pbdDMAT

July 23, 2013

`pbdDMAT-package` *Distributed Matrix Methods*

Description

A package for dense distributed matrix computations. Includes the use of PBLAS and ScaLAPACK libraries via pbdSLAP, communicating over MPI via the BLACS library and pbdMPI.

Details

Package:	pbdDMAT
Type:	Package
License:	GPL
LazyLoad:	yes

This package requires an MPI library (OpenMPI, MPICH2, or LAM/MPI).

Author(s)

Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel, with contributions from R Core team (some wrappers taken from the base and stats packages).

References

Programming with Big Data in R Website: http://r-pbd.org/

ddmatrix-class *Class ddmatrix*

Description

Distributed matrix class.

Creating Objects

```
new('ddmatrix', Data = ..., dim = ..., ldim = ...,  bldim = ..., ICTXT = ...)
```

Slots

Data: LOCAL: Object of class matrix

dim: GLOBAL: Object of class numeric

ldim: LOCAL: Object of class numeric

bldim: GLOBAL: Object of class numeric

ICTXT: GLOBAL: Object of class numeric

Prototype

matrix Data matrix(0.0)

numeric dim c(1,1)

numeric ldim c(1,1)

numeric bldim c(1,1)

numeric ICTXT 0

Details

ddmatrix is the container for ScaLAPACK-friendly parallel block-cyclically distributed matrices. The class object is instantiated in SPMD fashion, whereby each process owns a piece of the "whole" matrix (which no single R process need ever own in its entirety), and each process stores its piece of the whole into a container with a name common to all processes.

The Data slot contains the data (submatrix) belonging to that process. Accessible via submatrix(). Values in the Data slot will vary from process to process.

The dim slot contains the global dimension; the dimension of the full matrix. Accessible via dim(). The dim slot is global, i.e. each process stores the same information in this slot.

The ldim slot contains the local dimension; here, all(ldim == dim(Data)). Accessible via ldim(). Values in the Data slot will vary from process to process.

The bldim slot contains the blocking factor for the block- cyclic distribution of the data. It consists of two numbers, namely the row and column blocking, respectively. Accessible via bldim(). The bldim slot is global, i.e. each process stores the same information in this slot.

The ICTXT slot contains the BLACS context onto which the matrix information is stored. This is mostly for internal bookkeeping, though advanced users might be able to effectively leverage differing BLACS contexts for performance improvements. For details, see InitGrid. Accessible via ictxt(). The ICTXT slot is global, i.e. each process stores the same information in this slot.

A very important piece of information is that every process must own something in the Data slot. This is essentially a ScaLAPACK "problem", but one that is not particularly hard to avoid so long as you are aware that it exists. A submatrix of `matrix(0, nrow=1, ncol=1)` is used if the matrix should not actually, technically, own part of the whole global matrix. You can easily still see if the stored submatrix is indeed part of the global matrix or just a placeholder with the `ownany()` function, which is just a wrapper on `numroc()` with argument `fixme=FALSE`.

See Also

InitGrid, SlotAccessors

---

pbdDMAT Control *Some default parameters for pbdDMAT.*

Description

This set of controls is used to provide default values in this package.

Format

Objects contain several parameters for communicators and methods.

Details

The default blocking `.BLDIM` is `c(4,4)`, which results in a 4 by 4 blocking dimension for distributed matrices. Any time a function takes the `bldim=` argument, it will default to this value unless the user specifies an alternative.

The default ICTXT is 0. This is the full 2-dimensional processor grid.

SlotAccessors *Accessor Functions for Distributed Matrix Slots*

Description

Functions to get dimension information, local storage, or current BLACS context from a distributed matrix.

Usage

```
  ## S4 method for signature 'ddmatrix'
nrow(x)
  ## S4 method for signature 'ddmatrix'
ncol(x)
  ## S4 method for signature 'ddmatrix'
NROW(x)
  ## S4 method for signature 'ddmatrix'
NCOL(x)
  ## S4 method for signature 'ddmatrix'
length(x)
  ## S4 method for signature 'ddmatrix'
dim(x)
  ## S4 method for signature 'ddmatrix'
submatrix(x)
  ## S4 method for signature 'ddmatrix'
ldim(x)
  ## S4 method for signature 'ddmatrix'
bldim(x)
  ## S4 method for signature 'ddmatrix'
ICTXT(x)
  ## S4 method for signature 'ddmatrix'
ownany(x, ...)
  ## S4 method for signature 'missing'
ownany(dim, bldim=.BLDIM, ICTXT=.ICTXT, x)
```

Arguments

`x`	numeric distributed matrix
`dim`	global dimension.
`bldim`	blocking dimension.
`ICTXT`	BLACS context.
`...`	Extra arguments.

Details

The functions `nrow()`, `ncol()`, `length()` and `dim()` are the natural extensions of their ordinary matrix counterparts.

`ldim()` will give the dimension of the matrix stored locally on the process which runs the function. This is a local value, so its return is process-dependent. For example, if the 3x3 global matrix `x` is distributed as the `ddmatrix` `dx` across two processors with process 0 owning the first two rows and process 1 owning the third, then `ldim(dx)` will return `2 3` on process 0 and `1 3` on process 1.

`bldim()` will give the blocking dimension that was used to block-cyclically distribute the distributed matrix.

`submatrix()` will give the local storage for the requested object.

`ICTXT()` will give the current BLACS context (slot ICTXT) for the requested object.

ownany() is intended mostly for developers. It answers the question "do I own any of the data?". The user can either pass a distributed matrix object or the dim, bldim, and ICTXT of one.

Value

Each of dim(), ldim(), bldim() return a length 2 vector.

Each of nrow(), ncol(), and length() return a length 1 vector. Likewise, so does ICTXT().

submatrix() returns a matrix; namely, submatrix(x) returns a matrix of dimensions ldim(x).

