Package ‘pbdDMAT’

October 21, 2016

Title  ‘pbdR’ Distributed Matrix Methods
Version  0.4-2
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 (>= 3.0.0), pbdMPI (>= 0.3-1), pbdBASE (>= 0.4-5), 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
NeedsCompilation  yes
URL  http://r-pbd.org/
BugReports  http://group.r-pbd.org/
MailingList  Please send questions and comments regarding pbdR to RBigData@gmail.com
Maintainer  Drew Schmidt <schmidt@math.utk.edu>
RoxygenNote  5.0.1
Author  Drew Schmidt [aut, cre], Wei-Chen Chen [aut], George Ostrouchov [aut], Pragneshkumar Patel [aut], Zhao Kang Wang [ctb], Michael Lawrence [ctb], R Core team [ctb] (some wrappers taken from the base and stats packages)
R topics documented:

- pbdDMAT-package .................................................. 3
- Accessors .......................................................... 3
- arithmetic ........................................................ 6
- as.ddmatrix ......................................................... 8
- as.matrix .......................................................... 9
- as.rowcyclic ....................................................... 11
- as.vector .......................................................... 13
- binds .............................................................. 14
- chol2inv ........................................................... 15
- companion .......................................................... 16
- Comparators ....................................................... 17
- condns .............................................................. 20
- covariance .......................................................... 21
- ddmatrix-apply ..................................................... 22
- ddmatrix-chol ...................................................... 23
- ddmatrix-class ..................................................... 24
- ddmatrix-constructors ........................................... 25
- ddmatrix-eigen .................................................... 27
- ddmatrix-lu ........................................................ 28
- ddmatrix-norm ..................................................... 29
- ddmatrix-prcomp .................................................. 30
- ddmatrix-print .................................................... 31
- ddmatrix-scale ..................................................... 33
- ddmatrix-solve ..................................................... 34
- ddmatrix-summary ................................................ 35
- ddmatrix-sumstats ................................................ 36
- ddmatrix-svd ....................................................... 37
- diag-constructors ................................................ 38
- eigen2 .............................................................. 40
- expm ............................................................... 41
- extract ............................................................. 42
- getLocal ........................................................... 44
- headsortails ....................................................... 45
- Hilbert ............................................................. 46
- insert .............................................................. 47
- isdot ............................................................... 48
- isSymmetric ....................................................... 49
- lm.fit .............................................................. 50
- math ............................................................... 51
- matmult ........................................................... 53
- na ................................................................. 54
- pbdDMAT Control .................................................. 55
Accessors

qr ................................................................. 56
redistribute .................................................. 57
reductions ...................................................... 59
rounding ........................................................ 61
sd ................................................................. 62
sparsity .......................................................... 63
sweep .............................................................. 64
transpose ......................................................... 65

Index 66

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/

Accessors  Accessor Functions for Distributed Matrix Slots

Description

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

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'
Accessors

\texttt{length(x)}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} \text{numeric distributed matrix}
\item \texttt{...} \hspace{1cm} \text{Extra arguments.}
\item \texttt{dim} \hspace{1cm} \text{global dimension.}
\item \texttt{bldim} \hspace{1cm} \text{blocking dimension.}
\item \texttt{ICTXT} \hspace{1cm} \text{BLACS context.}
\end{itemize}

Details

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

\texttt{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 \texttt{x} is distributed as the \texttt{ddmatrix dx} across two processors with process 0 owning the first two rows and process 1 owning the third, then \texttt{ldim(dx)} will return 2 \hspace{1cm} 3 on process 0 and 1 \hspace{1cm} 3 on process 1.

\texttt{bldim()} will give the blocking dimension that was used to block-cyclically distribute the distributed matrix.

\texttt{submatrix()} will give the local storage for the requested object.

\texttt{ICTXT()} will give the current BLACS context (slot ICTXT) for the requested object.

\texttt{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 \texttt{dim()}, \texttt{ldim()}, \texttt{bldim()} return a length 2 vector.

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

\texttt{submatrix()} returns a matrix; namely, \texttt{submatrix(x)} returns a matrix of dimensions \texttt{ldim(x)}.

