# Package ‘DelayedMatrixStats’

October 17, 2020

**Type** Package

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

**Version** 1.10.1

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

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

**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** 7.1.0

**Depends** DelayedArray (>= 0.14.0)

**Imports** methods, matrixStats (>= 0.56.0), Matrix, S4Vectors (>= 0.17.5), IRanges, HDF5Array (>= 1.16.1), BiocParallel

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

**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** RELEASE_3_11

**git_last_commit** 016f867

**git_last_commit_date** 2020-06-28

**Date/Publication** 2020-10-16

# R topics documented:



---

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

---

## Description

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

## Usage

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

colAnys(
```

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

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

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

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

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

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

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

## Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `value` | A value to search for. |
| `na.rm` | If `TRUE`, `NA`s are excluded first, otherwise not. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `...` | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Details

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

## Value

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

## Logical `value`

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

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

## Author(s)

Peter Hickey

Peter Hickey

## See Also

rowCounts

## 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, ...)

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

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

rowAnyNAs(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 DelayedMatrix.
- `rows` A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done.
- `cols` A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array.

## 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)
```

---

## colAvgsPerRowSet — *Applies a row-by-row (column-by-column) averaging function to equally-sized subsets of matrix columns (rows)*

---

### Description

Applies a row-by-row (column-by-column) averaging function to equally-sized subsets of matrix columns (rows). Each subset is averaged independently of the others.

### Usage

```
colAvgsPerRowSet(
  X,
  W = NULL,
  cols = NULL,
  S,
  FUN = colMeans,
  ...,
  tFUN = FALSE
```

```
)

rowAvgsPerColSet(
  X,
  W = NULL,
  rows = NULL,
  S,
  FUN = rowMeans,
  ...,
  tFUN = FALSE
)

## S4 method for signature 'DelayedMatrix'
colAvgsPerRowSet(
  X,
  W = NULL,
  cols = NULL,
  S,
  FUN = colMeans,
  ...,
  force_block_processing = FALSE,
  tFUN = FALSE
)

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

## Arguments

| | |
|---|---|
| X | A NxM DelayedMatrix. |
| W | An optional numeric NxM matrix of weights. |
| cols | A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done. |
| S | An integer KxJ matrix specifying the J subsets. Each column holds K column (row) indices for the corresponding subset. |
| FUN | The row-by-row (column-by-column) function used to average over each subset of X. This function must accept a numeric NxK (KxM) matrix and the logical argument na.rm (which is automatically set), and return a numeric vector of length N (M). |
| ... | Additional arguments passed to specific methods. |
| tFUN | If TRUE, the NxK (KxM) matrix passed to FUN() is transposed first. |

rows
: A vector indicating 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.

## Details

If argument S is a single column vector with indices 1:N, then rowAvgsPerColSet(X,S = S,FUN = rowMeans) gives the same result as rowMeans(X). Analogously, for colAvgsPerRowSet().

## Value

Returns a numeric JxN (MxJ) matrix, where row names equal rownames(X) (colnames(S)) and column names colnames(S) (colnames(X)).

## Author(s)

Peter Hickey

## 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)))
colAvgsPerRowSet(dm_DF, S = matrix(1:2, ncol = 2))

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

---

colCollapse *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

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

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

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

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

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

## Arguments

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

## Value

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

## Author(s)

Peter Hickey

## See Also

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

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

---

## Description

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

## Usage

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

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

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

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

## Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `value` | A value to search for. |
| `na.rm` | If `TRUE`, `NA`s are excluded first, otherwise not. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `...` | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Value

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

## Author(s)

Peter Hickey

## See Also

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

---

## Description

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

## Usage

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

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

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

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

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

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

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

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

## S4 method for signature 'DelayedMatrix'
colCummaxs(
  x,
```

```
  rows = NULL,
  cols = NULL,
  dim. = dim(x),
  force_block_processing = FALSE,
  ...
)

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

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

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

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

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

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

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

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

## Arguments

| | |
|---|---|
| x | A NxK DelayedMatrix. |
| rows | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| cols | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| dim. | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| ... | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Value

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

## Author(s)

Peter Hickey

Peter Hickey

Peter Hickey

Peter Hickey

### See Also

See cumsum(), cumprod(), cummin(), and 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 — *Calculates difference for each row (column) in a matrix*

---

### Description

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

### Usage

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

```
  lag = 1L,
  differences = 1L,
  dim. = dim(x),
  ...
)

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

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

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

## Arguments

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

`...` 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array.

## Value

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

## Author(s)

Peter Hickey

## See Also

See also `diff2`().