Methods

signature(x = "ddmatrix")

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

x <- ddmatrix(1:9, 3, 3)
x <- as.ddmatrix(x)

y <- list(dim=dim(x), ldim=ldim(x), bldim=bldim(x))
comm.print(y)

finalize()

## End(Not run)
```

DistributedMatrixCreation

Distributed Matrix Creation

Description

Methods for simple construction of distributed matrices.

Usage

```
  ## S4 method for signature 'character'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
                                  ..., min = 0, max = 1, mean = 0, sd = 1,
                                  rate = 1, shape, scale = 1,
                                   bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'matrix'
```

```
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'missing'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'vector'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'character'
ddmatrix.local(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., min = 0, max = 1, mean = 0, sd = 1,
                                 rate = 1, shape, scale = 1,
                                  bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'matrix'
ddmatrix.local(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'missing'
ddmatrix.local(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'vector'
ddmatrix.local(data, nrow = 1, ncol = 1, byrow = FALSE,
                                 ..., bldim = .BLDIM, ICTXT = .ICTXT)
```

Arguments

`data`	optional data vector.
`nrow`	number of rows. Global rows for `ddmatrix()`. Local rows for `ddmatrix.local()`. See details below.
`ncol`	number of columns. Global columns for `ddmatrix()`. Local columns for `ddmatrix.local()`. See details below.
`byrow`	logical. If `FALSE` then the distributed matrix will be filled by column major storage, otherwise row-major.
`...`	Extra arguments
`min, max`	Min and max values for random uniform generation.
`mean,sd`	Mean and standard deviation for random normal generation.
`rate`	Rate for random exponential generation.
`shape,scale`	Shape and scale parameters for random weibull generation.
`bldim`	blocking dimension.
`ICTXT`	BLACS context number.

Details

These methods are simplified methods of creating distributed matrices, including random ones. These methods involve only local computations, i.e., no communication is performed in the construction of a `ddmatrix` using these methods (in contrast to using `as.ddmatrix()` et al).

For non-character inputs, the methods attempt to mimic R as closely as possible. So `ddmatrix(1:3, 5, 7)` produces the distributed analogue of `matrix(1:3, 5, 7)`.

For character inputs, you may also specify additional parametric family information.

The functions predicated with `.local` generate data with a fixed local dimension, i.e., each processor gets an identical amount of data. Likewise, the remaining functions generate a fixed global amount of data, and each processor may or may not have an identical amount of local data.

To ensure good random number generation, you should only consider using the character methods with the `comm.set.seed()` function from pbdMPI which uses the method of L'Ecuyer via the rlecuyer package.

Value

Returns a distributed matrix.

Methods

`signature(data = "character")`

`signature(data = "matrix")`

`signature(data = "missing")`

`signature(data = "vector")`

See Also

`as.ddmatrix`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

dx <- ddmatrix(data="rnorm", nrow=5, ncol=6, mean=10, sd=100)
dy <- ddmatrix(data=1:4, nrow=7, ncol=5)

print(dx)
print(dy)

finalize()

## End(Not run)
```

Diag *Distributed Matrix Diagonals*

Description

Get the diagonal of a distributed matrix, or construct a distributed matrix which is diagonal.

Usage

```
  ## S4 method for signature 'ddmatrix'
diag(x)
  ## S4 method for signature 'vector'
diag(x, nrow, ncol, type = "matrix", ...,
                          bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'character'
diag(x, nrow, ncol, type = "matrix", ...,
                             min = 0, max = 1, mean = 0, sd = 1,
                             rate = 1, shape, scale = 1,
                             bldim = .BLDIM, ICTXT = .ICTXT)
```

Arguments

`x`	distributed matrix or a vector.
`nrow,ncol`	in the case that `x` is a vector, these specify the global dimension of the diagonal distributed matrix to be created.
`type`	character. Options are 'matrix' or 'ddmatrix', with partial matching. This specifies the return type.
`...`	Extra arguments
`min, max`	Min and max values for random uniform generation.
`mean,sd`	Mean and standard deviation for random normal generation.
`rate`	Rate for random exponential generation.
`shape,scale`	Shape and scale parameters for random weibull generation.
`bldim`	blocking dimension.
`ICTXT`	BLACS context number.

Details

Gets the diagonal of a distributed matrix and stores it as a global R vector owned by all processes.

Value

If a distributed matrix is passed to `diag()` then it returns a global R vector.

If a vector (numeric or character) is passed to `diag()` and `type='ddmatrix'`, then the return is a diagonal distributed matrix.

Methods

signature(x = "ddmatrix")

signature(x = "vector")

signature(x = "character")

See Also

Extract

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:16, 4)
x <- as.ddmatrix(x)

y <- diag(x)
comm.print(y)

finalize()

## End(Not run)
```

---

as.ddmatrix *Simplified Syntax to Distribute Matrix Across Process Grid*

Description

A simplified interface to the `distribute()` and `redistribute()` functions.

Usage

```
  ## S4 method for signature 'NULL'
as.ddmatrix(x, bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'vector'
as.ddmatrix(x, bldim = .BLDIM, ICTXT = .ICTXT)
  ## S4 method for signature 'matrix'
as.ddmatrix(x, bldim = .BLDIM, ICTXT = .ICTXT)
```

Arguments

x	a numeric matrix
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid.
ICTXT	BLACS context number for return.

Details

A simplified wrapper for the `distribute()` function, especially in the case that the matrix `x` is global (which you really should not ever let happen outside of testing, but I won't stop you).

The function will only work if `x` is stored on all processes, or `x` is stored on a single process (does not matter which) and every other process has NULL stored for x.

If several processes own pieces of the matrix `x`, then you can not use this function. You will have to create an appropriate `ddmatrix` on all processes and redistriubte the data with the `redistribute()` function.

As usual, the `ICTXT` number is the BLACS context corresponding to the process grid onto which the output distributed matrix will be distributed.

Value

Returns a distributed matrix.

See Also

`Distribute`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

if (comm.rank()==0){
  x <- matrix(1:16, ncol=4)
} else {
  x <- NULL
}

dx <- as.ddmatrix(x, bldim=c(4,4))
print(dx)

finalize()

## End(Not run)
```

as.matrix *Distributed-to-non-distributed Matrix Converters*

Description

Converts objects of class `ddmatrix` to the requested non-distributed type.

