where new methods are added.

## Installation

To install this package, please do:

```
install.packages("matrixStats")
```

## Dependancies and other requirements

See DESCRIPTION file for now.

## To get started

### Counts and logicals:

- rowCounts()
- rowAlls()
- rowAnys()

### Sums and products:

- rowSums()

- rowProds()

**Estimates of the mean:**

- rowMeans()
- rowMedians()
- rowWeightedMeans()
- rowWeightedMedians()

**Estimates of the standard deviation, variance and more:**

- rowSds()
- rowMads()
- rowVars()
- rowIQRs()
- rowQuantiles()
- rowOrderStats()
- rowRanks()
- rowRanges()
- rowMins()
- rowMaxs()

**Miscellaneous:**

- rowDiffs()
- anyMissing()

**How to contribute**

This is an open-source project which embraces collaborations. If you have improvements on code and/or documentation, or new function, please consider contributing them to this package.

**For developers**

It is currently not decided whether the methods should be S4 or S3 methods. This is the reason why some methods are based on S4 and some on S3. The ones using S3 rely on the **R.methodsS3** package to define the methods. There are also dependancies on other packages. The plan is to remove all such dependancies as soon as the API settles, but until then, we keep the dependancies for conveniency and in order to avoid reduncancy of available implementations of identical methods.

**How to cite this package**

Henrik Bengtsson, Hector Corrada Bravo, Robert Gentleman and Harris Jaffee (2012). matrixStats: Methods that apply to rows and columns of a matrix. R package version 0.5.0.

**Author(s)**

Henrik Bengtsson, Hector Corrada Bravo, Robert Gentleman, Harris Jaffee.

---

anyMissing	<i>Checks if there are any missing values in an object or not</i>
------------	---

---

**Description**

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

**Usage**

```
anyMissing(x, ...)
```

**Arguments**

x	A <a href="#">vector</a> , a <a href="#">list</a> , a <a href="#">matrix</a> , a <a href="#">data.frame</a> , or <a href="#">NULL</a> .
...	Not use.

**Details**

The implementation of this method is optimized for both speed and memory. The method will return [TRUE](#) at the first detected missing value.

**Value**

Returns [TRUE](#) if a missing value was detected, otherwise [FALSE](#).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**Examples**

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

---

indexByRow	<i>Translates matrix indices by rows into indices by columns</i>
------------	--

---

**Description**

Translates matrix indices by rows into indices by columns.

**Usage**

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

**Arguments**

`x` A [matrix](#).  
`idxs` A [vector](#) of indices. If `NULL`, all indices are returned.  
... Not use.

**Value**

Returns an [integer vector](#) of indices.

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**Examples**

```
x <- matrix(NA, nrow=5, ncol=4)
y <- t(x)
idxs <- seq(along=x)

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

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

stopifnot(x == t(y))
```

---

`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, ...)
colCollapse(x, idxs, ...)
```

**Arguments**

`x` An `NxK` [matrix](#).  
`idxs` An index [vector](#) of (maximum) length `N` (`K`) specifying the columns (rows) to be extracted.  
... Not used.

**Value**

Returns a [vector](#) of length N (K).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**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)
yT <- 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, yT))
```

---

rowCounts

*Counts the number of TRUE values in each row (column) of a matrix*

---

**Description**

Counts the number of TRUE values in each row (column) of a matrix.

**Usage**

```
rowCounts(x, na.rm=FALSE, ...)
colCounts(x, na.rm=FALSE, ...)
rowAlls(x, na.rm=FALSE, ...)
colAlls(x, na.rm=FALSE, ...)
rowAnys(x, na.rm=FALSE, ...)
colAnys(x, na.rm=FALSE, ...)
```

**Arguments**

`x` A logical NxK matrix.  
`na.rm` If TRUE, NAs are excluded first, otherwise not.  
`...` Not used.

**Value**

`rowCounts()` (`colCounts()`) returns an integer vector of length N (K). The other methods returns a logical vector of length N (K).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**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
```

---

rowDiffs

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


---

**Description**

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

**Usage**

```
rowDiffs(x, ...)
colDiffs(x, ...)
```

**Arguments**

`x` A numeric NxK matrix.  
`...` Not used.

**Value**

Returns a `numeric`  $N \times (K-1)$  or  $(N-1) \times K$  `matrix`.

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**See Also**

Internally `diff()` is used.

---

`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, ...)  
colIQRs(x, ...)
```

**Arguments**

`x`                    A `numeric`  $N \times K$  `matrix`.  
`...`                Additional arguments passed to `rowQuantiles()` (`colQuantiles()`).

