Package 'pbdDMAT'

March 17, 2019

Title 'pbdR' Distributed Matrix Methods

Version 0.5-1

Description A set of classes for managing distributed matrices, and a collection of methods for computing linear algebra and statistics. Computation is handled mostly by routines from the 'pbdBASE' package, which itself relies on the 'ScaLAPACK' and 'PBLAS' numerical libraries for distributed computing.

License GPL (>= 2)

Depends R (>= 3.1.0), pbdMPI (>= 0.3-8), pbdBASE (>= 0.5-0), stats

Imports utils, methods

SystemRequirements OpenMPI (>= 1.5.4) on Solaris, Linux, Mac, and FreeBSD. MS-MPI (Microsoft HPC Pack 2012) or MPICH2 (>= 1.4.1p1) on Windows.

LazyLoad yes

LazyData yes

ByteCompile yes

URL https://pbdr.org/

BugReports http://group.pbdr.org/

MailingList Please send questions and comments regarding pbdR to RBigData@gmail.com

Maintainer Drew Schmidt <wrathematics@gmail.com>

RoxygenNote 6.1.1

NeedsCompilation yes

Author Drew Schmidt [aut, cre],
Wei-Chen Chen [aut],
Sebastien Lamy de la Chapelle [aut],
George Ostrouchov [aut],
Pragneshkumar Patel [aut],
ZhaoKang Wang [ctb],
Michael Lawrence [ctb],
R Core team [ctb] (some wrappers taken from the base and stats packages)

Repository CRAN

Date/Publication 2019-03-17 19:00:03 UTC

R topics documented:

pbdDMAT-package . . . 3
Accessors . . . 3
arithmetic . . . 5
as.ddmatrix . . . 7
as.matrix . . . 9
as.rowcyclic . . . 10
as.vector . . . 11
binds . . . 12
chol2inv . . . 13
companion . . . 13
Comparators . . . 14
condnums . . . 16
covariance . . . 17
ddmatrix-apply . . . 18
ddmatrix-chol . . . 19
ddmatrix-class . . . 20
ddmatrix-constructors . . . 20
ddmatrix-eigen . . . 22
ddmatrix-lu . . . 23
ddmatrix-norm . . . 23
ddmatrix-prcomp . . . 24
ddmatrix-print . . . 25
ddmatrix-scale . . . 26
ddmatrix-solve . . . 26
ddmatrix-summary . . . 27
ddmatrix-sumstats . . . 28
ddmatrix-svd . . . 29
diag-constructors . . . 30
eigen2 . . . 31
expm . . . 32
extract . . . 33
getLocal . . . 34
headsortails . . . 34
Hilbert . . . 35
insert . . . 36
isdot . . . 37
isSymmetric . . . 37
lm.fit . . . 38
math . . . 39
matmult . . . 41
na . . . 42
pbdDMAT Control . . . 42

qr 43
redistribute 44
reductions 45
rounding 47
sd 47
sparsity 48
sweep 49
transpose 50

Index **51**

`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 <wrathematics AT gmail.com>, 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: https://pbdr.org/

`Accessors` *Accessor Functions for Distributed Matrix Slots*

Description

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

Usage

```
nrow(x)

## S4 method for signature 'ddmatrix'
nrow(x)

NROW(x)

## S4 method for signature 'ddmatrix'
NROW(x)

ncol(x)

## S4 method for signature 'ddmatrix'
ncol(x)

NCOL(x)

## S4 method for signature 'ddmatrix'
NCOL(x)

submatrix(x, ...)

## S4 method for signature 'ddmatrix'
submatrix(x)

## S4 method for signature 'Linalg'
submatrix(x)

ldim(x, ...)

## S4 method for signature 'ddmatrix'
ldim(x)

bldim(x, ...)

## S4 method for signature 'ddmatrix'
bldim(x)

ICTXT(x, ...)

## S4 method for signature 'ddmatrix'
ICTXT(x)

## S4 method for signature 'ddmatrix'
dim(x)

## S4 method for signature 'ddmatrix'
```

```
length(x)
```

Arguments

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

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

list("signature(x = \"ddmatrix\")")

---

arithmetic *Arithmetic Operators*

Description

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

Usage

```
## S4 method for signature 'ddmatrix,numeric'
e1 + e2

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

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

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

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

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

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

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

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

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

## S4 method for signature 'ddmatrix,numeric'
e1 / e2

## S4 method for signature 'numeric,ddmatrix'
e1 / e2

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

## S4 method for signature 'ddmatrix,numeric'
e1 ^ e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 ^ e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 %% e2
```

```
## S4 method for signature 'ddmatrix,numeric'
e1 %% e2

## S4 method for signature 'numeric,ddmatrix'
e1 %% e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 %/% e2

## S4 method for signature 'numeric,ddmatrix'
e1 %/% e2

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

Arguments

e1, e2	numeric distributed matrices or numeric vectors

Details

If e1 and e2 are distributed matrices, then they must be conformable, on the same BLACS context, and have the same blocking dimension.

Value

Returns a distributed matrix.

---

as.ddmatrix *Non-Distributed object to Distributed Object Converters*

Description

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

Usage

```
as.ddmatrix(x, ...)

distribute(x, bldim = .pbd_env$BLDIM, xCTXT = 0,
  ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'matrix'
as.ddmatrix(x, bldim = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)

## S4 method for signature '`NULL`'
```

```
as.ddmatrix(x, bldim = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)

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

Arguments

x	a numeric matrix
...	Additional arguments.
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

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.