Methods

\begin{verbatim}
list("signature(x = "ddmatrix")")
\end{verbatim}

Examples

\begin{verbatim}
## 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, bldim=2)

y <- list(dim=dim(x), ldim=ldim(x), bldim=bldim(x))
\end{verbatim}
comm.print(y)
finalize()

## End(Not run)

---

**arithmetic**  

**Arithmetic Operators**

**Description**

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

**Usage**

```r
## 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 'ddmatrix,ddmatrix'
e1 %/% e2

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

## S4 method for signature 'ddmatrix,numeric'
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.

### 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, bldim=2)

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

finalize()

## End(Not run)

---

as.ddmatrix  Non-Distributed object to Distributed Object Converters

**Description**

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

**Usage**

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

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

```r
## 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.
as.matrix

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 redistribute 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

```r
## 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, bdim=2)
dx

### Can also be common to all ranks
y <- matrix(1:25, 5, bdim=2)
dy <- as.ddmatrix(y)
dy

finalize()

## End(Not run)
```
Description

Converts a distributed matrix into a non-distributed matrix.

Usage

```r
## 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

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

comm.print(y)
finalize()

## End(Not run)
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() function. 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 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.

**Examples**

```r
## 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)
```
as.vector

**Distributed object to Vector Converters**

### Description

Converts a distributed matrix into a non-distributed vector.

### Usage

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

```r
data <- ddmatrix(1:16, ncol=4, bldim=2)
y <- as.vector(data)
```

### 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.

### Examples

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

comm.print(y)
finalize()
```
binds

Row and Column binds for Distributed Matrices

Description

Row and column binds.

Usage

```r
rbind.ddmatrix(..., ICTXT = .pb docker$ICTXT, deparse.level = 1)
```

```r
cbind.ddmatrix(..., ICTXT = .pb docker$ICTXT, deparse.level = 1)
```

Arguments

- `...`: vectors, matrices, or distributed matrices.
- `ICTXT`: BLACS communicator number for return object.

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

```r
list("signature(... = \"ANY\")")
```

an R object.

Examples

```r
## 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()
```
Description

qr() takes the QR decomposition.

Usage

```r
## 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

```r
list("signature(x = "ddmatrix")")
list("signature(x = "ANY")")
```
Examples

```r
## 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)
initNgrid()
commNprint(id)
finalize()

## End(Not run)
```

---

**companion**

*Generate Companion Matrices*

Description

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

Usage

```r
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".
- **bldim**: blocking dimension.
- **ICTXT**: BLACS context number.
Details

For a degree n polynomial,

\[ x^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0 \]

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

<table>
<thead>
<tr>
<th>Logical Comparisons</th>
</tr>
</thead>
</table>

Description

Logical comparisons.

Usage

```r
## 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'
e1 < e2

## S4 method for signature 'ddmatrix'
e1 > e2
```
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'
Comparators

\[
e_1 \mid e_2
\]

```r
## S4 method for signature 'numeric,ddmatrix'
\[
e_1 \mid e_2
\]
## S4 method for signature 'ddmatrix,numeric'
\[
e_1 \& e_2
\]
## S4 method for signature 'numeric,ddmatrix'
\[
e_1 \& e_2
\]
```

**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.

**Examples**

```r
## 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)
```
Compute or estimate the Condition Number of a Distributed Matrix

**Description**
Computes or estimates the condition number.

**Usage**
```r
## S3 method for class 'ddmatrix'
kappa(z, exact = FALSE, norm = NULL, method = c("qr", "direct"), ...)
```
```r
## S4 method for signature 'ddmatrix'
rcond(x, norm = c("0", "1", "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 used 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.

**Examples**
```r
## 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, bdim=2)

cn <- rcond(x)

comm.print(cn)
```
covariance

finalize()

## End(Not run)

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 dealt 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.
ddmatrix-apply

Value

Returns a distributed matrix.

Examples

```r
## 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), bldim=2

cv <- cov(x)
print(cv)

finalize()

## End(Not run)
```

---

**ddmatrix-apply**  
*Apply Family of Functions*

Description

Apply a function to the margins of a distributed matrix.

Usage

```r
## 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
**ddmatrix-chol**

**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.

**Examples**

```r
## 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, bldim=2)
y <- as.vector(apply(x, 1, mean))
comm.print(y)
finalize()

## End(Not run)
```

**ddmatrix-chol**  

*Cholesky Factorization*

**Description**

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

**Usage**

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

**Arguments**

- `x`  
  numeric distributed matrices.
- `...`  
  Ignored.
ddmatrix-class

Details

Extensions of R linear algebra functions.