**Value**

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

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**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))
```

---

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

## Arguments

x	A <b>numeric</b> NxK <b>matrix</b> .
na.rm	If <b>TRUE</b> , NAs are excluded first, otherwise not.
...	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 and Harris Jaffee.

**See Also**

See `rowMedians()` and `colMedians()` for weighted medians. For mean estimates, see `rowMeans()` in `colSums()`.

---

rowOrderStats	<i>Gets an order statistic for each row (column) in a matrix</i>
---------------	--

---

**Description**

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

**Usage**

```
rowOrderStats(x, which, ...)  
colOrderStats(x, which, ...)
```

**Arguments**

x	A numeric N x K matrix.
which	An integer index in [1,K] ([1,N]) indicating which order statistic to be returned.
...	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. Currently, `colOrderStats(x)` is calling `rowOrderStats(t(x))`.

**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)**

Henrik Bengtsson (<http://www.braju.com/R/>) The native implementation of `rowOrderStats()` was adopted from Robert Gentleman's `rowQ()` in the **Biobase** package.

**See Also**

See `rowMeans()` in `colSums()`.

---

rowProds	<i>Calculates the product for each row (column) in a matrix</i>
----------	---

---

**Description**

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

**Usage**

```
rowProds(x, ...)  
colProds(x, ...)
```

**Arguments**

x	A numeric NxK matrix.
...	Arguments passed to <code>rowSums()</code> .

**Details**

Internally the product is calculated via the logarithmic transform, treating zeros and negative values specially.

**Value**

Returns a [numeric vector](#) of length N (K).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

---

rowQuantiles	<i>Estimates quantiles for each row (column) in a matrix</i>
--------------	--

---

**Description**

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

**Usage**

```
rowQuantiles(x, probs=seq(from=0, to=1, by=0.25), ..., drop=TRUE)  
colQuantiles(x, ...)
```

**Arguments**

x	A <a href="#">numeric NxK matrix</a> with $N \geq 0$ .
probs	A <a href="#">numeric vector</a> of J probabilities in [0,1].
...	Additional arguments passed to <a href="#">quantile</a> .
drop	If TRUE, singleton dimensions in the result are dropped, otherwise not.

**Value**

Returns a [numeric JxN \(JxK\) matrix](#).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**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)
q0 <- apply(x, MARGIN=1, FUN=quantile, probs=probs)
stopifnot(all.equal(q0, t(q)))

# Column IQRs
q <- colQuantiles(x, probs=probs)
print(q)
q0 <- apply(x, MARGIN=2, FUN=quantile, probs=probs)
stopifnot(all.equal(q0, 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, na.rm=FALSE, ...)
colRanges(x, na.rm=FALSE, ...)
rowMins(x, na.rm=FALSE, ...)
colMins(x, na.rm=FALSE, ...)
rowMaxs(x, na.rm=FALSE, ...)
colMaxs(x, na.rm=FALSE, ...)

```

**Arguments**

x	A <b>numeric</b> NxK <b>matrix</b> .
na.rm	If <b>TRUE</b> , <b>NA</b> s are excluded first, otherwise not.
...	Not used.

**Details**

The rowRanges() function uses the much faster rowOrderStats() if there are no missing values.

**Value**

rowRanges() (colRanges()) returns a **numeric** Nx2 (Kx2) **matrix**. rowMins()/rowMaxs() (colMins()/colMaxs()) returns a **numeric vector** of length N (K).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**See Also**

[rowOrderStats\(\)](#) and [rowRanges\(\)](#).

---

rowRanks

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


---

**Description**

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

**Usage**

```

rowRanks(x, ...)
colRanks(x, ...)

```

**Arguments**

x	A <b>numeric</b> or <b>integer</b> NxK <b>matrix</b> .
...	Not used.

**Details**

The row- (column-) ranks of  $x$  are collected as *rows* of the result matrix. 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. Currently, `colRanks(x)` is just `rowRanks(t(x))`. Any *names* of  $x$  are ignored and absent in the result.

**Value**

`rowRanks()` (`colRanks()`) returns an `integer`  $N \times K$  ( $K \times N$ ) *matrix*.

**Missing and non- values**

These are ranked as NA, as with `na.last="keep"` in the `rank()` function.

**Ties**

Ties are ranked equally, as with setting `ties.method="max"` in the `rank()` function.