Examples

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

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

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

  ### Can also be common to all ranks
```

```
  y <- matrix(1:25, 5, bldim=2)
  dy <- as.ddmatrix(y)
  dy

  finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 2L)
```

`as.matrix` *Distributed object to Matrix Converters*

Description

Converts a distributed matrix into a non-distributed matrix.

Usage

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

Arguments

`x`	numeric distributed matrix
`proc.dest`	destination process for storing the matrix
`attributes`	logical, specifies whether or not the current attributes should be preserved.
`...`	Additional arguments.

Details

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.

Examples

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

  dx <- ddmatrix(1:16, ncol=4, bldim=2)
  y <- as.matrix(dx, proc.dest=0)
```

```
  comm.print(y)

  finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 2L)
```

---

as.rowcyclic *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.rowcyclic(dx, bldim = .pbd_env$BLDIM)

as.colcyclic(dx, bldim = .pbd_env$BLDIM)

as.blockcyclic(dx, bldim = .pbd_env$BLDIM)

as.block(dx, square.bldim = TRUE)

as.rowblock(dx)

as.colblock(dx)
```

Arguments

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

Details

These functions are simple wrappers of the very general `redistribute()` funciton. 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.

---

as.vector *Distributed object to Vector Converters*

Description

Converts a distributed matrix into a non-distributed vector.

Usage

```
as.vector(x, ...)

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

Arguments

x	numeric distributed matrix
...	Additional arguments.
mode	A character string giving an atomic mode or "list", or (except for 'vector') "any".
proc.dest	destination process for storing the matrix

Details

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 vector.

---

binds *Row and Column binds for Distributed Matrices*

Description

Row and column binds.

Usage

```
## S3 method for class 'ddmatrix'
rbind(..., ICTXT = .pbd_env$ICTXT,
  deparse.level = 1)

## S3 method for class 'ddmatrix'
cbind(..., ICTXT = .pbd_env$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

list("signature(... = \"ANY\")") an R object.

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

list("signature(x = \"ddmatrix\")")

list("signature(x = \"ANY\")")

companion *Generate Companion Matrices*

Description

Methods for constructing companion matrices of an n-degree polynomial.

Usage

```
companion(coef, type = "matrix", ..., bldim = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)
```

Arguments

coef	Vector of polynomial coefficients, listed in increasing order (by index; see details below).
type	"matrix" or "ddmatrix".
...	Additional arguments.
bldim	blocking dimension.
ICTXT	BLACS context number.

Details

For a degree n polynomial,

$x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$

its associated companion matrix is a matrix of the form

$$\begin{bmatrix} 0 & 0 & 0 & \ldots & 0 & -a_0 \\ 1 & 0 & 0 & \ldots & 0 & -a_1 \\ 0 & 1 & 0 & \ldots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & -a_{n-1} \end{bmatrix}$$

In the function call, we assume that the argument 'coef' is ordered from a_0 to a_{n-1}.

NOTE that we assume that the leading coefficient is 1.

Value

Returns a matrix or a distributed matrix.

Comparators *Logical Comparisons*

Description

Logical comparisons.

Usage

```
## S4 method for signature 'ddmatrix,ddmatrix'
e1 == e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 != e2

## S4 method for signature 'ddmatrix'
all(x, na.rm = FALSE)
```

```
## S4 method for signature 'ddmatrix'
any(x, na.rm = FALSE)

## S4 method for signature 'ddmatrix,ddmatrix'
e1 < e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 > e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 <= e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 >= e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 | e2

## S4 method for signature 'ddmatrix,ddmatrix'
e1 & e2

## S4 method for signature 'ddmatrix,numeric'
e1 < e2

## S4 method for signature 'numeric,ddmatrix'
e1 < e2

## S4 method for signature 'ddmatrix,numeric'
e1 > e2

## S4 method for signature 'numeric,ddmatrix'
e1 > e2

## S4 method for signature 'ddmatrix,numeric'
e1 <= e2

## S4 method for signature 'numeric,ddmatrix'
e1 <= e2

## S4 method for signature 'ddmatrix,numeric'
e1 >= e2

## S4 method for signature 'numeric,ddmatrix'
e1 >= e2

## S4 method for signature 'ddmatrix,numeric'
e1 == e2
```

```
## S4 method for signature 'numeric,ddmatrix'
e1 == e2

## S4 method for signature 'ddmatrix,numeric'
e1 != e2

## S4 method for signature 'numeric,ddmatrix'
e1 != e2

## S4 method for signature 'ddmatrix,numeric'
e1 | e2

## S4 method for signature 'numeric,ddmatrix'
e1 | e2

## S4 method for signature 'ddmatrix,numeric'
e1 & e2

## S4 method for signature 'numeric,ddmatrix'
e1 & e2
```

Arguments

e1, e2, x	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.

---

condnums *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

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.
...	Extra arguments.
x, z	numeric distributed matrices.
triangular	logical. If true, only the lower triangle is used.

Value

Returns a number.

---

covariance *Covariance and Correlation*

Description

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'
cov(x, y = NULL, use = "everything",
  method = "pearson")

## S4 method for signature 'ddmatrix'
var(x, y = NULL, na.rm = FALSE, use)

## S4 method for signature 'ddmatrix'
cor(x, y = NULL, use = "everything",
  method = "pearson")

## S4 method for signature 'ddmatrix'
cov2cor(V)
```

Arguments

x, y, V	numeric distributed matrices.
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.
na.rm	logical, determines whether or not NA's should be dealth with.

