

# Package ‘DelayedMatrixStats’

February 25, 2021

**Type** Package

**Title** Functions that Apply to Rows and Columns of 'DelayedMatrix' Objects

**Version** 1.13.5

**Date** 2020-02-02

**Description** A port of the 'matrixStats' API for use with DelayedMatrix objects from the 'DelayedArray' package. High-performing functions operating on rows and columns of DelayedMatrix objects, 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.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**Depends** MatrixGenerics, DelayedArray (>= 0.17.6)

**Imports** methods, matrixStats (>= 0.56.0), sparseMatrixStats, Matrix, S4Vectors (>= 0.17.5), IRanges

**Suggests** testthat, knitr, rmarkdown, covr, BiocStyle, microbenchmark, profmem, HDF5Array

**VignetteBuilder** knitr

**URL** <https://github.com/PeteHaitch/DelayedMatrixStats>

**BugReports** <https://github.com/PeteHaitch/DelayedMatrixStats/issues>

**biocViews** Infrastructure, DataRepresentation, Software

**git\_url** <https://git.bioconductor.org/packages/DelayedMatrixStats>

**git\_branch** master

**git\_last\_commit** 65ee791

**git\_last\_commit\_date** 2021-02-05

**Date/Publication** 2021-02-24

**Author** Peter Hickey [aut, cre],  
Hervé Pagès [ctb],  
Aaron Lun [ctb]

**Maintainer** Peter Hickey <peter.hickey@gmail.com>

**R topics documented:**

colAlls,DelayedMatrix-method . . . . .	2
colAnyMissings . . . . .	4
colAvgPerRowSet,DelayedMatrix-method . . . . .	6
colCollapse,DelayedMatrix-method . . . . .	8
colCounts,DelayedMatrix-method . . . . .	9
colCummaxs,DelayedMatrix-method . . . . .	11
colDiffs,DelayedMatrix-method . . . . .	13
colIQRDiffs,DelayedMatrix-method . . . . .	15
colIQRs,DelayedMatrix-method . . . . .	18
colLogSumExps,DelayedMatrix-method . . . . .	20
colMads,DelayedMatrix-method . . . . .	22
colMeans2,DelayedMatrix-method . . . . .	24
colMedians,DelayedMatrix-method . . . . .	26
colOrderStats,DelayedMatrix-method . . . . .	27
colProds,DelayedMatrix-method . . . . .	29
colQuantiles,DelayedMatrix-method . . . . .	31
colRanks,DelayedMatrix-method . . . . .	32
colSums2,DelayedMatrix-method . . . . .	34
colTabulates,DelayedMatrix-method . . . . .	36
colVars,DelayedMatrix-method . . . . .	38
colWeightedMads,DelayedMatrix-method . . . . .	39
colWeightedMeans,DelayedMatrix-method . . . . .	41
colWeightedMedians,DelayedMatrix-method . . . . .	43
colWeightedSds,DelayedMatrix-method . . . . .	44
DelayedMatrixStats . . . . .	47
subset_by_Nindex . . . . .	47

<b>Index</b>	<b>48</b>
--------------	-----------

---

colAlls,DelayedMatrix-method

*Check if all elements in a row (column) of a matrix-like object are equal to a value*

---

**Description**

Check if all elements in a row (column) of a matrix-like object are equal to a value.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colAlls(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```

## S4 method for signature 'DelayedMatrix'
colAnys(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowAlls(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowAnys(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

```

## Arguments

<code>x</code>	A $N \times K$ <a href="#">DelayedMatrix</a> .
<code>rows</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>cols</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>value</code>	The value to search for.
<code>na.rm</code>	If <code>TRUE</code> , <code>NA</code> s are excluded first, otherwise not.
<code>force_block_processing</code>	<code>FALSE</code> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <code>TRUE</code> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .
<code>...</code>	Additional arguments passed to specific methods.

**Details**

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowAlls/matrixStats::colAlls`.

**Value**

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

**Author(s)**

Peter Hickey

Peter Hickey

**See Also**

- `matrixStats::rowAlls()` and `matrixStats::colAlls()` which are used when the input is a `matrix` or `numeric vector`.
- For checks if *any* element is equal to a value, see `rowAnys()`.
- `base::all()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'SolidRleArraySeed' seed
dm_rle <- RleArray(Rle(c(rep(1L, 5),
                        as.integer((0:4) ^ 2),
                        seq(-5L, -1L, 1L))),
                  dim = c(5, 3))

colAlls(dm_matrix, value = 1)
colAnys(dm_matrix, value = 2)
rowAlls(dm_rle, value = 1)
rowAnys(dm_rle, value = 2)
```

---

colAnyMissings

*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**

```

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

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

## S4 method for signature 'DelayedMatrix'
colAnyMissings(
  x,
  rows = NULL,
  cols = NULL,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
colAnyNAs(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowAnyMissings(
  x,
  rows = NULL,
  cols = NULL,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowAnyNAs(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating subset of elements (or rows and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
cols	A <a href="#">vector</a> indicating subset of elements (or rows and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
...	Additional arguments passed to specific methods.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .

**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)**

Peter Hickey

Peter Hickey

**See Also**

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

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'HDF5ArraySeed' seed
# NOTE: Requires that the HDF5Array package is installed
library(HDF5Array)
dm_HDF5 <- writeHDF5Array(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

dm_matrix[dm_matrix > 3] <- NA
colAnyNAs(dm_matrix)
dm_HDF5[dm_HDF5 > 3] <- NA
rowAnyNAs(dm_HDF5)
```

---

colAvgPerRowSet,DelayedMatrix-method

*Calculates for each row (column) a summary statistic for equally sized subsets of columns (rows)*

---

**Description**

Calculates for each row (column) a summary statistic for equally sized subsets of columns (rows).

