Package 'matter'

January 27, 2023

Type Package

Title A framework for rapid prototyping with file-based data structures

Version 2.0.1

Date 2016-10-11

Author Kylie A. Bemis <k.bemis@northeastern.edu>

Maintainer Kylie A. Bemis <k.bemis@northeastern.edu>

Description Memory-efficient reading, writing, and manipulation of structured binary data as file-based vectors, matrices, arrays, lists, and data frames.

License Artistic-2.0

Depends R (>= 3.5), BiocParallel, Matrix, methods, stats

Imports BiocGenerics, ProtGenerics, digest, irlba, biglm, utils

Suggests BiocStyle, knitr, testthat

VignetteBuilder knitr

Collate matterGenerics.R utils.R drle.R atoms.R ops.R matter.R matter_arr.R matter_fct.R matter_list.R matter_str.R signal.R search.R sparse_arr.R stream_stat.R stats.R rowStats.R apply.R scale.R biglm.R prcomp.R altrep.R

biocViews Infrastructure, DataRepresentation

URL https://github.com/kuwisdelu/matter

git_url https://git.bioconductor.org/packages/matter

git_branch RELEASE_3_16

git_last_commit 9cb1871

git_last_commit_date 2022-11-18

Date/Publication 2023-01-27

R topics documented:

`asearch` *Approximate Search with Interpolation*

Description

Search a set of values indexed by a sorted (non-decreasing) vector of keys. Finds the values corresponding to matches between the elements of the first argument and the keys. Approximate matching is allowed within a specified tolerance. Interpolation can be performed for key collisions and/or non-exact matches.

Usage

```
asearch(x, keys, values = seq_along(keys), tol = 0, tol.ref = "abs",
nomatch = NA_integer_, interp = "none")
```

Arguments

x	A vector of values to be matched. Only integer, numeric, and character vectors are supported.
keys	A sorted (non-decreasing) vector of keys to match against. Only integer, numeric, and character vectors are supported.
values	A vector of values corresponding to the keys. Only numeric types are supported.
tol	The tolerance for matching doubles. Must be nonnegative.
tol.ref	One of 'abs', 'x', or 'y'. If 'abs', then comparison is done by taking the absolute difference. If either 'x' or 'y', then relative differences are used, and this specifies which to use as the reference (target) value.
nomatch	The value to be returned in the case when no match is found, coerced to an integer. (Ignored if nearest = TRUE.)
interp	Interpolation scheme for non-exact matches or key collisions. One of 'none', 'mean', 'sum', 'max', 'min', 'area', 'linear', 'cubic', 'gaussian', or 'lanczos'.

Details

The algorithm is implemented in C and relies on binary search when the keys are sorted. See implementation details for bsearch for matching behavior when keys are sorted. For unsorted keys, a fallback to linear search is used.

Value

A vector of the same length as x, giving the values corresponding to matching keys.

Author(s)

Kylie A. Bemis

See Also

bsearch

Examples

```
keys <- c(1.11, 2.22, 3.33, 5.0, 5.1)
values <- keys^1.11

asearch(2.22, keys, values) # 2.42359
asearch(3.0, keys, values) # NA
asearch(3.0, keys, values, tol=0.1, tol.ref="y") # 3.801133
```

biglm *Using "biglm" with "matter"*

Description

This method allows matter_mat and sparse_mat matrices with the "biglm" package.

Usage

```
## S4 method for signature 'formula,matter_mat'
bigglm(formula, data, ..., chunksize = NULL, fc = NULL)

## S4 method for signature 'formula,sparse_mat'
bigglm(formula, data, ..., chunksize = NULL, fc = NULL)
```

Arguments

formula	A model formula.
data	A matter matrix with column names.
chunksize	An integer giving the maximum number of rows to process at a time. If left NULL, this will be calculated by dividing the chunksize of data by the number of variables in the formula.
fc	Either column indices or names of variables which are factors.
...	Additional options passed to bigglm.

Value

An object of class bigglm.

Author(s)

Kylie A. Bemis

See Also

bigglm

Examples

```
set.seed(1)

x <- matter_mat(rnorm(1000), nrow=100, ncol=10)

colnames(x) <- c(paste0("x", 1:9), "y")

fm <- paste0("y ~ ", paste0(paste0("x", 1:9), collapse=" + "))
fm <- as.formula(fm)
```

```
fit <- bigglm(fm, data=x, chunksize=50)
coef(fit)
```

---

binvec *Bin a vector*

Description

Bin a vector based on intervals or groups.

Usage

```
binvec(x, u, v, method = "sum")
```

Arguments

x	A numeric vector.
u, v	The (inclusive) lower and upper indices of the bins, or a factor providing the groupings.
method	The method used to bin the values. This is efficiently implemented for "sum", "mean", "min" or "max". Providing a function will use a less-efficient fallback.

Value

An vector of the summarized (binned) values.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)

x <- sort(runif(20))

binvec(x, c(1,6,11,16), c(5,10,15,20), method="mean")

binvec(x, seq(from=1, to=21, by=5), method="mean")
```

bsearch *Binary Search with Approximate Matching*

Description

Use a binary search to find approximate matches for the elements of its first argument among those in its second. This implementation allows for returning the index of the nearest match if there are no exact matches. It also allows specifying a tolerance for the comparison.

Usage

