

Package ‘bigmemory’

February 14, 2012

Version 4.2.11

Date 2011-07-26

Title Manage massive matrices with shared memory and memory-mapped files

Author Michael J. Kane <kaneplusplus@gmail.com> and John W. Emerson
<jayemerson@gmail.com>

Maintainer Jay and Mike <bigmemoryauthors@gmail.com>

Depends methods, utils

Enhances biganalytics, bigtabulate, bigalgebra, synchronicity

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

Copyright (C) 2011 John W. Emerson and Michael J. Kane

URL <http://www.bigmemory.org>

LazyLoad yes

Repository CRAN

Date/Publication 2011-07-26 17:36:16

R topics documented:

bigmemory-package	2
as.big.matrix-methods	4
big.matrix, filebacked.big.matrix, etc...	5
big.matrix-class	8
big.matrix.descriptor-class	10
deepcopy	11
describe, attach.big.matrix, attach.resource	12

flush	14
morder, mpermute	15
mwhich	16
mwhich-methods	18
sub.big.matrix, is.sub.big.matrix	19
write.big.matrix, read.big.matrix	20

Index	23
--------------	-----------

bigmemory-package	<i>Manage massive matrices with shared memory and memory-mapped files.</i>
-------------------	--

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 in R by an S4 class whose interface is similar to that of an R `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 R `matrix` objects.

Details

Package:	bigmemory
Type:	Package
Version:	4.2.11
Date:	2011-07-26
License:	LGPL-3
Copyright:	(C) 2011 Michael J. Kane and John W. Emerson
URL:	http://www.bigmemory.org
LazyLoad:	yes

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 R 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 R 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" by 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 R 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.

Versions < 4.0 supported a limited number of columns (due to mutex limitations): roughly 50,000 on a typical Linux system. This restriction has been removed in versions ≥ 4.0 . There were row limitations (due to a bug that has now been fixed) in versions ≤ 3.8 of roughly 1 billion, but this has been fixed in versions ≥ 3.82 .

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

Author(s)

Michael J. Kane and John W. Emerson

Maintainers: Mike and Jay <bigmemoryauthors@gmail.com>

References

The Bigmemory Project: <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") ...

big.matrix, filebacked.big.matrix, etc...

The core “big.matrix” operations.

Description

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

Usage

```
big.matrix(nrow, ncol, type = options()$bigmemory.default.type,
  init = NULL, dimnames = NULL, separated = FALSE,
  backingfile = NULL, backingpath = NULL, descriptorfile = NULL,
  shared = TRUE)
filebacked.big.matrix(nrow, ncol,
  type = options()$bigmemory.default.type, init = NULL,
  dimnames = NULL, separated = FALSE, backingfile = NULL,
  backingpath = NULL, descriptorfile = NULL)
as.big.matrix(x, type = NULL, separated = FALSE,
  backingfile = NULL, backingpath = NULL, descriptorfile = NULL,
  shared=TRUE)
is.big.matrix(x)
is.separated(x)
is.filebacked(x)
is.shared(x)
is.nil(address)
```

Arguments

<code>x</code>	a <code>matrix</code> , vector, or data frame for <code>as.big.matrix</code> ; if a vector, a one-column <code>big.matrix</code> is created by <code>as.big.matrix</code> ; if a data frame, see details. For the <code>is.*</code> functions, <code>x</code> is likely a <code>big.matrix</code> .
<code>nrow</code>	number of rows.
<code>ncol</code>	number of columns.
<code>type</code>	the type of the atomic element (<code>options()\$bigmemory.default.type</code> by default – “double” – but can be changed by the user to “integer”, “short”, or “char”).
<code>init</code>	a scalar value for initializing the matrix (NULL by default to avoid unnecessary time spent doing the initializing).
<code>dimnames</code>	a list of the row and column names; use with caution for large objects.
<code>separated</code>	use separated column organization of the data; see details.
<code>backingfile</code>	the root name for the file(s) for the cache of <code>x</code> .
<code>backingpath</code>	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.
shared	TRUE by default, and always TRUE if the <code>big.matrix</code> is file-backed. For a non-filebacked <code>big.matrix</code> , <code>shared=FALSE</code> 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.
address	an externalptr, so <code>is.nil(x@address)</code> 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 inadvertant 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>

References

The Bigmemory Project: <http://www.bigmemory.org/>.

