Package ‘houba’

February 20, 2026

Type Package

Title Manipulation of (Large) Memory-Mapped Objects (Vectors, Matrices and Arrays)

Version 0.1.1

Date 2026-02-20

Maintainer Hervé Perdry <herve.perdry@universite-paris-saclay.fr>

Description Manipulate data through memory-mapped files, as vectors, matrices or arrays. Basic arithmetic functions are implemented, but currently no matrix arithmetic. Can write and read descriptor files for compatibility with the 'bigmemory' package.

License CeCILL-2

Encoding UTF-8

Depends methods

Imports Rcpp (>= 1.0.11)

Suggests knitr, bigmemory

VignetteBuilder knitr

LinkingTo Rcpp

RoxygenNote 7.3.2

NeedsCompilation yes

Author Hervé Perdry [aut, cre], Juliette Meyniel [aut]

Repository CRAN

Date/Publication 2026-02-20 18:30:02 UTC

Contents

`apply` *Apply functions over margins of a mmatrix*

Description

This method generalizes ‘base::apply‘ to mmatrix objects

Usage

```
## S4 method for signature 'mmatrix'
apply(X, MARGIN, FUN, ..., simplify = TRUE)
```

Arguments

`X`	a mmatrix
`MARGIN`	an integer giving the subscript which the function will be applied over
`FUN`	the function to be applied
`...`	extra arguments for ‘FUN‘
`simplify`	a logical indicating whether the results should be simplified

Details

If ’simplify’ is TRUE the result will be a vector or a matrix, depending on the size of the values returned by ’FUN’. If the size of this object is greater than `houba(max.size)`, then it will be memory-mapped (i.e., either a mvector or a mmatrix). If ’simplify’ is FALSE, the result is a list.

The function extracts the rows or the columns of ’X’ one by one, to a R object, which is passed to ’FUN’.

Value

If 'simplify' is TRUE, a matrix (or a mmatrix) or a vector (or a mvector). If 'simplify' is FALSE, a list.

See Also

base:apply

Examples

```
a <- matrix(1:6, 2, 3)
A <- as.mmatrix(a)
apply(A, 1, var)
apply(A, 2, var)
```

`Arithmetic` *Arithmetic Operators*

Description

Arithmetic operators for memory mapped objects

Usage

```
## S4 method for signature 'mmatrixOrMarray,mvectorOrNumeric'
e1 + e2

## S4 method for signature 'mvectorOrNumeric,mmatrixOrMarray'
e1 + e2

## S4 method for signature 'mmatrixOrMarray,array'
e1 + e2

## S4 method for signature 'array,mmatrixOrMarray'
e1 + e2

## S4 method for signature 'mmatrixOrMarray,mmatrixOrMarray'
e1 + e2

## S4 method for signature 'mmatrixOrMarray,mvectorOrNumeric'
e1 - e2

## S4 method for signature 'mvectorOrNumeric,mmatrix'
e1 - e2

## S4 method for signature 'mvectorOrNumeric,marray'
```

```
e1 - e2

## S4 method for signature 'mmatrixOrMarray,array'
e1 - e2

## S4 method for signature 'matrix,mmatrix'
e1 - e2

## S4 method for signature 'array,marray'
e1 - e2

## S4 method for signature 'mmatrixOrMarray,mmatrixOrMarray'
e1 - e2

## S4 method for signature 'mmatrixOrMarray,missing'
e1 - e2

## S4 method for signature 'mmatrixOrMarray,mvectorOrNumeric'
e1 * e2

## S4 method for signature 'mvectorOrNumeric,mmatrixOrMarray'
e1 * e2

## S4 method for signature 'mmatrixOrMarray,array'
e1 * e2

## S4 method for signature 'array,mmatrixOrMarray'
e1 * e2

## S4 method for signature 'mmatrixOrMarray,mmatrixOrMarray'
e1 * e2

## S4 method for signature 'mmatrixOrMarray,mvectorOrNumeric'
e1 / e2

## S4 method for signature 'mvectorOrNumeric,mmatrix'
e1 / e2

## S4 method for signature 'mvectorOrNumeric,marray'
e1 / e2

## S4 method for signature 'mmatrixOrMarray,array'
e1 / e2

## S4 method for signature 'matrix,mmatrix'
e1 / e2

## S4 method for signature 'array,marray'
```

```
e1 / e2

## S4 method for signature 'mmatrixOrMarray,mmatrixOrMarray'
e1 / e2

## S4 method for signature 'mvector,mvectorOrNumeric'
e1 + e2

## S4 method for signature 'numeric,mvector'
e1 + e2

## S4 method for signature 'mvector,mvectorOrNumeric'
e1 - e2

## S4 method for signature 'numeric,mvector'
e1 - e2

## S4 method for signature 'mvector,missing'
e1 - e2

## S4 method for signature 'mvector,mvectorOrNumeric'
e1 * e2

## S4 method for signature 'numeric,mvector'
e1 * e2

## S4 method for signature 'mvector,mvectorOrNumeric'
e1 / e2

## S4 method for signature 'numeric,mvector'
e1 / e2
```