```
bsearch(x, table, tol = 0, tol.ref = "abs",
nomatch = NA_integer_, nearest = FALSE)

reldiff(x, y, ref = "abs")
```

Arguments

x	A vector of values to be matched. Only integer, numeric, and character vectors are supported.
y, table	A sorted (non-decreasing) vector of values to be matched against. Only integer, numeric, and character vectors are supported.
tol	The tolerance for matching doubles. Must be >= 0.
ref, tol.ref	One of 'abs', 'x', or 'y'. If 'abs', then comparison is done by taking the absolute difference. If either 'x' or 'y', then relative differences are used, and this specifies which to use as the reference (target) value. For strings, this uses the Hamming distance (number of errors), normalized by the length of the reference string for relative differences.
nomatch	The value to be returned in the case when no match is found, coerced to an integer. (Ignored if nearest = TRUE.)
nearest	Should the index of the closest match be returned if no exact matches are found?

Details

The algorithm is implemented in C and currently only works for 'integer', 'numeric', and 'character' vectors. If there are multiple matches, then the first match that is found will be returned, with no guarantees. If a nonzero tolerance is provided, the closest match will be returned.

The "nearest" match for strings when there are no exact matches is decided by the match with the most initial matching characters. Tolerance is ignored for strings and integers. Behavior is undefined and results may be unexpected if values includes NAs.

Value

A vector of the same length as x, giving the indexes of the matches in table.

Author(s)

Kylie A. Bemis

See Also

asearch, match, pmatch, findInterval

Examples

```
a <- c(1.11, 2.22, 3.33, 5.0, 5.1)

bsearch(2.22, a) # 2
bsearch(3.0, a) # NA
bsearch(3.0, a, nearest=TRUE) # 3
bsearch(3.0, a, tol=0.1, tol.ref="values") # 3

b <- c("hello", "world!")
bsearch("world!", b) # 2
bsearch("worl", b) # NA
bsearch("worl", b, nearest=TRUE) # 2
```

---

checksum *Calculate Checksums and Cryptographic Hashes*

Description

This is a generic function for applying cryptographic hash functions and calculating checksums for arbitrary R objects.

Usage

```
checksum(x, ...)

## S4 method for signature 'matter_'
checksum(x, algo = "sha1", ...)
```

Arguments

x	An object to be hashed.
algo	The hash function to use.
...	Additional arguments to be passed to the hash function.

Details

The method for matter objects calculates checksums of each of the files in the object's paths.

Value

A character vector giving the hash or hashes of the object.

Author(s)

Kylie A. Bemis

See Also

digest

Examples

```
x <- matter(1:10)
y <- matter(1:10)

checksum(x)
checksum(y) # should be the same
```

---

chunkApply *Apply Functions Over Chunks of a List, Vector, or Matrix*

Description

Perform equivalents of `apply`, `lapply`, and `mapply`, but over parallelized chunks of data. This is most useful if accessing the data is potentially time-consuming, such as for file-based `matter` objects. Operating on chunks reduces the number of I/O operations.

Usage

```
## Operate on elements/rows/columns
chunkApply(X, MARGIN, FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = FALSE, BPPARAM = bpparam())

chunkLapply(X, FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = FALSE, BPPARAM = bpparam())

chunkMapply(FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = FALSE, BPPARAM = bpparam())


## Operate on complete chunks
chunk_rowapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
```

```
    verbose = FALSE, BPPARAM = bpparam())

chunk_colapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = FALSE, BPPARAM = bpparam())

chunk_lapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = FALSE, BPPARAM = bpparam())

chunk_mapply(FUN, ..., MoreArgs = NULL,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = FALSE, BPPARAM = bpparam())
```

Arguments

X	A matrix for chunkApply(), a list or vector for chunkLapply(), or lists for chunkMapply(). These may be any class that implements suitable methods for [, [[, dim, and length().
MARGIN	If the object is matrix-like, which dimension to iterate over. Must be 1 or 2, where 1 indicates rows and 2 indicates columns. The dimension names can also be used if X has dimnames set.
FUN	The function to be applied.
MoreArgs	A list of other arguments to FUN.
...	Additional arguments to be passed to FUN.
simplify	Should the result be simplified into a vector, matrix, or higher dimensional array?
nchunks	The number of chunks to use. If NA (the default), this is inferred from getOption("matter.default.chu For IO-bound operations, using fewer chunks will often be faster, but use more memory.
depends	A list with length equal to the extent of X. Each element of depends should give a vector of indices which correspond to other elements of X on which each computation depends. These elements are passed to FUN. For time efficiency, no attempt is made to verify these indices are valid.
outpath	If non-NULL, a file path where the results should be written as they are processed. If specified, FUN must return a 'raw', 'logical', 'integer', or 'numeric' vector. The result will be returned as a matter object.
verbose	Should user messages be printed with the current chunk being processed?
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Details

For chunkApply(), chunkLapply(), and chunkMapply():

For vectors and lists, the vector is broken into some number of chunks according to chunks. The individual elements of the chunk are then passed to FUN.

For matrices, the matrix is chunked along rows or columns, based on the number of chunks. The individual rows or columns of the chunk are then passed to FUN.

In this way, the first argument of FUN is analogous to using the base apply, lapply, and mapply functions.

For chunk_rowapply(), chunk_colapply(), chunk_lapply(), and chunk_mapply():

In this situation, the entire chunk is passed to FUN, and FUN is responsible for knowing how to handle a sub-vector or sub-matrix of the original object. This may be useful if FUN is already a function that could be applied to the whole object such as rowSums or colSums.

When this is the case, it may be useful to provide a custom simplify function.

For convenience to the programmer, several attributes are made available when operating on a chunk.

- "chunkid":The index of the chunk currently being processed by FUN.
- "index":The indices of the elements of the chunk, as elements/rows/columns in the original matrix/vector.
- "depends" (optional):If depends is given, then this is a list of indices within the chunk. The length of the list is equal to the number of elements/rows/columns that should be processed in the chunk. Each list element is a vector of indices giving the elements/rows/columns of the chunk that should be processed.

The depends argument can be used to iterate over dependent elements of a vector, or dependent rows/columns of a matrix. This can be useful if the calculation for a particular row/column/element depends on the values of others.

When depends is provided, multiple rows/columns/elements will be passed to FUN. Each element of the depends list should be a vector giving the indices that should be passed to FUN.

For example, this can be used to implement a rolling apply function.

Value

Typically, a list if simplify=FALSE. Otherwise, the results may be coerced to a vector or array.

Author(s)

Kylie A. Bemis

See Also

apply, lapply, mapply,

Examples

