Package ‘bigmemory’

March 28, 2016

Version 4.5.19
Date 2016-03-28
Title Manage Massive Matrices with Shared Memory and Memory-Mapped Files
Author Michael J. Kane <kaneplusplus@gmail.com>, John W. Emerson
<jayemerson@gmail.com>, Peter Haverty <haverty.peter@gene.com>, and Charles Determan Jr. <cdetermanjr@gmail.com>
Maintainer Michael J. Kane <bigmemoryauthors@gmail.com>
Contact Mike, Jay, and Charles <bigmemoryauthors@gmail.com>
Depends R (>= 3.2.0), methods, utils, bigmemory.sri
Imports Rcpp
Enhances biganalytics, bigtabulate, synchronicity
LinkingTo BH, Rcpp
Description Create, store, access, and manipulate massive matrices. Matrices are allocated to shared memory and may use memory-mapped files. Packages 'biganalytics', 'bigtabulate', 'synchronicity', and 'bigalgebra' provide advanced functionality.
License LGPL-3 | Apache License 2.0
URL http://www.bigmemory.org
LazyLoad yes
Biarch yes
VignetteBuilder knitr
Suggests knitr, testthat
RoxygenNote 5.0.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2016-03-28 18:59:53
Description

Create, store, access, and manipulate massive matrices. Matrices are, by default, allocated to shared memory and may use memory-mapped files. Packages biganalytics, synchronicity, bigalgebra, and bigtabulate provide advanced functionality. Access to and manipulation of a big.matrix object is exposed by an S4 class whose interface is similar to that of an matrix. Use of these packages in parallel environments can provide substantial speed and memory efficiencies. bigmemory also provides a C++ framework for the development of new tools that can work both with big.matrix and native matrix objects.
Details

Index of functions/methods (grouped in a friendly way):

- big.matrix, filebacked.big.matrix, as.big.matrix
- is.big.matrix, is.separated, is.filebacked
- describe, attach.big.matrix, attach.resource
- sub.big.matrix, is.sub.big.matrix
- dim, dimnames, nrow, ncol, print, head, tail, typeof, length
- read.big.matrix, write.big.matrix
- mwhich
- morder, mpermute
- deepcopy
- flush

Multi-gigabyte data sets challenge and frustrate users, even on well-equipped hardware. Use of C/C++ can provide efficiencies, but is cumbersome for interactive data analysis and lacks the flexibility and power of R’s rich statistical programming environment. The package bigmemory and sister packages biganalytics, synchronicity, bigtabulate, and bigalgebra bridge this gap, implementing massive matrices and supporting their manipulation and exploration. The data structures may be allocated to shared memory, allowing separate processes on the same computer to share access to a single copy of the data set. The data structures may also be file-backed, allowing users to easily manage and analyze data sets larger than available RAM and share them across nodes of a cluster. These features of the Bigmemory Project open the door for powerful and memory-efficient parallel analyses and data mining of massive data sets.

This project (bigmemory and its sister packages) is still actively developed, although the design and current features can be viewed as "stable." Please feel free to email us with any questions: bigmemoryauthors@gmail.com.

Note

Various options are available. options(bigmemory.typecast.warning) can be set to avoid annoying warnings that might occur if, for example, you assign objects (typically type double) to char, short, or integer big.matrix objects. options(bigmemory.print.warning) protects against extracting and printing a massive matrix (which would involve the creation of a second massive copy of the matrix). options(bigmemory.allow.dimnames) by default prevents the setting of dimnames attributes, because they aren’t allocated to shared memory and changes will not be visible across processes. options(bigmemory.default.type) is “double” be default (a change in default behavior as of 4.1.1) but may be changed by the user.
Versions >=4.0 represent a major redesign, with the mutexes (locking) abstracted to package synchronicity, the exploratory data analysis functionality relocated to package biganalytics, and new linear algebra support available in package bigalgebra. Package bigtabulate extends the bigmemory package with table-, tapply-, and split-like behavior. The functions may also be used with regular matrices for speed and memory-efficiency gains. Package bigmemory itself is now minimalist, providing only the core functionality. As an example, the apply() method appears in biganalytics, supporting exploration and analysis, while mwhich, morder and mpermute appear in bigmemory as fundamental tools for data manipulation.

Note that you can’t simply use a big.matrix with many (most) existing functions (e.g. lm, kmeans). One nice exception is split, because this function only accesses subsets of the matrix.

Author(s)

Michael J. Kane, John W. Emerson, Peter Haverty, and Charles Determan Jr.

Maintainers: Michael J. Kane <bigmemoryauthors@gmail.com>

References

http://www.bigmemory.org

See Also

For example, big.matrix, mwhich, read.big.matrix

Examples

# Our examples are all trivial in size, rather than burning huge amounts of memory.

x <- big.matrix(5, 2, type="integer", init=0,
                    dimnames=list(NULL, c("alpha", "beta")))
x
x[1:2,]
x[,,1] <- 1:5
x[,"alpha"]
colnames(x)
options(bigmemory.allow.dimnames=TRUE)
colnames(x) <- NULL
x[,]
as.big.matrix-methods

Create a “big.matrix” from a matrix or vector.

Description

Create a big.matrix from a matrix or vector or data.frame; a vector will result in a big.matrix with one column. A data frame will have character vectors converted to factors, and then all factors converted to numeric factor levels. All labels or character values will be lost.

Methods

signature(x = "matrix") ...
signature(x = "vector") ...
signature(x = "data.frame") ...

as.matrix, big.matrix-method

Convert to base R matrix

Description

Extract values from a big.matrix object and convert to a base R matrix object

Usage

## S4 method for signature 'big.matrix'
as.matrix(x)

Arguments

x A big.matrix object
big.matrix \hspace{1cm} \textit{The core "big.matrix" operations.}

\textbf{Description}

Create a \texttt{big.matrix} (or check to see if an object is a \texttt{big.matrix}, or create a \texttt{big.matrix} from a \texttt{matrix}, and so on). The \texttt{big.matrix} may be file-backed.