Details

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.

Examples

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

x <- ddmatrix('rnorm', nrow=3, ncol=3), bldim=2

cv <- cov(x)
cv

finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 2L)
```

---

ddmatrix-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.

---

ddmatrix-chol *Cholesky Factorization*

Description

Cholesky factorization for distributed matrices with R-like syntax, with calculations performed by the PBLAS and ScaLAPACK libraries.

Usage

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

Arguments

x	numeric distributed matrices.
...	Ignored.

Details

Extensions of R linear algebra functions.

Value

chol() performs Cholesky factorization.

ddmatrix-class *Class ddmatrix*

Description

Distributed matrix class.

Slots

`Data` The local submatrix.

`bldim` Blocking factor.

`ICTXT` BLACS ICTXT value. Should be one of 0, 1, or 2 (initialized from `pbdBASE::init.grid()`) or a custom value greater than 2 (created from `pbdBASE::blacs_gridinit()`).

ddmatrix-constructors *Distributed Matrix Creation*

Description

Methods for simple construction of distributed matrices.

Usage

```
ddmatrix(data, ...)

## S4 method for signature 'ddmatrix'
ddmatrix(data, nrow = 1, ncol = 1,
  byrow = FALSE, ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'missing'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
  ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'vector'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
  ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'matrix'
ddmatrix(data, nrow = 1, ncol = 1, byrow = FALSE,
  ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## 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 = .pbd_env$BLDIM,
```

```
  ICTXT = .pbd_env$ICTXT)

ddmatrix.local(data, ...)

## S4 method for signature 'missing'
ddmatrix.local(data, nrow = 1, ncol = 1,
  byrow = FALSE, ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$BLDIM)

## S4 method for signature 'vector'
ddmatrix.local(data, nrow = 1, ncol = 1,
  byrow = FALSE, ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'matrix'
ddmatrix.local(data, nrow = 1, ncol = 1,
  byrow = FALSE, ..., bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$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 = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)
```

Arguments

data	A global value: a string (for random generation) or an optional data vector. In the data vector case, the data should be the same across all processes.
...	Extra arguments
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.
bldim	blocking dimension.
ICTXT	BLACS context number.
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.

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.

`ddmatrix-eigen` *eigen*

Description

Eigenvalue decomposition for distributed matrices with R-like syntax, with calculations performed by the PBLAS and ScaLAPACK libraries.

Usage

```
## S4 method for signature 'ddmatrix'
eigen(x, symmetric, only.values = FALSE,
  EISPACK = FALSE)
```

Arguments

`x`	numeric distributed matrices.
`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.
`EISPACK`	Ignored.

Details

Extensions of R linear algebra functions.

Value

`eigen()` computes the eigenvalues, and eigenvectors if requested. As

ddmatrix-lu *LU Factorization*

Description

LU factorization for distributed matrices with R-like syntax, with calculations performed by the PBLAS and ScaLAPACK libraries.

Usage

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

Arguments

x	numeric distributed matrices.

Details

Extensions of R linear algebra functions.

Value

`lu()` performs LU factorization.

ddmatrix-norm *Norm*

Description

Computes the norm of a distributed matrix.

Usage

```
## S4 method for signature 'ddmatrix,ANY'
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

list("signature(x = \"ddmatrix\")")

`ddmatrix-prcomp` *Principal Components Analysis*

Description

Performs the principal components analysis.

Usage

```
prcomp(x, ...)

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

Arguments

`x`	numeric distributed matrix.
`...`	Ignored.
`retx`	logical, indicates whether the rotated variables should be returned
`center`	logical value, determines whether or not columns are zero centered
`scale.`	logical value, determines whether or not columns are rescaled to unit variance
`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.

Value

Returns a list.

ddmatrix-print *Printing a Distributed Matrix*

Description

Print method for a distributed matrices.

Usage

```
print(x, ...)

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

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

Arguments

x, object	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.

ddmatrix-scale *Scale*

Description

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

Usage

```
scale(x, center = TRUE, scale = TRUE)

## S4 method for signature 'ddmatrix'
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

Value

Returns a distributed matrix.

ddmatrix-solve *Solve*

Description

Solving linear systems and matrix inversion for distributed matrices with R-like syntax, with calculations performed by the PBLAS and ScaLAPACK libraries.

Usage

```
## S4 method for signature 'ddmatrix,ANY'
solve(a)

## S4 method for signature 'ddmatrix,ddmatrix'
solve(a, b)
```

Arguments

a, b	numeric distributed matrices. Here, a and b must be on the same BLACS context and have the same blocking dimension.

Details

Extensions of R linear algebra functions.

Value

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

`ddmatrix-summary` *Distributed Matrix Summary*

Description

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

Usage

```
summary(object, ...)

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

Arguments

`object`	numeric distributed matrix
`...`	Additional arguments.

Details

The return is on process 0 only.

Value

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

ddmatrix-sumstats *Basic Summary Statistics*

Description

Get basic summary statistics.

