Package 'matter'

October 16, 2019

Type Package

Title A framework for rapid prototyping with binary data on disk

Version 1.10.0

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 on disk as vectors, matrices, arrays, lists, and data frames.

License Artistic-2.0

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

Imports BiocGenerics, digest, irlba, utils

Suggests BiocStyle, testthat

Collate matterGenerics.R utils.R drle.R atoms.R matter.R matter_vec.R matter_mat.R matter_arr.R matter_list.R matter_str.R matter_fc.R matter_tbl.R matter_vt.R matter_df.R sparse_mat.R virtual_mat.R rep_vt.R coerce.R stats.R apply.R scale.R biglm.R prcomp.R

biocViews Software, Infrastructure

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

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

git_branch RELEASE_3_9

git_last_commit 2a0b5e5

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

R topics documented:

`apply` *Apply Functions Over "matter" Matrices*

Description

An implementation of `apply` for `matter_mat`, `sparse_mat` and `virtual_mat` matrices.

Usage

```
## S4 method for signature 'matter_mat'
apply(X, MARGIN, FUN, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat'
apply(X, MARGIN, FUN, ..., BPPARAM = bpparam())

## S4 method for signature 'virtual_mat'
apply(X, MARGIN, FUN, ..., BPPARAM = bpparam())
```

Arguments

`X`	A `matter` matrix-like object.
`MARGIN`	Must be 1 or 2 for `matter_mat` matrices, 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.

...	Additional arguments to be passed to FUN.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Details

Because FUN must be executed by the interpreter in the appropriate R environment, the full row or column will be loaded into memory. The chunksize of X is ignored. For summary statistics, functions like colMeans and rowMeans offer greater control over memory pressure. When performed in parallel, the matter metadata is serialized to each R session, so all workers must be able to access the data via the same paths().

Value

See apply for details.

Warning

Applying a function over the rows of a column-major matrix (e.g., matter_matc) or over the columns of a row-major matrix (e.g., matter_matr) may be very slow.

Author(s)

Kylie A. Bemis

See Also

apply

Examples

```
register(SerialParam())

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

apply(x, 2, summary)
```

---

biglm *Using "biglm" with "matter"*

Description

This method allows matter_mat matrices to be used with the biglm and bigglm function from the "biglm" package.

Usage

```
## S4 method for signature 'formula,matter_df'
biglm(formula, data, weights = NULL, sandwich = FALSE)

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

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

Arguments

formula	A model formula.
data	A matter matrix with column names.
weights	A one-sided, single-term formula specifying weights.
sandwich	If TRUE, compute the Huber/White sandwich covariance matrix (uses p^4 memory rather than p^2).
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)
```

---

bsearch *Binary Search with Approximate Matching*

Description

Given a set of keys and a sorted (non-decreasing) vector of values, use a binary search to find the indexes in values that match the values of key. This implementation allows for returning the index of the nearest match if there are no exact matches. It also allows specifying a tolerance for comparison of doubles.

Usage

```
bsearch(key, values, tol = 0, tol.ref = "none",
nomatch = NA_integer_, nearest = FALSE)
```

Arguments

key	A vector of keys to match.
values	A sorted (non-decreasing) vector of values to be matched.
tol	The tolerance for matching doubles. Must be >= 0.
tol.ref	One of 'none', 'key', or 'values'. If 'none', then comparison of doubles is done by taking the absolute difference. If either 'key' or 'values', 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.)
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, the first match that is found will be returned, but there are no guarantees about *which* match is found.

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 key, giving the indexes of the matches in values.

Author(s)

Kylie A. Bemis

See Also

match, pmatch, findInterval

Examples

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

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

y <- c("hello", "world!")
bsearch("world!", y) # 2
bsearch("worl", y) # NA
bsearch("worl", y, 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 = c("sha1", "md5"), ...)
```

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
```

combiner *Get or Set combiner for an Object*

Description

This is a generic function for getting or setting the 'combiner' for an object with values to combine.