\textbf{Usage}

\begin{verbatim}
big.matrix(nrow, ncol, type = options()$bigmemory.default.type, init = NULL,
dimnames = NULL, separated = FALSE, backingfile = NULL,
backingpath = NULL, descriptorfile = NULL, binarydescriptor = FALSE,
shared = TRUE)

filebacked.big.matrix(nrow, ncol, type = options()$bigmemory.default.type,
init = NULL, dimnames = NULL, separated = FALSE, backingfile = NULL,
backingpath = NULL, descriptorfile = NULL, binarydescriptor = FALSE)

as.big.matrix(x, type = NULL, separated = FALSE, backingfile = NULL,
backingpath = NULL, descriptorfile = NULL, binarydescriptor = FALSE,
shared = TRUE)

is.big.matrix(x)

## S4 method for signature 'big.matrix'
is.big.matrix(x)

## S4 method for signature 'ANY'
is.big.matrix(x)

is.separated(x)

## S4 method for signature 'big.matrix'
is.separated(x)

is.filebacked(x)

## S4 method for signature 'big.matrix'
is.filebacked(x)

shared.name(x)

## S4 method for signature 'big.matrix'
shared.name(x)

file.name(x)
\end{verbatim}
### S4 method for signature 'big.matrix'

`file.name(x)`

`is.shared(x)`

### S4 method for signature 'big.matrix'

`is.shared(x)`

`is.readonly(x)`

### S4 method for signature 'big.matrix'

`is.readonly(x)`

`is.nil(address)`

#### Arguments

- `nrow`: number of rows.
- `ncol`: number of columns.
- `type`: the type of the atomic element (options()$bigmemory.default.type by default – "double" – but can be changed by the user to "integer", "short", or "char").
- `init`: a scalar value for initializing the matrix (NULL by default to avoid unnecessary time spent doing the initializing).
- `dimnames`: a list of the row and column names; use with caution for large objects.
- `separated`: use separated column organization of the data; see details.
- `backingfile`: the root name for the file(s) for the cache of `x`.
- `backingpath`: the path to the directory containing the file backing cache.
- `descriptorfile`: the name of the file to hold the backingfile description, for subsequent use with `attach.big.matrix`; if NULL, the backingfile is used as the root part of the descriptor file name. The descriptor file is placed in the same directory as the backing files.
- `binarydescriptor`: the flag to specify if the binary RDS format should be used for the backingfile description, for subsequent use with `attach.big.matrix`; if NULL of FALSE, the dput() file format is used.
- `shared`: TRUE by default, and always TRUE if the big.matrix is file-backed. For a non-filebacked big.matrix, shared=FALSE uses non-shared memory, which can be more stable for large (say, >50 fail in such cases due to exhausted shared-memory resources in the system.
- `x`: a matrix, vector, or data.frame for as.big.matrix; if a vector, a one-column big.matrix is created by as.big.matrix; if a data.frame, see details. For the is.* functions, x is likely a big.matrix.
- `address`: an externalptr, so is.nil(x@address) might be a sensible thing to want to check, but it's pretty obscure.
Details

A `big.matrix` consists of an object in R that does nothing more than point to the data structure implemented in C++. The object acts much like a traditional R matrix, but helps protect the user from many inadvertent memory-consuming pitfalls of traditional R matrices and data frames.

There are two `big.matrix` types which manage data in different ways. A standard, shared `big.matrix` is constrained to available RAM, and may be shared across separate R processes. A file-backed `big.matrix` may exceed available RAM by using hard drive space, and may also be shared across processes. The atomic types of these matrices may be `double`, `integer`, `short`, or `char` (8, 4, 2, and 1 bytes, respectively).

If `x` is a `big.matrix`, then `x[1:5,]` is returned as an R matrix containing the first five rows of `x`. If `x` is of type double, then the result will be numeric; otherwise, the result will be an integer R matrix. The expression `x` alone will display information about the R object (e.g. the external pointer) rather than evaluating the matrix itself (the user should try `x[,]` with extreme caution, recognizing that a huge R matrix will be created).

If `x` has a huge number of rows and/or columns, then the use of `rownames` and/or `colnames` will be extremely memory-intensive and should be avoided. If `x` has a huge number of columns and `separated=TRUE` is used (this isn’t typically recommended), the user might want to store the transpose as there is overhead of a pointer for each column in the matrix. If `separated` is `TRUE`, then the memory is allocated into separate vectors for each column. Use this option with caution if you have a large number of columns, as shared-memory segments are limited by OS and hardware combinations. If `separated` is `FALSE`, the matrix is stored in traditional column-major format. The function `is.separated()` returns the separation type of the `big.matrix`.

When a `big.matrix`, `x`, is passed as an argument to a function, it is essentially providing call-by-reference rather than call-by-value behavior. If the function modifies any of the values of `x`, the changes are not limited in scope to a local copy within the function. This introduces the possibility of side-effects, in contrast to standard R behavior.

A file-backed `big.matrix` may exceed available RAM in size by using a file cache (or possibly multiple file caches, if `separated=TRUE`). This can incur a substantial performance penalty for such large matrices, but less of a penalty than most other approaches for handling such large objects. A side-effect of creating a file-backed object is not only the file-backing(s), but a descriptor file (in the same directory) that is needed for subsequent attachments (see `attach.big.matrix`).

Note that we do not allow setting or changing the `dimnames` attributes by default; such changes would not be reflected in the descriptor objects or in shared memory. To override this, set `options(bigmemory.allow.dimnames=TRUE)`.