**Author(s)**

Hector Corrada Bravo and Harris Jaffee. The native implementation of `rowRanks()` was adapted from Robert Gentleman's `rowQ()` in the **Biobase** package.

**See Also**

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

---

rowSds

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


---

**Description**

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

**Usage**

```
rowSds(x, ...)
colSds(x, ...)
rowMads(x, centers=rowMedians(x,...), constant=1.4826, ...)
colMads(x, centers=colMedians(x,...), constant=1.4826, ...)
```

**Arguments**

<code>x</code>	A <code>numeric</code> $N \times K$ <i>matrix</i> .
<code>centers</code>	A optional <code>numeric vector</code> of length $N$ ( $K$ ) with centers. By default, they are calculated using <code>rowMedians()</code> .
<code>constant</code>	A scale factor. See <code>mad</code> for details.
<code>...</code>	Additional arguments passed to <code>rowVars()</code> and <code>rowMedians()</code> , respectively.

**Value**

Returns a [numeric vector](#) of length N (K).

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**See Also**

[sd](#), [mad](#) and [var](#). [rowIQRs\(\)](#).

---

rowTabulates	<i>Tabulates the values in a matrix by row (column)</i>
--------------	---

---

**Description**

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

**Usage**

```
rowTabulates(x, values=NULL, ...)  
colTabulates(x, values=NULL, ...)
```

**Arguments**

x	An <a href="#">integer</a> or <a href="#">raw</a> NxK <a href="#">matrix</a> .
values	An <a href="#">vector</a> of values of count. If <a href="#">NULL</a> , all (unique) values are counted.
...	Not used.

**Value**

Returns a NxJ (KxJ) [matrix](#) where J is the number of values counted.

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**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, center=NULL, ...)
colVars(x, ...)
```

### Arguments

`x` A numeric NxK matrix.  
`center` (optional) The center, defaults to the row means.  
`...` Additional arguments passed to `rowMeans()` and `rowSums()`.

### Value

Returns a numeric vector of length N (K).

### Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

### 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, 1), rowOrderStats(x, ncol(x))))

# Column ranges
print(colRanges(x))
print(cbind(colMins(x), colMaxs(x)))
print(cbind(colOrderStats(x, 1), colOrderStats(x, 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.matrix

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

---

## Description

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

**Usage**

```
## S3 method for class 'matrix'
rowWeightedMeans(x, w=NULL, na.rm=FALSE, ...)
## S3 method for class 'matrix'
colWeightedMeans(x, w=NULL, na.rm=FALSE, ...)
```

**Arguments**

x	A <b>numeric NxK matrix</b> .
w	A <b>numeric vector</b> of length K (N).
na.rm	If <b>TRUE</b> , 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 (<http://www.braju.com/R/>)

**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
xM0 <- rowMeans(x)
xM <- rowWeightedMeans(x)
stopifnot(all.equal(xM, xM0))

# Weighted row averages (uniform weights)
w <- rep(2.5, ncol(x))
xM <- rowWeightedMeans(x, w=w)
stopifnot(all.equal(xM, xM0))

# Weighted row averages (excluding some columns)
w <- c(1,1,0,1)
xM0 <- rowMeans(x[, (w == 1), drop=FALSE]);
xM <- rowWeightedMeans(x, w=w)
stopifnot(all.equal(xM, xM0))
```

```

# Weighted row averages (excluding some columns)
w <- c(0,1,0,0)
xM0 <- rowMeans(x[, (w == 1), drop=FALSE]);
xM <- rowWeightedMeans(x, w=w)
stopifnot(all.equal(xM, xM0))

# Weighted averages by rows and columns
w <- 1:4
xM1 <- rowWeightedMeans(x, w=w)
xM2 <- colWeightedMeans(t(x), w=w)
stopifnot(all.equal(xM2, xM1))

```

---

```
rowWeightedMedians.matrix
```

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

---

## Description

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

## Usage

```

## S3 method for class 'matrix'
rowWeightedMedians(x, w=NULL, na.rm=FALSE, ...)
## S3 method for class 'matrix'
colWeightedMedians(x, w=NULL, na.rm=FALSE, ...)

```

## Arguments

<code>x</code>	A <a href="#">numeric NxK matrix</a> .
<code>w</code>	A <a href="#">numeric vector</a> of length K (N).
<code>na.rm</code>	If <code>TRUE</code> , missing values are excluded from the calculation, otherwise not.
<code>...</code>	Additional arguments passed to <code>weightedMedian()</code> .