Usage

```
  ## S4 method for signature 'ddmatrix'
as.vector(x, mode = 'any', proc.dest = 'all')
  ## S4 method for signature 'ddmatrix'
as.matrix(x, proc.dest = 'all', attributes = TRUE)
```

Arguments

`x`	numeric distributed matrix
`mode`	A character string giving an atomic mode or "list", or (except for 'vector') "any".
`proc.dest`	destination process for storing the matrix
`attributes`	logical, specifies whether or not the current attributes should be preserved.

Details

Converts a distributed matrix into a non-distributed vector or matrix.

The `proc.dest=` argument accepts either the BLACS grid position or the MPI rank if the user desires a single process to own the matrix. Alternatively, passing the default value of `'all'` will result in all processes owning the matrix. If only a single process owns the undistributed matrix, then all other processes store `NULL` for that object.

Value

Returns an ordinary R matrix.

Methods

`signature(x = "ddmatrix")`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
```

```
x <- matrix(1:16, ncol=4)
dx <- as.ddmatrix(x)

y <- as.matrix(dx, proc.dest=0)

finalize()

## End(Not run)
```

`Distribute` *Distribute/Redistribute matrices across the process grid*

Description

Takes either an R matrix and distributes it as a distributed matrix, or takes a distributed matrix and redistributes it across a (possibly) new BLACS context, using a (possibly) new blocking dimension.

Usage

```
distribute(x, bldim = .BLDIM, xCTXT = 0, ICTXT = .ICTXT)
redistribute(dx, bldim = dx@bldim, ICTXT = .ICTXT)
```

Arguments

`x`	a numeric matrix
`dx`	numeric distributed matrix
`bldim`	the blocking dimension for block-cyclically distributing the matrix across the process grid.
`xCTXT`	the BLACS context number for initial distribution of the matrix `x`.
`ICTXT`	BLACS context number for return.

Details

`distribute()` takes an R matrix `x` stored on the processes in some fashion and distributes it across the process grid belonging to `ICTXT`. If a process is to call `distribute()` and does not yet have any ownership of the matrix `x`, then that process should store `NULL` for `x`.

How one might typically use this is to read in a non-distributed matrix on the first process, store that result as the R matrix `x`, and then have the other processes store `NULL` for `x`. Then calling `distribute()` returns the distributed matrix which was distributed according to the options `bldim` and `ICTXT`.

Using an `ICTXT` value other than zero is not recommended unless you have a good reason to. Use of other such contexts should only be considered for advanced users, preferably those with knowledge of ScaLAPACK.

`redistribute()` takes a distributed matrix and redistributes it to the (possibly) new process grid with BLACS context `ICTXT` and with the (possibly) new blocking dimension `bldim`. The original BLACS context is `dx@CTXT` and the original blocking dimension is `dx@bldim`.

These two functions are essentially simple wrappers for the ScaLAPACK function PDGEMR2D, with the above described behavior. Of note, for `distribute()`, `dx@CTXT` and `ICTXT` must share at least one process in common. Likewise for `redistribute()` with `xCTXT` and `ICTXT`.

Very general redistributions can be done with `redistribute()`, but thinking in these terms is an acquired skill. For this reason, several simple interfaces to this function have been written. See SimpleRedistributions for details.

Value

Returns a distributed matrix.

See Also

`as.ddmatrix`, `BLACS`, `InitGrid`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

if (comm.rank()==0){
  x <- matrix(1:16, ncol=4)
} else {
  x <- NULL
}

dx <- distribute(x, bldim=c(4,4))
print(dx)

dx <- redistribute(dx, bldim=c(3,3))
print(dx)


finalize()

## End(Not run)
```

`SimpleRedistributions` *Distribute/Redistribute matrices across the process grid*

Description

Takes either an R matrix and distributes it as a distributed matrix, or takes a distributed matrix and redistributes it across a (possibly) new BLACS context, using a (possibly) new blocking dimension.

Usage

```
as.block(dx, square.bldim = TRUE)
as.rowblock(dx)
as.colblock(dx)
as.rowcyclic(dx, bldim = .BLDIM)
as.colcyclic(dx, bldim = .BLDIM)
as.blockcyclic(dx, bldim = .BLDIM)
```

Arguments

`dx`	numeric distributed matrix
`square.bldim`	logical. Determines whether or not the blocking factor for the resulting redistributed matrix will be square or not.
`bldim`	the blocking dimension for block-cyclically distributing the matrix across the process grid.

Details

These functions are simple wrappers of the very general `redistribute()` funciton (see Distribute). Different distributed matrix distributions of note can be classified into three categories: block, cyclic, and block-cyclic.

`as.block()` will convert `ddmatrix` into one which is merely "block" distributed, i.e., the blocking factor is chosen in such a way that there will be no cycling. By default, this new blocking factor will be square. This can result in some raggedness (some processors owning less than others — or nothing) if the matrix is far from square itself. However, the methods of factoring `ddmatrix` objects, and therefore anything that relies on (distributed) matrix factorizations such as computing an inverse, least squares solution, etc., require that blocking factors be square. The matrix will not change BLACS contexts.

`as.rowblock()` will convert a distributed matrix into one which is distributed by row into a block distributed matrix. That is, the rows are stored contiguously, and different processors will own different rows, but with no cycling. In other words, it block redistributes the data across context 2.

`as.colblock()` is the column-wise analogue of `as.rowblock()`. In other words, it block redistributes the data across context 1.

`as.rowcyclic()` is a slightly more general version of `as.rowblock()`, in that the data will be distributed row-wise, but with the possibility of cycling, as determined by the blocking factor. In other words it block-cyclically redistributes the data across context 2.