Value

chol() performs Cholesky factorization.

Examples

```r
## 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, bldim=2)

y <- solve(crossprod(x))

finalize()

## End(Not run)
```

---

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

```r
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 = .pbdd.env$BLDIM, ICTXT = .pbdd.env$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 con-
struction 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 pro-
cessor 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.
### Examples

```r
## 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, bldim=2)
dx

dy <- ddmatrix(data=1:4, nrow=7, ncol=5, bldim=2)
dy

finalize()

## End(Not run)
```

---

### ddmatrix-eigen  eigen

#### Description

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

#### Usage

```r
## 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

Examples

```r
## 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, bldim=2)

y <- eigen(crossprod(x))

finalize()

## End(Not run)
```

---

**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

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)

---

### ddmatrix-norm

<table>
<thead>
<tr>
<th>Norm</th>
</tr>
</thead>
</table>

#### Description

Computes the norm of a distributed matrix.

#### Usage

```r
## 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

```r
list("signature(x = '\texttt{ddmatrix}')")
```
Examples

```r
## 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)
```

---

**ddmatrix-prcomp**  
**Principal Components Analysis**

Description

Performs the principal components analysis.

Usage

```r
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 calculated.

Value

Returns a list.

Examples

```r
## 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)
```

Description

Print method for a distributed matrices.

Usage

```r
print(x, ...)

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

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

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.

Examples

```r
## 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)
```
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.

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)
**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**

```r
## 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.

**Examples**

```r
## 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)

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

finalize()

## End(Not run)
```
**Description**

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

**Usage**

```r
summary(object, ...)
```

```r
## 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.

**Examples**

```r
## 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)
```
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.

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

Singular Value Decomposition

Description

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

Usage

```r
## 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

```r
## 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 <- svd(x)
print(y)

finalize()

## End(Not run)
```

Description

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

Usage

```r
## 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'
\texttt{diag(x)}

## S4 method for signature 'matrix'
\texttt{diag(x, nrow, ncol)}

### Arguments

\textbf{x} \hspace{1cm} \text{distributed matrix or a vector.}

\textbf{nrow, ncol} \hspace{1cm} \text{in the case that x is a vector, these specify the global dimension of the diagonal distributed matrix to be created.}

\textbf{type} \hspace{1cm} \text{character. Options are 'matrix' or 'ddmatrix', with partial matching. This specifies the return type.}

\textbf{...} \hspace{1cm} \text{Extra arguments}

\textbf{bldim} \hspace{1cm} \text{blocking dimension.}

\textbf{ICTXT} \hspace{1cm} \text{BLACS context number.}

\textbf{min, max} \hspace{1cm} \text{Min and max values for random uniform generation.}

\textbf{mean, sd} \hspace{1cm} \text{Mean and standard deviation for random normal generation.}

\textbf{rate} \hspace{1cm} \text{Rate for random exponential generation.}

\textbf{shape, scale} \hspace{1cm} \text{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 \texttt{diag()} then it returns a global R vector.

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

### Examples

```r
## 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()

### Grab diagonal of a ddmatrix
x <- ddmatrix(1:16, 4, bldim=2)
y <- diag(x)
comm.print(y)

### Construct a diagonal ddmatrix
```
Description

Compute eigenvalues and, optionally, eigenvectors of a real symmetric matrix by searching 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.
### Examples

```r
## 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()
init.grid()

comm.set.seed(seed=1234, diff=TRUE)

x <- crossprod(ddmatrix("rnorm", 10, 3, bldim=2))
y <- as.matrix(x)

comm.print(eigen(y))

### Look for eigenvalues in the range 0 to 10
ev <- eigen(x, range=c(0, 10), only.values=TRUE)
comm.print(ev)

finalize()

## End(Not run)
```

---

**expm**

*Matrix Exponentiation*

---

### Description

Routines for matrix exponentiation.

### Usage

```r
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 Padé’ 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

## Examples

```r
## 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 <- matrix("rnorm", 5, 5, bldim=2)
expm(x)
```

## Description

Operators to extract or replace parts of a distributed matrix.

## Usage

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

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

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.