Usage

```
combiner(object)

combiner(object) <- value
```

Arguments

object	An object with a combiner.
value	The value to set the combiner.

Author(s)

Kylie A. Bemis

See Also

sparse_mat

Examples

```
x <- sparse_mat(diag(10))
combiner(x)
combiner(x) <- "sum"
x[]
```

delayed-ops *Delayed Operations on "matter" Objects*

Description

Some arithmetic, comparison, and logical operations are available as delayed operations on matter objects. With these operations, no out-of-memory data is changed, and the operation is only executed when elements of the object are actually accessed.

Details

Currently the following delayed operations are supported:

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

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

'Logic': '&', 'I'

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

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

Delayed operations are applied at the 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 delayed operations.

Value

A new `matter` object with the registered delayed 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, delta = TRUE)

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.
delta	Should non-zero deltas be considered by the encoding? (Default TRUE.) If FALSE, then ordinary run-length-encoding is used.

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[]
```

keys *Get or Set Keys for an Object*

Description

This is a generic function for getting or setting 'keys' for an object with key-value pairs such as a map data structure.

Usage

```
keys(object)

keys(object) <- value
```

Arguments

object	An object with keys.
value	The value to set the keys.

Author(s)

Kylie A. Bemis

See Also

sparse_mat

Examples

```
x <- sparse_mat(diag(10))
keys(x)
keys(x) <- 1:10
x[]
```

lapply *Apply Functions Over "matter" Lists*

Description

An implementation of lapply and sapply for matter_list objects.

Usage

```
## S4 method for signature 'matter_list'
lapply(X, FUN, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_list'
sapply(X, FUN, ..., BPPARAM = bpparam(),
simplify = TRUE, USE.NAMES = TRUE)
```

Arguments

X	A matter list-like object.
FUN	The function to be applied.
...	Additional arguments to be passed to FUN.
simplify	Should the result be simplified into a vector, matrix, or higher dimensional array?
USE.NAMES	Use names(X) for the names of the answer. If X is a character, use X as names unless it has names already.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Details

Because FUN must be executed by the interpreter in the appropriate R environment, the full list element will be loaded into memory. The chunksize of X is ignored. When performed in parallel, the matter metadata is serialized to each R session, so all workers must be able to access the data via the same paths().

Value

See lapply for details.

Author(s)

Kylie A. Bemis

See Also

lapply

Examples

```
register(SerialParam())

x <- matter_list(list(1:10, b=11:20, 21:30), names=c("a", "b", "c"))

lapply(x, sum)

sapply(x, sum)
```

---

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 the information about locations of the data in virtual memory and within files.

`datamode`: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

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).

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

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

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

`readonly(x)`, `readonly(x) <- value`: A shortcut for getting or setting 'filemode'.

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

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_vec`, `matter_mat`, `matter_arr`, `matter_list`, `matter_fc`, `matter_str`, `matter_df`

Examples

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

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

---

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; this data mode is also use for `matter_str` to represent strings.
- `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.
- virtual:A number of matter objects do not necessarily represent out-of-memory data, or may include a number of components mixed between virtual memory and real memory; these will use this data mode.

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-utils *Internal Utilities for "matter" Package*

Description

Low-level utility functions, classes, and data defined in the **matter** package. They are not intended to be used directly.

---

matter_arr-class *Out-of-Memory Arrays*

Description

The matter_arr class implements out-of-memory arrays.

Usage

```
## Instance creation
matter_arr(data, datamode = "double", paths = NULL,
            filemode = ifelse(all(file.exists(paths)), "r", "rw"),
            offset = 0, extent = prod(dim), dim = 0, dimnames = NULL, ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to virtual memory if provided.
datamode	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.
paths	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using tempfile.
filemode	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
dim	A vector giving the dimensions of the array.
dimnames	The names of the matrix dimensions.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_arr.