```
register(SerialParam())

set.seed(1)
x <- matrix(rnorm(1000^2), nrow=1000, ncol=1000)

out <- chunkApply(x, 1L, mean, nchunks=10)
```

colscale *Scaling and Centering by Row or Column Based on Grouping*

Description

Apply the equivalent of `scale` to either columns or rows of a matrix, using a grouping variable.

Usage

```
## S4 method for signature 'ANY'
colscale(x, center=TRUE, scale=TRUE,
    group = NULL, ..., BPPARAM = bpparam())

## S4 method for signature 'ANY'
rowscale(x, center=TRUE, scale=TRUE,
    group = NULL, ..., BPPARAM = bpparam())
```

Arguments

`x`	A matrix-like object.
`center`	Either a logical value or a numeric vector of length equal to the number of columns of 'x' (for `colscale()`) or the number of the rows of 'x' (for `rowscale()`). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
`scale`	Either a logical value or a numeric vector of length equal to the number of columns of 'x' (for `colscale()`) or the number of the rows of 'x' (for `rowscale()`). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
`group`	A vector or factor giving the groupings with length equal to the number of rows of 'x' (for `colscale()`) or the number of the columns of 'x' (for `rowscale()`).
`...`	Arguments passed to `rowStats()` or `colStats()` respectively, if `center` or `scale` must be calculated.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`.

Details

See `scale` for details.

Value

A matrix-like object (usually of the same class as `x`) with either 'col-scaled:center' and 'col-scaled:scale' attributes or 'row-scaled:center' and 'row-scaled:scale' attributes.

Author(s)

Kylie A. Bemis

See Also

scale

Examples

```
x <- matter(1:100, nrow=10, ncol=10)

colscale(x)
```

---

colStats *Row and Column Summary Statistics Based on Grouping*

Description

These functions perform calculation of summary statistics over matrix rows and columns for each level of a grouping variable.

Usage

```
## S4 method for signature 'ANY'
rowStats(x, ..., BPPARAM = bpparam())

## S4 method for signature 'ANY'
colStats(x, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_mat'
rowStats(x, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_mat'
colStats(x, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat'
rowStats(x, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat'
colStats(x, ..., BPPARAM = bpparam())

.rowStats(x, stat, group = NULL,
    na.rm = FALSE, simplify = TRUE, drop = TRUE,
    iter.dim = 1L, BPPARAM = bpparam(), ...)

.colStats(x, stat, group = NULL,
    na.rm = FALSE, simplify = TRUE, drop = TRUE,
    iter.dim = 2L, BPPARAM = bpparam(), ...)
```

Arguments

x	A matrix on which to calculate summary statistics.
stat	The name of summary statistics to compute over the rows or columns of a matrix. Allowable values include: "min", "max", "prod", "sum", "mean", "var", "sd", "any", "all", and "nnzero".
group	A factor or vector giving the grouping. If not provided, no grouping will be used.
na.rm	If TRUE, remove NA values before summarizing.
simplify	Simplify the results from a list to a vector or array. This also drops any additional attributes (besides names).
drop	If only a single summary statistic is calculated, return the results as a vector (or matrix) rather than a list.
iter.dim	The dimension to iterate over. Must be 1 or 2, where 1 indicates rows and 2 indicates columns.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to chunk_rowapply() or chunk_colapply(), such as the number of chunks.

Details

The summary statistics methods are calculated over chunks of the matrix using s_colstats and s_rowstats. For matter objects, the iteration is performed over the major dimension for IO efficiency.

Value

A list for each stat requested, where each element is either a vector (if no grouping variable is provided) or a matrix where each column corresponds to a different level of group.

If drop=TRUE, and only a single statistic is requested, then the result will be unlisted and returned as a vector or matrix.

Author(s)

Kylie A. Bemis

See Also

colSums

Examples

```
register(SerialParam())

set.seed(1)

x <- matrix(runif(100^2), nrow=100, ncol=100)
```

```
g <- as.factor(rep(letters[1:5], each=20))

colStats(x, "mean", group=g)
```

---

colsweep *Sweep on Array Summaries Based on Grouping*

Description

Apply the equivalent of sweep to either columns or rows of a matrix, using a grouping variable.

Usage

```
## S4 method for signature 'ANY'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'matter_mat'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'sparse_mat'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'ANY'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'matter_mat'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'sparse_mat'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)
```

Arguments

x	A matrix-like object.
STATS	The summary statistic to be swept out, with length equal to the number of columns of 'x' (for colsweep()) or the number of the rows of 'x' (for rowsweep()). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
FUN	The function to be used to carry out the sweep.
group	A vector or factor giving the groupings with length equal to the number of rows of 'x' (for colsweep()) or the number of the columns of 'x' (for rowsweep()).
...	Ignored.

Details

See sweep for details.

Value

A matrix-like object (usually of the same class as x) with the statistics swept out.

Author(s)

Kylie A. Bemis

See Also

sweep

Examples

```
set.seed(1)

x <- matrix(1:100, nrow=10, ncol=10)

colsweep(x, colStats(x, "mean"))
```

---

deferred-ops *Deferred Operations on "matter" Objects*

Description

Some arithmetic, comparison, and logical operations are available as delayed operations on matter_arr and sparse_arr objects.

Details

Currently the following delayed operations are supported:

'Arith': '+', '-', '*', '/', '^', '

'Compare': '==', '>', '<', '!=', '<=', '>='

'Logic': '&', '|'

'Ops': 'Arith', 'Compare', 'Logic'

'Math': 'exp', 'log', 'log2', 'log10'

Arithmetic operations are applied in C++ layer immediately after the elements are read from virtual memory. This means that operations that are implemented in C and/or C++ for efficiency (such as summary statistics) will also reflect the execution of the deferred arithmetic operations.

Value

A new matter object with the registered deferred operation. Data in storage is not modified; only object metadata is changed.

Author(s)

Kylie A. Bemis

See Also

Arith, Compare, Logic, Ops, Math

Examples