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, bdim=2)

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

finalize()

## End(Not run)
**getLocal**

---

### Description

Get the value of the distributed matrix at global indices \( g_i \times g_j \).

### Usage

```r
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 \( g_i \times g_j \).

### Examples

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

library(pbdDMAT)
init.grid()

x <- ddmatrix(1:100, 10, bldim=c(2, 2))

val <- getLocal(x, 5, 1)
comm.print(val, all.rank=TRUE)

val <- getLocal(x, 5, 1, all.rank=FALSE)
comm.print(val, all.rank=TRUE)

finalize()

## End(Not run)
```
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

```r
## 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.

Examples

```r
## 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, bldim=2)

y <- head(y, 2)
p
print(y)

finalize()

## End(Not run)
```
Generate Hilbert Matrices

Description

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

Usage

```r
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.

Examples

```r
## 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 <- Hilbert(100, type="ddmatrix")

print(dx)
finalizer()

## End(Not run)
```
Description

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

Usage

```r
## 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:
```r
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.

Examples

```r
## 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()
```
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

Examples

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

suppressPackageStartupMessages(library(pbdDMAT, quiet=T))

init.grid()

comm.set.seed(seed=1234, diff=TRUE)

x <- ddmatrix("rnorm", 5, 5, bdim=2)
test <- comm.any(is.na(x))
comm.print(test)

x[1,1] <- NA
test <- comm.any(is.na(x))
comm.print(test)

finalize()

## End(Not run)
```

isSymmetric  isSymmetric

Description

Tests if a distributed matrix is symmetric.

Usage

```r
## 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**

```r
## 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

```r
## 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)
```

---

**Miscellaneous Mathematical Functions**

**Description**

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

**Usage**

```r
## 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)
```
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.
Examples

```r
## 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)
```

matmult  

### Matrix Multiplication

**Description**

Multiplies two distributed matrices, if they are conformable.

**Usage**

```r
## 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.

Examples

```r
## 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)
```

---

**Handle Missing Values in Distributed Matrices**

Description

Dealing with NA's and NaN's.

Usage

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

## S4 method for signature 'ddmatrix'

```r
da.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.

Examples

```r
## 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)
```

---

**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**

```r
## 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)
```

**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

\[ \text{qr.Q() recovers } Q \text{ from the output of } \text{qr().} \]
\[ \text{qr.R() recovers } R \text{ from the output of } \text{qr().} \]
\[ \text{qr.qy() multiplies } y \text{ by } Q. \]
\[ \text{qr.qy()} \text{ multiplies } y \text{ by the transpose of } Q. \]

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

Value

\[ \text{qr()} \text{ returns a list consisting of: } q_r - \text{rank} - \text{calculated numerical rank}, \tau - \text{pivot} - \text{"class" - attribute } "qr". \]

Examples

```r
## 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)
```

---

**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

```r
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 xCTX 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.

Examples

```r
## 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)

---

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

**Description**

Arithmetic reductions for distributed matrices.

**Usage**

- `rowMin(x, ...)`
- `rowMax(x, ...)`
- `colMin(x, ...)`
- `colMax(x, ...)`
- `rowSums(x, na.rm = FALSE)`
- `colSums(x, na.rm = FALSE)`
- `rowMeans(x, na.rm = FALSE)`
- `colMeans(x, na.rm = FALSE)`
- `rowMin(x, na.rm = FALSE)`
- `colMin(x, na.rm = FALSE)`
## S4 method for signature 'matrix'

\[
\text{colMin}(x, \text{na.rm} = \text{FALSE})
\]

## S4 method for signature 'ddmatrix'

\[
\text{rowMax}(x, \text{na.rm} = \text{FALSE})
\]

## S4 method for signature 'matrix'

\[
\text{rowMax}(x, \text{na.rm} = \text{FALSE})
\]

## S4 method for signature 'ddmatrix'

\[
\text{colMax}(x, \text{na.rm} = \text{FALSE})
\]

## S4 method for signature 'matrix'

\[
\text{colMin}(x, \text{na.rm} = \text{FALSE})
\]

**Arguments**

- **x**: numeric distributed matrix
- **...**: additional arguments
- **na.r\(m\)**: logical. Should missing (including NaN) be removed?

**Details**

Performs the reduction operation on a distributed matrix.

There are several legitimately new operations, including \(\text{rowMin()}\), \(\text{rowMax()}\), \(\text{colMin()}\), and \(\text{colMax()}\). These implementations are not really necessary in R because one can easily (and reasonably efficiently) do something like

\[
\text{apply}(X=x, \text{MARGIN}=1, \text{FUN}=\text{min}, \text{na.rm}=\text{TRUE})
\]

But \(\text{apply()}\) on a \(\text{ddmatrix}\) is very costly, and should be used sparingly.

\(\text{sd()}\) will compute the standard deviations of the columns, equivalent to calling \(\text{apply}(x, \text{MARGIN}=2, \text{FUN}=\text{sd})\) (which will work for distributed matrices, by the way). However, this should be much faster and use less memory than \(\text{apply()}\). If \(\text{reduce=FALSE}\) 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 \(\text{proc.dest}\).

**Value**

Returns a global numeric vector.

**Examples**