It should also be noted that a user can create an “anonymous” file-backed `big.matrix` by specifying `""` as the filebacking argument. In this case, the backing resides in the temporary directory and a descriptor file is not created. These should be used with caution since even anonymous backings use disk space which could eventually fill the hard drive. Anonymous backings are removed either manually, by a user, or automatically, when the operating system deems it appropriate.

Finally, note that as `big.matrix` can coerce data frames. It does this by making any character columns into factors, and then making all factors numeric before forming the `big.matrix`. Level labels are not preserved and must be managed by the user if desired.

Value

A `big.matrix` is returned (for `big.matrix` and `filebacked.big.matrix`, and `as.big.matrix`), and `TRUE` or `FALSE` for `is.big.matrix` and the other functions.
Author(s)

John W. Emerson and Michael J. Kane <<bigmemoryauthors@gmail.com>>

See Also

`bigmemory`, and perhaps the class documentation of `big.matrix`; `attach.big.matrix` and `describe`. Sister packages `biganalytics`, `bigtabulate`, `synchronicity`, and `bigalgebra` provide advanced functionality.

Examples

```r
library(bigmemory)
x <- big.matrix(10, 2, type='integer', init=-5)
options(bigmemory.allow.dmmames=TRUE)
colnames(x) <- c("alpha", "beta")
is.big.matrix(x)
dim(x)
colnames(x)
rownames(x)
x[,,]
x[1:8,1] <- 11:18
colnames(x) <- NULL
x[,,]

# The following shared memory example is quite silly, as you wouldn't
# likely do this in a single R session. But if zdescription were
# passed to another R session via SNOW, foreach, or even by a
# simple file read/write, then the attach.big.matrix() within the
# second R process would give access to the same object in memory.
# Please see the package vignette for real examples.

z <- big.matrix(3, 3, type='integer', init=3)
z[,,]
dim(z)
z[1,1] <- 2
z[,,]
zdescription <- describe(z)
zdescription
y <- attach.big.matrix(zdescription)
y[,,]
y
z
y[1,1] <- -100
y[,,]
z[,,]
```

---

**big.matrix-class**

*Class "big.matrix"*
Description

The big.matrix class is designed for matrices with elements of type double, integer, short, or char. A big.matrix acts much like a traditional R matrix, but helps protect the user from many inadvertent memory-consuming pitfalls of traditional R matrices and data frames. The objects are allocated to shared memory, and if file-backing is used they may exceed virtual memory in size. Sadly, 32-bit operating system constraints – largely Windows and some MacOS versions – will be a limiting factor with file-backed matrices; 64-bit operating systems are recommended.

Objects from the Class

Unlike many R objects, objects should not be created by calls of the form new("big.matrix", ...). The functions big.matrix() and filebacked.big.matrix() are intended for the user.

Slots

address: Object of class "externalPtr" points to the memory location of the C++ data structure.

Methods

As you would expect:

signature(x = "big.matrix", i = "ANY", j = "ANY"): ...

[<signature(x = "big.matrix", i = "ANY", j = "missing"): ...

[< signature(x = "big.matrix", i = "missing", j = "ANY"): ...

[< signature(x = "big.matrix", i = "missing", j = "missing"): ...

[< signature(x = "big.matrix", i = "matrix", j = "missing"): ...

signature(x = "big.matrix", i = "ANY", j = "ANY", drop = "missing"): ...

signature(x = "big.matrix", i = "ANY", j = "missing", drop = "missing"): ...

signature(x = "big.matrix", i = "ANY", j = "missing", drop = "logical"): ...

signature(x = "big.matrix", i = "missing", j = "missing", drop = "logical"): ...

signature(x = "big.matrix", i = "matrix", j = "missing", drop = "logical"): ...

signature(x = "big.matrix", i = "missing", j = "ANY", drop = "missing"): ...

signature(x = "big.matrix", i = "missing", j = "ANY", drop = "logical"): ...

signature(x = "big.matrix", i = "missing", j = "missing", drop = "missing"): ...

signature(x = "big.matrix", i = "missing", j = "missing", drop = "logical"): ...

The following are probably more interesting:

describe signature(x = "big.matrix"): provide necessary and sufficient information for the sharing or re-attaching of the object.

dim signature(x = "big.matrix"): returns the dimension of the big.matrix.

length signature(x = "big.matrix"): returns the product of the dimensions of the big.matrix.

dimnames<- signature(x = "big.matrix", value = "list"): set the row and column names, prohibited by default (see bigmemory to override).

dimnames signature(x = "big.matrix"): get the row and column names.
The `deepcopy` function is needed to make a duplicate of a `big.matrix`, with the new copy optionally filebacked.

**Usage**

```r
deepcopy(x, cols = NULL, rows = NULL, y = NULL, type = NULL,
        separated = NULL, backingfile = NULL, backingpath = NULL,
        descriptorfile = NULL, binarydescriptor = FALSE, shared = TRUE)
```

**Description**

This is needed to make a duplicate of a `big.matrix`, with the new copy optionally filebacked.
Arguments

`x`  
a big.matrix.

cols  
possible subset of columns for the deepcopy; could be numeric, named, or logical.

`rows`  
possible subset of rows for the deepcopy; could be numeric, named, or logical.

`y`  
optional destination object (matrix or big.matrix); if not specified, a big.matrix will be created.

type  
preferably specified, "integer" for example.

separated  
use separated column organization of the data instead of column-major organization; use with caution if the number of columns is large.

`backingfile`  
the root name for the file(s) for the cache of x.

`backingpath`  
the path to the directory containing the file-backing cache.

`descriptorfile`  
we recommend specifying this for file-backing.

`binarydescriptor`  
the flag to specify if the binary RDS format should be used for the backingfile description, for subsequent use with attach.big.matrix; if NULL or FALSE, the dput() file format is used.