Slots

data: This slot stores the information about locations of the data in virtual memory and within the files.

datamode: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

paths: A 'character' vector of the paths to the files where the data are stored.

filemode: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

chunksize: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

length: The length of the data.

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: Delayed operations to be applied on atoms.

Extends

matter

Creating Objects

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

Methods

Standard generic methods:

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

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_arr(1:125, dim=c(5,5,5))
x[]
```

`matter_df-class` *Out-of-Memory Data Frames*

Description

The `matter_df` class implements out-of-memory data frames.

The columns may be `matter` objects or ordinary R objects. If they are ordinary R objects, those columns remain in memory. Columns that should be stored out-of-memory should be manually coerced to `matter` objects, if they are not already.

Calling `as.matter()` on an ordinary R `data.frame` will do this automatically.

Usage

```
## Instance creation
matter_df(..., row.names = NULL)

## Additional methods documented below
```

Arguments

`...`	These arguments become the data columns or data frame variables. They should be named.
`row.names`	A character vector giving the row names.

Value

An object of class `matter_df`.

Slots

`data`: This slot stores the information about locations of the data in virtual memory and within the files.

`datamode`: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

Extends

`matter`

Creating Objects

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

Methods

Standard generic methods:

`x$name, x$name <- value`: Get or set a single column.

`x[[i]], x[[i]] <- value`: Get or set a single column.

`x[i], x[i] <- value`: Get or set multiple columns.

`x[i, j, ..., drop], x[i, j] <- value`: Get or set the elements of the data frame.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_df(a=as.matter(1:10), b=11:20, c=as.matter(letters[1:10]))
x[]
x[1:2]
x[[2]]
x[["c"]]
x[,"c"]
x[1:5,c("a","c")]
x$c
x$c[1:5]
```

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

Description

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

Usage

```
## Instance creation
matter_fc(data, datamode = "int", paths = NULL,
            filemode = ifelse(all(file.exists(paths)), "r", "rw"),
            offset = 0, extent = length, length = 0L, names = NULL,
            levels = base::levels(as.factor(data)), ...)

## Additional methods documented below
```

Arguments

`data`	An optional data vector which will be initially written to the data in virtual memory if provided.
`datamode`	Must be an integral type for factors.
`paths`	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using `tempfile`.
`filemode`	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
`offset`	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
`length`	An optional number giving the total length of the data across all files, equal to the sum of 'extent'. This is ignored and calculated automatically if 'extent' is specified.
`names`	The names of the data elements.
`levels`	The levels of the factor.
`...`	Additional arguments to be passed to constructor.

Value

An object of class `matter_fc`.

Slots

`data`: This slot stores the information about locations of the data in virtual memory and within the files.

`datamode`: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

`levels`: The levels of the factor.

Extends

`matter`, `matter_vec`

Creating Objects

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

Methods

Standard generic methods:

`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_fc(c("a", "a", "b"), 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, datamode = "double", paths = NULL,
            filemode = ifelse(all(file.exists(paths)), "r", "rw"),
            offset = c(0, cumsum(sizeof(datamode) * extent)[-length(extent)]),
            extent = lengths, lengths = 0, names = NULL, dimnames = NULL, ...)

## Additional methods documented below
```

Arguments

data	An optional data list which will be initially written to the data in virtual memory if provided.
datamode	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.
paths	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using tempfile.
filemode	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
lengths	A vector giving the length of each element of the list.
names	The names of the data elements.
dimnames	The names of the data elements' data elements.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_list.

Slots

data**:** This slot stores the information about locations of the data in virtual memory and within the files.

datamode**:** The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

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_mat-class` *Out-of-Memory Matrices*

Description

The `matter_mat` class implements out-of-memory matrices.

Usage

```
## Instance creation
matter_mat(data, datamode = "double", paths = NULL,
            filemode = ifelse(all(file.exists(paths)), "r", "rw"),
            offset = c(0, cumsum(sizeof(datamode) * extent)[-length(extent)]),
            extent = if (rowMaj) rep(ncol, nrow) else rep(nrow, ncol),
            nrow = 0, ncol = 0, rowMaj = FALSE, dimnames = NULL, ...)