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colAvgPerRowSet(
  X,
  W = NULL,
  cols = NULL,
  S,
  FUN = colMeans,
  ...,
  force_block_processing = FALSE,
  na.rm = NA,
  tFUN = FALSE
)
```

```
## S4 method for signature 'DelayedMatrix'
rowAvsPerColSet(
  X,
  W = NULL,
  rows = NULL,
  S,
  FUN = rowMeans,
  ...,
  force_block_processing = FALSE,
  na.rm = NA,
  tFUN = FALSE
)
```

### Arguments

X	A NxM <a href="#">DelayedMatrix</a> .
W	An optional numeric NxM matrix of weights.
cols	A <a href="#">vector</a> indicating the subset (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
S	An <a href="#">integer</a> KxJ matrix that specifying the J subsets. Each column hold K column (row) indices for the corresponding subset. The range of values is [1, M] ([1,N]).
FUN	A row-by-row (column-by-column) summary statistic function. It is applied to each column (row) subset of X that is specified by S.
...	Additional arguments passed to specific methods.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
na.rm	(logical) Argument passed to FUN() as <code>na.rm = na.rm</code> . If NA (default), then <code>na.rm = TRUE</code> is used if X or S holds missing values, otherwise <code>na.rm = FALSE</code> .
tFUN	If TRUE, X is transposed before it is passed to FUN.
rows	A <a href="#">vector</a> indicating the subset (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.

### Details

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowAvsPerColSet` / `matrixStats::colAvsPerRowSet`.

### Value

Returns a numeric JxN (MxJ) matrix.

### Author(s)

Peter Hickey

**See Also**

- `matrixStats::rowAvsPerColSet()` and `matrixStats::colAvsPerRowSet()` which are used when the input is a matrix or numeric vector.

**Examples**

```
# A DelayedMatrix with a 'DataFrame' seed
dm_DF <- DelayedArray(S4Vectors::DataFrame(C1 = rep(1L, 5),
                                           C2 = as.integer((0:4) ^ 2),
                                           C3 = seq(-5L, -1L, 1L)))

colAvsPerRowSet(dm_DF, S = matrix(1:2, ncol = 2))

rowAvsPerColSet(dm_DF, S = matrix(1:2, ncol = 1))
```

---

colCollapse,DelayedMatrix-method

*Extract one cell from each row (column) of a matrix-like object*

---

**Description**

Extract one cell from each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colCollapse(x, idxs, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowCollapse(x, idxs, rows = NULL, force_block_processing = FALSE, ...)
```

**Arguments**

<code>x</code>	A $N \times K$ <a href="#">DelayedMatrix</a> .
<code>idxs</code>	An index <a href="#">vector</a> with the position to extract. It is recycled to match the number of rows (column)
<code>cols</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>force_block_processing</code>	<code>FALSE</code> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <code>TRUE</code> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link{DelayedArray}{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
<code>...</code>	Additional arguments passed to specific methods.
<code>rows</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.

**Details**

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowCollapse / matrixStats::colCollapse`.



**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- [matrixStats::rowCollapse\(\)](#) and [matrixStats::colCollapse\(\)](#) which are used when the input is a matrix or numeric vector.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'HDF5ArraySeed' seed
# NOTE: Requires that the HDF5Array package is installed
library(HDF5Array)
dm_HDF5 <- writeHDF5Array(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# Extract the 4th row as a vector
# NOTE: An ordinary vector is returned regardless of the backend of
#       the DelayedMatrix object
colCollapse(dm_matrix, 4)
colCollapse(dm_HDF5, 4)

# Extract the 2nd column as a vector
# NOTE: An ordinary vector is returned regardless of the backend of
#       the DelayedMatrix object
rowCollapse(dm_matrix, 2)
rowCollapse(dm_HDF5, 2)
```

---

colCounts,DelayedMatrix-method

*Count how often an element in a row (column) of a matrix-like object is equal to a value*

---

**Description**

Count how often an element in a row (column) of a matrix-like object is equal to a value.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colCounts(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowCounts(
  x,
  rows = NULL,
  cols = NULL,
  value = TRUE,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
value	The value to search for.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowCounts/matrixStats::colCounts`.

**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowCounts()` and `matrixStats::colCounts()` which are used when the input is a matrix or numeric vector.
- For checks if any element is equal to a value, see `rowAnys()`. To check if all elements are equal, see `rowAlls()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'DataFrame' seed
dm_DF <- DelayedArray(S4Vectors::DataFrame(C1 = rep(1L, 5),
                                           C2 = as.integer((0:4) ^ 2),
                                           C3 = seq(-5L, -1L, 1L)))

colCounts(dm_matrix, value = 1)
# Only count those in the first 4 rows
colCounts(dm_matrix, rows = 1:4, value = 1)

rowCounts(dm_DF, value = 5)
# Only count those in the odd-numbered rows of the 2nd column
rowCounts(dm_DF, rows = seq(1, nrow(dm_DF), 2), cols = 2, value = 5)
```

---

colCummaxs,DelayedMatrix-method

*Calculates the cumulative maxima for each row (column) of a matrix-like object*

---

**Description**

Calculates the cumulative maxima for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colCummaxs(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
colCummins(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
colCumprods(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
colCumsums(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowCummaxs(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)
```

```
## S4 method for signature 'DelayedMatrix'
rowCummins(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowCumprods(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowCumsums(x, rows = NULL, cols = NULL, force_block_processing = FALSE, ...)
```

### Arguments

**x** A NxK [DelayedMatrix](#).

**rows** A [vector](#) indicating the subset of rows (and/or columns) to operate over. If [NULL](#), no subsetting is done.

**cols** A [vector](#) indicating the subset of rows (and/or columns) to operate over. If [NULL](#), no subsetting is done.

**force\_block\_processing** FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on `\link[DelayedArray]{getAutoBlockSize}()`) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary [base::array](#).

**...** Additional arguments passed to specific methods.

### Details

The S4 methods for `x` of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowCummaxs / matrixStats::colCummaxs`.

### Value

Returns a [numeric matrix](#) with the same dimensions as `x`.

### Author(s)

Peter Hickey  
 Peter Hickey  
 Peter Hickey  
 Peter Hickey

### See Also

- `matrixStats::rowCummaxs()` and `matrixStats::colCummaxs()` which are used when the input is a [matrix](#) or [numeric vector](#).
- For single maximum estimates, see [rowMaxs\(\)](#).
- `base::cummax()`.

**Examples**

```

# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                           as.integer((0:4) ^ 2),
                                           seq(-5L, -1L, 1L)),
                                           ncol = 3))

colCummaxs(dm_matrix)

colCummins(dm_matrix)

colCumprods(dm_matrix)

colCumsums(dm_matrix)

# Only use rows 2-4
rowCummaxs(dm_Matrix, rows = 2:4)

# Only use rows 2-4
rowCummins(dm_Matrix, rows = 2:4)

# Only use rows 2-4
rowCumprods(dm_Matrix, rows = 2:4)

# Only use rows 2-4
rowCumsums(dm_Matrix, rows = 2:4)

```

---

colDiffs,DelayedMatrix-method

*Calculates the difference between each element of a row (column) of a matrix-like object*

---

**Description**

Calculates the difference between each element of a row (column) of a matrix-like object.

**Usage**