`shared`  
TRUE by default, and always TRUE if the big.matrix is file-backed. For a non-filebacked big.matrix, shared=FALSE uses non-shared memory, which can be more stable for large (say, >50% of RAM) objects. Shared memory allocation can sometimes fail in such cases due to exhausted shared-memory resources in the system.

Details

This is needed to make a duplicate of a big.matrix, because traditional syntax would only copy the object (the pointer to the big.matrix rather than the big.matrix itself). It can also make a copy of only a subset of columns.

Value

a big.matrix.

See Also

big.matrix

Examples

```r
x <- as.big.matrix(matrix(1:30, 10, 3))
y <- deepcopy(x, -1)  # Don't include the first column.
x
y
head(x)
head(y)
```
The basic “big.matrix” operations for sharing and re-attaching.

Description

The `describe` function returns the information needed by `attach.big.matrix` to reference a shared or file-backed `big.matrix` object. The `attach.big.matrix` and `attach.resource` functions create a new `big.matrix` object based on the descriptor information referencing previously allocated shared-memory or file-backed matrices.

Usage

```r
## S4 method for signature 'big.matrix'
describe(x)

attach.big.matrix(obj, ...)
```

Arguments

- `x` a `big.matrix` object
- `obj` an object as returned by `describe()` or, optionally, the filename of the descriptor for a filebacked matrix, assumed to be in the directory specified by the path (if one is provided)
- `...` possibly path which gives the path where the descriptor and/or filebacking can be found

Details

The `describe` function returns a list of the information needed to attach to a `big.matrix` object. A descriptor file is automatically created when a new filebacked `big.matrix` is created.

Value

- `describe` returns a list of the information needed to attach to a `big.matrix` object.
- `attach.big.matrix` return a new instance of type `big.matrix` corresponding to a shared-memory or file-backed `big.matrix`.

Author(s)

Michael J. Kane and John W. Emerson <<bigmemoryauthors@gmail.com>>

See Also

`bigmemory`, `big.matrix`, or the class documentation `big.matrix`. 
Examples

# The example is quite silly, as you wouldn't likely do this in a
# single R session. But if zdescription were passed to another R session
# via SNOW, foreach, or even by a simple file read/write,
# then the attach of the second R process would give access to the
# same object in memory. Please see the package vignette for real examples.

z <- big.matrix(3, 3, type='integer', init=3)
z[,]
dim(z)
z[1,1] <- 2
z[,]
zdescription <- describe(z)
zdescription
y <- attach.big.matrix(zdescription)
y[,]
y
z
zz <- attach.resource(zdescription)
zz[1,1] <- -100
y[,]
z[,]
The first column of the submatrix
of the submatrix if not NULL
required path to the filebacked object, if applicable
The filename of the descriptor for a filebacked matrix, assumed ot be in the
directory specified
possibly path which gives the path where the descriptor and/or filebacking can be found.

Objects from the Class

Objects should not be created by calls of the form `new("big.matrix.descriptor", ...)`, but should use the `describe` function.

Slots

description: Object of class "list"; details omitted.

Extends

Class "descriptor", directly.

Methods

`attach.resource` signature(obj = "big.matrix.descriptor"): ...

`sub.big.matrix` signature(x = "big.matrix.descriptor"): ...

Note

We provide `attach.resource` for convenience, but expect most users will prefer `attach.big.matrix`.

Author(s)

John W. Emerson and Michael J. Kane

References

Other types of descriptors are defined in package `synchronicity`.

See Also

See also `attach.big.matrix`.

Examples

`showClass("big.matrix.descriptor")`
**dim, big.matrix-method**  
*Dimensions of a big.matrix object*

**Description**
Retrieve the dimensions of a big.matrix object

**Usage**
```r
## S4 method for signature 'big.matrix'
dim(x)
```

**Arguments**
- `x`: A big.matrix object

---

**dimnames, big.matrix-method**  
*Dimnames of a big.matrix Object*

**Description**
Retrieve or set the dimnames of an object

**Usage**
```r
## S4 method for signature 'big.matrix'
dimnames(x)
```
```r
## S4 replacement method for signature 'big.matrix,list'
dimnames(x) <- value
```

**Arguments**
- `x`: A big.matrix object
- `value`: A possible value for dimnames(x)
Description

Extract or replace big.matrix elements

Usage

```r
## S4 method for signature 'big.matrix,ANY,ANY,missing'
x[i, j, drop]

## S4 method for signature 'big.matrix,ANY,ANY,logical'
x[i, j, drop]

## S4 method for signature 'big.matrix,missing,ANY,missing'
x[i, j, drop]

## S4 method for signature 'big.matrix,missing,ANY,logical'
x[i, j, drop]

## S4 method for signature 'big.matrix,ANY,missing,missing'
# S4 method for signature 'big.matrix,missing,ANY,missing'
x[i, j, drop]

## S4 method for signature 'big.matrix,ANY,missing,logical'
# S4 method for signature 'big.matrix,missing,ANY,logical'
x[i, j, drop]

## S4 method for signature 'big.matrix,missing,missing,missing'
# S4 method for signature 'big.matrix,missing,missing,logical'
x[i, j, drop]

## S4 method for signature 'big.matrix,missing,missing,missing'
# S4 method for signature 'big.matrix,missing,missing,missing'
x[i, j, drop]

## S4 method for signature 'big.matrix,ANY,ANY'
x[i, j] <- value

## S4 replacement method for signature 'big.matrix,missing,ANY'
x[i, j] <- value

## S4 replacement method for signature 'big.matrix,ANY,missing'
x[i, j] <- value

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

```c
CC sT method for signature
G bigNmatrixLanyLanyLmissing
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLanyLanyLlogical
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLmissingLanyLmissing
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLmissingLanyLlogical
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLanyLmissingLmissing
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLanyLmissingLlogical
G x[iL jL drop]