See Also

[bigmemory](#), and perhaps the class documentation of [big.matrix](#); [attach.big.matrix](#) and [describe.big.matrix](#). Sister packages [biganalytics](#), [bigtabulate](#), [synchronicity](#), and [bigalgebra](#) provide advanced functionality.

Examples

```
x <- big.matrix(10, 2, type='integer', init=-5)
options(bigmemory.allow.dimnames=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[,]
```

```
x <- as.big.matrix(matrix(-5, 10, 2))
colnames(x) <- c("alpha", "beta")
is.big.matrix(x)
dim(x)
colnames(x)
rownames(x)
x[1:8,1] <- 11:18
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[,]

# A short filebacked example, showing the creation of associated files:
files <- dir()
files[grep("example.bin", files)]
z <- filebacked.big.matrix(3, 3, type='integer', init=123,
                           backingfile="example.bin",
                           descriptorfile="example.desc",
                           dimnames=list(c('a','b','c'), c('d', 'e', 'f')))

z[,]
files <- dir()
files[grep("example.bin", files)]
zz <- attach.big.matrix("example.desc")
zz[,]
zz[1,1] <- 0
zzz <- attach.big.matrix(describe(z))
zzz[,]

is.nil(z@address)

```

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

head signature(x = "big.matrix"): get the first 6 (or n) rows.

as.matrix signature(x = "big.matrix"): coerce a big.matrix to a matrix.

is.big.matrix signature(x = "big.matrix"): return TRUE if it's a big.matrix.

is.filebacked signature(x = "big.matrix"): return TRUE if there is a file-backing.

is.separated signature(x = "big.matrix") : return TRUE if the big.matrix is organized as a separated column vectors.

is.sub.big.matrix signature(x = "big.matrix"): return TRUE if this is a sub-matrix of a big.matrix.

ncol signature(x = "big.matrix"): returns the number of columns.

nrow signature(x = "big.matrix"): returns the number of rows.

print signature(x = "big.matrix"): a traditional print() is intentionally disabled, and returns head(x) unless options()\$bm.print.warning==FALSE; in this case, print(x[,]) is the result, which could be very big!

sub.big.matrix signature(x = "big.matrix"): for contiguous submatrices.

tail signature(x = "big.matrix"): returns the last 6 (or n) rows.

typeof signature(x = "big.matrix"): return the type of the atomic elements of the big.matrix.

write.big.matrix signature(bigMat = "big.matrix", fileName = "character"): produce an ASCII file from the big.matrix.

apply signature(x = "big.matrix"): apply() where MARGIN may only be 1 or 2, but otherwise conforming to what you would expect from apply().

See Also

[big.matrix](#)

Examples

```
showClass("big.matrix")
```

```
big.matrix.descriptor-class
  Class "big.matrix.descriptor"
```

Description

An object of this class contains necessary and sufficient information to “attach” a shared or file-backed [big.matrix](#).

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](#).

References

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

See Also

See also [attach.big.matrix](#).

Examples

```
showClass("big.matrix.descriptor")
```

deepcopy

Produces a physical copy of a "big.matrix"

Description

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

Usage

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

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 (<code>matrix</code> or <code>big.matrix</code>); if not specified, a <code>big.matrix</code> 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.
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 R syntax would only copy the R 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

```
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)
```

describe, attach.big.matrix, attach.resource

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

```
describe(x)
attach.big.matrix(obj, ...)
attach.resource(obj, ...)
```

Arguments

x	a <code>big.matrix</code> object
obj	an object as returned by <code>describe()</code> 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 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[, ]
```

flush

Updating a big.matrix filebacking.

Description

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

Usage

```
flush(con)
```

Arguments

con a filebacked `big.matrix`.

Details

This function flushes any modified data (in RAM) of a file-backed `big.matrix` to disk. This may be useful for improving performance in cases where allowing the operating system to decide on flushing creates a bottleneck (likely near the threshold of available RAM).

Value

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

Examples