Arguments

`e1`	first operand
`e2`	second operand

Details

The usual operations are performed. Values are recycled if necessary. There's no type promotion: if one of the operands is a R object and the other is a memory-mapped object, the result will be a memory mapped object with same data type as the operand. If both operand are memory mapped objects with different data types, the result will be a memory mapped object with the same data type than the left operand.

Value

an object of class mvector, mmatrix or marray depending on the operand classes.

See Also

inplace

Examples

```
x <- as.mvector(2**(1:4))
y <- 2*x
x <- x/2
x + c(1,2) / y
```

---

as.array.marray *Converting memory-mapped objects to R objects*

Description

Converting memory-mapped objects to R objects

Usage

```
## S3 method for class 'marray'
as.array(x, ...)

## S3 method for class 'marray'
as.vector(x, mode = "any")

## S3 method for class 'mmatrix'
as.matrix(x, ...)

## S3 method for class 'mmatrix'
as.vector(x, mode = "any")

## S3 method for class 'mvector'
as.vector(x, mode = "any")
```

Arguments

x	memory-mapped object to convert
...	extra parameters (ignored)
mode	the mode oh the created vector

Value

an array

Examples

```
a <- array( 1:24, c(2,3,4) )
A <- as.marray(a)
all(as.array(A) == a)
as.vector(A)
```

---

as.marray *Conversion of R objects to memory mapped objects*

Description

Conversion of R objects to memory mapped objects

Usage

```
as.marray(x, datatype, filename)

## S4 method for signature 'array'
as.marray(x, datatype, filename)

as.mmatrix(x, datatype, filename)

## S4 method for signature 'matrix'
as.mmatrix(x, datatype, filename)

as.mvector(x, datatype, filename)

## S4 method for signature 'numeric'
as.mvector(x, datatype, filename)
```

Arguments

`x`	an r object
`datatype`	(optional) type of the memory mapped object
`filename`	(optional) path to file

Details

If 'filename' is a path to an existing file, the function will raise an error. If you need to overwrite a file, unlink it first.

Value

A memmory-mapped object, of class 'mvector', 'mmatrix' or 'marray'

Examples

```
a <- matrix(1:6, 2)
A <- as.mmatrix(a)
B <- as.mmatrix(a, "float")
A
B
```

---

colMeans,mmatrix-method

Row and Columns sums and means

Description

Methods generalizing the base methods to mmatrix objects

Usage

```
## S4 method for signature 'mmatrix'
colMeans(x, output.type)

## S4 method for signature 'mmatrix'
colSums(x, output.type)

## S4 method for signature 'mmatrix'
rowMeans(x, output.type)

## S4 method for signature 'mmatrix'
rowSums(x, output.type)
```

Arguments

`x`	a dual matrix or array
`output.type`	type of the result, if it's a mvector (see details)

Details

If the size of the result is greater than `houba(max.size)`, then it will be a mvector instead of R object. In this case its type will be determined using 'output.type'. If 'output.type' is missing, a coherent choice will be made (integer or double).

Value

a mvector or a R vector, depending on the size of the result.