## 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` *Estimation of scale based on sequential-order differences*

---

## Description

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

**Usage**

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

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

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

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

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

rowMadDiffs(
```

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

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

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

## 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 DelayedMatrix.
- `rows` — A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done.
- `cols` — A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done.
- `na.rm` — If `TRUE`, `NA`s are excluded, otherwise not.
- `diff` — The positional distance of elements for which the difference should be calculated.
- `trim` — A `double` in [0,1/2] specifying the fraction of observations to be trimmed from each end of (sorted) `x` before estimation.
- `...` — 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array.

### Details

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

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

## Value

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

## Author(s)

Peter Hickey

Peter Hickey

Peter Hickey

Peter Hickey

## References

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

## See Also

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

## 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 *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

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

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

## 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 DelayedMatrix. |
| rows | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| cols | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| na.rm | If `TRUE`, missing values are dropped first, otherwise not. |
| ... | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

### Value

Returns a numeric vector of length N (K).

### Missing values

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

### Author(s)

Peter Hickey

### See Also

See IQR. See rowSds().

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

---

### Description

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

### Usage

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

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

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

```
  na.rm = FALSE,
  dim. = dim(lx),
  force_block_processing = FALSE,
  ...
)

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

## Arguments

| | |
|---|---|
| `lx` | A NxK DelayedMatrix. Typically, `lx` are $log(x)$ values. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `na.rm` | If `TRUE`, any missing values are ignored, otherwise not. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `...` | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Value

A `numeric` `vector` of length N (K).

## Benchmarking

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

## Author(s)

Peter Hickey

## See Also

To calculate the same on vectors, `logSumExp`().

## Examples

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

---

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

---

## Description

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

## Usage

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

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

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

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

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

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

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

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

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

```
  ...
)
```

## Arguments

x
: A NxK DelayedMatrix.

rows
: A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.

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

center
: (optional) The center, defaults to the row means for the SD estimators and row medians for the MAD estimators.

constant
: A scale factor. See mad for details.

na.rm
: If TRUE, NAs are excluded first, otherwise not.

dim.
: An integer vector of length two specifying the dimension of x, also when not a matrix.

...
: Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array.

## Value

Returns a numeric vector of length N (K).

## Author(s)

Peter Hickey

Peter Hickey

## See Also

sd, mad and var. rowIQRs().

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

## Description

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

## Usage

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

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

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

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

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

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

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

## Arguments

| | |
|---|---|
| x | A NxK DelayedMatrix. |

rows
: A vector indicating subset of rows (and/or columns) to operate over. If NULL, no subsetting is done.

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

na.rm
: If TRUE, NAs are excluded first, otherwise not.

dim.
: An integer vector of length two specifying the dimension of x, also when not a matrix.

...
: 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array.

## Details

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

## Value

Returns a numeric vector of length N (K).

## Author(s)

Peter Hickey

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

## Description

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

## Usage

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

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

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

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

## Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `na.rm` | If `TRUE`, `NA`s are excluded first, otherwise not. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `...` | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array.

## 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)

Peter Hickey

## See Also

See rowWeightedMedians() and colWeightedMedians() for weighted medians. 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 *Gets an order statistic for each row (column) in a matrix*

---

## Description

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

## Usage

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

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

## S4 method for signature 'DelayedMatrix'
colOrderStats(
  x,
```

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

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

## Arguments

| | |
|---|---|
| x | A NxK DelayedMatrix. |
| rows | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| cols | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| which | An `integer` index in [1,K] ([1,N]) indicating which order statistic to be returned. |
| dim. | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| ... | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Details

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

## Value

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

## Missing values

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

## Author(s)

Peter Hickey

## See Also

See rowMeans() in colSums().

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

---

## Description

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

## Usage

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

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

## 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

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `na.rm` | If `TRUE`, missing values are ignored, otherwise not. |
| `method` | A `character` string specifying how each product is calculated. |
| `...` | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Details

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

### Value

Returns a numeric vector of length N (K).

### Missing values

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

### Author(s)

Peter Hickey

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

---

### Description

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

### Usage

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

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

## 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 DelayedMatrix. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `probs` | A `numeric` `vector` of J probabilities in [0, 1]. |
| `na.rm` | If `TRUE`, `NA`s are excluded first, otherwise not. |
| `type` | An `integer` specify the type of estimator. See `quantile` for more details. |
| `...` | Additional arguments passed to specific methods. |
| `drop` | If TRUE, singleton dimensions in the result are dropped, 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.

## Value

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

## Author(s)

Peter Hickey

## See Also

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

---

## Description

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

## Usage

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

rowRanks(
```