```
x <- matter(1:100)
y <- 2 * x + 1

x[1:10]
y[1:10]

mean(x)
mean(y)
```

---

drle-class *Delta Run Length Encoding*

Description

The drle class stores delta-run-length-encoded vectors. These differ from other run-length-encoded vectors provided by other packages in that they allow for runs of values that each differ by a common difference (delta).

Usage

```
## Instance creation
drle(x, cr_threshold = 0)

is.drle(x)
## Additional methods documented below
```

Arguments

x	An integer or numeric vector to convert to delta run length encoding for drle(); an object to test if it is of class drle for is.drle().
cr_threshold	The compression ratio threshold to use when converting a vector to delta run length encoding. The default (0) always converts the object to drle. Values of cr_threshold < 1 correspond to compressing even when the output will be larger than the input (by a certain ratio). For values > 1, compression will only take place when the output is (approximately) at least cr_threshold times smaller.

Value

An object of class drle.

Slots

`values`: The values that begin each run.

`lengths`: The length of each run.

`deltas`: The difference between the values of each run.

Creating Objects

`drle` instances can be created through `drle()`.

Methods

Standard generic methods:

`x[i]`: Get the elements of the uncompressed vector.

`length(x)`: Get the length of the uncompressed vector.

`c(x, ...)`: Combine vectors.

Author(s)

Kylie A. Bemis

See Also

`rle`

Examples

```
## Create a drle vector
x <- c(1,1,1,1,1,6,7,8,9,10,21,32,33,34,15)
y <- drle(x)

# Check that their elements are equal
x == y[]
```

---

findpeaks *Find Peaks Based on Local Maxima*

Description

Find peaks in a signal based on its local maxima, as determined by a sliding window.

Usage

```
findpeaks(x, width = 5)

locmax(x, width = 5)
```

Arguments

x	A numeric vector.
width	The number of signal elements to consider when determining if the center of the sliding window is a local maximum.

Details

For this function, a local maximum is defined as an element greater than all of the elements within width / 2 elements to the left of it, and greater than or equal to all of the elements within width / 2 elements to the right of it.

The peaks in this case are simply the local maxima of the signal. The peak boundaries are found by descending a local maximum until a local minimum is found on either side, using the same criteria as above.

Value

For locmax(), an logical vector indicating whether each element is a local maximum.

For findpeaks(), an integer vector giving the indices of the peaks, with attributes 'left_bounds' and 'right_bounds' giving the left and right boundaries of the peak as determined using the rule above.

Author(s)

Kylie A. Bemis

Examples

```
x <- c(0, 1, 1, 2, 3, 2, 1, 4, 5, 1, 1, 0)

locmax(x)

findpeaks(x)
```

---

matter-class *Vectors, Matrices, and Arrays Stored in Virtual Memory*

Description

The matter class and its subclasses are designed for easy on-demand read/write access to binary virtual memory data structures, and working with them as vectors, matrices, arrays, lists, and data frames.

Usage

```
## Instance creation
matter(...)

# Check if an object is a matter object
is.matter(x)

# Coerce an object to a matter object
as.matter(x)

## Additional methods documented below
```

Arguments

`...`	Arguments passed to subclasses.
`x`	An object to check if it is a matter object or coerce to a matter object.

Value

An object of class `matter`.

Slots

`data`: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

`type`: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

`dim`: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

`names`: The names of the data elements for vectors.

`dimnames`: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

Creating Objects

`matter` is a virtual class and cannot be instantiated directly, but instances of its subclasses can be created through `matter()`.

Methods

Class-specific methods:

`atomdata(x)`: Access the 'data' slot.

`adata(x)`: An alias for atomdata(x).

`type(x)`, `type(x) <- value`: Get or set data 'type'.

Standard generic methods:

`length(x)`, `length(x) <- value`: Get or set length.

`dim(x)`, `dim(x) <- value`: Get or set 'dim'.

`names(x)`, `names(x) <- value`: Get or set 'names'.

`dimnames(x)`, `dimnames(x) <- value`: Get or set 'dimnames'.

Author(s)

Kylie A. Bemis

See Also

`matter_arr`, `matter_mat`, `matter_vec`, `matter_fct`, `matter_list`, `matter_str`

Examples