```

## S4 method for signature 'DelayedMatrix'
colDiffs(
  x,
  rows = NULL,
  cols = NULL,
  lag = 1L,
  differences = 1L,
  force_block_processing = FALSE,
  ...
)

```

```
## S4 method for signature 'DelayedMatrix'
rowDiffs(
  x,
  rows = NULL,
  cols = NULL,
  lag = 1L,
  differences = 1L,
  force_block_processing = FALSE,
  ...
)
```

### Arguments

x	A $N \times K$ <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
lag	An integer specifying the lag.
differences	An integer specifying the order of difference.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

### Details

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowDiffs/matrixStats::colDiffs`.

### Value

Returns a [numeric matrix](#) with one column (row) less than x:  $N \times (K - 1)$  or  $(N - 1) \times K$ .

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowDiffs()` and `matrixStats::colDiffs()` which are used when the input is a [matrix](#) or [numeric vector](#).
- `base::diff()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'HDF5ArraySeed' seed
# NOTE: Requires that the HDF5Array package is installed
library(HDF5Array)
dm_HDF5 <- writeHDF5Array(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

colDiffs(dm_matrix)

rowDiffs(dm_HDF5)
# In reverse column order
rowDiffs(dm_HDF5, cols = seq(ncol(dm_HDF5), 1, -1))
```

---

colIQRDiffs,DelayedMatrix-method

*Calculates the interquartile range of the difference between each element of a row (column) of a matrix-like object*

---

**Description**

Calculates the interquartile range of the difference between each element of a row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colIQRDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
colMadDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
```

```
    force_block_processing = FALSE,
    ...
)

## S4 method for signature 'DelayedMatrix'
colSdDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
colVarDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowIQRDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowMadDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)
```



```
## S4 method for signature 'DelayedMatrix'
rowSdDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowVarDiffs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  diff = 1L,
  trim = 0,
  force_block_processing = FALSE,
  ...
)
```

### Arguments

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
diff	An integer specifying the order of difference.
trim	A double in [0,1/2] specifying the fraction of observations to be trimmed from each end of (sorted) x before estimation.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

### Details

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowIQRDiffs / matrixStats::colIQRDiffs`.

### Value

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

**Author(s)**

Peter Hickey  
 Peter Hickey  
 Peter Hickey  
 Peter Hickey

**See Also**

- `matrixStats::rowIQRDiffs()` and `matrixStats::colIQRDiffs()` which are used when the input is a matrix or numeric vector.
- For the direct interquartile range see also [rowIQRs](#).

**Examples**

```
# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                         as.integer((0:4) ^ 2),
                                         seq(-5L, -1L, 1L)),
                                         ncol = 3))

# A DelayedMatrix with a 'SolidRleArraySeed' seed
dm_Rle <- RleArray(Rle(c(rep(1L, 5),
                        as.integer((0:4) ^ 2),
                        seq(-5L, -1L, 1L))),
                  dim = c(5, 3))

colIQRDiffs(dm_Matrix)

colMadDiffs(dm_Matrix)

colSdDiffs(dm_Matrix)

colVarDiffs(dm_Matrix)

# Only using rows 2-4
rowIQRDiffs(dm_Rle, rows = 2:4)

# Only using rows 2-4
rowMadDiffs(dm_Rle, rows = 2:4)

# Only using rows 2-4
rowSdDiffs(dm_Rle, rows = 2:4)

# Only using rows 2-4
rowVarDiffs(dm_Rle, rows = 2:4)
```

---

colIQRs,DelayedMatrix-method

*Calculates the interquartile range for each row (column) of a matrix-like object*

---

**Description**

Calculates the interquartile range for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colIQRs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowIQRs(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , NAs are excluded first, otherwise not.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowIQRs/matrixStats::colIQRs`.

**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowIQRs()` and `matrixStats::colIQRs()` which are used when the input is a matrix or numeric vector.
- For a non-robust analog, see `rowSds()`. For a more robust version see `rowMads()`
- `stats::IQR()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                           as.integer((0:4) ^ 2),
                                           seq(-5L, -1L, 1L)),
                                           ncol = 3))

colIQRs(dm_matrix)

# Only using rows 2-4
rowIQRs(dm_matrix, rows = 2:4)
```

---

colLogSumExps,DelayedMatrix-method

*Accurately calculates the logarithm of the sum of exponentials for each row (column) of a matrix-like object*

---

**Description**

Accurately calculates the logarithm of the sum of exponentials for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colLogSumExps(
  lx,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowLogSumExps(
  lx,
  rows = NULL,
  cols = NULL,
```

```

na.rm = FALSE,
force_block_processing = FALSE,
...
)

```

### Arguments

<code>lx</code>	A NxK <a href="#">DelayedMatrix</a> . Typically, <code>lx</code> are $\log(x)$ values.
<code>rows</code>	A <a href="#">vector</a> indicating the subset (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>cols</code>	A <a href="#">vector</a> indicating the subset (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>na.rm</code>	If <code>TRUE</code> , NAs are excluded first, otherwise not.
<code>force_block_processing</code>	<code>FALSE</code> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <code>TRUE</code> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .
<code>...</code>	Additional arguments passed to specific methods.

### Details

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowLogSumExps / matrixStats::colLogSumExps`.

### Value

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

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowLogSumExps()` and `matrixStats::colLogSumExps()` which are used when the input is a `matrix` or `numeric` vector.
- `rowSums2()`

### Examples

```

x <- DelayedArray(matrix(runif(10), ncol = 2))
colLogSumExps(log(x))
rowLogSumExps(log(x))

```

---

colMads,DelayedMatrix-method

*Calculates the median absolute deviation for each row (column) of a matrix-like object*

---

### Description

Calculates the median absolute deviation for each row (column) of a matrix-like object.

### Usage

```
## S4 method for signature 'DelayedMatrix'
colMads(
  x,
  rows = NULL,
  cols = NULL,
  center = NULL,
  constant = 1.4826,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
colSds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
rowMads(
  x,
  rows = NULL,
  cols = NULL,
  center = NULL,
  constant = 1.4826,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
rowSds(
  x,
  rows = NULL,
  cols = NULL,
```

```

na.rm = FALSE,
center = NULL,
force_block_processing = FALSE,
...
)

```

### Arguments

x	A N x K <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
center	(optional) the center, defaults to the row means
constant	A scale factor. See <code>stats::mad()</code> for details.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

### Details

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowMads / matrixStats::colMads`.

### Value

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

### Author(s)

Peter Hickey  
Peter Hickey

### See Also

- `matrixStats::rowMads()` and `matrixStats::colMads()` which are used when the input is a [matrix](#) or [numeric vector](#).
- For mean estimates, see `rowMeans2()` and `rowMeans()`.
- For non-robust standard deviation estimates, see `rowSds()`.

### Examples