`as.colcyclic()` is a the column-wise analogue of `as.rowcyclic()`. In other words, it block-cyclically redistributes the data across context 1.

`as.blockcyclic()` moves the distributed matrix into a general block-cyclic distribution across a 2-dimensional process grid. In other words, it block-cyclically redistributes the data across context 0.

Value

Returns a distributed matrix.

See Also

as.ddmatrix, Distribute, BLACS

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

dx <- ddmatrix(1:30, nrow=10)

x <- as.block(dx)
x

x <- as.rowblock(dx)
x

x <- as.colblock(dx)
x

x <- as.rowcyclic(dx)
x

x <- as.colcyclic(dx)
x

x <- as.blockcyclic(dx)
x

finalize()

## End(Not run)
```

Print *Printing a Distributed Matrix*

Description

Print method for a distributed matrices.

Usage

```
  ## S4 method for signature 'ddmatrix'
print(x, ..., all = FALSE, name = "x")
```

Arguments

`x`	numeric distributed matrix
`...`	additional arguments
`all`	control for whether the entire distributed matrix should be printed to standard output
`name`	character string that will be printed to standard output along with the matrix elements

Details

Print method for class `ddmatrix`.

If argument `all=TRUE`, then a modified version of the ScaLAPACK TOOLS routine PDLAPRNT is used to print the entire distributed matrix. The matrix will be printed in column-major fashion, with one element of the matrix per line. If `all=FALSE` then the `name=` argument is ignored.

Value

The function silently returns 0.

Methods

`signature(x = "ddmatrix")`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:16, ncol=4)
dx <- as.ddmatrix(x)

print(dx)

print(dx, all=T)

finalize()

## End(Not run)
```

Summary *Distributed Matrix Summary*

Description

Summarize a distributed matrix. Gives min, max, mean, etc. by column.

Usage

```
  ## S4 method for signature 'ddmatrix'
summary(object)
```

Arguments

object	numeric distributed matrix

Details

The return is on process 0 only.

Value

A table on processor 0, NULL on all other processors.

Methods

signature(x = "ddmatrix")

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:16, ncol=4)
dx <- as.ddmatrix(x)

summary(dx)

finalize()

## End(Not run)
```

Extract *Extract or Replace Parts of a Distributed Matrix*

Description

Operators to extract or replace parts of a distributed matrix.

Usage

```
  x[i, j, ..., ICTXT]

  ## S3 method for class 'ddmatrix'
head(x, n = 6L, ...)
  ## S3 method for class 'ddmatrix'
tail(x, n = 6L, ...)
```

Arguments

x	numeric distributed matrix.
i, j	indices specifying elements to extract or replace. Indices can be numeric, character, empty, or NULL.
n	a single integer. If positive, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the n last/first number of elements of x.
...	additional arguments.
ICTXT	optional BLACS context number for output

Details

[can be used to extract/replace for a distributed matrix exactly as you would with an ordinary matrix.

The functions rely on reblocking across different BLACS contexts. If i is not empty, then the input distributed matrix will be redistributed along context 1, where extracting/deleting rows does not destroy block-cyclicality. Likewise, if j is not empty, then the input distributed matrix will be redistributed along context 2. When extraction is complete, the matrix will be redistributed across its input context.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

See Also

BLACS, InitGrid

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- x[, -1]
y <- head(y, 2)
print(y)

finalize()

## End(Not run)
```

Insert — *Directly Insert Into Distributed Matrix Submatrix Slot*

Description

Allows you to directly replace the submatrix of a distributed matrix.

Usage

```
x[i, j] <- value
submatrix(x) <- value
```

Arguments

x	numeric distributed matrix.
i, j	global integer indices.
value	replacement value. Can be a global vector or a ddmatrix.

Details

[<- allows the user to insert values into a distributed matrix in exactly the same way one would with an ordinary matrix. The indices here are global, meaning that x[i, j] refers to the (i, j)'th element of the "full", global matrix, and not necessarily the (i, j)'th element of the local submatrix.

On the other hand, submatrix<- is different. It is basically syntactic sugar for:

x@Data <- newMatrix

It does not alter the distributed matrix x's dim or bldim. It *does* adjust the ldim automatically. However, using this can be dangerous. It is merely provided to give consistent behavior with the submatrix() function.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

See Also

BLACS, InitGrid

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

x[1, ] <- 0
comm.print(submatrix(x), all.rank=T)

finalize()

## End(Not run)
```

Comparators *Logical Comparisons*

Description

Logical comparisons.

Usage

```
  x == y
  x < y
  x > y
  x >= y
  x <= y
  x != y
  x & y
  x | y
  ## S4 method for signature 'ddmatrix'
any(x, na.rm=FALSE)
  ## S4 method for signature 'ddmatrix'
all(x, na.rm=FALSE)
```

Arguments

`x, y`	distributed matrix or numeric vector
`na.rm`	logical, indicating whether or not `NA`'s should first be removed. If not and an NA is present, `NA` is returned.

Details

Performs the indicated logical comparison.

If `na.rm` is `TRUE` and only `NA`'s are present, then `TRUE` is returned.

Value

Returns a distributed matrix.

Methods

`signature(x = "ddmatrix")`

See Also

Type

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(sample(0, 1, 9, replace=T), 3)
comm.print(x)

x <- as.ddmatrix(x, bldim=2)

y <- any(x)
comm.print(y)

finalize()

## End(Not run)
```

Type *Type Checks, Including NA, NaN, etc.*

Description

Functions to check for various types.

Usage

```
  is.ddmatrix(x)
  ## S4 method for signature 'ddmatrix'
is.numeric(x)
  ## S4 method for signature 'ddmatrix'
is.na(x)
  ## S4 method for signature 'ddmatrix'
is.nan(x)
  ## S4 method for signature 'ddmatrix'
is.infinite(x)
```

Arguments

x	numeric distributed matrix

Details

Performs the appropriate type check.