```
## Create a matter_vec vector
x <- matter(1:100, length=100)
x

## Create a matter_mat matrix
y <- matter(1:100, nrow=10, ncol=10)
y
```

`matter-datatypes` *Data Types for "matter" Objects*

Description

The `matter` package defines a number of data types for translating between data elements stored in virtual memory and data elements loaded into R. These are typically set and stored via the `datamode` argument and slot.

At the R level, `matter` objects may be any of the following data modes:

- `raw`:`matter` objects of this mode are typically vectors of raw bytes.
- `logical`:Any `matter` object that represents a logical vector or has had any Compare or Logic delayed operations applied to it will be of this type.
- `integer`:`matter` objects represented as integers in R.
- `numeric`:`matter` objects represented as doubles in R.
- `character`:`matter` objects representated as character vectors in R.

In virtual memory, `matter` objects may be composed of atomic units of the following data types:

- `char`:8-bit signed integer; defined as `char`.
- `uchar`:8-bit unsigned integer; used for 'Rbyte' or 'raw'; defined as `unsigned char`.
- `short`:16-bit signed integer; defined as `int16_t`.

- ushort:16-bit unsigned integer; defined as uint16_t.
- int:32-bit signed integer; defined as int32_t.
- uint:32-bit unsigned integer; defined as uint32_t.
- long:64-bit signed integer; defined as int64_t.
- ulong:64-bit unsigned integer; defined as uint64_t.
- float:Platform dependent, but usually a 32-bit float; defined as float.
- double:Platform dependent, but usually a 64-bit float; defined as double.

While a substantial effort is made to coerce data elements properly between data types, sometimes this cannot be done losslessly. This will generate a warning (typically *many* such warnings) that can be silenced by setting options(matter.cast.warning=FALSE).

Note that the unsigned data types do not support NA; coercion to signed short and long attempts to preserve missingness. The special values NaN, Inf, and -Inf are only supported by the floating-point types, and will be set to NA for signed integral types, and to 0 for unsigned integral types.

---

matter-options *Options for "matter" Objects*

Description

The matter package provides the following options:

- options(matter.cast.warning=TRUE):Should a warning be emitted when casting between data types results in a loss of precision?
- options(matter.compress.atoms=3):The compression ratio threshold to be used to determine when to compress atoms in a matter object. Setting to 0 or FALSE means that atoms are never compressed.
- options(matter.default.nchunks=20L):The default number of chunks to use when iterating over matter objects.
- options(matter.default.chunksize=1000000L):The default chunksize for new matter objects. This is the (suggested) maximum number of elements which should be accessed at once by summary functions and linear algebra. Ignored when explicitly subsetting the dataset. Must be an integer.
- options(matter.show.head=TRUE):Should a preview of the beginning of the data be displayed when the object is printed?
- options(matter.show.head.n=6):The number of elements, rows, and/or columns to be displayed by the object preview.
- options(matter.coerce.altrep=FALSE):When coercing matter objects to native R objects (such as matrix), should a matter-backed ALTREP object be returned instead? The initial coercion will be cheap, and the result will look like a native R object. This does not guarantee that the full data is never read into memory. Not all functions are ALTREP-aware at the C-level, so some operations may still trigger the full data to be read into memory. This should only ever happen once, as long as the object is not duplicated, though.

- options(matter.coerce.altrep.list=FALSE):Should a matter-backed ALTREP list be returned when coercing matter_list lists to native R lists? Lists are treated differently, because the coercion is more costly, as the metadata for each list element must be uncompressed and converted to separate ALTREP representations. (Note that this does not affect matter_df data frames, which do not compress metadata about the columns, because the columns are regular matter vectors.)
- options(matter.wrap.altrep=FALSE):When coercing to a matter-backed ALTREP object, should the object be wrapped in an ALTREP wrapper? (This is always done in cases where the coercion preserves existing attributes.) This allows setting of attributes without triggering a (potentially expensive) duplication of the object when safe to do so.
- options(matter.dump.dir=tempdir()):Temporary directory where matter object files should be dumped when created without user-specified file paths.

---

matter_arr-class *Out-of-Memory Arrays*

Description

The matter_arr class implements out-of-memory arrays.

Usage

```
## Instance creation
matter_arr(data, type = "double", path = NULL,
    dim = NA_integer_, dimnames = NULL, offset = 0, extent = NA_real_,
    readonly = NA, rowMaj = FALSE, ...)

matter_mat(data, type = "double", path = NULL,
    nrow = NA_integer_, ncol = NA_integer_, dimnames = NULL,
    offset = 0, extent = NA_real_, readonly = NA, rowMaj = FALSE, ...)

matter_vec(data, type = "double", path = NULL,
    length = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, rowMaj = FALSE, ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to virtual memory if provided.
type	A 'character' vector giving the storage mode of the data in virtual memory. Allowable values are the C types ('char', 'uchar', short', 'ushort', 'int', 'uint', 'long', 'ulong', 'float') and their R equivalents ('raw', 'logical', 'integer', 'numeric'). See ?datatypes for details.

path	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.
dim, nrow, ncol, length	The dimensions of the array, or the number of rows and columns, or the length.
dimnames, names	The names of the matrix dimensions or vector elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
rowMaj	Whether the data is stored in row-major or column-major order. The default is to use column-major order, which is the same as native R matrices.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_arr.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

ops: Deferred arithmetic operations.

transpose: Indicates whether the data is stored in row-major order (TRUE) or column-major order (FALSE). For a matrix, switching the order that the data is read is equivalent to transposing the matrix (without changing any data).

indexed: For matter_mat only. Indicates whether the pointers to rows or columns are indexed for quick access or not.

Extends

matter

Creating Objects

matter_arr instances can be created through matter_arr() or matter(). Matrices and vectors can also be created through matter_mat() and matter_vec().

Methods

Standard generic methods:

`length(x)`, `length(x) <- value`: Get or set length.

`dim(x)`, `dim(x) <- value`: Get or set 'dim'.

`names(x)`, `names(x) <- value`: Get or set 'names'.

`dimnames(x)`, `dimnames(x) <- value`: Get or set 'dimnames'.

`x[...]`, `x[...] <- value`: Get or set the elements of the array.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_arr(1:1000, dim=c(10,10,10))
x
```

`matter_fct-class` *Out-of-Memory Factors*

Description

The `matter_fct` class implements out-of-memory factors.

Usage

```
## Instance creation
matter_fct(data, levels, type = "integer", path = NULL,
    length = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, labels = as.character(levels), ...)