```

# A DelayedMatrix with a 'data.frame' seed
dm_df <- DelayedArray(data.frame(C1 = rep(1L, 5),
                                C2 = as.integer((0:4) ^ 2),
                                C3 = seq(-5L, -1L, 1L)))
# A DelayedMatrix with a 'DataFrame' seed

```

```
dm_DF <- DelayedArray(S4Vectors::DataFrame(C1 = rep(1L, 5),
                                           C2 = as.integer((0:4) ^ 2),
                                           C3 = seq(-5L, -1L, 1L)))

colMads(dm_df)

colSds(dm_df)

rowMads(dm_DF)

rowSds(dm_DF)
```

---

colMeans2,DelayedMatrix-method

*Calculates the mean for each row (column) of a matrix-like object*

---

## Description

Calculates the mean for each row (column) of a matrix-like object.

## Usage

```
## S4 method for signature 'DelayedMatrix'
colMeans2(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'Matrix'
colMeans2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)

## S4 method for signature 'SolidRleArraySeed'
colMeans2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)

## S4 method for signature 'DelayedMatrix'
rowMeans2(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'Matrix'
rowMeans2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```



**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , NAs are excluded first, otherwise not.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowMeans2/matrixStats::colMeans2`.

**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowMeans2()` and `matrixStats::colMeans2()` which are used when the input is a [matrix](#) or [numeric vector](#).
- See also `rowMeans()` for the corresponding function in base R.
- For variance estimates, see `rowVars()`.
- See also the base R version `base::rowMeans()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                              ncol = 3))

# A DelayedMatrix with a 'SolidRleArraySeed' seed
dm_rle <- RleArray(Rle(c(rep(1L, 5),
                        as.integer((0:4) ^ 2),
                        seq(-5L, -1L, 1L))),
                  dim = c(5, 3))

colMeans2(dm_matrix)

# NOTE: Temporarily use verbose output to demonstrate which method is
#       which method is being used
```

```

options(DelayedMatrixStats.verbose = TRUE)
# By default, this uses a seed-aware method for a DelayedMatrix with a
# 'SolidRleArraySeed' seed
rowMeans2(dm_Rle)
# Alternatively, can use the block-processing strategy
rowMeans2(dm_Rle, force_block_processing = TRUE)
options(DelayedMatrixStats.verbose = FALSE)

```

---

colMedians,DelayedMatrix-method

*Calculates the median for each row (column) of a matrix-like object*

---

## Description

Calculates the median for each row (column) of a matrix-like object.

## Usage

```

## S4 method for signature 'DelayedMatrix'
colMedians(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowMedians(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

```

## Arguments

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , NAs are excluded first, otherwise not.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .

... Additional arguments passed to specific methods.

### Details

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowMedians / matrixStats::colMedians`.

### Value

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

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowMedians()` and `matrixStats::colMedians()` which are used when the input is a `matrix` or `numeric vector`.
- For mean estimates, see `rowMeans2()` and `rowMeans()`.

### Examples

```
# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                         as.integer((0:4) ^ 2),
                                         seq(-5L, -1L, 1L)),
                                         ncol = 3))

colMedians(dm_Matrix)

rowMedians(dm_Matrix)
```

---

colOrderStats,DelayedMatrix-method

*Calculates an order statistic for each row (column) of a matrix-like object*

---

### Description

Calculates an order statistic for each row (column) of a matrix-like object.

### Usage

```
## S4 method for signature 'DelayedMatrix'
colOrderStats(
  x,
  rows = NULL,
  cols = NULL,
  which,
  force_block_processing = FALSE,
  ...
```



```

# Only using columns 2-3
colOrderStats(dm_Matrix, cols = 2:3, which = 1)

# Different algorithms, specified by `which`, may give different results
rowOrderStats(dm_Matrix, which = 1)
rowOrderStats(dm_Matrix, which = 2)

```

---

colProds,DelayedMatrix-method

*Calculates the product for each row (column) of a matrix-like object*

---

## Description

Calculates the product for each row (column) of a matrix-like object.

## Usage

```

## S4 method for signature 'DelayedMatrix'
colProds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  method = c("direct", "expSumLog"),
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'SolidRleArraySeed'
colProds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  method = c("direct", "expSumLog"),
  ...
)

## S4 method for signature 'DelayedMatrix'
rowProds(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  method = c("direct", "expSumLog"),
  force_block_processing = FALSE,
  ...
)

```

**Arguments**

<code>x</code>	A NxK <a href="#">DelayedMatrix</a> .
<code>rows</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>cols</code>	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <code>NULL</code> , no subsetting is done.
<code>na.rm</code>	If <code>TRUE</code> , NAs are excluded first, otherwise not.
<code>method</code>	A character vector of length one that specifies the how the product is calculated. Note, that this is not a generic argument and not all implementation have to provide it.
<code>force_block_processing</code>	<code>FALSE</code> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <code>TRUE</code> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .
<code>...</code>	Additional arguments passed to specific methods.

**Details**

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowProds/matrixStats::colProds`.

**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowProds()` and `matrixStats::colProds()` which are used when the input is a `matrix` or `numeric` vector.
- For sums across rows (columns), see `rowSums2()` (`colSums2()`)
- `base::prod()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'HDF5ArraySeed' seed
# NOTE: Requires that the HDF5Array package is installed
library(HDF5Array)
dm_HDF5 <- writeHDF5Array(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))
```

```
colProds(dm_matrix)
```

```
rowProds(dm_matrix)
```

---

```
colQuantiles,DelayedMatrix-method
```

*Calculates quantiles for each row (column) of a matrix-like object*

---

## Description

Calculates quantiles for each row (column) of a matrix-like object.

## Usage

```
## S4 method for signature 'DelayedMatrix'
colQuantiles(
  x,
  rows = NULL,
  cols = NULL,
  probs = seq(from = 0, to = 1, by = 0.25),
  na.rm = FALSE,
  type = 7L,
  force_block_processing = FALSE,
  ...,
  drop = TRUE
)
```

```
## S4 method for signature 'DelayedMatrix'
rowQuantiles(
  x,
  rows = NULL,
  cols = NULL,
  probs = seq(from = 0, to = 1, by = 0.25),
  na.rm = FALSE,
  type = 7L,
  force_block_processing = FALSE,
  ...,
  drop = TRUE
)
```

## Arguments

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
probs	A numeric vector of J probabilities in [0, 1].
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.

type	An integer specifying the type of estimator. See <code>stats::quantile()</code> . for more details.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .
...	Additional arguments passed to specific methods.
drop	If TRUE a vector is returned if $J == 1$ .