```r
## 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()
```
rounding

# 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)

---

### Description

Extensions of R rounding functions for distributed matrices.

### Usage

```r
## 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.
Examples

```r
## 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)
finalization()

## End(Not run)
```

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

```r
## 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.

Examples

```r
## 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)
```

---

**sparsity**  
*Sparsity of Matrix Objects*

Description

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

Usage

```r
## 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(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.
**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.

**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)
```
Index

!),ddmatrix,ddmatrix-method (Comparators), 17
!,ddmatrix,numeric-method (Comparators), 17
!,numeric,ddmatrix-method (Comparators), 17

*Topic Algebra
  chol2inv, 15
  condnums, 20
  ddmatrix-chol, 23
  ddmatrix-eigen, 27
  ddmatrix-lu, 28
  ddmatrix-norm, 29
  ddmatrix-solve, 34
  eigen2, 40
  expm, 41
  matmult, 53
  qr, 56
  transpose, 65
*Topic BLACS
  as.rowcyclic, 11
  redistribute, 57
*Topic Classes
  ddmatrix-class, 24
*Topic ConditionNumbers
  condnums, 20
*Topic Data
  as.ddmatrix, 8
  as.rowcyclic, 11
  companion, 16
  ddmatrix-constructors, 25
  Hilbert, 46
  redistribute, 57
*Topic Distributing
  as.ddmatrix, 8
  as.rowcyclic, 11
  redistribute, 57
*Topic Extraction
  Comparators, 17
  ddmatrix-apply, 22
  diag-constructors, 38
  extract, 42
  headsortails, 45
  insert, 47
  lm.fit, 50
  na, 54
*Topic Generation
  companion, 16
  ddmatrix-constructors, 25
  Hilbert, 46
*Topic Linear
  chol2inv, 15
  condnums, 20
  ddmatrix-chol, 23
  ddmatrix-eigen, 27
  ddmatrix-lu, 28
  ddmatrix-norm, 29
  ddmatrix-solve, 34
  eigen2, 40
  expm, 41
  matmult, 53
  qr, 56
  transpose, 65
*Topic Methods,Sparse
  sparsity, 63
*Topic Methods
  Accessors, 3
  arithmetic, 6
  as.matrix, 9
  as.vector, 13
  binds, 14
  chol2inv, 15
  Comparators, 17
  condnums, 20
  covariance, 21
  ddmatrix-apply, 22
  ddmatrix-chol, 23
  ddmatrix-eigen, 27
INDEX

ddmatrix-lu, 28
ddmatrix-norm, 29
ddmatrix-prcomp, 30
ddmatrix-print, 31
ddmatrix-scale, 33
ddmatrix-solve, 34
ddmatrix-summary, 35
ddmatrix-sumstats, 36
diag: constructors, 38
eigen2, 40
expm, 41
extract, 42
headsortails, 45
insert, 47
isdot, 48
lm.fit, 50
math, 51
matmult, 53
na, 54
qr, 56
reductions, 59
rounding, 61
sd, 62
sweep, 64
transpose, 65

*Topic **Norm**
ddmatrix-norm, 29

*Topic **Package**
pbdDMAT-package, 3

*Topic **Type**
Comparators, 17
isdot, 48
na, 54
*,ddmatrix,ddmatrix-method (arithmetic), 6
*,ddmatrix,numeric-method (arithmetic), 6
*,numeric,ddmatrix-method (arithmetic), 6
+,ddmatrix,ddmatrix-method (arithmetic), 6
+,ddmatrix,numeric-method (arithmetic), 6
+,numeric,ddmatrix-method (arithmetic), 6
-,ddmatrix,ddmatrix-method (arithmetic), 6
-,ddmatrix,missing-method (arithmetic), 6
-,numeric,ddmatrix-method (arithmetic), 6
/,ddmatrix,ddmatrix-method (arithmetic), 6
/,numeric,ddmatrix-method (arithmetic), 6
<,ddmatrix,ddmatrix-method (Comparators), 17
<,ddmatrix,numeric-method (Comparators), 17