## Additional methods documented below
```

Arguments

`data`	An optional data matrix which will be initially written to the data in virtual memory if provided.
`datamode`	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.
`paths`	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using `tempfile`.
`filemode`	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
`offset`	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
`nrow`	An optional number giving the total number of rows.
`ncol`	An optional number giving the total number of columns.
`rowMaj`	Whether the data should be stored in row-major order (as opposed to column-major order) in virtual memory. Defaults to 'FALSE', for efficient access to columns. Set to 'TRUE' for more efficient access to rows instead.
`dimnames`	The names of the matrix dimensions.
`...`	Additional arguments to be passed to constructor.

Value

An object of class `matter_mat`.

Slots

`data`: This slot stores the information about locations of the data in virtual memory and within the files.

`datamode`: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

Extends

`matter`

Creating Objects

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

Methods

Standard generic methods:

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

`x %*% y`: Matrix multiplication. At least one matrix must be an in-memory R matrix (or vector).

`crossprod(x, y)`: Alias for t(x) %*% y.

`tcrossprod(x, y)`: Alias for x %*% t(y).

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

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

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_mat(1:100, nrow=10, ncol=10)
x[]
```

`matter_str-class` *Out-of-Memory Strings*

Description

The `matter_str` class implements out-of-memory strings.

Usage

```
## Instance creation
matter_str(data, datamode = "uchar", paths = NULL,
            filemode = ifelse(all(file.exists(paths)), "r", "rw"),
            offset = c(0, cumsum(sizeof("uchar") * extent)[-length(extent)]),
            extent = nchar, nchar = 0, names = NULL,
            encoding = "unknown", ...)

## Additional methods documented below
```

Arguments

`data`	An optional character vector which will be initially written to the data in virtual memory if provided.
`datamode`	Must be "uchar" (or "raw") for strings.
`paths`	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using `tempfile`.
`filemode`	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
`offset`	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
`nchar`	A vector giving the length of each element of the character vector.
`names`	The names of the data elements.
`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 the information about locations of the data in virtual memory and within the files.

`datamode`: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

`encoding`: The character encoding of the strings.

Extends

`matter`

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(c("hello", "world!"))
x[]
```

matter_vec-class *Out-of-Memory Vectors*

Description

The matter_vec class implements out-of-memory vectors.

Usage

```
## Instance creation
matter_vec(data, datamode = "double", paths = NULL,
           filemode = ifelse(all(file.exists(paths)), "r", "rw"),
           offset = 0, extent = length, length = 0L, names = NULL, ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to the data in virtual memory if provided.
datamode	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.
paths	A 'character' vector of the paths to the files where the data are stored. If 'NULL', then a temporary file is created using tempfile.
filemode	The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'paths', 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 'paths', specifying the number of elements of size 'datamode' to be accessed from each file.
length	An optional number giving the total length of the data across all files, equal to the sum of 'extent'. This is ignored and calculated automatically if 'extent' is specified.
names	The names of the data elements.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_vec.

Slots

data: This slot stores the information about locations of the data in virtual memory and within the files.

datamode: The storage mode of the *accessed* data when read into R. This is a 'character' vector of with possible values 'raw', 'logical', 'integer', 'numeric', or 'virtual'.

`paths`: A 'character' vector of the paths to the files where the data are stored.

`filemode`: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

`chunksize`: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

`length`: The length of the data.

`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`: Delayed operations to be applied on atoms.