### Details

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowQuantiles / matrixStats::colQuantiles`.

### Value

a `numeric`  $N \times J$  ( $K \times J$ ) `matrix`, where  $N$  ( $K$ ) is the number of rows (columns) for which the  $J$  values are calculated.

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowQuantiles()` and `matrixStats::colQuantiles()` which are used when the input is a `matrix` or `numeric` vector.
- `stats::quantile`

### Examples

```
# A DelayedMatrix with a 'data.frame' seed
dm_df <- DelayedArray(data.frame(C1 = rep(1L, 5),
                                C2 = as.integer((0:4) ^ 2),
                                C3 = seq(-5L, -1L, 1L)))

# colnames, if present, are preserved as rownames on output
colQuantiles(dm_df)

# Input has no rownames so output has no rownames
rowQuantiles(dm_df)
```

---

colRanks,DelayedMatrix-method

*Calculates the rank of the elements for each row (column) of a matrix-like object*

---

### Description

Calculates the rank of the elements for each row (column) of a matrix-like object.



**Usage**

```
## S4 method for signature 'DelayedMatrix'
colRanks(
  x,
  rows = NULL,
  cols = NULL,
  ties.method = c("max", "average", "first", "last", "random", "max", "min", "dense"),
  preserveShape = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowRanks(
  x,
  rows = NULL,
  cols = NULL,
  ties.method = c("max", "average", "first", "last", "random", "max", "min", "dense"),
  force_block_processing = FALSE,
  ...
)
```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
ties.method	A character string specifying how ties are treated. Note that the default specifies fewer options than the original <code>matrixStats</code> package.
preserveShape	If TRUE the output matrix has the same shape as the input x. Note, that this is not a generic argument and not all implementation of this function have to provide it.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <code>base::array</code> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type `matrix`, `array`, or `numeric` call `matrixStats::rowRanks/matrixStats::colRanks`.

The `matrixStats::rowRanks()` function can handle a lot of different values for the `ties.method` argument. Users of the generic function should however only rely on `max` and `average` because the other ones are not guaranteed to be implemented:

`max` for values with identical values the maximum rank is returned

`average` for values with identical values the average of the ranks they cover is returned. Note, that in this case the return value is of type `numeric`.

**Value**

a matrix of type `integer` is returned unless `ties.method = "average"`. It has dimensions  $N \times J$  ( $K \times J$ ) `matrix`, where  $N$  ( $K$ ) is the number of rows (columns) of the input `x`.

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowRanks()` and `matrixStats::colRanks()` which are used when the input is a `matrix` or numeric vector.
- `base::rank`

**Examples**

```
# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                         as.integer((0:4) ^ 2),
                                         seq(-5L, -1L, 1L)),
                                         ncol = 3))

colRanks(dm_Matrix)

rowRanks(dm_Matrix)
```

---

colSums2,DelayedMatrix-method

*Calculates the sum for each row (column) of a matrix-like object*

---

**Description**

Calculates the sum for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colSums2(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'Matrix'
colSums2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)

## S4 method for signature 'SolidRleArraySeed'
colSums2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```

```
## S4 method for signature 'DelayedMatrix'
rowSums2(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'Matrix'
rowSums2(x, rows = NULL, cols = NULL, na.rm = FALSE, ...)
```

### Arguments

x	A $N \times K$ <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

### Details

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowSums2/matrixStats::colSums2`.

### Value

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

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowSums2()` and `matrixStats::colSums2()` which are used when the input is a [matrix](#) or [numeric vector](#).
- For mean estimates, see `rowMeans2()` and `rowMeans()`.
- `base::sum()`.

**Examples**

```

# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'Matrix' seed
dm_Matrix <- DelayedArray(Matrix::Matrix(c(rep(1L, 5),
                                           as.integer((0:4) ^ 2),
                                           seq(-5L, -1L, 1L)),
                                           ncol = 3))

colSums2(dm_matrix)

# NOTE: Temporarily use verbose output to demonstrate which method is
#       which method is being used
options(DelayedMatrixStats.verbose = TRUE)
# By default, this uses a seed-aware method for a DelayedMatrix with a
# 'SolidRleArraySeed' seed
rowSums2(dm_Matrix)
# Alternatively, can use the block-processing strategy
rowSums2(dm_Matrix, force_block_processing = TRUE)
options(DelayedMatrixStats.verbose = FALSE)

```

---

colTabulates,DelayedMatrix-method

*Tabulates the values in a matrix-like object by row (column)*


---

**Description**

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

**Usage**

```

## S4 method for signature 'DelayedMatrix'
colTabulates(
  x,
  rows = NULL,
  cols = NULL,
  values = NULL,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowTabulates(
  x,
  rows = NULL,
  cols = NULL,
  values = NULL,
  force_block_processing = FALSE,
  ...
)

```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
values	the values to search for.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowTabulates / matrixStats::colTabulates`.

**Value**

a [numeric](#) NxJ (KxJ) [matrix](#), where N (K) is the number of rows (columns) for which the J values are calculated.

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowTabulates()` and `matrixStats::colTabulates()` which are used when the input is a [matrix](#) or [numeric](#) vector.
- `base::table()`

**Examples**

```
# A DelayedMatrix with a 'DataFrame' seed
dm_DF <- DelayedArray(S4Vectors::DataFrame(C1 = rep(1L, 5),
                                           C2 = as.integer((0:4) ^ 2),
                                           C3 = seq(-5L, -1L, 1L)))

colTabulates(dm_DF)

rowTabulates(dm_DF)
```

---

 colVars,DelayedMatrix-method

*Calculates the variance for each row (column) of a matrix-like object*


---

### Description

Calculates the variance for each row (column) of a matrix-like object.

### Usage

```
## S4 method for signature 'DelayedMatrix'
colVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
rowVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  force_block_processing = FALSE,
  ...
)
```

### Arguments

x	A NxK <a href="#">DelayedMatrix</a> .
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , NAs are excluded first, otherwise not.
center	(optional) the center, defaults to the row means.
force_block_processing	FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowVars / matrixStats::colVars`.