## Additional methods documented below
```

Arguments

`data`	An optional data vector which will be initially written to the data in virtual memory if provided.
`levels`	The levels of the factor.
`type`	Should be 'integer' type for factors.
`path`	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using `tempfile`.

length	The length of the factor.
names	The names of the data elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
labels	An optional character vector of labels for the factor levels.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_fct.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

levels: The levels of the factor.

labels: The labels for the levels.

Extends

matter, matter_vec

Creating Objects

matter_fct instances can be created through matter_fct() or matter().

Methods

Standard generic methods:

length(x), length(x) <- value: Get or set length.

names(x), names(x) <- value: Get or set 'names'.

x[i], x[i] <- value: Get or set the elements of the factor.

levels(x), levels(x) <- value: Get or set the levels of the factor.

Author(s)

Kylie A. Bemis

See Also

matter, matter_vec

Examples

```
x <- matter_fct(rep(c("a", "a", "b"), 5), levels=c("a", "b", "c"))
x
```

---

matter_list-class *Out-of-Memory Lists of Vectors*

Description

The matter_list class implements out-of-memory lists.

Usage

```
## Instance creation
matter_list(data, type = "double", path = NULL,
    lengths = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to virtual memory if provided.
type	A 'character' vector giving the storage mode of the data in virtual memory. Allowable values are the C types ('char', 'uchar', short', 'ushort', 'int', 'uint', 'long', 'ulong', 'float') and their R equivalents ('raw', 'logical', 'integer', 'numeric'). See ?datatypes for details.
path	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.
lengths	The lengths of the list elements.
names	The names of the list elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
...	Additional arguments to be passed to constructor.

Value

An object of class `matter_list`.

Slots

`data`: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

`type`: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

`dim`: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

`names`: The names of the data elements for vectors.

`dimnames`: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

Extends

`matter`

Creating Objects

`matter_list` instances can be created through `matter_list()` or `matter()`.

Methods

Standard generic methods:

`x[[i]]`, `x[[i]] <- value`: Get or set a single element of the list.

`x[[i, j]]`: Get the `j`th sub-elements of the `i`th element of the list.

`x[i]`, `x[i] <- value`: Get or set the `i`th elements of the list.

`lengths(x)`: Get the lengths of all elements in the list.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_list(list(c(TRUE,FALSE), 1:5, c(1.11, 2.22, 3.33)), lengths=c(2,5,3))
x[]
x[1]
x[[1]]

x[[3,1]]
x[[2,1:3]]
```

---

matter_str-class *Out-of-Memory Strings*

Description

The matter_str class implements out-of-memory strings.

Usage

```
## Instance creation
matter_str(data, type = "character", path = NULL,
    nchar = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, encoding = "unknown", ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to virtual memory if provided.
type	A 'character' vector giving the storage mode of the data in virtual memory. Allowable values are the C types ('char', 'uchar', short', 'ushort', 'int', 'uint', 'long', 'ulong', 'float') and their R equivalents ('raw', 'logical', 'integer', 'numeric'). See ?datatypes for details.
path	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.
nchar	A vector giving the length of each element of the character vector.
names	The names of the data elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
encoding	The character encoding to use (if known).
...	Additional arguments to be passed to constructor.

Value

An object of class matter_str.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

encoding: The string encoding used.

Extends

matter_list

Creating Objects

matter_str instances can be created through matter_str() or matter().

Methods

Standard generic methods:

x[i], x[i] <- value: Get or set the string elements of the vector.

lengths(x): Get the number of characters (in bytes) of all string elements in the vector.

Author(s)

Kylie A. Bemis

See Also

matter

Examples

```
x <- matter_str(rep(c("hello", "world!"), 50))
x
```

prcomp *Principal Components Analysis for "matter" Matrices*

Description

This method allows computation of a truncated principal components analysis of a matter_mat matrix using the implicitly restarted Lanczos method irlba.

Usage

```
## S4 method for signature 'matter_mat'
prcomp(x, n = 3, retx = TRUE, center = TRUE, scale. = FALSE, ...)
```

Arguments

x	A matter matrix.
n	The number of principal componenets to return, must be less than min(dim(x)).
retx	A logical value indicating whether the rotated variables should be returned.
center	A logical value indicating whether the variables should be shifted to be zero-centered, or a centering vector of length equal to the number of columns of x. The centering is performed implicitly and does not change the out-of-memory data in x.
scale.	A logical value indicating whether the variables should be scaled to have unit variance, or a scaling vector of length equal to the number of columns of x. The scaling is performed implicitly and does not change the out-of-memory data in x.
...	Additional options passed to irlba.

Value

An object of class 'prcomp'. See ?prcomp for details.

Note

The 'tol' truncation argument found in the default prcomp method is not supported. In place of the truncation tolerance in the original function, the argument n explicitly gives the number of principal components to return. A warning is generated if the argument 'tol' is used.

Author(s)

Kylie A. Bemis

See Also

bigglm

Examples

```
set.seed(1)

x <- matter_mat(rnorm(1000), nrow=100, ncol=10)

prcomp(x)
```

`profmem` *Profile Memory Use*

Description

These are utility functions for profiling memory used by objects and by R during the execution of an expression.

Usage

```
profmem(expr)

mem(x, reset = FALSE)
```

Arguments

`expr`	An expression to be evaluated.
`x`	An object, to identify how much memory it is using.
`reset`	Should the maximum memory used by R be reset?

Details

These are wrappers around the built-in `gc` function. Note that they only count memory managed by R.

Value

For `profmem`, a vector giving [1] the amount of memory used at the start of execution, [2] the amount of memory used at the end of execution, [3] the maximum amount of memory used during execution, [4] the memory overhead as defined by the maximum memory used minus the starting memory use, and [5] the execution time in seconds.

For `mem`, either a single numeric value giving the memory used by an object, or a vector providing a more readable version of the information returned by `gc` (see its help page for details).

Author(s)

Kylie A. Bemis

See Also

`gc`,

Examples

```
x <- 1:100

mem(x)

profmem(mean(x + 1))
```

`sparse_arr-class` *Sparse Vectors and Matrices*

Description

The `sparse_mat` class implements sparse matrices, potentially stored out-of-memory. Both compressed-sparse-column (CSC) and compressed-sparse-row (CSR) formats are supported. Sparse vectors are also supported through the `sparse_vec` class.