Value

Returns boolean in the case of `is.numeric()` and `is.ddmatrix()`, otherwise a distributed matrix.

Methods

`signature(x = "ddmatrix")`

See Also

NAs

NAs *Handle Missing Values in Distributed Matrices*

Description

Dealing with NA's and NaN's.

Usage

```
  ## S4 method for signature 'ddmatrix'
na.exclude(object, ..., ICTXT)
```

Arguments

`object`	numeric distributed matrix
`...`	extra arguments
`ICTXT`	optional BLACS context number for output

Details

Removes rows containing NA's and NaN's.

The function relies on reblocking across different BLACS contexts. The input distributed matrix will be redistributed along context 1, where extracting/deleting rows does not destroy block-cyclicality.

Only advanced users should supply an `ICTXT` value. Most should simply leave this argument blank.

The context of the return is dependent on the function arguments. If the `ICTXT=` argument is missing, then the return will be redistributed across its input context `object@CTXT`. Otherwise, the return will be redistributed across the supplied `ICTXT`.

Methods

`signature(object = "ddmatrix")`

See Also

Type

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x[1, 1] <- NA
x <- as.ddmatrix(x)

y <- na.exclude(x)
comm.print(y)

finalize()

## End(Not run)
```

`Apply` *Apply Family of Functions*

Description

Apply a function to the margins of a distributed matrix.

Usage

```
  ## S4 method for signature 'ddmatrix'
apply(X, MARGIN, FUN, ..., reduce = FALSE,
  proc.dest="all")
```

Arguments

`X`	distributed matrix
`MARGIN`	subscript over which the function will be applied
`FUN`	the function to be applied
`...`	additional arguments to FUN
`reduce`	logical or string. See details
`proc.dest`	Destination process (or 'all') if a reduction occurs

Details

If reduce==TRUE then a global matrix or vector (whichever is more appropriate) will be returned. The argument proc.dest= behaves exactly as in the as.vector() and as.matrix() functions of **pbdDMAT**. If reduce=FALSE then a distributed matrix is returned. Other acceptable arguments are reduce="matrix" and reduce="vector" which demand global matrix or vector return, respectively. This should generally be slightly more efficient than running apply and then calling as.vector() or as.matrix().

Value

Returns a distributed matrix unless a reduction is requested, then a global matrix/vector is returned.

Methods

signature(x = "ddmatrix")

Author(s)

Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

See Also

prcomp

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- head(x[, -1], 2)
print(y)

finalize()

## End(Not run)
```

Binders *Row and Column binds for Distributed Matrices*

Description

Row and column binds.

Usage

```
  ## S4 method for signature '...'
rbind(..., ICTXT = .ICTXT, deparse.level = 1)
  ## S4 method for signature '...'
cbind(..., ICTXT = .ICTXT, deparse.level = 1)
```

Arguments

`...`	vectors, matrices, or distributed matrices.
`ICTXT`	BLACS communicator number for return object.
`deparse.level`	integer controlling the construction of labels in the case of non-matrix-like arguments. Does nothing for distributed matrices.

Details

The `...` list of arguments can be vectors, matrices, or distributed matrices so long as non-distributed objects are not used with distributed objects. This kind of mixing-and-matching will lead to chaos. Currently no check is performed to prevent the user from this mixing-and-matching for performance reasons (it is slow enough already).

Value

Returns a vector, matrix, or distributed matrix, depending on input.

Methods

`signature(...  = "ANY")` an R object.

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:16, ncol=4)
dx <- as.ddmatrix(x)
```

```
y <- rbind(dx, dx)

print(y)

finalize()

## End(Not run)
```

Reductions *Arithmetic Reductions: Sums, Means, and Prods*

Description

Arithmetic reductions for distributed matrices.

Usage

```
  ## S4 method for signature 'ddmatrix'
sum(x, ..., na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
mean(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
median(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
prod(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
rowSums(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
colSums(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
rowMeans(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
colMeans(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
min(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
max(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
rowMin(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
colMin(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
rowMax(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
colMax(x, na.rm = FALSE)
```

Arguments

`x`	numeric distributed matrix
`na.rm`	logical. Should missing (including `NaN`) be removed?
`...`	additional arguments

Details

Performs the reduction operation on a distributed matrix.

There are four legitimately new operations, namely `rowMin()`, `rowMax()`, `colMin()`, and `colMax()`. These implementations are not really necessary in R because one can easily (and reasonably efficiently) do something like

`apply(X=x, MARGIN=1L, FUN=min, na.rm=TRUE)`

But `apply()` on a `ddmatrix` is *very* costly, and should be used sparingly.

Value

Returns a global numeric vector.

Methods

`signature(x = "ddmatrix")`

See Also

`Arithmetic`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- sum(colMeans(x))
comm.print(y)

finalize()

## End(Not run)
```

Arithmetic *Arithmetic Operators*

Description

Binary operations for distributed matrix/distributed matrix and distributed matrix/vector operations.

Usage

```
x + y
x - y
-y
x * y
x / y
x ^ y
x %% y
x %/% y
```

Arguments

`x, y`	numeric distributed matrices or numeric vectors

Details

If `x` and `y` are distributed matrices, then they must be conformable, on the same BLACS context, and have the same blocking dimension.

Value

Returns a distributed matrix.