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'
prod(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'
median(x, na.rm = FALSE)
```

Arguments

x	numeric distributed matrix
...	Additional arguments.
na.rm	Handling of NA's.

Details

The return is on process 0 only.

Value

A single value, owned by all ranks in the MPI communicator.

ddmatrix-svd *Singular Value Decomposition*

Description

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

Usage

```
## S4 method for signature 'ANY'
La.svd(x, nu = min(n, p), nv = min(n, p))

## S4 method for signature 'ddmatrix'
La.svd(x, nu = min(n, p), nv = min(n, p))

## S4 method for signature 'ANY'
svd(x, nu = min(n, p), nv = min(n, p),
  LINPACK = FALSE)

## S4 method for signature 'ddmatrix'
svd(x, nu = min(n, p), nv = min(n, p))
```

Arguments

x	numeric distributed matrices.
nu	number of left singular vectors to return when calculating singular values.
nv	number of right singular vectors to return when calculating singular values.
LINPACK	Ignored.

Details

Extensions of R linear algebra functions.

Value

`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.

Examples

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

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

  y <- svd(A)
  y

  finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 2L)
```

---

diag-constructors *Distributed Matrix Diagonals*

Description

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

Usage

```
## S4 method for signature 'vector'
diag(x, nrow, ncol, type = "matrix", ...,
  bldim = .pbd_env$BLDIM, ICTXT = .pbd_env$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 = .pbd_env$BLDIM, ICTXT = .pbd_env$ICTXT)

## S4 method for signature 'ddmatrix'
diag(x)

## S4 method for signature 'matrix'
diag(x, nrow, ncol)
```

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
bldim	blocking dimension.
ICTXT	BLACS context number.
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.

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.

`eigen2` *eigen2*

Description

Compute eigenvalues and, optionally, eigenvectors of a real symmetric matrix by seraching over ranges of values or ranges of indices.

Usage

```
eigen2(x, range = c(-Inf, Inf), range.type = "interval",
  only.values = FALSE, abstol = 1e-08, orfac = 0.001)
```

Arguments

x	symmetric, numeric ddmatrix.
range	A set of interval endpoints, i.e. a numeric pair. Controls the set of values over which the eigenvalue search occurs.
range.type	Controls whether interval range refers to a set of possible values for the eigenvalues, or a set of indices for the eigenvalues. Options are "interval" and "index".
only.values	logical. Determines whether only the eigenvalues should be computed, or if the eigenvectors should as well.
abstol	The absolute error tolerance for the eigenvalues.
orfac	Specifies which eigenvectors should be reorthogonalized. Eigenvectors that correspond to eigenvalues which are within tol=orfac*norm(A)of each other are to be reorthogonalized.

Details

This new method computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Value

Returns a distributed matrix.

---

expm *Matrix Exponentiation*

Description

Routines for matrix exponentiation.

Usage

```
expm(x, t = 1, p = 6)

## S4 method for signature 'matrix'
expm(x, t = 1, p = 6)

## S4 method for signature 'ddmatrix'
expm(x, t = 1, p = 6)
```

Arguments

x	A numeric matrix or a numeric distributed matrix.
t	Scaling parameter for x.
p	Order of the Pade' approximation.

Details

Formally, the exponential of a square matrix X is a power series:

$expm(X) = id + X/1! + X^2/2! + X^3/3! + \ldots$

where the powers on the matrix correspond to matrix-matrix multiplications.

expm() directly computes the matrix exponential of a distributed, dense matrix. The implementation uses Pade' approximations and a scaling-and-squaring technique (see references).

Value

Returns a distributed matrix.

References

"New Scaling and Squaring Algorithm for the Matrix Exponential" Awad H. Al-Mohy and Nicholas J. Higham, August 2009

`extract` *Extract or Replace Parts of a Distributed Matrix*

Description

Operators to extract or replace parts of a distributed matrix.

Usage

```
## S4 method for signature 'ddmatrix'
x[i, j, ICTXT]
```

Arguments

`x`	numeric distributed matrix.
`i, j`	indices specifying elements to extract or replace. Indices can be `numeric`, `character`, empty, or `NULL`. 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`.
`ICTXT`	optional BLACS context number for output
`...`	additional arguments.

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.

getLocal *getLocal*

Description

Get the value of the distributed matrix at global indices gi x gj.

Usage

```
getLocal(x, gi, gj, all.rank = TRUE, gridinfo)
```

Arguments

x	A distributed matrix.
gi, gj	Global row and column indices, respectively.
all.rank	Logical; if TRUE, then all processes will hold the desired value on exit. Otherwise, only the process who owns the local value returns this value, while every other process returns NULL.
gridinfo	An optional parameter; each local data lookup requires the data contained in gridinfo(ICTXT(x)). So you may specify it yourself (and if you are making many function calls, this is preferable performance-wise), or the lookup will be performed for you.

Value

The value at global index gi x gj.

headsortails *Head and Tail of a Distributed Matrix*

Description

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.

Usage

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

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

Arguments

`x`	numeric distributed matrix.
`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.

Value

Returns a distributed matrix.

`Hilbert` *Generate Hilbert Matrices*

Description

Methods for constructing Hilbert matrices: H[i,j] = 1/(i+j-1)

Usage

```
Hilbert(n, type = "matrix", ..., bldim = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)
```

Arguments

`n`	number of rows and columns.
`type`	"matrix" or "ddmatrix".
`...`	Additional arguments.
`bldim`	blocking dimension.
`ICTXT`	BLACS context number.