Examples

```
a <- matrix(1:20, 4, 5)
A <- as.mmatrix(a, "float")
colMeans(A)
rowSums(A)
```

---

copy *Copy memory mapped object*

Description

Copy memory mapped object

Usage

```
copy(x, filename)

## S4 method for signature 'mvector'
copy(x, filename)

## S4 method for signature 'mmatrix'
copy(x, filename)

## S4 method for signature 'marray'
copy(x, filename)
```

Arguments

`x`	a memory mapped object
`filename`	(optional) a file name for the new object

Details

Creates a new memory mapped object, identical to `x`.

Value

A memory mapped object.

Examples

```
a <- as.mvector(1:4)
b <- copy(a)
a
b
```

copy.values *Copy values to memory mapped object*

Description

Copy values to memory mapped object

Usage

```
copy.values(x, values)

## S4 method for signature 'memoryMapped,numericOrArray'
copy.values(x, values)

## S4 method for signature 'memoryMapped,memoryMapped'
copy.values(x, values)
```

Arguments

x	a memory mapped object
values	a R object or a memory mmaped object

Details

Copy values to x, recycling if necessary. This function modifies x in-place.

Value

None.

Examples

```
A <- mvector("double", 3)
copy.values(A, 1:3)
B <- mvector("double", 6)
copy.values(B, A)
B
```

descriptor.file *Descriptor file*

Description

Descriptor file

Usage

```
descriptor.file(object)

## S4 method for signature 'mmatrix'
descriptor.file(object)

## S4 method for signature 'mvector'
descriptor.file(object)

## S4 method for signature 'marray'
descriptor.file(object)
```

Arguments

`object`	a memory mapped object

Details

Creates a descriptor file, similar to the descriptor files of the package 'bigmemomry'. This descriptor allows to map the object with the package bigmemory, or the read.descriptor function in this package. Its name is obtained by appending ".desc' to the name of the file mapped by 'object'.

A method is available for marrays as well, but the resulting descriptor can't be read by 'bigmemory' as this package doesn't handle arrays. The function 'read.descriptor' in houba can read it.

Value

None.

See Also

`read.descriptor`

Examples

```
A <- mmatrix("short", 10, 20)
A[] <- sample.int(200)

# create descriptor file
dsc <- descriptor.file(A)
```

```
# linking file to other object
B <- read.descriptor(dsc, readonly = FALSE)
all(as.matrix(A) == as.matrix(B)) # TRUE

B[1:10] <- 0
all(A[1:10] == 0) # TRUE
```

dim *Change object dimensions*

Description

Change object dimensions

Usage

```
## S4 replacement method for signature 'memoryMapped,numeric'
dim(x) <- value

## S4 replacement method for signature 'memoryMapped,NULL'
dim(x) <- value
```

Arguments

x	a memory mapped object
value	or NULL new dimensions

Details

The new dimensions must match the object size. This function can change the class of the object, e.g. from mvector to mmatrix or the reverse.

If the value is NULL, then x is turned into a mvector.

Examples

```
x <- as.mvector(1:6)
x
dim(x) <- 2:3
x
dim(x) <- NULL
x
```

extract — *Read/write access to memory-mapped objects*

Description

Read/write access to memory-mapped objects

Usage

```
## S4 replacement method for signature 'marray,numeric,numeric,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,missing,numeric,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,numeric,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,missing,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,numeric,numeric,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,missing,numeric,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,numeric,missing,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'marray,missing,missing,memoryMapped'
x[i, j, ...] <- value

## S4 method for signature 'marray,numeric,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'marray,missing,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'marray,numeric,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'marray,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'mmatrix,numeric,numeric,numeric'
x[i, j, ...] <- value
```

```
## S4 replacement method for signature 'mmatrix,missing,numeric,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,numeric,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,missing,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,numeric,numeric,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,missing,numeric,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,numeric,missing,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'mmatrix,missing,missing,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'mvector,numeric,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'mvector,missing,missing,numeric'
x[i, j, ...] <- value

## S4 replacement method for signature 'mvector,numeric,missing,memoryMapped'
x[i, j, ...] <- value

## S4 replacement method for signature 'mvector,missing,missing,memoryMapped'
x[i, j, ...] <- value

## S4 method for signature 'mmatrix,numeric,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'mmatrix,missing,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'mmatrix,numeric,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'mmatrix,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'mvector,numeric,missing'
x[i, j, ..., drop = TRUE]
```

```
## S4 method for signature 'mvector,missing,missing'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	memory-mapped object
`i, j`	indices of elements to extract or replace
`...`	supplementary indices (for arrays)
`value`	replacement value
`drop`	for dual matrices or array.

Value

a R object or a memory-mapped object (depending on `houba("max.size")`)

Examples