**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- `matrixStats::rowVars()` and `matrixStats::colVars()` which are used when the input is a `matrix` or `numeric vector`.
- For mean estimates, see `rowMeans2()` and `rowMeans()`.
- For standard deviation estimates, see `rowSds()`.
- `stats::var()`.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

# A DelayedMatrix with a 'HDF5ArraySeed' seed
# NOTE: Requires that the HDF5Array package is installed
library(HDF5Array)
dm_HDF5 <- writeHDF5Array(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

colVars(dm_matrix)

rowVars(dm_matrix)
```

---

colWeightedMads,DelayedMatrix-method

*Calculates the weighted median absolute deviation for each row (column) of a matrix-like object*

---

**Description**

Calculates the weighted median absolute deviation for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colWeightedMads(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  constant = 1.4826,
  center = NULL,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowWeightedMads(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  constant = 1.4826,
  center = NULL,
  force_block_processing = FALSE,
  ...
)
```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
w	A <a href="#">numeric</a> vector of length K (N) that specifies by how much each element is weighted.
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
constant	A scale factor. See <code>stats::mad()</code> for details.
center	(optional) the center, defaults to the row means
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

**Details**

The S4 methods for x of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowWeightedMads / matrixStats::colWeightedMads`.



**Value**

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

**Author(s)**

Peter Hickey

**See Also**

- [matrixStats::rowWeightedMads\(\)](#) and [matrixStats::colWeightedMads\(\)](#) which are used when the input is a matrix or numeric vector.
- See also [rowMads](#) for the corresponding unweighted function.

**Examples**

```
# A DelayedMatrix with a 'matrix' seed
dm_matrix <- DelayedArray(matrix(c(rep(1L, 5),
                                as.integer((0:4) ^ 2),
                                seq(-5L, -1L, 1L)),
                                ncol = 3))

colWeightedMads(dm_matrix, w = 1:5)

rowWeightedMads(dm_matrix, w = 3:1)
```

---

colWeightedMeans,DelayedMatrix-method

*Calculates the weighted mean for each row (column) of a matrix-like object*

---

**Description**

Calculates the weighted mean for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colWeightedMeans(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowWeightedMeans(
  x,
  w = NULL,
```



```
colWeightedMeans(dm_Matrix)
# Specifying weights inversely proportional to rowwise variances
colWeightedMeans(dm_Matrix, w = 1 / rowVars(dm_Matrix))
rowWeightedMeans(dm_Matrix, w = 1:3)
```

---

colWeightedMedians,DelayedMatrix-method

*Calculates the weighted median for each row (column) of a matrix-like object*

---

## Description

Calculates the weighted median for each row (column) of a matrix-like object.

## Usage

```
## S4 method for signature 'DelayedMatrix'
colWeightedMedians(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)

## S4 method for signature 'DelayedMatrix'
rowWeightedMedians(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

## Arguments

x	A NxK <a href="#">DelayedMatrix</a> .
w	A <a href="#">numeric</a> vector of length K (N) that specifies by how much each element is weighted.
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.

`force_block_processing`  
 FALSE (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to TRUE (typically not advised). The block-processing strategy loads one or more (depending on `\link[DelayedArray]{getAutoBlockSize}()`) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary `base::array`.

... Additional arguments passed to specific methods.

### Details

The S4 methods for `x` of type `matrix`, `array`, or `numeric` call `matrixStats::rowWeightedMedians` / `matrixStats::colWeightedMedians`.

### Value

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

### Author(s)

Peter Hickey

### See Also

- `matrixStats::rowWeightedMedians()` and `matrixStats::colWeightedMedians()` which are used when the input is a matrix or numeric vector.
- See also `rowMedians` for the corresponding unweighted function.

### Examples

```
# A DelayedMatrix with a 'SolidRleArraySeed' seed
dm_Rle <- RleArray(Rle(c(rep(1L, 5),
                        as.integer((0:4) ^ 2),
                        seq(-5L, -1L, 1L))),
                  dim = c(5, 3))

# Specifying weights inversely proportional to rowwise MADs
colWeightedMedians(dm_Rle, w = 1 / rowMads(dm_Rle))
```

---

colWeightedSds,DelayedMatrix-method

*Calculates the weighted standard deviation for each row (column) of a matrix-like object*

---

### Description

Calculates the weighted standard deviation for each row (column) of a matrix-like object.

**Usage**

```
## S4 method for signature 'DelayedMatrix'
colWeightedSds(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
colWeightedVars(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
rowWeightedSds(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

```
## S4 method for signature 'DelayedMatrix'
rowWeightedVars(
  x,
  w = NULL,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

**Arguments**

x	A NxK <a href="#">DelayedMatrix</a> .
w	A <a href="#">numeric</a> vector of length K (N) that specifies by how much each element is weighted.
rows	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.

cols	A <a href="#">vector</a> indicating the subset of rows (and/or columns) to operate over. If <a href="#">NULL</a> , no subsetting is done.
na.rm	If <a href="#">TRUE</a> , <a href="#">NAs</a> are excluded first, otherwise not.
force_block_processing	<a href="#">FALSE</a> (the default) means that a seed-aware, optimised method is used (if available). This can be overridden to use the general block-processing strategy by setting this to <a href="#">TRUE</a> (typically not advised). The block-processing strategy loads one or more (depending on <code>\link[DelayedArray]{getAutoBlockSize}()</code> ) columns ( <code>colFoo()</code> ) or rows ( <code>rowFoo()</code> ) into memory as an ordinary <a href="#">base::array</a> .
...	Additional arguments passed to specific methods.

### Details

The S4 methods for `x` of type [matrix](#), [array](#), or [numeric](#) call `matrixStats::rowWeightedSds / matrixStats::colWeightedSds`.

### Value

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

### Author(s)

Peter Hickey

Peter Hickey

### See Also

- `matrixStats::rowWeightedSds()` and `matrixStats::colWeightedSds()` which are used when the input is a [matrix](#) or [numeric vector](#).
- See also [rowSds](#) for the corresponding unweighted function.

### Examples

```
# A DelayedMatrix with a 'SolidRleArraySeed' seed
dm_Rle <- RleArray(Rle(c(rep(1L, 5),
                        as.integer((0:4) ^ 2),
                        seq(-5L, -1L, 1L))),
                  dim = c(5, 3))

colWeightedSds(dm_Rle, w = 1 / rowMeans2(dm_Rle))

# Specifying weights inversely proportional to rowwise means
colWeightedVars(dm_Rle, w = 1 / rowMeans2(dm_Rle))

# Specifying weights inversely proportional to columnwise means
rowWeightedSds(dm_Rle, w = 1 / colMeans2(dm_Rle))

# Specifying weights inversely proportional to columnwise means
rowWeightedVars(dm_Rle, w = 1 / colMeans2(dm_Rle))
```

---

DelayedMatrixStats	<i>DelayedMatrixStats: Functions that apply to rows and columns of DelayedMatrix objects.</i>
--------------------	---

---

## Description

**DelayedMatrixStats** is a part of the **matrixStats** API to work with *DelayedMatrix* objects from the **DelayedArray** package. High-performing functions operating on rows and columns of *DelayedMatrix* objects, e.g. `colMedians()` / `rowMedians()`, `colRanks()` / `rowRanks()`, and `colSds()` / `rowSds()`. Functions optimized per data type and for subsetted calculations such that both memory usage and processing time is minimized.

