Package ‘pbdDMAT’

February 20, 2015

Title Programming with Big Data -- Distributed Matrix Methods
Version 0.2-3
Date 2013-3-4
Depends R (>= 2.14.0), methods, rlecuyer, pbdMPI (>= 0.2-1), pbdSLAP
(>= 0.1-6), pbdBASE (>= 0.2-3)
LazyLoad yes
LazyData yes
Description pbdDMAT contains high level S3 and S4 methods for creating,
modifying, and performing computations with dense, distributed
matrices. This includes a new class, 'ddmatrix', for storing all
of the distributed data details. Computation is handled mostly
by routines from the pbdBASE package.
SystemRequirements OpenMPI (>= 1.5.4) on Solaris, Linux and Mac,
MPICH2 (>= 1.4.1p1) on Windows
License GPL (>= 2)
NeedsCompilation No
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>
Author Drew Schmidt [aut, cre],
Wei-Chen Chen [aut],
George Ostrouchov [aut],
Pragneshkumar Patel [aut],
R Core team [ctb] (some wrappers taken from the base and stats
packages)
Repository CRAN
Date/Publication 2013-12-16 20:20:19
R topics documented:

- pbdDMAT-package ................................................. 3
- Apply .......................................................... 3
- Arithmetic ......................................................... 5
- as.ddmatrix ....................................................... 6
- as.matrix ........................................................ 7
- Binders ............................................................ 9
- chol2inv ........................................................... 10
- Companion ......................................................... 11
- Comparators ....................................................... 12
- ddmatrix-class .................................................... 13
- Diag ............................................................... 15
- Distribute .......................................................... 16
- DistributedMatrixCreation ................................. 18
- eigen2 .............................................................. 20
- Expm ............................................................... 21
- Extract ............................................................. 22
- Hilbert ............................................................. 23
- Insert .............................................................. 24
- kappa ............................................................... 25
- LinAlg .............................................................. 27
- lm.fit ............................................................... 29
- MatMult ............................................................ 30
- MiscMath .......................................................... 31
- NAs ................................................................. 33
- Norm ............................................................... 34
- pbdDMAT Control ................................................ 35
- PCA ................................................................. 36
- Print ............................................................... 37
- QR Decomposition .............................................. 38
- Reductions .......................................................... 40
- Round .............................................................. 42
- scale ............................................................... 43
- SimpleRedistributions ......................................... 44
- SlotAccessors ....................................................... 46
- Summary .......................................................... 48
- sweep .............................................................. 49
- Type ............................................................... 50
- Variance/Covariance .......................................... 51

Index ............................................................. 53
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/

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

Details

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

Value

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

Methods

signature(x = "ddmatrix")

Author(s)

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

See Also

- `prcomp`

Examples

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

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

Usage

\[
\begin{align*}
x + y \\
x - y \\
-y \\
x \times y \\
x / y \\
x ^ y \\
x \% y \\
x \%\% y
\end{align*}
\]

Arguments

\(x, y\) numeric distributed matrices or numeric vectors

Details

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

Value

Returns a distributed matrix.

Methods

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

See Also

Arithmetic, LinAlg, MatMult
Examples

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

finalize()

## End(Not run)
```

---

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

**Description**

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

**Usage**

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

**Arguments**

- `x`  
a numeric matrix

- `bldim`  
the blocking dimension for block-cyclically distributing the matrix across the process grid.

- `ICTXT`  
BLACS context number for return.

**Details**

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

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

---

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

**Description**

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

**Usage**

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

**Arguments**

- `x`  
a numeric matrix

- `bldim`  
the blocking dimension for block-cyclically distributing the matrix across the process grid.

- `ICTXT`  
BLACS context number for return.

**Details**

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

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

---

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

**Description**

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

**Usage**

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

**Arguments**

- `x`  
a numeric matrix

- `bldim`  
the blocking dimension for block-cyclically distributing the matrix across the process grid.

- `ICTXT`  
BLACS context number for return.

**Details**

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

The function will only work if `x` is stored on all processes, or `x` is stored on a single process (does not matter which) and every other process has NULL stored for `x`.
If several processes own pieces of the matrix `x`, then you can not use this function. You will have to create an appropriate `ddmatrix` on all processes and 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.

**See Also**

`Distribute`

**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=c(4,4))
print(dx)

finalize()
## End(Not run)
```

---

**as.matrix**  

**Distributed-to-non-distributed Matrix Converters**

**Description**

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

**Usage**

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

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

Details

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

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

Value

Returns an ordinary R matrix.