<,numeric,ddmatrix-method (Comparators), 17
<=,ddmatrix,ddmatrix-method (Comparators), 17
<=,ddmatrix,numeric-method (Comparators), 17
<=,numeric,ddmatrix-method (Comparators), 17
>=,ddmatrix,ddmatrix-method (Comparators), 17
>=,ddmatrix,numeric-method (Comparators), 17
>=,numeric,ddmatrix-method (Comparators), 17
[^,ddmatrix,ddmatrix-method (Comparators), 17
[^,ddmatrix,numeric-method (Comparators), 17
[^,numeric,ddmatrix-method (Comparators), 17
[^,numeric,ddmatrix-method (Comparators), 17
[^,numeric,ddmatrix-method (Comparators), 17
[>,ddmatrix,ddmatrix-method (Comparators), 17
[>,ddmatrix,numeric-method (Comparators), 17
[>,numeric,ddmatrix-method (Comparators), 17
[>,numeric,ddmatrix-method (Comparators), 17
[>,numeric,ddmatrix-method (Comparators), 17
[>,numeric,ddmatrix-method (Comparators), 17
[<,ddmatrix,ANY,ANY,ANY-method (insert), 47
[<,ddmatrix,ANY,ANY,ddmatrix-method (insert), 47
[<,ddmatrix,ANY,ANY,ddmatrix-method (insert), 47
%*%,ddmatrix,ddmatrix-method (matmult), 53
as.in, ddmatrix-method (math), 51
atan, ddmatrix-method (math), 51
binds, 14
bldim (Accessors), 3
bldim, ddmatrix-method (Accessors), 3
cbind, ddmatrix (binds), 14
ceiling, ddmatrix-method (rounding), 61
chol (ddmatrix-chol), 23
chol, ddmatrix-method (ddmatrix-chol), 23
chol2inv, 15
chol2inv, ddmatrix-method (chol2inv), 15
chol2inv-method (chol2inv), 15
colMax (reductions), 59
colMax, ddmatrix-method (reductions), 59
colMeans, ddmatrix-method (reductions), 59
colMin (reductions), 59
colMin, ddmatrix-method (reductions), 59
colSums, matrix-method (reductions), 59
colSums, ddmatrix-method (reductions), 59
companion, 16
Comparators, 17
condnums, 20
cor, ddmatrix-method (covariance), 21
cos, ddmatrix-method (math), 51
cosh, ddmatrix-method (math), 51
cov, ddmatrix-method (covariance), 21
cov2cor, ddmatrix-method (covariance), 21
covariance, 21
crossprod, ddmatrix-method (matmult), 53
ddmatrix (ddmatrix-constructors), 25
ddmatrix, character-method (ddmatrix-constructors), 25
ddmatrix, ddmatrix-method (ddmatrix-constructors), 25
ddmatrix, matrix-method (ddmatrix-constructors), 25
ddmatrix, missing-method (ddmatrix-constructors), 25
ddmatrix, vector-method (ddmatrix-constructors), 25
ddmatrix-apply, 22
ddmatrix-chol, 23
ddmatrix-class, 24
ddmatrix-constructors, 25
ddmatrix-eigen, 27

%/%, ddmatrix, ddmatrix-method (arithmetic), 6
%/%, ddmatrix, numeric-method (arithmetic), 6
%/%, numeric, ddmatrix-method (arithmetic), 6
%, ddmatrix, ddmatrix-method (arithmetic), 6
%, ddmatrix, numeric-method (arithmetic), 6
%, numeric, ddmatrix-method (arithmetic), 6
&-, ddmatrix, ddmatrix-method (Comparators), 17
&-, ddmatrix, numeric-method (Comparators), 17
&-, numeric, ddmatrix-method (Comparators), 17
^-, ddmatrix, numeric-method (arithmetic), 6

abs, ddmatrix-method (math), 51
Accessors, 3
acos, ddmatrix-method (math), 51
all, ddmatrix-method (Comparators), 17
any, ddmatrix-method (Comparators), 17
apply (ddmatrix-apply), 22
apply, ddmatrix-method (ddmatrix-apply), 22
arithmetic, 6
as.block (as.rowcyclic), 11
as.blockcyclic (as.rowcyclic), 11
as.colblock (as.rowcyclic), 11
as.colcyclic (as.rowcyclic), 11
as.dmatrix, 8
as.ddmatrix, matrix-method (as.ddmatrix), 8
as.ddmatrix, NULL-method (as.ddmatrix), 8
as.ddmatrix, vector-method (as.ddmatrix), 8
as.matrix, 9
as.matrix, ddmatrix-method (as.matrix), 9
as.rowblock (as.rowcyclic), 11
as.rowcyclic, 11
as.vector, 13