---

subset_by_Nindex	subset_by_Nindex
------------------	------------------

---

## Description

`subset_by_Nindex()` is an internal generic function not aimed to be used directly by the user. It is basically an S4 generic for `DelayedArray:::subset_by_Nindex`.

## Usage

```
subset_by_Nindex(x, Nindex)
```

## Arguments

<code>x</code>	An array-like object.
<code>Nindex</code>	An unnamed list of subscripts as positive integer vectors, one vector per dimension in <code>x</code> . Empty and missing subscripts (represented by <code>integer(0)</code> and <code>NULL</code> list elements, respectively) are allowed. The subscripts can contain duplicated indices. They cannot contain NAs or non-positive values.

## Details

`subset_by_Nindex(x, Nindex)` conceptually performs the operation `x[Nindex[1], ..., Nindex[length(Nindex)]]`. `subset_by_Nindex()` methods need to support empty and missing subscripts, e.g., `subset_by_Nindex(x, list(NULL, integer(0)))` must return an `M x 0` object of class `class(x)` and `subset_by_Nindex(x, list(integer(0), integer(0)))` a `0 x 0` object of class `class(x)`.

Also, subscripts are allowed to contain duplicate indices so things like `subset_by_Nindex(x, list(c(1:3, 3:1), 2L))` need to be supported.

## Value

A object of class `class(x)` of the appropriate type (e.g., `integer`, `double`, etc.). For example, if `x` is a `data.frame` representing an `M x N` matrix of integers, `subset_by_Nindex(x, list(NULL, 2L))` must return its 2nd column as a `data.frame` with `M` rows and 1 column of type `integer`.

# Index

- all, [4](#)
- array, [4](#), [7](#), [8](#), [10](#), [12](#), [14](#), [17](#), [19](#), [21](#), [23](#), [25](#),  
[27](#), [28](#), [30](#), [32](#), [33](#), [35](#), [37](#), [39](#), [40](#), [42](#),  
[44](#), [46](#)
- base::array, [3](#), [5](#), [7](#), [8](#), [10](#), [12](#), [14](#), [17](#), [19](#), [21](#),  
[23](#), [25](#), [26](#), [28](#), [30](#), [32](#), [33](#), [35](#), [37](#), [38](#),  
[40](#), [42](#), [44](#), [46](#)
- base::rank, [34](#)
- colAlls, [4](#)
- colAlls, DelayedMatrix-method, [2](#)
- colAnyMissings, [4](#)
- colAnyMissings, DelayedMatrix-method  
(colAnyMissings), [4](#)
- colAnyNAs, DelayedMatrix-method  
(colAnyMissings), [4](#)
- colAnys, DelayedMatrix-method  
(colAlls, DelayedMatrix-method),  
[2](#)
- colAvgPerRowSet, [8](#)
- colAvgPerRowSet, DelayedMatrix-method,  
[6](#)
- colCollapse, [9](#)
- colCollapse, DelayedMatrix-method, [8](#)
- colCounts, [11](#)
- colCounts, DelayedMatrix-method, [9](#)
- colCummaxs, [12](#)
- colCummaxs, DelayedMatrix-method, [11](#)
- colCummins, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- colCumprods, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- colCumsums, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- colDiffs, [14](#)
- colDiffs, DelayedMatrix-method, [13](#)
- colIQRDiffs, [18](#)
- colIQRDiffs, DelayedMatrix-method, [15](#)
- colIQRs, [20](#)
- colIQRs, DelayedMatrix-method, [18](#)
- colLogSumExps, [21](#)
- colLogSumExps, DelayedMatrix-method, [20](#)
- colMadDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
[15](#)
- colMads, [23](#)
- colMads, DelayedMatrix-method, [22](#)
- colMeans2, [25](#)
- colMeans2, DelayedMatrix-method, [24](#)
- colMeans2, Matrix-method  
(colMeans2, DelayedMatrix-method),  
[24](#)
- colMeans2, SolidRleArraySeed-method  
(colMeans2, DelayedMatrix-method),  
[24](#)
- colMedians, [27](#)
- colMedians, DelayedMatrix-method, [26](#)
- colOrderStats, [28](#)
- colOrderStats, DelayedMatrix-method, [27](#)
- colProds, [30](#)
- colProds, DelayedMatrix-method, [29](#)
- colProds, SolidRleArraySeed-method  
(colProds, DelayedMatrix-method),  
[29](#)
- colQuantiles, [32](#)
- colQuantiles, DelayedMatrix-method, [31](#)
- colRanks, [34](#)
- colRanks, DelayedMatrix-method, [32](#)
- colSdDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
[15](#)
- colSds, DelayedMatrix-method  
(colMads, DelayedMatrix-method),  
[22](#)
- colSums2, [35](#)
- colSums2(), [30](#)
- colSums2, DelayedMatrix-method, [34](#)
- colSums2, Matrix-method  
(colSums2, DelayedMatrix-method),  
[34](#)
- colSums2, SolidRleArraySeed-method  
(colSums2, DelayedMatrix-method),  
[34](#)