Usage

```
## Instance creation
sparse_mat(data, index, type = "double",
    nrow = NA_integer_, ncol = NA_integer_, dimnames = NULL,
    pointers = NULL, domain = NULL, offset = 0L, rowMaj = FALSE,
    tolerance = c(abs=0), sampler = "none", ...)

sparse_vec(data, index, type = "double",
    length = NA_integer_, names = NULL,
    domain = NULL, offset = 0L, rowMaj = FALSE,
    tolerance = c(abs=0), sampler = "none", ...)

# Check if an object is a sparse matrix
is.sparse(x)

# Coerce an object to a sparse matrix
as.sparse(x, ...)

## Additional methods documented below
```

Arguments

`data`	TODO
`index`	TODO
`type`	A 'character' vector giving the storage mode of the data in virtual memory. Allowable values are R numeric and logical types ('logical', 'integer', 'numeric') and their C equivalents.
`nrow, ncol, length`	The number of rows and columns, or the length of the array.

domain	Either NULL or a vector with length equal to the number of rows (for CSC matrices) or the number of columns (for CSR matrices). If NULL, then the 'key' portion of the key-value pairs that make up the non-zero elements are assumed to be row or column indices. If a vector, then they define the how the non-zero elements are matched to rows or columns. The 'key' portion of each non-zero element is matched against this canonical set of keys using binary search. Allowed types for keys are 'integer', 'numeric', and 'character'.
offset	TODO
rowMaj	Whether the data should be stored using compressed-sparse-row (CSR) representation (as opposed to compressed-sparse-column (CSC) representation). Defaults to 'FALSE', for efficient access to columns. Set to 'TRUE' for more efficient access to rows instead.
dimnames	The names of the sparse matrix dimensions.
names	The names of the sparse vector elements.
tolerance	For 'numeric' keys, the tolerance used for floating-point equality when determining key matches. The vector should be named. Use 'absolute' to use absolute differences, and 'relative' to use relative differences.
sampler	In the case of collisions when matching keys, how the row- or column-vectors should be combined. Acceptable values are "identity", "min", "max", "sum", and "mean". A user-specified function may also be provided. Using "identity" means collisions result in an error. Using "sum" or "mean" results in binning all matches.
pointers	TODO
x	An object to check if it is a sparse matrix or coerce to a sparse matrix.
...	Additional arguments to be passed to constructor.

Value

An object of class `sparse_mat`.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

index: TODO

pointers: TODO

`domain`: TODO

`offset`: TODO

`tolerance`: The tolerance to be used when matching indices from `index` to the `domain`. An attribute 'tol_type' gives whether 'absolute' or 'relative' differences should be used for the comparison.

`sampler`: This is a function determining how the row- or column-vectors should be combined (or not) when index matching collisions occur.

`ops`: Deferred arithmetic operations.

`transpose` Indicates whether the data is stored in row-major order (TRUE) or column-major order (FALSE). For a matrix, switching the order that the data is read is equivalent to transposing the matrix (without changing any data).

Warning

If 'data' is given as a length-2 list of key-value pairs, no checking is performed on the validity of the key-value pairs, as this may be a costly operation if the list is stored in virtual memory. Each element of the 'keys' element must be *sorted* in increasing order, or behavior may be unexpected.

Assigning a new data element to the sparse matrix will always sort the key-value pairs of the row or column into which it was assigned.

Extends

`matter`

Creating Objects

`sparse_mat` instances can be created through `sparse_mat()`.

Methods

Standard generic methods:

`x[i, j, ..., drop]`, `x[i, j] <- value`: Get or set the elements of the sparse matrix. Use `drop = NULL` to return a subset of the same class as the object.

`cbind(x, ...)`, `rbind(x, ...)`: Combine sparse matrices by row or column.

`t(x)`: Transpose a matrix. This is a quick operation which only changes metadata and does not touch the data representation.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matrix(rbinom(100, 1, 0.2), nrow=10, ncol=10)

y <- sparse_mat(x)
y[]
```

---

stream-stats *Streaming Summary Statistics*

Description

These functions allow calculation of streaming statistics. They are useful, for example, for calculating summary statistics on small chunks of a larger dataset, and then combining them to calculate the summary statistics for the whole dataset.

This is not particularly interesting for simpler, commutative statistics like `sum()`, but it is useful for calculating non-commutative statistics like running `sd()` or `var()` on pieces of a larger dataset.

Usage

```
# calculate streaming univariate statistics
s_range(x, ..., na.rm = FALSE)

s_min(x, ..., na.rm = FALSE)

s_max(x, ..., na.rm = FALSE)

s_prod(x, ..., na.rm = FALSE)

s_sum(x, ..., na.rm = FALSE)

s_mean(x, ..., na.rm = FALSE)

s_var(x, ..., na.rm = FALSE)

s_sd(x, ..., na.rm = FALSE)

s_any(x, ..., na.rm = FALSE)

s_all(x, ..., na.rm = FALSE)

s_nnzero(x, ..., na.rm = FALSE)