Extends

`matter`

Creating Objects

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

Methods

Standard generic methods:

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

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

`t(x)`: Transpose a vector (to a row matrix). This is a quick operation which only changes metadata and does not touch the out-of-memory data.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_vec(1:100)
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))
```

`rep_vt-class` *Virtual Replication of Vectors*

Description

The `rep_vt` class simulates the behavior of the base function `rep` without actually allocating memory for the duplication. Only the original vector and the expected length of the result are stored. All attributes of the original vector (including names) are dropped.

Usage

```
## Instance creation
rep_vt(x, times, length.out = length(x) * times)

## Additional methods documented below
```

Arguments

`x`	A vector (of any mode).
`times`	The number of times to repeat the whole vector.
`length.out`	The desired length of the result.

Value

An object of class `rep_vt`.

Slots

`data`: The original vector.

`length`: The expected length of the repeated virtual vector.

Creating Objects

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

Methods

Standard generic methods:

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

`x[[i]]`: Get a single element of the uncompressed vector.

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

Author(s)

Kylie A. Bemis

See Also

[base]{rep}

Examples

```
## Create a rep_vt vector
init <- 1:3
x <- rep(init, length.out=100)
y <- rep_vt(init, length.out=100)

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

---

scale *Scaling and Centering of "matter" Matrices*

Description

An implementation of scale for matter_mat matrices.

Usage

```
## S4 method for signature 'matter_mat'
scale(x, center = TRUE, scale = TRUE)
```

Arguments

x	A matter_mat object.
center	Either a logical value or a numeric vector of length equal to the number of columns of 'x'.
scale	Either a logical value or a numeric vector of length equal to the number of columns of 'x'.

Details

See scale for details.

Value

A matter_mat object with the appropriate 'scaled:center' and 'scaled:scale' attributes set. No data in virtual memory is changed, but the scaling will be applied any time the data is read. This includes but is not limited to loading data elements via subsetting, summary statistics methods, and matrix multiplication.

Author(s)

Kylie A. Bemis

See Also

scale

Examples

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

scale(x)
```

---

sparse_mat-class *Sparse 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. Non-zero elements are internally represented as key-value pairs.

Usage

```
## Instance creation
sparse_mat(data, datamode = "double", nrow = 0, ncol = 0,
            rowMaj = FALSE, dimnames = NULL, keys = NULL,
            tolerance = c(abs=0), combiner = "identity", ...)

# 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	Either a length-2 'list' with elements 'keys' and 'values' which provide the halves of the key-value pairs of the non-zero elements, or a data matrix that will be used to initialized the sparse matrix. If a list is given, all 'keys' elements must be *sorted* in increasing order.
datamode	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	An optional number giving the total number of rows.
ncol	An optional number giving the total number of columns.
keys	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'.

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.
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.
combiner	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.
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: A length-2 'list' with elements 'keys' and 'values' which provide the halves of the key-value pairs of the non-zero elements.

datamode: The storage mode of the accessed data when read into R. This should a 'character' vector of length one with value 'integer' or 'numeric'.

paths: A 'character' vector of the paths to the files where the data are stored.

filemode: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

chunksize: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

length: The length of the data.

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: Delayed operations to be applied on atoms.

keys 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'.

tolerance: For 'numeric' keys, the tolerance used for floating-point equality when determining key matches. An attribute 'type' gives whether 'absolute' or 'relative' differences should be used for the comparison.

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

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

```
keys <- list(
    c(1,4,8,10),
    c(2,3,5),
    c(1,2,7,9))

values <- list(
    rnorm(4),
    rnorm(3),
    rnorm(4))

init1 <- list(keys=keys, values=values)

x <- sparse_mat(init1, nrow=10)
x[]

init2 <- matrix(rbinom(100, 1, 0.2), nrow=10, ncol=10)
```

```
y <- sparse_mat(init2, keys=letters[1:10])
y[]
```

`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, filemode = "rw", 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.
`filemode`	The mode to use to open the file.
`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` objects. For matrices, they operate efficiently on both rows and columns.