as.vector, ddmatrix-method (as.vector), 13
ddmatrix-\text{lu}, 28
ddmatrix-\text{norm}, 29
ddmatrix-\text{prcomp}, 30
ddmatrix-\text{print}, 31
ddmatrix-\text{scale}, 33
ddmatrix-\text{solve}, 34
ddmatrix-\text{summary}, 35
ddmatrix-\text{sumstats}, 36
ddmatrix-\text{svd}, 37
\text{ddmatrix}\.local(\text{ddmatrix-constructors}), 25
\text{ddmatrix}\.local, \text{character-method (ddmatrix-constructors)}, 25
\text{ddmatrix}\.local, \text{matrix-method (ddmatrix-constructors)}, 25
\text{ddmatrix}\.local, \text{missing-method (ddmatrix-constructors)}, 25
\text{ddmatrix}\.local, \text{vector-method (ddmatrix-constructors)}, 25
diag(diag-\text{constructors}), 38
\text{diag}, \text{character-method (diag-\text{constructors})}, 38
\text{diag}, \text{ddmatrix-method (diag-\text{constructors})}, 38
\text{diag}, \text{matrix-method (diag-\text{constructors})}, 38
\text{diag}, \text{vector-method (diag-\text{constructors})}, 38
diag-\text{constructors}, 38
dim, \text{ddmatrix-method (Accessors)}, 3
\text{distribute (as.ddmatrix)}, 8
eigen(\text{ddmatrix-eigen}), 27
eigen, \text{ddmatrix-method (ddmatrix-eigen)}, 27
eigen2, 40
\text{exp}, \text{ddmatrix-method (math)}, 51
\text{expm}, 41
\text{expm}, \text{ddmatrix-method (expm)}, 41
\text{expm}, \text{matrix-method (expm)}, 41
\text{extract}, 42
\text{floor}, \text{ddmatrix-method (rounding)}, 61
\text{getLocal}, 44
\text{head.ddmatrix (headsor\text{tails})}, 45
\text{headsortails}, 45
\text{Hilbert}, 46
\text{ICTXT (Accessors)}, 3
\text{ICTXT}, \text{ddmatrix-method (Accessors)}, 3
\text{insert}, 47
\text{is.ddmatrix (isdot)}, 48
\text{is.infinite, ddmatrix-method (isdot)}, 48
\text{is.na, ddmatrix-method (isdot)}, 48
\text{is.nan, ddmatrix-method (isdot)}, 48
\text{is.numeric, ddmatrix-method (isdot)}, 48
\text{isdot}, 48
\text{isSymmetric}, 49
\text{isSymmetric}, \text{ddmatrix-method (isSymmetric)}, 49
\kappa \text{.ddmatrix (condnums)}, 20
\text{La.svd (ddmatrix-\text{svd})}, 37
\text{La.svd, ANY-method (ddmatrix-\text{svd})}, 37
\text{La.svd, ddmatrix-method (ddmatrix-\text{svd})}, 37
\text{l\text{dim} (Accessors)}, 3
\text{\text{l\text{dim}, ddmatrix-method (Accessors)}}, 3
\text{length, ddmatrix-method (Accessors)}, 3
\text{\text{l\text{m}.fit}}, 50
\text{\text{l\text{m}.fit, ddmatrix, ddmatrix-method (\text{l\text{m}.fit})}}, 50
\text{log, ddmatrix-method (math)}, 51
\text{log10, ddmatrix-method (method)}, 51
\text{log1p, ddmatrix-method (math)}, 51
\text{log2, ddmatrix-method (math)}, 51
\text{\text{lu (ddmatrix-\text{lu})}}, 28
\text{\text{lu, ddmatrix-method (ddmatrix-\text{lu})}}, 28
\text{\text{math}}, 51
\text{\text{matmult}}, 53
\text{\text{max, ddmatrix-method (ddmatrix-\text{sumstats})}}, 36
\text{\text{mean, ddmatrix-method (ddmatrix-\text{sumstats})}}, 36
\text{\text{median, ddmatrix-method (ddmatrix-\text{sumstats})}}, 36
\text{\text{min, ddmatrix-method (ddmatrix-\text{sumstats})}}, 36
\text{\text{na}}, 54
\text{\text{NCOL (Accessors)}}, 3
\text{\text{ncol (Accessors)}}, 3
\text{\text{NCOL, ddmatrix-method (Accessors)}}, 3
\text{\text{ncol, ddmatrix-method (Accessors)}}, 3
\text{\text{norm (ddmatrix-norm)}}, 29
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>norm</code></td>
<td><code>ddmatrix</code>, ANY-method</td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-norm</code>)</td>
</tr>
<tr>
<td><code>NROW(Accessors)</code></td>
<td></td>
</tr>