CC sT method for signature
G bigNmatrixLmissingLmissingLmissing
G x[iL jL drop]
```

## S4 replacement method for signature
G bigNmatrixLanyLany
G x[iL j]

## S4 replacement method for signature
G bigNmatrixLmissingLany
G x[iL j]

## S4 replacement method for signature
G bigNmatrixLanyLmissing
G x[iL j]

## S4 replacement method for signature
G bigNmatrixLmissingLmissing
G x[iL j] <- value ...

## S4 replacement method for signature
G bigNmatrixLmissingLmissingLlogical
G x[iL j] <- value ...

## S4 replacement method for signature
G bigNmatrixLmatrixLmissingLmissing
G x[iL j] <- value ...
```
## flush

### Arguments

- `x`: A `big.matrix` object
- `i`: Indices specifying the rows
- `j`: Indices specifying the columns
- `drop`: Logical indication if reduce to minimum dimensions
- `value`: Typically an array-like R object of similar class

### Description

For a file-backed `big.matrix` object, `flush()` forces any modified information to be written to the file-backing.

### Usage

```r
flush(con)
```

### Value

TRUE or FALSE (invisible), indicating whether or not the flush was successful.

### Author(s)

John W. Emerson and Michael J. Kane
Examples

```r
temp_dir = tempdir()
if (!dir.exists(temp_dir)) dir.create(temp_dir)
x <- big.matrix(nrow=3, ncol=3, backingfile="flushtest.bin",
                descriptorfile="flushdesc", backingpath=temp_dir,
                type='integer')
x[1,1] <- 0
flush(x)
```

---

**GetMatrixSize**

**big.matrix size**

**Description**

Returns the size of the created matrix in bytes

**Usage**

```r
GetMatrixSize(bigMat)
```

**Arguments**

- **bigMat**
  - a big.matrix object

---

**head, big.matrix-method**

**Return First or Last Part of a big.matrix Object**

**Description**

Returns the first or last parts of a big.matrix object.

**Usage**

```r
## S4 method for signature 'big.matrix'
head(x, n = 6)
```

```r
## S4 method for signature 'big.matrix'
tail(x, n = 6)
```

**Arguments**

- **x**  
  - A big.matrix object

- **n**  
  - A single integer for the number of rows to return
is.float

**Check if Float**

**Description**

Check to see if the elements of a big.matrix object are floats.

**Usage**

```r
is.float(x)
```

**Arguments**

- `x` An object to be evaluated if float

is.float,numeric-method

**Is Float?**

**Description**

Check if R numeric value has float flag

**Usage**

```r
## S4 method for signature 'numeric'
is.float(x)
```

**Arguments**

- `x` A numeric value

is.sub.big.matrix

**Submatrix support**

**Description**

This doesn’t create a copy, it just provides a new version of the class which provides behavior for a contiguous submatrix of the big.matrix. Non-contiguous submatrices are not supported.
Usage

is.sub.big.matrix(x)

## S4 method for signature 'big.matrix'
is.sub.big.matrix(x)

sub.big.matrix(x, firstRow = 1, lastRow = NULL, firstCol = 1, lastCol = NULL, backingpath = NULL)

## S4 method for signature 'big.matrix'
sub.big.matrix(x, firstRow = 1, lastRow = NULL, firstCol = 1, lastCol = NULL, backingpath = NULL)

Arguments

x either a big.matrix or a descriptor.
firstRow the first row of the submatrix.
lastRow the last row of the submatrix if not NULL.
firstCol the first column of the submatrix.
lastCol the last column of the submatrix if not NULL.
backingpath required path to the filebacked object, if applicable.

Details

The sub.big.matrix function allows a user to create a big.matrix object that references a contiguous set of columns and rows of another big.matrix object.

The is.sub.big.matrix function returns TRUE if the specified argument is a sub.big.matrix object and return FALSE otherwise.

Value

A big.matrix which is actually a submatrix of a larger big.matrix. It is not a physical copy. Only contiguous blocks may form a submatrix.

Author(s)

John W. Emerson and Michael J. Kane

See Also

big.matrix

Examples

x <- big.matrix(10, 5, init=0, type="double")
x[,] <- 1:50
y <- sub.big.matrix(x, 2, 9, 2, 3)
y[,]
```r
y[1,1] <- -99
x[,] rm(x)
```

---

**length, big.matrix-method**

*Length of a big.matrix object*

---

**Description**

Get the length of a big.matrix object

**Usage**

```r
## S4 method for signature 'big.matrix'
length(x)
```

**Arguments**

- `x`: A big.matrix object

---

**morder**

*Ordering and Permuting functions for “big.matrix” and “matrix” objects*

---

**Description**

The `morder` function returns a permutation of row indices which can be used to rearrange an object according to the values in the specified columns (a multi-column ordering). The `mpermute` function actually reorders the rows of a big.matrix or matrix based on an order vector or a desired ordering on a set of columns.

**Usage**

```r
morder(x, cols, na.last = TRUE, decreasing = FALSE)
morderCols(x, rows, na.last = TRUE, decreasing = FALSE)
mpermute(x, order = NULL, cols = NULL, allow.duplicates = FALSE, ...)
mpermuteCols(x, order = NULL, rows = NULL, allow.duplicates = FALSE, ...)
```
Arguments

- **x**: A big.matrix or matrix object with numeric values.
- **cols**: The columns of x to get the ordering for or reorder on.
- **na.last**: for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
- **decreasing**: logical. Should the sort order be increasing or decreasing?
- **rows**: The rows of x to get the ordering for or reorder on.
- **order**: A vector specifying the reordering of rows, i.e., the result of a call to order or morder.
- **allow.duplicates**: If TRUE, allows a row to be duplicated in the resulting big.matrix or matrix (i.e., in this case, order would not need to be a permutation of 1:nrow(x)).
- **...**: optional parameters to pass to morder when cols is specified instead of just using order.