Methods

signature(x = "ddmatrix")

Examples

```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)

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

finalize()

## End(Not run)
```
Row and Column binds for Distributed Matrices

Description

Row and column binds.

Usage

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

Arguments

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

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
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()

# don't do this in production code
x <- matrix(1:16, ncol=4)
dx <- as.ddmatrix(x)
```
chol2inv

Inverse from Choleski (or QR) Decomposition

Description

qr() takes the QR decomposition.

Usage

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

Arguments

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

Details

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

Value

A numeric distributed matrix.

Methods

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

See Also

lm.fit

```r
y <- rbind(dx, dx)
print(y)
finalize()
## End(Not run)
```
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", 3, 3)
R <- qr.R(qr(x))
xtx.inv <- chol2inv(R)

id <- xtx.inv

print(id)

finalize()

## End(Not run)
```

Generate Companion Matrices

Description

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

Usage

```r
companion(coef, type="matrix", ..., bldim=.BLDIM, ICTXT=.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,

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

its associated companion matrix is a matrix of the form

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

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

- \( x == y \)
- \( x < y \)
- \( x > y \)
- \( x >= y \)
- \( x <= y \)
- \( x != y \)
- \( x & y \)
- \( x | y \)

## S4 method for signature 'ddmatrix'

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

## S4 method for signature 'ddmatrix'

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

Arguments

- \( x, y \)
  distributed matrix or numeric vector
- \( \text{na.rm} \)
  logical, indicating whether or not NA’s should first be removed. If not and an NA is present, NA is returned.
ddmatrix-class

Details
Performs the indicated logical comparison.
If na.rm is TRUE and only NA’s are present, then TRUE is returned.

Value
Returns a distributed matrix.

Methods
signature(x = "ddmatrix")

See Also
Type

Examples
## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r
library(pbdDMAT, quiet = TRUE)
init.grid()

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

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

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

finalize()

## End(Not run)

---

ddmatrix-class 

Class ddmatrix

Description
Distributed matrix class.

Creating Objects
new('ddmatrix', Data = ..., dim = ..., ldim = ..., bldim = ..., ICTXT = ...)
Slots

Data: LOCAL: Object of class matrix
dim: GLOBAL: Object of class numeric
ldim: LOCAL: Object of class numeric
bldim: GLOBAL: Object of class numeric
ICTXT: GLOBAL: Object of class numeric

Prototype

matrix Data  matrix(0.0)
numeric dim  c(1,1)
numeric ldim  c(1,1)
numeric bldim  c(1,1)
numeric ICTXT  0

Details

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

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

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

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

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

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

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

See Also

SlotAccessors
Diag

Distributed Matrix Diagonals

Description

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

Usage

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

Arguments

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

Details

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

Value

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

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

signature(x = "ddmatrix")
signature(x = "vector")
signature(x = "character")

See Also
Extract

Examples

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

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

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

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

finalize()

## End(Not run)

---

Distribute

Distribute/Redistribute matrices across the process grid

Description

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

Usage

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

Arguments

- `x`: a numeric matrix
- `dx`: numeric distributed matrix
Distribute

- **bldim**: the blocking dimension for block-cyclically distributing the matrix across the process grid.
- **xctxt**: the BLACS context number for initial distribution of the matrix x.
- **ictxt**: BLACS context number for return.

**Details**

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

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

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

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

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

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

**Value**

Returns a distributed matrix.

**See Also**

- as.ddmatrix

**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
```
DistributedMatrixCreation

Distributed Matrix Creation

Description

Methods for simple construction of distributed matrices.

Usage

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

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

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

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

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

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

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

ddmatrix.local(data, nrow = 1, ncol = 1, byrow = FALSE,
    ..., bldim = .BLDIM, ICTXT = .ICTXT)

### Arguments

data  
optional data vector.

nrow  
number of rows. Global rows for ddmatrix(). Local rows for ddmatrix.local().  
See details below.

ncol  
number of columns. Global columns for ddmatrix(). Local columns for ddmatrix.local().  
See details below.

byrow  
logical. If FALSE then the distributed matrix will be filled by column major  
storage, otherwise row-major.

...  
Extra arguments

min, max  
Min and max values for random uniform generation.

mean, sd  
Mean and standard deviation for random normal generation.

rate  
Rate for random exponential generation.

shape, scale  
Shape and scale parameters for random weibull generation.

bldim  
blocking dimension.