Details

This constructs the square Hilbert matrix of order `n`. The return is either a matrix or a distributed matrix depending on the argument `type=`.

Value

Returns a matrix or a distributed matrix.

insert *Directly Insert Into Distributed Matrix Submatrix Slot*

Description

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

Usage

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

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

Arguments

x	numeric distributed matrix.
i, j	global integer indices.
...	Additional arguments.
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.

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

Description

Functions to check for various types.

Usage

```
is.ddmatrix(x)

## S4 method for signature 'ddmatrix'
is.na(x)

## S4 method for signature 'ddmatrix'
is.nan(x)

## S4 method for signature 'ddmatrix'
is.numeric(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.

isSymmetric *isSymmetric*

Description

Tests if a distributed matrix is symmetric.

Usage

```
## S4 method for signature 'ddmatrix'
isSymmetric(object, tol = 100 * .Machine$double.eps,
  ...)
```

Arguments

object	Distributed matrix
tol	Numerical tolerance for the comparison.
...	Additional arguments passed to all.equal().

Details

The test is performed by comparing the object against its transpose.

---

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) solution to the linear least squares problem
`residuals`	(distributed matrix) difference in the numerical fit and the observed
`effects`	(distributed matrix) `t(Q) %*% y`
`rank`	(global numeric) numerical column rank
`fitted.values`	(distributed matrix) Numerical fit `A %*% x`
`assign`	`NULL` if `lm.fit()` is called directly
`qr`	list, same as return from `qr()`
`df.residual`	(global numeric) degrees of freedom of residuals

Examples

```
spmd.code = "
  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)
  fit

  finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks=2L)
```

`math` *Miscellaneous Mathematical Functions*

Description

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

Usage

```
## S4 method for signature 'ddmatrix'
sqrt(x)

## S4 method for signature 'ddmatrix'
```

```
abs(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.

`matmult` *Matrix Multiplication*

Description

Multiplies two distributed matrices, if they are conformable.

Usage

```
## S4 method for signature 'ddmatrix,ddmatrix'
x %*% y

## S4 method for signature 'ddmatrix'
crossprod(x, y = NULL)

## S4 method for signature 'ddmatrix'
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.

na *Handle Missing Values in Distributed Matrices*

Description

Dealing with NA's and NaN's.

Usage

```
na.exclude(object, ...)

## 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@ICTXT. Otherwise, the return will be redistributed across the supplied ICTXT.

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 `dmat_opts$BLDIM` is `c(16,16)`, which results in a 16 by 16 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.

---

qr *QR Decomposition Methods*

Description

`qr()` takes the QR decomposition.

Usage

```
## S4 method for signature 'ddmatrix'
qr(x, tol = 1e-07)

## S4 method for signature 'ANY'
qr.Q(x, complete = FALSE, Dvec)

## 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)

lq(x)

lq.Q(x, complete = FALSE)
```

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

`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.

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".

`redistribute` *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

```
reblock(dx, bldim = dx@bldim, ICTXT = .pbd_env$ICTXT)

redistribute(dx, bldim = dx@bldim, ICTXT = .pbd_env$ICTXT)
```

Arguments

`dx`	numeric distributed matrix
`bldim`	the blocking dimension for block-cyclically distributing the matrix across the process grid.
`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@ICTXT 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@ICTXT 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.

Value

Returns a distributed matrix.

---

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

Description

Arithmetic reductions for distributed matrices.

Usage

```
rowMin(x, ...)

rowMax(x, ...)

colMin(x, ...)

colMax(x, ...)

## 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'
```

```
rowMin(x, na.rm = FALSE)

## S4 method for signature 'matrix'
rowMin(x, na.rm = FALSE)

## S4 method for signature 'ddmatrix'
colMin(x, na.rm = FALSE)

## S4 method for signature 'matrix'
colMin(x, na.rm = FALSE)

## S4 method for signature 'ddmatrix'
rowMax(x, na.rm = FALSE)

## S4 method for signature 'matrix'
rowMax(x, na.rm = FALSE)

## S4 method for signature 'ddmatrix'
colMax(x, na.rm = FALSE)

## S4 method for signature 'matrix'
colMin(x, na.rm = FALSE)
```

Arguments

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

Details

Performs the reduction operation on a distributed matrix.

There are several legitimately new operations, including `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.

`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`.

Value

Returns a global numeric vector.

rounding *Rounding of Numbers*

Description

Extensions of R rounding functions for distributed matrices.

Usage

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

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

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

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.

sd *Covariance and Correlation*

Description

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.