## 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 (<http://www.braju.com/R/>)

**See Also**

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

**Examples**

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

# Non-weighted row averages
xM0 <- rowMedians(x)
xM <- rowWeightedMedians(x)
stopifnot(all.equal(xM, xM0))

# Weighted row averages (uniform weights)
w <- rep(2.5, ncol(x))
xM <- rowWeightedMedians(x, w=w)
stopifnot(all.equal(xM, xM0))

# Weighted row averages (excluding some columns)
w <- c(1,1,0,1)
xM0 <- rowMedians(x[, (w == 1), drop=FALSE]);
xM <- rowWeightedMedians(x, w=w)
stopifnot(all.equal(xM, xM0))

# Weighted row averages (excluding some columns)
w <- c(0,1,0,0)
xM0 <- rowMedians(x[, (w == 1), drop=FALSE]);
xM <- rowWeightedMedians(x, w=w)
stopifnot(all.equal(xM, xM0))

# Weighted averages by rows and columns
w <- 1:4
xM1 <- rowWeightedMedians(x, w=w)
xM2 <- colWeightedMedians(t(x), w=w)
stopifnot(all.equal(xM2, xM1))
```

---

varDiff

*Estimation of discrepancies based on sequential order differences in a vector*


---

**Description**

Estimation of discrepancies based on sequential order differences in a vector.

**Usage**

```
varDiff(x, na.rm=FALSE, diff=1, ...)
sdDiff(x, na.rm=FALSE, diff=1, ...)
madDiff(x, na.rm=FALSE, diff=1, ...)
```

**Arguments**

x	A <a href="#">numeric vector</a> of length N.
na.rm	If <code>TRUE</code> , <code>NA</code> s are excluded, otherwise not.
diff	The positional distance of elements for which the difference should be calculated.
...	Not used.

**Value**

Returns a [numeric](#) scalar.

**Author(s)**

Henrik Bengtsson (<http://www.braju.com/R/>)

**See Also**

See [diff\(\)](#).

---

weightedMad	<i>Weighted Median Absolute Deviation (MAD)</i>
-------------	---

---

**Description**

Computes a weighted MAD of a numeric vector.

**Usage**

```
## Default S3 method:
weightedMad(x, w, na.rm=FALSE, constant=1.4826, center=NULL, ...)
```

**Arguments**

x	a <a href="#">numeric vector</a> 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.
na.rm	a logical value indicating whether <code>NA</code> values in x should be stripped before the computation proceeds, or not. If <code>NA</code> , no check at all for <code>NA</code> s is done. Default value is <code>NA</code> (for efficiency).
constant	A <a href="#">numeric</a> scale factor, cf. <a href="#">mad</a> .
center	Optional <a href="#">numeric</a> scalar specifying the center location of the data. If <code>NULL</code> , 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.

**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))
```

---

weightedMedian

*Weighted Median Value*

---

**Description**

Computes a weighted median of a numeric vector.

**Usage**

```
## Default S3 method:
weightedMedian(x, w, na.rm=NA, interpolate=is.null(ties), ties=NULL, method=c("quick", "shell"), ...)
```

**Arguments**

x	a <b>numeric vector</b> 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.
na.rm	a logical value indicating whether <b>NA</b> values in x should be stripped before the computation proceeds, or not. If <b>NA</b> , no check at all for <b>NA</b> s is done. Default value is <b>NA</b> (for efficiency).
interpolate	If <b>TRUE</b> , linear interpolation is used to get a consistent estimate of the weighted median.
ties	If <code>interpolate == FALSE</code> , 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", the smaller value of the two is returned and when it is "max", the larger value is returned. If ties is "mean", the mean of the two values is returned and if it is "both", both values are returned. Finally, if ties is "weighted" (or <b>NULL</b> ) a weighted average of the two are returned, where the weights are weights of all values $x[i] \leq x[k]$ and $x[i] > x[k]$ , respectively.
method	If "shell", then <code>order()</code> is used and when <code>method="quick"</code> , then internal <code>qsort()</code> is used.
...	Not used.

**Details**

For the  $n$  elements  $x = c(x[1], x[2], \dots, x[n])$  with positive weights  $w = c(w[1], w[2], \dots, w[n])$  such that  $\text{sum}(w) = S$ , 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]).

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

If one or more weights are **Inf**, it is the same as these weights have the same weight and the others has zero. This makes things easier for cases where the weights are result of a division with zero. In this case `median()` is used internally.

When all the weights are the same (after values with weight zero are excluded and **Inf**'s are taken care of), `median` is used internally.