# calculate streaming matrix statistics
s_rowstats(x, stat, group, na.rm = FALSE, ...)

s_colstats(x, stat, group, na.rm = FALSE, ...)
```

```
# calculate combined summary statistics
stat_c(x, y, ...)
```

Arguments

x, y, ...	Object(s) on which to calculate a summary statistic, or a summary statistic to combine.
stat	The name of a summary statistic to compute over the rows or columns of a matrix. Allowable values include: "range", "min", "max", "prod", "sum", "mean", "var", "sd", "any", "all", and "nnzero".
group	A factor or vector giving the grouping. If not provided, no grouping will be used.
na.rm	If TRUE, remove NA values before summarizing.

Details

These summary statistics methods are intended to be applied to chunks of a larger dataset. They can then be combined either with the individual summary statistic functions, or with `stat_c()`, to produce the combined summary statistic for the full dataset. This is most useful for calculating running variances and standard deviations iteratively, which would be difficult or impossible to calculate on the full dataset.

The variances and standard deviations are calculated using running sum of squares formulas which can be calculated iteratively and are accurate for large floating-point datasets (see reference).

Value

For all univariate functions except `s_range()`, a single number giving the summary statistic. For `s_range()`, two numbers giving the minimum and the maximum values.

For `s_rowstats()` and `s_colstats()`, a vector of summary statistics.

Author(s)

Kylie A. Bemis

References

B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," Technometrics, vol. 4, no. 3, pp. 1-3, Aug. 1962.

B. O'Neill, "Some Useful Moment Results in Sampling Problems," The American Statistician, vol. 68, no. 4, pp. 282-296, Sep. 2014.

See Also

Summary

Examples

```
set.seed(1)
x <- sample(1:100, size=10)
y <- sample(1:100, size=10)

sx <- s_var(x)
sy <- s_var(y)

var(c(x, y))
stat_c(sx, sy) # should be the same

sxy <- stat_c(sx, sy)

# calculate with 1 new observation
var(c(x, y, 99))
stat_c(sxy, 99)

# calculate over rows of a matrix
set.seed(2)
A <- matrix(rnorm(100), nrow=10)
B <- matrix(rnorm(100), nrow=10)

sx <- s_rowstats(A, "var")
sy <- s_rowstats(B, "var")

apply(cbind(A, B), 1, var)
stat_c(sx, sy) # should be the same
```

---

struct *C-Style Structs Stored in Virtual Memory*

Description

This is a convenience function for creating and reading C-style structs in a single file stored in virtual memory.

Usage

```
struct(..., filename = NULL, readonly = FALSE, offset = 0)
```

Arguments

`...`	Named integers giving the members of the `struct`. They should be of the form `name=c(type=length)`.
`filename`	A single string giving the name of the file.
`readonly`	Should the file be treated as read-only?
`offset`	A scalar integer giving the offset from the beginning of the file.

Details

This is simply a convenient wrapper around the wrapper around matter_list that allows easy specification of C-style structs in a file.

Value

A object of class matter_list.

Author(s)

Kylie A. Bemis

See Also

matter_list

Examples

```
x <- struct(first=c(int=1), second=c(double=1))

x$first <- 2L
x$second <- 3.33

x$first
x$second
```

---

summary-stats *Summary Statistics for "matter" Objects*

Description

These functions efficiently calculate summary statistics for matter_arr objects. For matrices, they operate efficiently on both rows and columns.

Usage

```
## S4 method for signature 'matter_arr'
range(x, ..., na.rm)
## S4 method for signature 'matter_arr'
min(x, ..., na.rm)
## S4 method for signature 'matter_arr'
max(x, ..., na.rm)
## S4 method for signature 'matter_arr'
prod(x, ..., na.rm)
## S4 method for signature 'matter_arr'
mean(x, ..., na.rm)
## S4 method for signature 'matter_arr'
sum(x, ..., na.rm)
```

```
## S4 method for signature 'matter_arr'
sd(x, na.rm)
## S4 method for signature 'matter_arr'
var(x, na.rm)
## S4 method for signature 'matter_arr'
any(x, ..., na.rm)
## S4 method for signature 'matter_arr'
all(x, ..., na.rm)
## S4 method for signature 'matter_mat'
colMeans(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
colSums(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
rowMeans(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
rowSums(x, na.rm, dims = 1, ...)
```

Arguments

x	A matter_arr object.
...	Arguments passed to chunk_lapply().
na.rm	If TRUE, remove NA values before summarizing.
dims	Not used.

Details

These summary statistics methods operate on chunks of data which are loaded into memory and then freed before reading the next chunk.

For row and column summaries on matrices, the iteration scheme is dependent on the layout of the data. Column-major matrices will always be iterated over by column, and row-major matrices will always be iterated over by row. Row statistics on column-major matrices and column statistics on row-major matrices are calculated iteratively.

Variance and standard deviation are calculated using a running sum of squares formula which can be calculated iteratively and is accurate for large floating-point datasets (see reference).

Value

For mean, sd, and var, a single number. For the column summaries, a vector of length equal to the number of columns of the matrix. For the row summaries, a vector of length equal to the number of rows of the matrix.

Author(s)

Kylie A. Bemis

References

B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," Technometrics, vol. 4, no. 3, pp. 1-3, Aug. 1962.

See Also

stream_stat

Examples

```
x <- matter(1:100, nrow=10, ncol=10)

sum(x)
mean(x)
var(x)
sd(x)

colSums(x)
colMeans(x)

rowSums(x)
rowMeans(x)
```

---

uuid *Universally Unique Identifiers*

Description

Generate a UUID.

Usage

```
uuid(uppercase = FALSE)

hex2raw(x)

raw2hex(x, uppercase = FALSE)
```

Arguments

x	A vector of to convert between raw bytes and hexadecimal strings.
uppercase	Should the result be in uppercase?

Details

uuid generates a random universally unique identifier.

hex2raw converts a hexadecimal string to a raw vector.

raw2hex converts a raw vector to a hexadecimal string.

Value

For `uuid`, a list of length 2:

- `string`: A character vector giving the UUID.
- `bytes`: The `raw` bytes of the UUID.

For `hex2raw`, a `raw` vector.

For `raw2hex`, a character vector of length 1.

Author(s)

Kylie A. Bemis

Examples

```
id <- uuid()
id
hex2raw(id$string)
raw2hex(id$bytes)
```

Index