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)
```

Arguments

x	numeric distributed matrices.
na.rm	Logical; if TRUE, then na.exclude() is called first.
reduce	logical or string. See details
proc.dest	Destination process (or 'all') if a reduction occurs

Value

Returns a distributed matrix.

---

sparsity *Sparsity of Matrix Objects*

Description

Determine the sparsity of a matrix, distributed, dense, or otherwise.

Usage

```
## S4 method for signature 'matrix'
sparsity(x, count = "zero", out = "count",
  tol = .Machine$double.eps)

## S4 method for signature 'vector'
sparsity(x, count = "zero", out = "count",
  tol = .Machine$double.eps)

## S4 method for signature 'dmat'
sparsity(x, count = "zero", out = "count",
  tol = .Machine$double.eps)
```

Arguments

x	numeric matrix
count	character; options are "zero" and "other". The former counts the number of zeros, while the latter counts the number of non-zeros ('other' elements).

out	character; options are "count", "proportion", and "percent". This determines whether a pure count, proportion of count elements in the matrix, or percentage of count elements in the matrix.
tol	numeric; the tolerance for numerical zero. This is ignored if the input data is integer/logical.

Details

The sparsity count of a matrix is returned.

---

sweep *Sweep*

Description

Sweep vector or ddmatrix from a distributed matrix.

Usage

```
sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)

## 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().
check.margin, ...	Ignored.

Value

Returns a distributed matrix.

transpose *Distributed Matrix Transpose*

Description

Transposes a distributed dense matrix.

Usage

```
t(x)

## S4 method for signature 'ddmatrix'
t(x)
```

Arguments

x	numeric distributed matrix.

Value

The transposed matrix.

Index

!=,ddmatrix,ddmatrix-method (Comparators), 14
!=,ddmatrix,numeric-method (Comparators), 14
!=,numeric,ddmatrix-method (Comparators), 14
∗Topic **Algebra**
chol2inv, 13
condnums, 16
ddmatrix-chol, 19
ddmatrix-eigen, 22
ddmatrix-lu, 23
ddmatrix-norm, 23
ddmatrix-solve, 26
eigen2, 31
expm, 32
matmult, 41
qr, 43
transpose, 50
∗Topic **BLACS**
as.rowcyclic, 10
redistribute, 44
∗Topic **Classes**
ddmatrix-class, 20
∗Topic **ConditionNumbers**
condnums, 16
∗Topic **Data**
as.ddmatrix, 7
as.rowcyclic, 10
companion, 13
ddmatrix-constructors, 20
Hilbert, 35
redistribute, 44
∗Topic **Distributing**
as.ddmatrix, 7
as.rowcyclic, 10
redistribute, 44
∗Topic **Extraction**
Comparators, 14
ddmatrix-apply, 18
diag-constructors, 30
extract, 33
headsortails, 34
insert, 36
lm.fit, 38
na, 42
∗Topic **Generation**
companion, 13
ddmatrix-constructors, 20
Hilbert, 35
∗Topic **Linear**
chol2inv, 13
condnums, 16
ddmatrix-chol, 19
ddmatrix-eigen, 22
ddmatrix-lu, 23
ddmatrix-norm, 23
ddmatrix-solve, 26
eigen2, 31
expm, 32
matmult, 41
qr, 43
transpose, 50
∗Topic **Methods,Sparse**
sparsity, 48
∗Topic **Methods**
Accessors, 3
arithmetic, 5
as.matrix, 9
as.vector, 11
binds, 12
chol2inv, 13
Comparators, 14
condnums, 16
covariance, 17
ddmatrix-apply, 18
ddmatrix-chol, 19
ddmatrix-eigen, 22

ddmatrix-lu, 23
ddmatrix-norm, 23
ddmatrix-prcomp, 24
ddmatrix-print, 25
ddmatrix-scale, 26
ddmatrix-solve, 26
ddmatrix-summary, 27
ddmatrix-sumstats, 28
diag-constructors, 30
eigen2, 31
expm, 32
extract, 33
headsortails, 34
insert, 36
isdot, 37
lm.fit, 38
math, 39
matmult, 41
na, 42
qr, 43
reductions, 45
rounding, 47
sd, 47
sweep, 49
transpose, 50
∗Topic **Norm**
ddmatrix-norm, 23
∗Topic **Package**
pbdDMAT-package, 3
∗Topic **Type**
Comparators, 14
isdot, 37
na, 42
*,ddmatrix,ddmatrix-method (arithmetic), 5
*,ddmatrix,numeric-method (arithmetic), 5
*,numeric,ddmatrix-method (arithmetic), 5
+,ddmatrix,ddmatrix-method (arithmetic), 5
+,ddmatrix,numeric-method (arithmetic), 5
+,numeric,ddmatrix-method (arithmetic), 5
-,ddmatrix,ddmatrix-method (arithmetic), 5
-,ddmatrix,missing-method (arithmetic), 5
-,ddmatrix,numeric-method (arithmetic), 5
-,numeric,ddmatrix-method (arithmetic), 5
/,ddmatrix,ddmatrix-method (arithmetic), 5
/,ddmatrix,numeric-method (arithmetic), 5
/,numeric,ddmatrix-method (arithmetic), 5
<,ddmatrix,ddmatrix-method (Comparators), 14
<,ddmatrix,numeric-method (Comparators), 14
<,numeric,ddmatrix-method (Comparators), 14
<=,ddmatrix,ddmatrix-method (Comparators), 14
<=,ddmatrix,numeric-method (Comparators), 14
<=,numeric,ddmatrix-method (Comparators), 14
==,ddmatrix,ddmatrix-method (Comparators), 14
==,ddmatrix,numeric-method (Comparators), 14
==,numeric,ddmatrix-method (Comparators), 14
>,ddmatrix,ddmatrix-method (Comparators), 14
>,ddmatrix,numeric-method (Comparators), 14
>,numeric,ddmatrix-method (Comparators), 14
>=,ddmatrix,ddmatrix-method (Comparators), 14
>=,ddmatrix,numeric-method (Comparators), 14
>=,numeric,ddmatrix-method (Comparators), 14
[,ddmatrix-method (extract), 33
[<-,ddmatrix,ANY,ANY,ANY-method (insert), 36
[<-,ddmatrix,ANY,ANY,ddmatrix-method (insert), 36
%*%,ddmatrix,ddmatrix-method (matmult), 41