Details

The morder function behaves similar to order, returning a permutation of 1:nrow(x) which rearranges objects according to the values in the specified columns. However, morder takes a big.matrix or an R matrix (with numeric type) and a set of columns (cols) with which to determine the ordering; morder does not incur the same memory overhead required by order, and runs more quickly. The mpermute function changes the row ordering of a big.matrix or matrix based on a vector order or an ordering based on a set of columns specified by cols. It should be noted that this function has side-effects, that is x is changed when this function is called.

Value

morder returns an ordering vector. mpermute returns nothing but does change the contents of x. This type of a side-effect is generally frowned upon in R, but we “break” the rules here to avoid memory overhead and improve performance.

Author(s)

Michael J. Kane <<bigmemoryauthors@gmail.com>>

See Also

order

Examples

```r
m = matrix(as.double(as.matrix(iris)), nrow=nrow(iris))
morder(m, 1)
order(m[,1])
m[order(m[,1], 2]
mpermute(m, cols=1)
m[,2]
```
**mwhich**

*Expanded “which”-like functionality.*

**Description**

Implements `which`-like functionality for a `big.matrix`, with additional options for efficient comparisons (executed in C++); also works for regular numeric matrices without the memory overhead.

**Usage**

```
mwhich(x, cols, vals, comps, op = "AND")
```

**Arguments**

- `x`: a `big.matrix` (or a numeric matrix; see below).
- `cols`: a vector of column indices or names.
- `vals`: a list (one component for each of `cols`) of vectors of length 1 or 2; length 1 is used to test equality (or inequality), while vectors of length 2 are used for checking values in the range (`-Inf` and `Inf` are allowed). If a scalar or vector of length 2 is provided instead of a list, it will be replicated `length(cols)` times.
- `comps`: a list of operators (one component for each of `cols`), including 'eq', 'neq', 'le', 'lt', 'ge' and 'gt'. If a single operator, it will be replicated `length(cols)` times.
- `op`: the comparison operator for combining the results of the individual tests, either 'AND' or 'OR'.

**Details**

To improve performance and avoid the creation of massive temporary vectors in R when doing comparisons, `mwhich()` efficiently executes column-by-column comparisons of values to the specified values or ranges, and then returns the row indices satisfying the comparison specified by the `op` operator. More advanced comparisons are then possible (and memory-efficient) in R by doing set operations (union and intersect, for example) on the results of multiple `mwhich()` calls.

Note that NA is a valid argument in conjunction with 'eq' or 'neq', replacing traditional is.na() calls. And both -Inf and Inf can be used for one-sided inequalities.

If `mwhich()` is used with a regular numeric R matrix, we access the data directly and thus incur no memory overhead. Interested developers might want to look at our code for this case, which uses a handy pointer trick (accessor) in C++.

**Value**

A vector of row indices satisfying the criteria.

**Author(s)**

John W. Emerson <<bigmemoryauthors@gmail.com>>
See Also

big.matrix, which

Examples

```r
x <- as.big.matrix(matrix(1:30, 10, 3))
options(bigmemory.allow.dimnames=TRUE)
colnames(x) <- c("A", "B", "C")
x[,]
x[mwhich(x, 1:2, list(c(2,3), c(11,17)),
  list(c('ge','le'), c('gt', 'lt')), 'OR'),]
x[mwhich(x, c("A","B"), list(c(2,3), c(11,17)),
  list(c('ge','le'), c('gt', 'lt')), 'AND'),]

# These should produce the same answer with a regular matrix:
y <- matrix(1:30, 10, 3)
y[mwhich(y, 1:2, list(c(2,3), c(11,17)),
  list(c('ge','le'), c('gt', 'lt')), 'OR'),]
y[mwhich(y, -3, list(c(2,3), c(11,17)),
  list(c('ge','le'), c('gt', 'lt')), 'AND'),]

x[1,1] <- NA
mwhich(x, 1:2, NA, 'eq', 'OR')
mwhich(x, 1:2, NA, 'neq', 'AND')

# Column 1 equal to 4 and/or column 2 less than or equal to 16:
mwhich(x, 1:2, list(4, 16), list('eq', 'le'), 'OR')
mwhich(x, 1:2, list(4, 16), list('eq', 'le'), 'AND')

# Column 2 less than or equal to 15:
mwhich(x, 2, 15, 'le')

# No NAs in either column, and column 2 strictly less than 15:
mwhich(x, c(1:2,2), list(NA, NA, 15), list('neq', 'neq', 'lt'), 'AND')

x <- big.matrix(4, 2, init=1, type="double")
x[1,1] <- Inf
mwhich(x, 1, Inf, 'eq')
mwhich(x, 1, 1, 'gt')
mwhich(x, 1, 1, 'le')
```

mwhich-methods  
Expanded “which”-like functionality.
Description

Implements `which`-like functionality for a `big.matrix`, with additional options for efficient comparisons (executed in C++); also works for regular numeric matrices without the memory overhead.

test

Methods

signature(x = "big.matrix=" cols = "ANY", vals = "ANY", " comps = "ANY", op = "character")
...signature(x = "big.matrix", cols = "ANY", vals = "ANY", " comps = "ANY", op = "missing")
...signature(x = "matrix", cols = "ANY", vals = "ANY", " comps = "ANY", op = "character")
...signature(x = "matrix", cols = "ANY", vals = "ANY", " comps = "ANY", op = "missing")
...

See Also

`big.matrix`, `which`, `mwhich`

---

**ncol, big.matrix-method**

*The Number of Rows/Columns of a big.matrix*

Description

`nrow` and `ncol` return the number of rows or columns present in a `big.matrix` object.