Methods

`signature(x = "ddmatrix", y = "ddmatrix")`

`signature(x = "numeric", y = "ddmatrix")`

`signature(x = "ddmatrix", y = "numeric")`

See Also

`Arithmetic`, `LinAlg`, `MatMult`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- (2*x) - x^(.5)
print(y)

finalize()

## End(Not run)
```

MatMult *Matrix Multiplication*

Description

Multiplies two distributed matrices, if they are conformable.

Usage

```
  x %*% y
  ## S4 method for signature 'ddmatrix,ANY'
crossprod(x, y = NULL)
  ## S4 method for signature 'ddmatrix,ANY'
tcrossprod(x, y = NULL)
```

Arguments

`x, y`	numeric distributed matrices

Details

`x` and `y` must be conformable, on the same BLACS context, but they need not be blocked with the same blocking dimension. The return will default to the blocking dimension of `x`.

If you need to use `x` and `y` with differing blocking dimensions and you want the return to have blocking different from that of `x`, then use the function `base.rpdgemm()`.

The `crossprod()` and `tcrossprod()` functions behave exactly as their R counterparts.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix", y = "ddmatrix")

signature(x = "ddmatrix", y = "ANY")

See Also

Arithmetic, LinAlg, MatMult

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- x %*% x
print(y)

finalize()

## End(Not run)
```

MiscMath *Miscellaneous Mathematical Functions*

Description

Binary operations for distributed matrix/distributed matrix and distributed matrix/vector operations.

Usage

```
  ## S4 method for signature 'ddmatrix'
abs(x)
  ## S4 method for signature 'ddmatrix'
sqrt(x)
  ## S4 method for signature 'ddmatrix'
exp(x)
  ## S4 method for signature 'ddmatrix'
log(x, base = exp(1))
  ## S4 method for signature 'ddmatrix'
log2(x)
  ## S4 method for signature 'ddmatrix'
```

```
log10(x)
  ## S4 method for signature 'ddmatrix'
log1p(x)
  ## S4 method for signature 'ddmatrix'
sin(x)
  ## S4 method for signature 'ddmatrix'
cos(x)
  ## S4 method for signature 'ddmatrix'
tan(x)
  ## S4 method for signature 'ddmatrix'
asin(x)
  ## S4 method for signature 'ddmatrix'
acos(x)
  ## S4 method for signature 'ddmatrix'
atan(x)
  ## S4 method for signature 'ddmatrix'
sinh(x)
  ## S4 method for signature 'ddmatrix'
cosh(x)
  ## S4 method for signature 'ddmatrix'
tanh(x)
```

Arguments

x	numeric distributed matrix
base	a positive number: the base with respect to which logarithms are computed. Defaults to e='exp(1)'.

Details

Performs the miscellaneous mathematical calculation on a distributed matrix.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

See Also

Arithmetic, Reductions

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r
```

```
library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- sqrt(abs(log(x/10)))
comm.print(y)

finalize()

## End(Not run)
```

Round *Rounding of Numbers*

Description

Extensions of R rounding functions for distributed matrices.

Usage

```
  ## S4 method for signature 'ddmatrix'
ceiling(x)
  ## S4 method for signature 'ddmatrix'
floor(x)

  ## S4 method for signature 'ddmatrix'
round(x, digits = 0)
```

Arguments

`x`	numeric distributed matrix
`digits`	integer indicating the number of decimal places (`round()`) or significant digits (`signif()`) to be used. Negative values are allowed (see 'Details').

Details

Rounding to a negative number of digits means rounding to a power of ten, so for example `round(x, digits = -2)` rounds to the nearest hundred.

Value

Returns a distributed matrix.

Methods

`signature(x = "ddmatrix")`

See Also

MiscMath, NAs

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- ceiling(x/3)
print(y)

finalize()

## End(Not run)
```

LinAlg *Linear Algebra Functions*

Description

Linear alegbra functions for distributed matrices with R-like syntax, with calculations performed by the PBLAS and ScaLAPACK libraries.

Usage

```
  ## S4 method for signature 'ddmatrix'
isSymmetric(object, tol = 100 * .Machine$double.eps, ...)
  ## S4 method for signature 'ddmatrix'
t(x)
  ## S4 method for signature 'ddmatrix,ddmatrix'
solve(a, b)
  ## S4 method for signature 'ddmatrix,ANY'
solve(a)
  ## S4 method for signature 'ddmatrix'
La.svd(x, nu, nv)
  ## S4 method for signature 'ddmatrix'
svd(x, nu, nv)
  ## S4 method for signature 'ddmatrix'
eigen(x, symmetric, only.values = FALSE)
  ## S4 method for signature 'ddmatrix'
```

```
chol(x)
  ## S4 method for signature 'ddmatrix'
lu(x)
```

Arguments

`object, x, a, b`
: numeric distributed matrices. If applicable, `a` and `b` must be on the same BLACS context and have the same blocking dimension.

`tol`
: precision tolerance.

`...`
: additional arguments.

`nu`
: number of left singular vectors to return when calculating singular values.

`nv`
: number of right singular vectors to return when calculating singular values.

`symmetric`
: logical, if `TRUE` then the matrix is assumed to be symmetric and only the lower triangle is used. Otherwise `x` is inspected for symmetry.

`only.values`
: logical, if `TRUE` then only the eigenvalues are returned. Otherwise both eigenvalues and eigenvectors are returned.

Details

Extensions of R linear algebra functions.

Value

`t()` returns the transposed matrix.

`solve()` solves systems and performs matrix inversion when argument `b=` is missing.

`La.svd()` performs singular value decomposition, and returns the transpose of right singular vectors if any are requested. Singular values are stored as a global R vector. Left and right singular vectors are unique up to sign. Sometimes core R (via LAPACK) and ScaLAPACK will disagree as to what the left/right singular vectors are, but the disagreement is always only up to sign.

`svd()` performs singular value decomposition. Differs from `La.svd()` in that the right singular vectors, if requested, are returned non-transposed. Singular values are stored as a global R vector. Sometimes core R (via LAPACK) and ScaLAPACK will disagree as to what the left/right singular vectors are, but the disagreement is always only up to sign.

`eigen()` computes the eigenvalues, and eigenvectors if requested. As with `svd()`, eigenvalues are stored in a global R vector.

`chol()` performs Cholesky factorization.

`lu()` performs LU factorization.