%/%,ddmatrix,ddmatrix-method (arithmetic), 5
%/%,ddmatrix,numeric-method (arithmetic), 5
%/%,numeric,ddmatrix-method (arithmetic), 5
%%,ddmatrix,ddmatrix-method (arithmetic), 5
%%,ddmatrix,numeric-method (arithmetic), 5
%%,numeric,ddmatrix-method (arithmetic), 5
&,ddmatrix,ddmatrix-method (Comparators), 14
&,ddmatrix,numeric-method (Comparators), 14
&,numeric,ddmatrix-method (Comparators), 14
^,ddmatrix,ddmatrix-method (arithmetic), 5
^,ddmatrix,numeric-method (arithmetic), 5

abs,ddmatrix-method (math), 39
Accessors, 3
acos,ddmatrix-method (math), 39
all,ddmatrix-method (Comparators), 14
any,ddmatrix-method (Comparators), 14
apply (ddmatrix-apply), 18
apply,ddmatrix-method (ddmatrix-apply), 18
arithmetic, 5
as.block (as.rowcyclic), 10
as.blockcyclic (as.rowcyclic), 10
as.colblock (as.rowcyclic), 10
as.colcyclic (as.rowcyclic), 10
as.ddmatrix, 7
as.ddmatrix,matrix-method (as.ddmatrix), 7
as.ddmatrix,NULL-method (as.ddmatrix), 7
as.ddmatrix,vector-method (as.ddmatrix), 7
as.matrix, 9
as.matrix,ddmatrix-method (as.matrix), 9
as.rowblock (as.rowcyclic), 10
as.rowcyclic, 10
as.vector, 11
as.vector,ddmatrix-method (as.vector), 11
asin,ddmatrix-method (math), 39
atan,ddmatrix-method (math), 39

binds, 12
bldim (Accessors), 3
bldim,ddmatrix-method (Accessors), 3

cbind.ddmatrix (binds), 12
ceiling,ddmatrix-method (rounding), 47
chol (ddmatrix-chol), 19
chol,ddmatrix-method (ddmatrix-chol), 19
chol2inv, 13
chol2inv,ddmatrix-method (chol2inv), 13
chol2inv-method (chol2inv), 13
colMax (reductions), 45
colMax,ddmatrix-method (reductions), 45
colMeans,ddmatrix-method (reductions), 45
colMin (reductions), 45
colMin,ddmatrix-method (reductions), 45
colMin,matrix-method (reductions), 45
colSums,ddmatrix-method (reductions), 45
companion, 13
Comparators, 14
condnums, 16
cor,ddmatrix-method (covariance), 17
cos,ddmatrix-method (math), 39
cosh,ddmatrix-method (math), 39
cov,ddmatrix-method (covariance), 17
cov2cor,ddmatrix-method (covariance), 17
covariance, 17
crossprod,ddmatrix-method (matmult), 41

ddmatrix (ddmatrix-constructors), 20
ddmatrix,character-method (ddmatrix-constructors), 20
ddmatrix,ddmatrix-method (ddmatrix-constructors), 20
ddmatrix,matrix-method (ddmatrix-constructors), 20
ddmatrix,missing-method (ddmatrix-constructors), 20
ddmatrix,vector-method (ddmatrix-constructors), 20
ddmatrix-apply, 18
ddmatrix-chol, 19
ddmatrix-class, 20
ddmatrix-constructors, 20
ddmatrix-eigen, 22

ddmatrix-lu, 23
ddmatrix-norm, 23
ddmatrix-prcomp, 24
ddmatrix-print, 25
ddmatrix-scale, 26
ddmatrix-solve, 26
ddmatrix-summary, 27
ddmatrix-sumstats, 28
ddmatrix-svd, 29
ddmatrix.local *(ddmatrix-constructors)*, 20
ddmatrix.local,character-method *(ddmatrix-constructors)*, 20
ddmatrix.local,matrix-method *(ddmatrix-constructors)*, 20
ddmatrix.local,missing-method *(ddmatrix-constructors)*, 20
ddmatrix.local,vector-method *(ddmatrix-constructors)*, 20
diag *(diag-constructors)*, 30
diag,character-method *(diag-constructors)*, 30
diag,ddmatrix-method *(diag-constructors)*, 30
diag,matrix-method *(diag-constructors)*, 30
diag,vector-method *(diag-constructors)*, 30
diag-constructors, 30
dim,ddmatrix-method *(Accessors)*, 3
distribute *(as.ddmatrix)*, 7

eigen *(ddmatrix-eigen)*, 22
eigen,ddmatrix-method *(ddmatrix-eigen)*, 22
eigen2, 31
exp,ddmatrix-method *(math)*, 39
expm, 32
expm,ddmatrix-method *(expm)*, 32
expm,matrix-method *(expm)*, 32
extract, 33