```
a <- matrix(1:6, 2, 3)
A <- as.mmatrix(a)
A[1,]
A[2,] <- A[1,] * 2
A[,3] <- 6:7
A
```

`flush` *Flushes changes from a memory-mapped matrix*

Description

Sync makes sure that the data written to the file linked with the object.

Usage

```
flush(con)

## S4 method for signature 'memoryMapped'
flush(con)
```

Arguments

`con`	a memory mapped object

Details

An error will be raised if the object is read-only, or if the operation failed.

Value

None

Examples

```
x <- as.mvector(1:50)
x <- x + 1
flush(x)
```

---

houba — *Options for package houba*

Description

Options for package houba

Usage

```
houba(...)
```

Arguments

`...`	options to be defined, using 'name = value', or name(s) of option(s) to get.

Details

`houba()` sends back the list of all options. `houba(option = value)` sets the option value. `houba("option")` sends back the value of an option.

Currently the only supported option is "max.size". Use `houba("max.size")` to to get its value and, for example, `houba(max.size = 1e3)`, to set it to 1000.

When subsetting an mvector or an mmatrix, if the size of the resulting object is greater than ‘max.size‘, then the result will be a memory mapped object (mvector or mmatrix), else if will be a R object (vector or matrix). The default value is 1e6. Set ‘max.size‘ to ‘0‘ to always get a memory mapped object and to ‘Inf‘ to always get a R object.

Value

a named list with options values, or a single option value.

Examples

```
houba()
houba("maxsize")
```

inplace *In-place arithmetic operations*

Description

In-place arithmetic operations

Usage

```
inplace.inverse(x)

inplace.opposite(x)

inplace.sum(x, y)

## S4 method for signature 'memoryMapped,numeric'
inplace.sum(x, y)

## S4 method for signature 'memoryMapped,memoryMapped'
inplace.sum(x, y)

inplace.minus(x, y)

## S4 method for signature 'memoryMapped,numeric'
inplace.minus(x, y)

## S4 method for signature 'memoryMapped,memoryMapped'
inplace.minus(x, y)

inplace.prod(x, y)

## S4 method for signature 'memoryMapped,numeric'
inplace.prod(x, y)

## S4 method for signature 'memoryMapped,memoryMapped'
inplace.prod(x, y)

inplace.div(x, y)

## S4 method for signature 'memoryMapped,numeric'
inplace.div(x, y)

## S4 method for signature 'memoryMapped,memoryMapped'
inplace.div(x, y)
```

Arguments

x	a memory mapped object

y	a R object or a memory mapped object

Details

These functions will modify x in-place, performing the operation indicated by their name.

Value

None

Examples

```
x <- as.mvector( 2**(1:3) )
inplace.inverse(x)
inplace.opposite(x)
inplace.sum(x, 0.5)
inplace.prod(x, 8)
x
y <- copy(x)
inplace.prod(x, y)
x
```

length,mvector-method *Length of mvector*

Description

returns the length of a mvector

Usage

```
## S4 method for signature 'mvector'
length(x)
```

Arguments

x	mvector

Value

an integer

marray — *Creation of memory mapped objects*

Description

These functions create memory mapped vectors, matrices or arrays, possibly from an existing file. It is also possible to create objects in memory, by passing the argument `filename = ""`.

Usage