Methods

`signature(x = "ddmatrix")`

`signature(a = "ddmatrix")`

`signature(b = "ddmatrix")`

See Also

Arithmetic, Reductions, MatMult, MiscMath

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

y <- solve(t(A) %*% A)
print(y)

finalize()

## End(Not run)
```

QR Decomposition *QR Decomposition Methods*

Description

qr() takes the QR decomposition.

qr.Q() recovers Q from the output of qr().

qr.R() recovers R from the output of qr().

qr.qy() multiplies y by Q.

qr.qty() multiplies y by the transpose of Q.

Usage

```
  ## S4 method for signature 'ddmatrix'
qr(x, tol = 1e-07)
  ## S4 method for signature 'ANY'
qr.Q(x, complete = FALSE,  Dvec = rep.int(if (cmplx) 1 +
  (0+0i) else 1, if (complete) dqr[1] else min(dqr)))
  ## S4 method for signature 'ANY'
qr.R(x, complete = FALSE)
  ## S4 method for signature 'ANY'
qr.qy(x, y)
  ## S4 method for signature 'ANY'
qr.qty(x, y)
```

Arguments

`x, y`	numeric distributed matrices for `qr()`. Otherwise, `x` is a list, namely the return from `qr()`.
`tol`	logical value, determines whether or not columns are zero centered.
`complete`	logical expression of length 1. Indicates whether an arbitrary orthogonal completion of the Q or X matrices is to be made, or whether the R matrix is to be completed by binding zero-value rows beneath the square upper triangle.
`Dvec`	Not implemented for objects of class `ddmatrix`. vector (not matrix) of diagonal values. Each column of the returned Q will be multiplied by the corresponding diagonal value. Defaults to all 1's.

Details

Functions for forming a QR decomposition and for using the outputs of these numerical QR routines.

Value

`qr()` returns a list consisting of: `qr` - `rank` - calculated numerical rank, `tau` - `pivot` - `"class"` - attribute "qr".

Methods

`signature(x = "ddmatrix")`

`signature(x = "ANY")`

See Also

`lm.fit`

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(1:9, 3)
x <- as.ddmatrix(x)

Q <- qr.Q(qr(x))
print(Q)

finalize()

## End(Not run)
```

chol2inv *Inverse from Choleski (or QR) Decomposition*

Description

qr() takes the QR decomposition.

Usage

```
  ## S4 method for signature 'ddmatrix'
chol2inv(x, size = NCOL(x))
```

Arguments

x	numeric distributed matrices for
size	number of columns of x containing the Choleski factorization.

Details

The function returns the inverse of a choleski factored matrix, or the inverse of crossprod(x) if qr.R(qr(x)) is passed.

Value

A numeric distributed matrix.

Methods

signature(x = "ddmatrix")

signature(x = "ANY")

See Also

lm.fit

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)
x <- ddmatrix("rnorm", 3, 3)

R <- qr.R(qr(x))
```

```
xtx.inv <- chol2inv(R)

id <- xtx.inv

print(id)

finalize()

## End(Not run)
```

kappa *Compute or estimate the Condition Number of a Distributed Matrix*

Description

Computes or estimates the condition number.

Usage

```
  ## S3 method for class 'ddmatrix'
kappa(z, exact = FALSE, norm = NULL,
                             method = c("qr", "direct"), ...)
  ## S4 method for signature 'ddmatrix'
rcond(x, norm = c("O", "I", "1"),
                             triangular = FALSE, ...)
```

Arguments

`x,z`	numeric distributed matrices.
`exact`	logical. Determines whether exact condition number or approximation should be computed.
`norm`	character. Determines which matrix norm is to be used.
`method`	character. Determines the method use in computing condition number.
`triangular`	logical. If true, only the lower triangle is used.
`...`	Extra arguments.

Value

Returns a number.

Methods

`signature(x = "ddmatrix")`

`signature(z = "ddmatrix")`

See Also

Norm

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)
x <- ddmatrix("rnorm", 10, 10)

cnm <- rcond(x)

comm.print(cnm)

finalize()

## End(Not run)
```

Norm *Compute the Norm of a Distributed Matrix*

Description

Computes the norm.

Usage

```
## S4 method for signature 'ddmatrix'
norm(x, type = c("O", "I", "F", "M", "2"))
```

Arguments

x	numeric distributed matrices.
type	character. Determines which matrix norm is to be used.

Value

Returns a number.

Methods

signature(x = "ddmatrix")

See Also

ConditionNumbers

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)
x <- ddmatrix("rnorm", 10, 10)

nrm <- norm(x)

comm.print(nrm)

finalize()

## End(Not run)
```

---

scale *Scale*

Description

Centers and/or scales the columns of a distributed matrix.

Usage

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

Arguments

x	numeric distributed matrix.
center	logical value, determines whether or not columns are zero centered
scale	logical value, determines whether or not columns are rescaled to unit variance

Details

Centers and/or scales the columns of a distributed matrix.

Value

Returns a distributed matrix.

Methods

```
signature(x = "ddmatrix", center="ANY", scale="ANY")
```

Author(s)

R Core Team, Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

See Also

prcomp

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)

x <- ddmatrix("rnorm", 10, 10)
y <- scale(x)

print(y)

finalize()

## End(Not run)
```

---

sweep *Sweep*

Description

Sweep vector or ddmatrix from a distributed matrix.

Usage

```
  ## S4 method for signature 'ddmatrix,ANY,vector'
sweep(x, MARGIN, STATS, FUN = "-")
  ## S4 method for signature 'ddmatrix,ANY,ddmatrix'
sweep(x, MARGIN, STATS, FUN = "-")
```

Arguments

x	numeric distributed matrix.
MARGIN	subscript over which the function will be applied
STATS	array to be swept out.
FUN	function used in the sweep. Only +, -, *, and / are accepted. For more general operations, use apply().