floor,ddmatrix-method *(rounding)*, 47

getLocal, 34

head.ddmatrix *(headsortails)*, 34
headsortails, 34
Hilbert, 35

ICTXT *(Accessors)*, 3
ICTXT,ddmatrix-method *(Accessors)*, 3
insert, 36
is.ddmatrix *(isdot)*, 37
is.infinite,ddmatrix-method *(isdot)*, 37
is.na,ddmatrix-method *(isdot)*, 37
is.nan,ddmatrix-method *(isdot)*, 37
is.numeric,ddmatrix-method *(isdot)*, 37
isdot, 37
isSymmetric, 37
isSymmetric,ddmatrix-method *(isSymmetric)*, 37

kappa.ddmatrix *(condnums)*, 16

La.svd *(ddmatrix-svd)*, 29
La.svd,ANY-method *(ddmatrix-svd)*, 29
La.svd,ddmatrix-method *(ddmatrix-svd)*, 29
ldim *(Accessors)*, 3
ldim,ddmatrix-method *(Accessors)*, 3
length,ddmatrix-method *(Accessors)*, 3
lm.fit, 38
lm.fit,ddmatrix,ddmatrix-method *(lm.fit)*, 38
log,ddmatrix-method *(math)*, 39
log10,ddmatrix-method *(math)*, 39
log1p,ddmatrix-method *(math)*, 39
log2,ddmatrix-method *(math)*, 39
lq *(qr)*, 43
lu *(ddmatrix-lu)*, 23
lu,ddmatrix-method *(ddmatrix-lu)*, 23

math, 39
matmult, 41
max,ddmatrix-method *(ddmatrix-sumstats)*, 28
mean,ddmatrix-method *(ddmatrix-sumstats)*, 28
median,ddmatrix-method *(ddmatrix-sumstats)*, 28
min,ddmatrix-method *(ddmatrix-sumstats)*, 28

na, 42
NCOL *(Accessors)*, 3
ncol *(Accessors)*, 3
NCOL,ddmatrix-method *(Accessors)*, 3
ncol,ddmatrix-method *(Accessors)*, 3

norm (ddmatrix-norm), 23
norm,ddmatrix,ANY-method (ddmatrix-norm), 23
NROW (Accessors), 3
nrow (Accessors), 3
NROW,ddmatrix-method (Accessors), 3
nrow,ddmatrix-method (Accessors), 3

pbdDMAT Control, 42
pbdDMAT-package, 3
prcomp (ddmatrix-prcomp), 24
prcomp,ddmatrix-method (ddmatrix-prcomp), 24
print (ddmatrix-print), 25
print,ddmatrix-method (ddmatrix-print), 25
prod,ddmatrix-method (ddmatrix-sumstats), 28

qr, 43
qr,ddmatrix-method (qr), 43
qr.Q (qr), 43
qr.Q,ANY-method (qr), 43
qr.qty (qr), 43
qr.qty,ANY-method (qr), 43
qr.qy (qr), 43
qr.qy,ANY-method (qr), 43
qr.R (qr), 43
qr.R,ANY-method (qr), 43

rbind.ddmatrix (binds), 12
rcond,ddmatrix-method (condnums), 16
reblock (redistribute), 44
redistribute, 44
reductions, 45
round,ddmatrix-method (rounding), 47
rounding, 47
rowMax (reductions), 45
rowMax,ddmatrix-method (reductions), 45
rowMax,matrix-method (reductions), 45
rowMeans,ddmatrix-method (reductions), 45
rowMin (reductions), 45
rowMin,ddmatrix-method (reductions), 45
rowMin,matrix-method (reductions), 45
rowSums,ddmatrix-method (reductions), 45

scale (ddmatrix-scale), 26
scale,ddmatrix-method (ddmatrix-scale), 26
sd, 47
sd,ANY-method (sd), 47
sd,ddmatrix-method (sd), 47
show,ddmatrix-method (ddmatrix-print), 25
sin,ddmatrix-method (math), 39
sinh,ddmatrix-method (math), 39
solve,ddmatrix,ANY-method (ddmatrix-solve), 26
solve,ddmatrix,ddmatrix-method (ddmatrix-solve), 26
sparsity, 48
sparsity,dmat-method (sparsity), 48
sparsity,matrix-method (sparsity), 48
sparsity,vector-method (sparsity), 48
sparsity-method (sparsity), 48
sqrt,ddmatrix-method (math), 39
submatrix (Accessors), 3
submatrix,ddmatrix-method (Accessors), 3
submatrix,Linalg-method (Accessors), 3
sum,ddmatrix-method (ddmatrix-sumstats), 28
summary (ddmatrix-summary), 27
summary,ddmatrix-method (ddmatrix-summary), 27
svd (ddmatrix-svd), 29
svd,ANY-method (ddmatrix-svd), 29
svd,ddmatrix-method (ddmatrix-svd), 29
sweep, 49
sweep,ddmatrix,ANY,ddmatrix-method (sweep), 49
sweep,ddmatrix,ANY,vector-method (sweep), 49

t (transpose), 50
t,ddmatrix-method (transpose), 50
tail.ddmatrix (headsortails), 34
tan,ddmatrix-method (math), 39
tanh,ddmatrix-method (math), 39
tcrossprod,ddmatrix-method (matmult), 41
transpose, 50

var,ddmatrix-method (covariance), 17