```
x <- big.matrix(nrow=3, ncol=3, backingfile='flusstest.bin',
  descriptorfile='flusstest.desc', type='integer')
x[1,1] <- 0
flush(x)
```

morder, mpermute	<i>Ordering and row-permuting functions for “big.matrix” and “matrix” objects</i>
------------------	---

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

```
morder(x, cols, na.last=TRUE, decreasing = FALSE)
mpermute(x, order=NULL, cols=NULL, allow.duplicates=FALSE, ...)
```

Arguments

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

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

```
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

```
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.

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](#)

sub.big.matrix, 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

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

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.

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[,]
y[1,1] <- -99
x[,]
```

```
write.big.matrix, read.big.matrix
```

File interface for a "big.matrix"

Description

Create a [big.matrix](#) by reading from a suitably-formatted ASCII file, or write the contents of a [big.matrix](#) to a file.

Usage

```
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, extraCols = NULL,
                shared=TRUE)
```

Arguments

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

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.
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[, ]
write.big.matrix(x, "foo.txt")

# Just for fun, I'll read it back in as character (1-byte integers):
y <- read.big.matrix("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, "bar.txt")  
w <- read.big.matrix("bar.txt", type="double")  
w[,]  
w <- read.big.matrix("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, 'IrisData.txt', col.names=TRUE, row.names=TRUE)  
y <- read.big.matrix("IrisData.txt", header=TRUE, has.row.names=TRUE)  
head(y)  
  