The weighted median solves the following optimization problem:

$$\alpha^* = \arg_{\alpha} \min \sum_{k=1}^K w_k |x_k - \alpha|$$

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

### Value

Returns the weighted median.

### Benchmarks

When implementing this function speed has been highly prioritized and it also making use of the internal quick sort algorithm (from R v1.5.0). The result is that `weightedMedian(x)` is about half as slow as `median(x)`. It is hard to say how much since it depends on the data set, but it is also hard to time it exactly since internal garbage collector etc might mess up the measurements.

Initial test also indicates that `method="shell"`, which uses `order()` is slower than `method="quick"`, which uses internal `qsort()`. Non-weighted median can use partial sorting which is faster because all values do not have to be sorted.

See examples below for some simple benchmarking tests.

### Author(s)

Henrik Bengtsson and Ola Hössjer, 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 [weighted.mean](#).

### Examples

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

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

w <- rep(1, 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,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, n)
weightedMedian(x, w)                  # NA

# Simple benchmarking
bench <- function(N=1e5, K=10) {
  x <- rnorm(N)
  t <- c()
  gc()
  t[1] <- system.time(for (k in 1:K) median(x))[3]
  gc()
  t[2] <- system.time(for (k in 1:K) weightedMedian(x, method="quick"))[3]
  gc()
  t[3] <- system.time(for (k in 1:K) weightedMedian(x, method="shell"))[3]
  t <- t / t[1]
  t[4] <- t[2]/t[3]
  names(t) <- c("median", "wMed-quick", "wMed-shell", "quick/shell")
  t
}