- colTabulates, [37](#)
- colTabulates, DelayedMatrix-method, [36](#)
- colVarDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
[15](#)
- colVars, [39](#)
- colVars, DelayedMatrix-method, [38](#)
- colWeightedMads, [41](#)
- colWeightedMads, DelayedMatrix-method,  
[39](#)
- colWeightedMeans, [42](#)
- colWeightedMeans, DelayedMatrix-method,  
[41](#)
- colWeightedMedians, [44](#)
- colWeightedMedians, DelayedMatrix-method,  
[43](#)
- colWeightedSds, [46](#)
- colWeightedSds, DelayedMatrix-method,  
[44](#)
- colWeightedVars, DelayedMatrix-method  
(colWeightedSds, DelayedMatrix-method),  
[44](#)
- cummax, [12](#)
- data.frame, [47](#)
- DelayedMatrix, [3](#), [5](#), [7](#), [8](#), [10](#), [12](#), [14](#), [17](#), [19](#),  
[21](#), [23](#), [25](#), [26](#), [28](#), [30](#), [31](#), [33](#), [35](#), [37](#),  
[38](#), [40](#), [42](#), [43](#), [45](#)
- DelayedMatrixStats, [47](#)
- diff, [14](#)
- FALSE, [5](#)
- integer, [7](#), [10](#), [34](#)
- IQR, [20](#)
- logical, [4](#)
- mad, [23](#), [40](#)
- matrix, [4](#), [7](#), [8](#), [10](#), [12](#), [14](#), [17](#), [19](#), [21](#), [23](#), [25](#),  
[27](#), [28](#), [30](#), [32–35](#), [37](#), [39](#), [40](#), [42](#), [44](#),  
[46](#)
- NA, [3](#), [10](#), [17](#), [19](#), [21](#), [23](#), [25](#), [26](#), [30](#), [31](#), [35](#), [38](#),  
[40](#), [42](#), [43](#), [46](#)
- NULL, [3](#), [5](#), [7](#), [8](#), [10](#), [12](#), [14](#), [17](#), [19](#), [21](#), [23](#), [25](#),  
[26](#), [28](#), [30](#), [31](#), [33](#), [35](#), [37](#), [38](#), [40](#), [42](#),  
[43](#), [45](#), [46](#)
- numeric, [4](#), [7–10](#), [12](#), [14](#), [17](#), [19](#), [21](#), [23](#), [25](#),  
[27](#), [28](#), [30](#), [32](#), [33](#), [35](#), [37](#), [39–46](#)
- prod, [30](#)
- quantile, [32](#)
- rowAlls, [4](#), [11](#)
- rowAlls, DelayedMatrix-method  
(colAlls, DelayedMatrix-method),  
[2](#)
- rowAnyMissings (colAnyMissings), [4](#)
- rowAnyMissings, DelayedMatrix-method  
(colAnyMissings), [4](#)
- rowAnyNAs, DelayedMatrix-method  
(colAnyMissings), [4](#)
- rowAnys, [4](#), [11](#)
- rowAnys, DelayedMatrix-method  
(colAlls, DelayedMatrix-method),  
[2](#)
- rowAvsPerColSet, [8](#)
- rowAvsPerColSet, DelayedMatrix-method  
(colAvsPerRowSet, DelayedMatrix-method),  
[6](#)
- rowCollapse, [9](#)
- rowCollapse, DelayedMatrix-method  
(colCollapse, DelayedMatrix-method),  
[8](#)
- rowCounts, [11](#)
- rowCounts, DelayedMatrix-method  
(colCounts, DelayedMatrix-method),  
[9](#)
- rowCummaxs, [12](#)
- rowCummaxs, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- rowCummins, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- rowCumprods, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- rowCumsums, DelayedMatrix-method  
(colCummaxs, DelayedMatrix-method),  
[11](#)
- rowDiffs, [14](#)
- rowDiffs, DelayedMatrix-method  
(colDiffs, DelayedMatrix-method),  
[13](#)
- rowIQRDiffs, [18](#)
- rowIQRDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
[15](#)
- rowIQRs, [18](#), [20](#)
- rowIQRs, DelayedMatrix-method  
(colIQRs, DelayedMatrix-method),  
[18](#)
- rowLogSumExps, [21](#)
- rowLogSumExps, DelayedMatrix-method

- (colLogSumExps, DelayedMatrix-method),  
20
- rowMadDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
15
- rowMads, 23, 41
- rowMads(), 20
- rowMads, DelayedMatrix-method  
(colMads, DelayedMatrix-method),  
22
- rowMaxs, 12
- rowMeans, 23, 25, 27, 35, 39
- rowMeans2, 23, 25, 27, 35, 39, 42
- rowMeans2, DelayedMatrix-method  
(colMeans2, DelayedMatrix-method),  
24
- rowMeans2, Matrix-method  
(colMeans2, DelayedMatrix-method),  
24
- rowMedians, 27, 44
- rowMedians, DelayedMatrix-method  
(colMedians, DelayedMatrix-method),  
26
- rowOrderStats, 28
- rowOrderStats, DelayedMatrix-method  
(colOrderStats, DelayedMatrix-method),  
27
- rowProds, 30
- rowProds, DelayedMatrix-method  
(colProds, DelayedMatrix-method),  
29
- rowQuantiles, 32
- rowQuantiles, DelayedMatrix-method  
(colQuantiles, DelayedMatrix-method),  
31
- rowRanks, 34
- rowRanks, DelayedMatrix-method  
(colRanks, DelayedMatrix-method),  
32
- rowSdDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
15
- rowSds, 20, 23, 39, 46
- rowSds, DelayedMatrix-method  
(colMads, DelayedMatrix-method),  
22
- rowSums2, 30, 35
- rowSums2(), 21
- rowSums2, DelayedMatrix-method  
(colSums2, DelayedMatrix-method),  
34
- rowSums2, Matrix-method  
(colSums2, DelayedMatrix-method),  
34
- rowTabulates, 37
- rowTabulates, DelayedMatrix-method  
(colTabulates, DelayedMatrix-method),  
36
- rowVarDiffs, DelayedMatrix-method  
(colIQRDiffs, DelayedMatrix-method),  
15
- rowVars, 25, 39
- rowVars, DelayedMatrix-method  
(colVars, DelayedMatrix-method),  
38
- rowWeightedMads, 41
- rowWeightedMads, DelayedMatrix-method  
(colWeightedMads, DelayedMatrix-method),  
39
- rowWeightedMeans, 42
- rowWeightedMeans, DelayedMatrix-method  
(colWeightedMeans, DelayedMatrix-method),  
41
- rowWeightedMedians, 44
- rowWeightedMedians, DelayedMatrix-method  
(colWeightedMedians, DelayedMatrix-method),  
43
- rowWeightedSds, 46
- rowWeightedSds, DelayedMatrix-method  
(colWeightedSds, DelayedMatrix-method),  
44
- rowWeightedVars, DelayedMatrix-method  
(colWeightedSds, DelayedMatrix-method),  
44
- stats::quantile, 32
- subset\_by\_Nindex, 47
- sum, 35
- table, 37
- TRUE, 3, 5, 10, 17, 19, 21, 23, 25, 26, 30, 31,  
35, 38, 40, 42, 43, 46
- var, 39
- vector, 3–5, 7–10, 12, 14, 17, 19, 21, 23,  
25–28, 30, 31, 33, 35, 37–46