# The following would fail with a dimension mismatch:  
if (FALSE) y <- read.big.matrix("IrisData.txt", header=TRUE)
```

Index

*Topic **classes**

- big.matrix,
 - filebacked.big.matrix, etc..., [5](#)
- big.matrix-class, [8](#)
- big.matrix.descriptor-class, [10](#)
- describe, attach.big.matrix,
 - attach.resource, [12](#)

*Topic **methods**

- as.big.matrix-methods, [4](#)
- big.matrix,
 - filebacked.big.matrix, etc..., [5](#)
- deepcopy, [11](#)
- describe, attach.big.matrix,
 - attach.resource, [12](#)
- flush, [14](#)
- mwhich, [16](#)
- mwhich-methods, [18](#)
- sub.big.matrix, is.sub.big.matrix, [19](#)
- write.big.matrix, read.big.matrix, [20](#)

*Topic **package**

- bigmemory-package, [2](#)
- [,big.matrix,ANY,ANY,logical-method (big.matrix-class), [8](#)
- [,big.matrix,ANY,ANY,missing-method (big.matrix-class), [8](#)
- [,big.matrix,ANY,missing,logical-method (big.matrix-class), [8](#)
- [,big.matrix,ANY,missing,missing-method (big.matrix-class), [8](#)
- [,big.matrix,missing,ANY,logical-method (big.matrix-class), [8](#)
- [,big.matrix,missing,ANY,missing-method (big.matrix-class), [8](#)
- [,big.matrix,missing,missing,logical-method (big.matrix-class), [8](#)

- [,big.matrix,missing,missing,missing-method (big.matrix-class), [8](#)
- [.big.matrix (big.matrix, filebacked.big.matrix, etc...), [5](#)
- [<-,big.matrix,ANY,ANY-method (big.matrix-class), [8](#)
- [<-,big.matrix,ANY,missing-method (big.matrix-class), [8](#)
- [<-,big.matrix,missing,ANY-method (big.matrix-class), [8](#)
- [<-,big.matrix,missing,missing-method (big.matrix-class), [8](#)

- as.big.matrix (big.matrix, filebacked.big.matrix, etc...), [5](#)

- as.big.matrix,data.frame-method (as.big.matrix-methods), [4](#)
- as.big.matrix,matrix-method (as.big.matrix-methods), [4](#)
- as.big.matrix,vector-method (as.big.matrix-methods), [4](#)
- as.big.matrix-methods, [4](#)

- as.matrix,big.matrix-method (big.matrix-class), [8](#)

- attach.big.matrix, [6](#), [7](#), [11](#)
- attach.big.matrix (describe, attach.big.matrix, attach.resource), [12](#)

- attach.resource (describe, attach.big.matrix, attach.resource), [12](#)

- attach.resource,big.matrix.descriptor-method (big.matrix.descriptor-class), [10](#)

- attach.resource,character-method (big.matrix.descriptor-class), [10](#)

- big.matrix, 2–4, 7, 10–14, 16–21
- big.matrix (big.matrix, filebacked.big.matrix, etc...), 5
- big.matrix, filebacked.big.matrix, etc..., 5
- big.matrix-class, 8
- big.matrix.descriptor-class, 10
- bigmemory, 7, 9, 13
- bigmemory (bigmemory-package), 2
- bigmemory-package, 2
- data.frame, 4
- deepcopy, 11
- describe, 7, 10
- describe (describe, attach.big.matrix, attach.resource), 12
- describe, attach.big.matrix, attach.resource, 12
- describe, big.matrix-method (big.matrix-class), 8
- descriptor, 10
- descriptor-class (big.matrix.descriptor-class), 10
- dim, big.matrix-method (big.matrix-class), 8
- dimnames, big.matrix-method (big.matrix-class), 8
- dimnames<-, big.matrix, list-method (big.matrix-class), 8
- filebacked.big.matrix (big.matrix, filebacked.big.matrix, etc...), 5
- flush, 14
- flush, big.matrix-method (big.matrix-class), 8
- head, big.matrix-method (big.matrix-class), 8
- intersect, 17
- is.big.matrix (big.matrix, filebacked.big.matrix, etc...), 5
- is.big.matrix, ANY-method (big.matrix-class), 8
- is.big.matrix, big.matrix-method (big.matrix-class), 8
- is.filebacked (big.matrix, filebacked.big.matrix, etc...), 5
- is.filebacked, big.matrix-method (big.matrix-class), 8
- is.nil (big.matrix, filebacked.big.matrix, etc...), 5
- is.separated (big.matrix, filebacked.big.matrix, etc...), 5
- is.separated, big.matrix-method (big.matrix-class), 8
- is.shared (big.matrix, filebacked.big.matrix, etc...), 5
- is.shared, big.matrix-method (big.matrix-class), 8
- is.sub.big.matrix (sub.big.matrix, is.sub.big.matrix), 19
- is.sub.big.matrix, big.matrix-method (big.matrix-class), 8
- kmeans, 3
- length, big.matrix-method (big.matrix-class), 8
- lm, 3
- matrix, 2, 5
- morder, 3
- morder (morder, mpermute), 15
- morder, mpermute, 15
- mpermute, 3
- mpermute (morder, mpermute), 15
- mwhich, 3, 4, 16, 18
- mwhich, big.matrix, ANY, ANY, ANY, character-method (mwhich-methods), 18
- mwhich, big.matrix, ANY, ANY, ANY, missing-method (mwhich-methods), 18
- mwhich, matrix, ANY, ANY, ANY, character-method (mwhich-methods), 18
- mwhich, matrix, ANY, ANY, ANY, missing-method (mwhich-methods), 18
- mwhich-methods, 18
- ncol, big.matrix-method (big.matrix-class), 8
- nrow, big.matrix-method (big.matrix-class), 8

order, 16

print, big.matrix-method
 (big.matrix-class), 8

read.big.matrix, 4

read.big.matrix (write.big.matrix,
 read.big.matrix), 20

read.big.matrix, character-method
 (big.matrix-class), 8

split, 3

sub.big.matrix (sub.big.matrix,
 is.sub.big.matrix), 19

sub.big.matrix, is.sub.big.matrix, 19

sub.big.matrix, big.matrix-method
 (big.matrix-class), 8

sub.big.matrix, big.matrix.descriptor-method
 (big.matrix.descriptor-class),
 10

tail, big.matrix-method
 (big.matrix-class), 8

typeof, big.matrix-method
 (big.matrix-class), 8

union, 17

which, 16–18

write.big.matrix (write.big.matrix,
 read.big.matrix), 20

write.big.matrix, read.big.matrix, 20

write.big.matrix, big.matrix, character-method
 (big.matrix-class), 8