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

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

## Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `ties.method` | A `character` string specifying how ties are treated. For details, see below. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `preserveShape` | A `logical` specifying whether the `matrix` returned should preserve the input shape of `x`, or not. |
| `...` | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Details

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

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

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

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

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

## Value

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

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

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

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

## Missing values

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

## Performance

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

## Author(s)

Peter Hickey

## See Also

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

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

---

## Description

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

## Usage

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

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

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

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

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

## S4 method for signature 'DelayedMatrix'
rowSums2(
  x,
  rows = NULL,
  cols = NULL,
```

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

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

## Arguments

x A NxK DelayedMatrix.

rows A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done.

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

na.rm If `TRUE`, `NA`s are excluded first, otherwise not.

dim. An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`.

... Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array.

## Details

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

## Value

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

## Author(s)

Peter Hickey

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

---

## Description

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

## Usage

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

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

## 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 DelayedMatrix. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |

| | |
|---|---|
| `values` | An `vector` of J values of count. If `NULL`, all (unique) values are counted. |
| `...` | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## Details

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

## Value

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

## Author(s)

Peter Hickey

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

---

## Description

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

## Usage

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

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

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

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

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

### Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `na.rm` | If `TRUE`, missing values are excluded first, otherwise not. |
| `center` | (optional) The center, defaults to the row means. |
| `dim.` | An `integer` `vector` of length two specifying the dimension of `x`, also when not a `matrix`. |
| `...` | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array.

## Value

Returns a numeric vector of length N (K).

## Author(s)

Peter Hickey

## See Also

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

## 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 *Weighted Median Absolute Deviation (MAD)*

---

## Description

Computes a weighted MAD of a numeric vector.

## Usage

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

```
  constant = 1.4826,
  center = NULL,
  ...
)

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

## 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 DelayedMatrix. |
| `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. |
| `rows` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of elements (or rows and/or columns) to operate over. If `NULL`, no subsetting is done. |

| | |
|---|---|
| na.rm | a logical value indicating whether NA values in x should be stripped before the computation proceeds, or not. If NA, no check at all for NAs is done. Default value is NA (for efficiency). |
| constant | A numeric scale factor, cf. mad. |
| center | Optional numeric scalar specifying the center location of the data. If NULL, it is estimated from data. |
| ... | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array. |

## Value

Returns a numeric scalar.

## Missing values

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

## Author(s)

Peter Hickey

## See Also

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

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

---

## Description

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

### Usage

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

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

## 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,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  force_block_processing = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `x` | A NxK DelayedMatrix. |
| `w` | A `numeric` `vector` of length K (N). |
| `rows` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `cols` | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| `na.rm` | If `TRUE`, missing values are excluded from the calculation, otherwise not. |
| `...` | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

### 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)

Peter Hickey

### See Also

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

### 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))

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

---

### Description

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

### Usage

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

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

## 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 DelayedMatrix. |
| w | A `numeric` `vector` of length K (N). |
| rows | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| cols | A `vector` indicating subset of rows (and/or columns) to operate over. If `NULL`, no subsetting is done. |
| na.rm | If `TRUE`, missing values are excluded from the calculation, otherwise not. |
| ... | Additional arguments passed to 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 \link[DelayedArray]{getAutoBlockSize}()) columns (`colFoo()`) or rows (`rowFoo()`) into memory as an ordinary base::array. |

## 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)

Peter Hickey

## See Also

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

## 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 *Weighted variance and weighted standard deviation*

## Description

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

## Usage

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

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

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

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

## 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 DelayedMatrix. |
| 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. |
| rows | A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done. |
| cols | A vector indicating subset of elements (or rows and/or columns) to operate over. If NULL, no subsetting is done. |
| na.rm | a logical value indicating whether NA values in x should be stripped before the computation proceeds, or not. If NA, no check at all for NAs is done. Default value is NA (for efficiency). |
| ... | 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 \link[DelayedArray]{getAutoBlockSize}()) columns (colFoo()) or rows (rowFoo()) into memory as an ordinary base::array. |

## Details

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

## Value

Returns a numeric scalar.

## Missing values

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

## Author(s)

Peter Hickey

Peter Hickey

### See Also

For the non-weighted variance, see var.

### 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 — *DelayedMatrixStats: Functions that apply to rows and columns of* DelayedMatrix *objects.*

---

### Description

**DelayedMatrixStats** is a port 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

| | |
|---|---|
| x | An array-like object. |
| Nindex | An unnamed list of subscripts as positive integer vectors, one vector per dimension in x. Empty and missing subscripts (represented by integer(0) and NULL 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,i 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