```
marray(
  datatype = c("double", "float", "integer", "short"),
  dim,
  filename,
  readonly
)

mmatrix(
  datatype = c("double", "float", "integer", "short"),
  nrow,
  ncol,
  filename,
  readonly
)

mvector(
  datatype = c("double", "float", "integer", "short"),
  length,
  filename,
  readonly
)
```

Arguments

`datatype`	the data type
`dim`	dimension of marray
`filename`	(optional) path to file
`readonly`	(optional) if `TRUE`, the object will be read-only.
`nrow`	number of rows of mmatrix
`ncol`	number of columns of mmatrix
`length`	length of mvector

Details

Currently `datatype` can only be double, float, int, or short. Short will always be a 16 bits integer (int16_t).

If `filename` is missing, a temporary filename will be generated using `tempfile`. If it is an empty string, the object will be created by allocating memory. Otherwise, `filename` must be a valid path file; if the file exists, it will be opened (if its size is compatible with the dimension of the object); if the file does not exist, it will be created.

If `readonly` is missing, it will be set to `TRUE` when opening an existing file, and to `FALSE` when the file is created by the function.

Value

a memory mapped object, of class 'mvector', 'mmatrix' or 'marray'

Examples

```
a <- mmatrix("float", 4, 3)
a[] <- 1:12
a[1,]
```

`marray-class` *Class* `"marray"`

Description

S4 class for manipulating memory-mapped files as arrays

Slots

`ptr` `externalptr` to an instance of the C++ `MMatrix` class

`file` `character` with the path (absolute) of the file used to store the marray.

`dim` An integer vector giving the dimensions of the marray.

`datatype` `character` giving the C++ underlying datatype.

`readonly` `logical` Indicates if the array if read-only.

Objects from the Class

Objects can be created by calling marray.

See Also

mmatrix-class, mvector-class

`mmatrix-class` *Class* `"mmatrix"`

Description

S4 class for manipulating memory-mapped files as matrices

Slots

`ptr` `externalptr` to an instance of the C++ `MMatrix` class

`file` `character` with the path (absolute) of the file used to store the mmatrix

`dim` An integer vector giving the dimensions of the mmatrix

`datatype` `character` giving the C++ underlying datatype.

`readonly` `logical` Indicates if the array is read-only.

Objects from the Class

Objects can be created by calling mmatrix.

See Also

marray-class, mvector-class

`mvector-class` *Class* `mvector`

Description

S4 class for manipulating memory-mapped files as vectors

Slots

`ptr` `externalptr` to an instance of the C++ `MMatrix` class

`file` `character` with the path (absolute) of the file used to store the mvector

`length` An integer giving the length of the mvector

`datatype` `character` giving the C++ underlying datatype.

`readonly` `logical` Indicates if the vector if read-only.

Objects from the Class

Objects can be created by calling mvector.

See Also

marray-class, mmatrix-class

read.descriptor *Read big memory descriptor file*

Description

Read big memory descriptor file

Usage

```
read.descriptor(descriptor, readonly)
```

Arguments

descriptor	name of descriptor file
readonly	TRUE by default, specifies if the object should be readonly

Details

Creates a memory-mapped object by reading a 'bigmemory'-like descriptor file.

Value

a mvector or a mmatrix

See Also

descriptor.file

Examples

```
A <- mmatrix("short", 10, 20)
A[] <- sample.int(200)

# create descriptor file
dsc <- descriptor.file(A)

# linking file to other object
B <- read.descriptor(dsc, readonly = FALSE)
all(as.matrix(A) == as.matrix(B)) # TRUE

B[1:10] <- 0
all(A[1:10] == 0) # TRUE
```

restore — *Restore memory-mapped matrix*

Description

When the external pointer is broken, attempt to recreate a valid object, if the file still exists.

Usage

```
restore(object)

## S4 method for signature 'marray'
restore(object)

## S4 method for signature 'mmatrix'
restore(object)

## S4 method for signature 'mvector'
restore(object)
```

Arguments

`object`	a memory mapped matrix

Value

a memory-mapped object

Examples

```
a <- matrix(1:24, 4, 6)
A <- as.mmatrix(a, "float")
rdsfile <- tempfile(fileext = ".rds")
saveRDS(A, rdsfile)
A <- readRDS(rdsfile)
A
A <- restore(A)
A
```

type	*Type of a memory-mapped object*

Description

Type of a memory-mapped object

Usage

```
type(x)

## S4 method for signature 'memoryMapped'
type(x)
```

Arguments

x	a memory mapped object

Details

Sends back the stored data type (currently "double", "float", "integer" or "short").

Value

a string

Examples

```
x <- mvector("integer", 6)
type(x)
```

Index