Details

Sweep vector or ddmatrix from a distributed matrix.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix", MARGIN = "ANY", STATS = "vector")

signature(x = "ddmatrix", MARGIN = "ANY", STATS = "ddmatrix")

Variance/Covariance *Variance, Covariance, and Correlation*

Description

sd() forms the vector of column standard deviations. cov() and var() form the variance-covariance matrix. cor() forms the correlation matrix. cov2cor() scales a covariance matrix into a correlation matrix.

Usage

```
  ## S4 method for signature 'ddmatrix'
sd(x, na.rm = FALSE, reduce = FALSE, proc.dest="all")
  ## S4 method for signature 'ANY'
sd(x, na.rm = FALSE)
  ## S4 method for signature 'ddmatrix'
var(x, y = NULL, na.rm = FALSE, use)
  ## S4 method for signature 'ddmatrix'
cov(x, y = NULL, use = "everything", method =
  "pearson")
  ## S4 method for signature 'ddmatrix'
cov2cor(V)
```

Arguments

x, y, V	numeric distributed matrices.
na.rm	logical, determines whether or not NA's should be dealth with.
reduce	logical or string. See details
proc.dest	Destination process (or 'all') if a reduction occurs
use	character indicating how missing values should be treated. Acceptable values are the same as R's, namely "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".
method	character argument indicating which method should be used to calculate covariances. Currently only "spearman" is available for ddmatrix.

Details

sd() will compute the standard deviations of the columns, equivalent to calling apply(x, MARGIN=2, FUN=sd) (which will work for distributed matrices, by the way). However, this should be much faster and use less memory than apply(). If reduce=FALSE then the return is a distributed matrix consisting of one (global) row; otherwise, an R vector is returned, with ownership of this vector determined by proc.dest.

cov() forms the variance-covariance matrix. Only method="pearson" is implemented at this time.

var() is a shallow wrapper for cov() in the case of a distributed matrix.

cov2cor() scales a covariance matrix into a correlation matrix.

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

signature(V = "ddmatrix")

Author(s)

R Core Team, Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

See Also

prcomp

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
```

```
init.grid()

x <- ddmatrix("rnorm", nrow=3, ncol=3)

cv <- cov(x)
print(cv)

finalize()

## End(Not run)
```

PCA *Principal Components Analysis*

Description

Performs the principal components analysis.

Usage

```
  ## S4 method for signature 'ddmatrix'
prcomp(x, retx = TRUE, center = TRUE,
  scale. = FALSE, tol = NULL)
```

Arguments

`x`	numeric distributed matrix.
`center`	logical value, determines whether or not columns are zero centered
`scale.`	logical value, determines whether or not columns are rescaled to unit variance
`retx`	logical, indicates whether the rotated variables should be returned
`tol`	a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to `tol` times the standard deviation of the first component.) With the default null setting, no components are omitted. Other settings for tol could be `tol = 0` or `tol = sqrt(.Machine$double.eps)`, which would omit essentially constant components

Details

`prcomp()` performs the principal components analysis on the data matrix by taking the SVD. Sometimes core R and pbdDMAT will disagree slightly in what the rotated variables are because of how the SVD is caluclated. See the details section of `La.svd()` under `LinAlg` for details. more details.

Value

Returns a list.

Methods

signature(x = "ddmatrix")

Author(s)

R Core Team, Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)

x <- ddmatrix("rnorm", 10, 10)

y <- prcomp(x)
comm.print(y)

finalize()

## End(Not run)
```

`lm.fit` *Fitter for Linear Models*

Description

Fits a real linear model via QR with a "limited pivoting strategy", as in R's DQRDC2 (fortran).

Usage

```
  ## S4 method for signature 'ddmatrix,ddmatrix'
lm.fit(x, y, tol = 1e-07, singular.ok = TRUE)
```

Arguments

x, y	numeric distributed matrices
tol	tolerance for numerical rank estimation in QR decomposition.
singular.ok	logical. If FALSE then a singular model (rank-deficient x) produces an error.

Details

Solves the linear least squares problem, which is to find an x (possibly non-uniquely) such that || Ax - b ||^2 is minimized, where A is a given n-by-p model matrix, b is a "right hand side" n-by-1 vector (multiple right hand sides can be solved at once, but the solutions are independent, i.e. not simultaneous), and "||" is the l2 norm.

Uses level 3 PBLAS and ScaLAPACK routines (modified PDGELS) to get a linear least squares solution, using the 'limited pivoting strategy' from R's DQRDC2 (unsed in DQRLS) routine as a way of dealing with (possibly) rank deficient model matrices.

A model matrix with many dependent columns will likely experience poor performance, especially at scale, due to all the data swapping that must occur to handle rank deficiency.

Value

Returns a list of values similar to R's `lm.fit()`. Namely, the list contains: `coefficients` - distributed matrix, `residuals` - distributed matrix, `effects` - distributed matrix, `rank` - global numeric, `fitted.values` - distributed matrix, `assign` - `NULL` if `lm.fit()` is called directly, `qr` - list, same as return from `qr()`, `df.residual` - global numeric.

The return values are, respectively: (1) a solution `x` to the linear least squares problem, (2) the difference in the numerical fit `A %*% x` and the observed `b`, (3) `t(Q) %*% b`, where `Q` is the orthogonal matrix from a QR-decomposition of `A`, (4) the numerical column rank of `A`, (5) the numerical fit `A %*% x`, (6) `NULL` if `lm.fit()` is directly called, (7) a list containing the return of QR decomposition performed by a modified PDGEQPF, (8) degrees of freedom of residuals, i.e. `n` minus the column rank of `A`.

Methods

`signature(x = "ddmatrix", y = "ddmatrix")`

See Also

QR

Examples

```
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r

library(pbdDMAT, quiet = TRUE)
init.grid()

# don't do this in production code
x <- matrix(rnorm(9), 3)
y <- matrix(rnorm(3))

dx <- as.ddmatrix(x)
dy <- as.ddmatrix(y)

fit <- lm.fit(x=dx, y=dy)
```

```
print(fit)

finalize()

## End(Not run)
```

Index