Usage

```r
## S4 method for signature 'big.matrix'
ncol(x)
```

```r
## S4 method for signature 'big.matrix'
nrow(x)
```

Arguments

- `x` A `big.matrix` object

Value

An integer of length 1
### `print.big.matrix-method`

#### Print Values

**Description**

`print` will print out the elements within a `big.matrix` object.

**Usage**

```r
## S4 method for signature 'big.matrix'
print(x)
```

**Arguments**

- `x`: A `big.matrix` object

**Note**

By default, this will only return the head of a `big.matrix` to prevent console overflow. If you turn off the `bigmemory.print.warning` option then it will convert to a base R matrix and print all elements.

### `typeof.big.matrix-method`

#### The Type of a `big.matrix` Object

**Description**

`typeof` returns the storage type of a `big.matrix` object.

**Usage**

```r
## S4 method for signature 'big.matrix'
typeof(x)
```

**Arguments**

- `x`: A `big.matrix` object
write.big.matrix  

File interface for a “big.matrix”

Description

Create a \texttt{big.matrix} by reading from a suitably-formatted ASCII file, or write the contents of a \texttt{big.matrix} to a file.

\texttt{@importFrom stats na.omit}

Usage

\begin{verbatim}
write.big.matrix(x, filename, row.names = FALSE, col.names = FALSE, 
sep = ",",)

## S4 method for signature 'big.matrix,character'
write.big.matrix(x, filename, 
row.names = FALSE, col.names = FALSE, sep = ",",)

read.big.matrix(filename, sep = ",", header = FALSE, col.names = NULL, 
row.names = NULL, has.row.names = FALSE, ignore.row.names = FALSE, 
type = NA, skip = 0, separated = FALSE, backingfile = NULL, 
backingpath = NULL, descriptorfile = NULL, binarydescriptor = FALSE, 
extraCols = NULL, shared = TRUE)

## S4 method for signature 'character'
read.big.matrix(filename, sep = ",", header = FALSE, 
col.names = NULL, row.names = NULL, has.row.names = FALSE, 
ignore.row.names = FALSE, type = NA, skip = 0, separated = FALSE, 
backingfile = NULL, backingpath = NULL, descriptorfile = NULL, 
binarydescriptor = FALSE, extraCols = NULL, shared = TRUE)
\end{verbatim}

Arguments

\begin{itemize}
\item \textbf{x} \hspace{2cm} \texttt{a big.matrix}.
\item \textbf{filename} \hspace{2cm} the name of an input/output file.
\item \textbf{row.names} \hspace{2cm} a vector of names, use them even if row names appear to exist in the file.
\item \textbf{col.names} \hspace{2cm} a vector of names, use them even if column names exist in the file.
\item \textbf{sep} \hspace{2cm} a field delimiter.
\item \textbf{header} \hspace{2cm} if \texttt{TRUE}, the first line (after a possible skip) should contain column names.
\item \textbf{has.row.names} \hspace{2cm} if \texttt{TRUE}, then the first column contains row names.
\item \textbf{ignore.row.names} \hspace{2cm} if \texttt{TRUE} when \texttt{has.row.names==TRUE}, the row names will be ignored.
\item \textbf{type} \hspace{2cm} preferably specified, \"integer\" for example.
\item \textbf{skip} \hspace{2cm} number of lines to skip at the head of the file.
\end{itemize}
write.big.matrix

separated  use separated column organization of the data instead of column-major organization.

backingfile  the root name for the file(s) for the cache of x.

backingpath  the path to the directory containing the file backing cache.

descriptorfile  the file to be used for the description of the filebacked matrix.

binarydescriptor  the flag to specify if the binary RDS format should be used for the backingfile description, for subsequent use with attach.big.matrix; if NULL or FALSE, the dput() file format is used.

extraCols  the optional number of extra columns to be appended to the matrix for future use.

shared  if TRUE, the resulting big.matrix can be shared across processes.

Details

Files must contain only one atomic type (all integer, for example). You, the user, should know whether your file has row and/or column names, and various combinations of options should be helpful in obtaining the desired behavior.

When reading from a file, if type is not specified we try to make a reasonable guess for you without making any guarantees at this point. Unless you have really large integer values, we recommend you consider "short". If you have something that is essentially categorical, you might even be able use "char", with huge memory savings for large data sets.

Any non-numeric entry will be ignored and replaced with NA, so reading something that traditionally would be a data.frame won’t cause an error. A warning is issued.

Wishlist: we’d like to provide an option to ignore specified columns while doing reads. Or perhaps to specify columns targeted for factor or character conversion to numeric values. Would you use such features? Email us and let us know!

Value

a big.matrix object is returned by read.big.matrix, while write.big.matrix creates an output file (a path could be part of filename).

Author(s)

John W. Emerson and Michael J. Kane <<bigmemoryauthors@gmail.com>>

See Also

big.matrix

Examples

# Without specifying the type, this big.matrix x will hold integers.

x <- as.big.matrix(matrix(1:10, 5, 2))
x[2,2] <- NA
x[,]   
 temp_dir = tempdir() 
 if (!dir.exists(temp_dir)) dir.create(temp_dir) 
 write.big.matrix(x, file.path(temp_dir, "foo.txt")) 

 # Just for fun, I'll read it back in as character (1-byte integers):  
y <- read.big.matrix(file.path(temp_dir, "foo.txt"), type="char")  
y[,]  

 # Other examples:  
w <- as.big.matrix(matrix(1:10, 5, 2), type='double')  
w[1,2] <- NA  
w[2,2] <- -Inf  
w[3,2] <- Inf  
w[4,2] <- NaN  
w[,]  
write.big.matrix(w, file.path(temp_dir, "bar.txt"))  
w[,]  
w <- read.big.matrix(file.path(temp_dir, "bar.txt"), type="double")  
w[,]  
w <- read.big.matrix(file.path(temp_dir, "bar.txt"), type="short")  
w[,]  