print(bench(N= 5, K=1000))
print(bench(N=100, K=1000))
print(bench(N=1e3, K=100))
print(bench(N=1e5, K=10))
print(bench(N=1e6, K=1))

```

# Index

## \*Topic **array**

- rowCounts, 6
- rowDiffs, 7
- rowIQRs, 8
- rowMedians, 9
- rowOrderStats, 10
- rowProds, 11
- rowQuantiles, 11
- rowRanges, 12
- rowRanks, 13
- rowSds, 14
- rowVars, 16
- rowWeightedMeans.matrix, 17
- rowWeightedMedians.matrix, 19

## \*Topic **iteration**

- anyMissing, 4
- indexByRow, 4
- rowCounts, 6
- rowDiffs, 7
- rowIQRs, 8
- rowMedians, 9
- rowOrderStats, 10
- rowProds, 11
- rowQuantiles, 11
- rowRanges, 12
- rowRanks, 13
- rowSds, 14
- rowVars, 16
- rowWeightedMeans.matrix, 17
- rowWeightedMedians.matrix, 19
- varDiff, 20

## \*Topic **logic**

- anyMissing, 4
- indexByRow, 4
- rowCounts, 6

## \*Topic **methods**

- rowWeightedMeans.matrix, 17
- rowWeightedMedians.matrix, 19

## \*Topic **package**

- matrixStats-package, 2

## \*Topic **robust**

- rowDiffs, 7
- rowIQRs, 8
- rowMedians, 9
- rowOrderStats, 10
- rowProds, 11
- rowQuantiles, 11
- rowRanges, 12
- rowRanks, 13
- rowSds, 14
- rowVars, 16
- rowWeightedMeans.matrix, 17
- rowWeightedMedians.matrix, 19
- varDiff, 20
- weightedMad, 21
- weightedMedian, 22

## \*Topic **univar**

- rowCounts, 6
- rowDiffs, 7
- rowIQRs, 8
- rowMedians, 9
- rowOrderStats, 10
- rowProds, 11
- rowQuantiles, 11
- rowRanges, 12
- rowRanks, 13
- rowSds, 14
- rowVars, 16
- rowWeightedMeans.matrix, 17
- rowWeightedMedians.matrix, 19
- varDiff, 20
- weightedMad, 21
- weightedMedian, 22

## \*Topic **utilities**

- rowCollapse, 5
- rowTabulates, 15

[, 6

- anyMissing, 4

- anyMissing, character-method
  - (anyMissing), 4
- anyMissing, complex-method (anyMissing), 4
- anyMissing, data.frame-method
  - (anyMissing), 4
- anyMissing, list-method (anyMissing), 4
- anyMissing, logical-method (anyMissing), 4
- anyMissing, matrix-method (anyMissing), 4
- anyMissing, NULL-method (anyMissing), 4
- anyMissing, numeric-method (anyMissing), 4
  
- colAlls (rowCounts), 6
- colAlls, matrix-method (rowCounts), 6
- colAnys (rowCounts), 6
- colAnys, matrix-method (rowCounts), 6
- colCollapse (rowCollapse), 5
- colCollapse, matrix-method (rowCollapse), 5
- colCounts (rowCounts), 6
- colCounts, matrix-method (rowCounts), 6
- colDiffs (rowDiffs), 7
- colIQRs (rowIQRs), 8
- colMads (rowSds), 14
- colMaxs (rowRanges), 12
- colMaxs, matrix-method (rowRanges), 12
- colMedians (rowMedians), 9
- colMedians, matrix-method (rowMedians), 9
- colMins (rowRanges), 12
- colMins, matrix-method (rowRanges), 12
- colOrderStats (rowOrderStats), 10
- colOrderStats, matrix-method (rowOrderStats), 10
- colProds (rowProds), 11
- colQuantiles (rowQuantiles), 11
- colRanges (rowRanges), 12
- colRanges, matrix-method (rowRanges), 12
- colRanks (rowRanks), 13
- colRanks, matrix-method (rowRanks), 13
- colSds (rowSds), 14
- colSds, matrix-method (rowSds), 14
- colSums, 10, 11, 16, 18
- colTabulates (rowTabulates), 15
- colTabulates, matrix-method (rowTabulates), 15
- colVars (rowVars), 16
- colVars, matrix-method (rowVars), 16
  
- colWeightedMeans
  - (rowWeightedMeans.matrix), 17
- colWeightedMedians
  - (rowWeightedMedians.matrix), 19
  
- data.frame, 4
- diff, 8, 21
- double, 9, 10, 14
  
- FALSE, 4
  
- indexByRow, 4
- indexByRow, matrix-method (indexByRow), 4
- integer, 5, 7, 9, 10, 13–15
- IQR, 8
  
- list, 4
- logical, 7
  
- mad, 14, 15, 21, 22
- madDiff (varDiff), 20
- madDiff, numeric-method (varDiff), 20
- matrix, 4, 5, 7–16, 18, 19
- matrixStats (matrixStats-package), 2
- matrixStats-package, 2
- mean, 24
- median, 24
  
- NA, 7, 9, 13, 21, 23
- names, 14
- NULL, 4, 5, 15, 21, 23
- numeric, 7–16, 18, 19, 21–23
  
- quantile, 12
  
- rank, 14
- raw, 15
- rowAlls (rowCounts), 6
- rowAlls, matrix-method (rowCounts), 6
- rowAnys (rowCounts), 6
- rowAnys, matrix-method (rowCounts), 6
- rowCollapse, 5
- rowCollapse, matrix-method (rowCollapse), 5
- rowCounts, 6
- rowCounts, matrix-method (rowCounts), 6
- rowDiffs, 7
- rowDTabulates (rowTabulates), 15
- rowIQRs, 8, 15
- rowMads (rowSds), 14

rowMaxs (rowRanges), 12  
rowMaxs, matrix-method (rowRanges), 12  
rowMedians, 9, 10, 14, 19, 20  
rowMedians, matrix-method (rowMedians), 9  
rowMins (rowRanges), 12  
rowMins, matrix-method (rowRanges), 12  
rowOrderStats, 10, 13  
rowOrderStats, matrix-method  
    (rowOrderStats), 10  
rowProds, 11  
rowQuantiles, 8, 11  
rowRanges, 12, 13  
rowRanges, matrix-method (rowRanges), 12  
rowRanks, 13  
rowRanks, matrix-method (rowRanks), 13  
rowSds, 8, 14  
rowSds, matrix-method (rowSds), 14  
rowSums, 11  
rowTabulates, 15  
rowTabulates, matrix-method  
    (rowTabulates), 15  
rowVars, 14, 16  
rowVars, matrix-method (rowVars), 16  
rowWeightedMeans  
    (rowWeightedMeans.matrix), 17  
rowWeightedMeans.matrix, 17  
rowWeightedMedians  
    (rowWeightedMedians.matrix), 19  
rowWeightedMedians.matrix, 19  
  
sd, 15  
sdDiff (varDiff), 20  
sdDiff, numeric-method (varDiff), 20  
  
TRUE, 4, 7, 9, 13, 18, 19, 21–23  
  
var, 15  
varDiff, 20  
varDiff, numeric-method (varDiff), 20  
vector, 4–16, 18, 19, 21, 23  
  
weighted.mean, 18, 24  
weightedMad, 21  
weightedMedian, 19, 20, 22, 22