Usage

```
## S4 method for signature 'matter'
range(x, na.rm)
## S4 method for signature 'matter'
min(x, na.rm)
## S4 method for signature 'matter'
max(x, na.rm)
## S4 method for signature 'matter'
prod(x, na.rm)
## S4 method for signature 'matter'
mean(x, na.rm)
## S4 method for signature 'matter'
sum(x, na.rm)
## S4 method for signature 'matter'
sd(x, na.rm)
## S4 method for signature 'matter'
var(x, na.rm)
## S4 method for signature 'matter'
any(x, na.rm)
## S4 method for signature 'matter'
all(x, na.rm)
## S4 method for signature 'matter_mat'
colMeans(x, na.rm)
## S4 method for signature 'matter_mat'
colSums(x, na.rm)
## S4 method for signature 'matter_mat'
colSds(x, na.rm)
## S4 method for signature 'matter_mat'
colVars(x, na.rm)
## S4 method for signature 'matter_mat'
rowMeans(x, na.rm)
## S4 method for signature 'matter_mat'
rowSums(x, na.rm)
## S4 method for signature 'matter_mat'
rowSds(x, na.rm)
## S4 method for signature 'matter_mat'
rowVars(x, na.rm)
```

Arguments

`x`	A `matter` object.
`na.rm`	If `TRUE`, remove `NA` values before summarizing.

Details

These summary statistics methods operate on chunks of data (equal to the `chunksize` of `x`) 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.

The efficiency of these methods is entirely dependent on the `chunksize` of `x`. Larger chunks will yield faster calculations, but greater memory usage. The row and column summary methods may be more or less efficient than the equivalent call to `apply`, depending on the chunk size.

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

`colSums`

Examples

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

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

colSums(x)
colMeans(x)
colVars(x)
colSds(x)

rowSums(x)
rowMeans(x)
rowVars(x)
rowSds(x)
```

tolerance *Get or Set Tolerance for an Object*

Description

This is a generic function for getting or setting 'tolerance' for an object which tests floating point equality.

Usage

```
tolerance(object)

tolerance(object) <- value
```

Arguments

object	An object with tolerance.
value	The value to set the tolerance.

Author(s)

Kylie A. Bemis

See Also

sparse_mat

Examples

```
x <- sparse_mat(diag(10), keys=rnorm(10))
tolerance(x)
tolerance(x) <- c(absolute=0.1)
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)
```

---

virtual_mat-class *Virtual Matrices*

Description

The virtual_mat class implements virtual matrices, which may hold any matrix-like objects. It is provided primarily to allow combining of matter matrix classes that could not be combined otherwise.

Usage

```
## Instance creation
virtual_mat(data, datamode = "double", rowMaj = FALSE,
            dimnames = NULL, index = NULL, ...)

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

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

## Additional methods documented below
```

Arguments

data	A list of matrices or vectors to combine.
datamode	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.
rowMaj	Whether the matrices in data are combined by row (TRUE) or by column (FALSE.
dimnames	The names of the virtual matrix dimensions.
index	A length-2 list of row and column indices giving a submatrix, if desired.
x	An object to check if it is a virtual matrix or coerce to a virtual matrix.
...	Additional arguments to be passed to constructor.

Value

An object of class virtual_mat.

Slots

data: A list of the original matrices or row/column-vectors.

datamode: The storage mode of the accessed data when read into R. This should a 'character' vector of length one with value 'integer' or 'numeric'.

paths: A 'character' vector of the paths to the files where the data are stored.

filemode: The read/write mode of the files where the data are stored. This should be 'r' for read-only access, or 'rw' for read/write access.

chunksize: The maximum number of elements which should be loaded into memory at once. Used by methods implementing summary statistics and linear algebra. Ignored when explicitly subsetting the dataset.

length: The length of the data.

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: Delayed operations to be applied on atoms.

index A length-2 list of row and column indices giving a virtual submatrix.

transpose TRUE if the virtual matrix should be transposed, and FALSE otherwise.

Extends

matter

Creating Objects

virtual_mat instances can be created through virtual_mat().

Methods

Standard generic methods:

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

`cbind(x, ...)`, `rbind(x, ...)`: Combine virtual 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(runif(50), nrow=10, ncol=5)

x <- virtual_mat(list(x, x))
x[]
```

Index