<tr>
<td><code>nrow(Alternatives)</code></td>
<td></td>
</tr>
<tr>
<td><code>NROW(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>nrow(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>pbddMAT Control</code></td>
<td>55</td>
</tr>
<tr>
<td><code>pbddMAT-package</code></td>
<td>3</td>
</tr>
<tr>
<td><code>prcomp(ddmatrix-prcomp)</code></td>
<td></td>
</tr>
<tr>
<td><code>prcomp(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-prcomp</code>)</td>
</tr>
<tr>
<td><code>print(ddmatrix-print)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>prod(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-sumstats</code>)</td>
</tr>
<tr>
<td><code>qr</code></td>
<td>56</td>
</tr>
<tr>
<td><code>qr(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>qr</code>)</td>
</tr>
<tr>
<td><code>qr.Q(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>qr.qty(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>qr.qy(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>qr.R(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rbind(ddmatrix)</code></td>
<td>14</td>
</tr>
<tr>
<td><code>rcond(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>reblock(redistribute)</code></td>
<td></td>
</tr>
<tr>
<td><code>reductions</code></td>
<td>59</td>
</tr>
<tr>
<td><code>round(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rounding</code></td>
<td>61</td>
</tr>
<tr>
<td><code>rowMax(reductions)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMax(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMax(matrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMeans(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMin(reductions)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMin(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowMin(matrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>rowSums(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>scale(ddmatrix-scale)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-scale</code>)</td>
</tr>
<tr>
<td><code>sd</code></td>
<td>62</td>
</tr>
<tr>
<td><code>sd(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>sd, ANY-method</code></td>
<td>(<code>sd</code>)</td>
</tr>
<tr>
<td><code>solve(ddmatrix, ANY-method)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-solve</code>)</td>
</tr>
<tr>
<td><code>sin</code></td>
<td>51</td>
</tr>
<tr>
<td><code>sinh</code></td>
<td>51</td>
</tr>
<tr>
<td><code>sqrt(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>sqrt</code></td>
<td>51</td>
</tr>
<tr>
<td><code>submatrix</code></td>
<td>3</td>
</tr>
<tr>
<td><code>submatrix(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>submatrix(Linalg-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>sum(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<code>ddmatrix-sumstats</code>)</td>
</tr>
<tr>
<td><code>summary(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>sweep</code></td>
<td>64</td>
</tr>
<tr>
<td><code>svd(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>svd, ANY-method</code></td>
<td>(<code>svd</code>)</td>
</tr>
<tr>
<td><code>svd(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>transpose</code></td>
<td>65</td>
</tr>
<tr>
<td><code>t(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>var(ddmatrix-method)</code></td>
<td></td>
</tr>
<tr>
<td><code>var(ddmatrix-method)</code></td>
<td></td>
</tr>
</tbody>
</table>