 # Another example using row names (which we don't like).  
x <- as.big.matrix(as.matrix(iris), type='double')  
rownames(x) <- as.character(1:nrow(x))  
head(x)  
write.big.matrix(x, file.path(temp_dir, 'IrisData.txt'), col.names=TRUE,  
                   row.names=TRUE)  
y <- read.big.matrix(file.path(temp_dir, "IrisData.txt"), header=TRUE,  
                      has.row.names=TRUE)  
head(y)  

 # The following would fail with a dimension mismatch:  
if (FALSE) y <- read.big.matrix(file.path(temp_dir, "IrisData.txt"),  
                                header=TRUE)
Index

*Topic classes
  big.matrix, 6
  big.matrix-class, 9
  describe.big.matrix-method, 13
  descriptor-class, 14
*Topic methods
  as.big.matrix-methods, 5
  big.matrix, 6
  deepcopy, 11
  describe.big.matrix-method, 13
  flush, 18
  is.sub.big.matrix, 20
  mwhich, 24
  mwhich-methods, 25
  write.big.matrix, 28
*Topic package
  bigmemory-package, 2
  [, big.matrix, ANY, ANY, logical-method
           (Extract, big.matrix), 17
  [, big.matrix, ANY, ANY, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, ANY, missing, logical-method
           (Extract, big.matrix), 17
  [, big.matrix, ANY, missing, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, matrix, missing, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY, logical-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, missing, logical-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, missing, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, ANY, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, ANY, missing-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  [, big.matrix, missing, ANY-method
           (Extract, big.matrix), 17
  as.big.matrix (big.matrix), 6
  as.big.matrix, data.frame-method
           (as.big.matrix-methods), 5
  as.big.matrix, matrix-method
           (as.big.matrix-methods), 5
  as.big.matrix, vector-method
           (as.big.matrix-methods), 5
  as.big.matrix-methods, 5
  as.matrix, big.matrix-method, 5
  attach.big.matrix, 7–9, 12, 15, 29
  attach.big.matrix
           (describe.big.matrix-method), 13
  attach.resource
           (describe.big.matrix-method), 13
  attach.resource, big.matrix.descriptor-method
           (descriptor-class), 14
  attach.resource, character-method
           (descriptor-class), 14
  big.matrix, 2–5, 6, 9, 11–14, 18, 21, 24–26, 28, 29
  big.matrix-class, 9
  big.matrix.descriptor-class
           (descriptor-class), 14
  bigmemory, 9, 10, 13
  bigmemory (bigmemory-package), 2
  bigmemory-package, 2
  data.frame, 5
  deepcopy, 11
  describe, 9, 15
describe(describe, big.matrix-method), 13
describe, big.matrix-method, 13
descriptor, 15
descriptor-class, 14
dim, big.matrix-method, 16
dimnames, big.matrix-method, 16
dimnames<-, big.matrix, list-method
  (dimnames, big.matrix-method), 16

Extract, big.matrix, 17

file.name (big.matrix), 6
file.name, big.matrix-method
  (big.matrix), 6
filebacked.big.matrix (big.matrix), 6
flush, 18
flush, big.matrix-method (flush), 18

GetMatrixSize, 19

head, big.matrix-method, 19

intersect, 24
is.big.matrix (big.matrix), 6
is.big.matrix, ANY-method (big.matrix), 6
is.big.matrix, big.matrix-method
  (big.matrix), 6
is.filebacked (big.matrix), 6
is.filebacked, big.matrix-method
  (big.matrix), 6
is.float, 20
is.float, numeric-method, 20
is.nil (big.matrix), 6
is.readonly (big.matrix), 6
is.readonly, big.matrix-method
  (big.matrix), 6
is.separated (big.matrix), 6
is.separated, big.matrix-method
  (big.matrix), 6
is.shared (big.matrix), 6
is.shared, big.matrix-method
  (big.matrix), 6
is.sub.big.matrix, 20
is.sub.big.matrix, big.matrix-method
  (is.sub.big.matrix), 20
kmeans, 4

length, big.matrix-method, 22
lm, 4

matrix, 2, 6
morder, 4, 22
morderCols (morder), 22
mpermute, 4
mpermute (morder), 22
mpermuteCols (morder), 22
mwhich, 4, 24, 26
mwhich, big.matrix, ANY, ANY, ANY, character-method
  (mwhich-methods), 25
mwhich, big.matrix, ANY, ANY, ANY, missing-method
  (mwhich-methods), 25
mwhich, big.matrix, ANY, ANY, ANY, character-method
  (mwhich-methods), 25
mwhich, big.matrix, ANY, ANY, ANY, missing-method
  (mwhich-methods), 25
mwhich-methods, 25

ncol, big.matrix-method, 26
nrow, big.matrix-method
  (ncol, big.matrix-method), 26

order, 23

print, big.matrix-method, 27

read.big.matrix, 4
read.big.matrix (write.big.matrix), 28
read.big.matrix, character-method
  (write.big.matrix), 28

shared.name (big.matrix), 6
shared.name, big.matrix-method
  (big.matrix), 6

split, 4

sub.big.matrix (is.sub.big.matrix), 20
sub.big.matrix, big.matrix-method
  (is.sub.big.matrix), 20
sub.big.matrix, big.matrix.descriptor-method
  (descriptor-class), 14

tail, big.matrix-method
  (head, big.matrix-method), 19

typeof, big.matrix-method, 27

union, 24

which, 24–26
write.big.matrix, 28
write.big.matrix, big.matrix, character-method
  (write.big.matrix), 28