ICTXT  
BLACS context number.

### Details

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

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

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

The functions predicated with .local generate data with a fixed local dimension, i.e., each 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.

### Methods

signature(data = "character")
signature(data = "matrix")
signature(data = "missing")
signature(data = "vector")
See Also

as.ddmatrix

Examples

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

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

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

print(dx)
print(dy)

finalize()

## End(Not run)

eigen2  eigen2

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-8, orfac=1e-3)

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 eigen-
values, 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 cor-
respond 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**

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

**Arguments**

- `x` A numeric matrix or a numeric distributed matrix.

**Details**

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

$$
\text{expm}(x) = X/1! + X^2/2! + X^3/3! + \ldots
$$

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.

**Methods**

```r
signature(x = "ddmatrix")
signature(x = "matrix")
```
References

Matrix exponentiation using Padé’ approximations and scaling and squaring from: "New Scaling and Squaring Algorithm for the Matrix Exponential" Awad H. Al-Mohy and Nicholas J. Higham, August 2009

See Also

Arithmetic, Reductions, MatMult, LinAlg

---

Extract

Extract or Replace Parts of a Distributed Matrix

Description

Operators to extract or replace parts of a distributed matrix.

Usage

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

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

Arguments

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

Details

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

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

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

Examples

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

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

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

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

finalize()

## End(Not run)

---

<table>
<thead>
<tr>
<th>Hilbert</th>
<th>Generate Hilbert Matrices</th>
</tr>
</thead>
</table>

Description

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

Usage

Hilbert(n, type="matrix", ..., bldim=BLDIM, ICTXT=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 \( \text{type} \).

Value

Returns a matrix or a distributed matrix.

See Also

as.ddmatrix

Examples

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

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

dx <- Hilbert(100, type="ddmatrix")

print(dx)

finalize()

# End(Not run)
```

---

### Insert

Directly Insert Into Distributed Matrix Submatrix Slot

**Description**

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

**Usage**

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

**Arguments**

- \( x \) : numeric distributed matrix.
- \( i, j \) : global integer indices.
- \( \text{value} \) : replacement value. Can be a global vector or a ddmatrix.
kappa

Details

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

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

x@Data <- newMatrix

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

Value

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")

Examples

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

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

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

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

finalize()

## End(Not run)

---

kappa  
Compute or estimate the Condition Number of a Distributed Matrix

Description

Computes or estimates the condition number.
Usage

## S3 method for class 'ddmatrix'

\texttt{kappa}(z, exact = FALSE, norm = NULL, 
\hspace{1cm} method = c("qr", "direct"), ...)

## S4 method for signature 'ddmatrix'

\texttt{rcond}(x, norm = c("0", "1", "\infty"), 
\hspace{1cm} triangular = FALSE, ...)

Arguments

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

Value

Returns a number.

Methods

\begin{itemize}
  \item \texttt{signature(x = "ddmatrix")}
  \item \texttt{signature(z = "ddmatrix")}
\end{itemize}

See Also

\texttt{Norm}

Examples

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

\begin{verbatim}
library(pbdDMAT, quiet = TRUE)
init.grid()

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

cnm <- rcond(x)

comm.print(cnm)

finalize()
\end{verbatim}
## Linear Algebra Functions

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

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

### Arguments
- **object, x, a, b**
  numeric distributed matrices. If applicable, `a` and `b` must be on the same BLACS context and have the same blocking dimension.
- **tol**
  precision tolerance.
- **...**
  additional arguments.
- **nu**
  number of left singular vectors to return when calculating singular values.
- **nv**
  number of right singular vectors to return when calculating singular values.
- **symmetric**
  logical, if TRUE then the matrix is assumed to be symmetric and only the lower triangle is used. Otherwise `x` is inspected for symmetry.
- **only.values**
  logical, if TRUE then only the eigenvalues are returned. Otherwise both eigenvalues and eigenvectors are returned.
Details

Extensions of R linear algebra functions.

Value

t() returns the transposed matrix.
solve() solves systems and performs matrix inversion when argument b= is missing.
La.svd() performs singular value decomposition, and returns the transpose of right singular vectors if any are requested. Singular values are stored as a global R vector. Left and right singular vectors are unique up to sign. Sometimes core R (via LAPACK) and ScaLAPACK will disagree as to what the left/right singular vectors are, but the disagreement is always only up to sign.
svd() performs singular value decomposition. Differs from La.svd() in that the right singular vectors, if requested, are returned non-transposed. Singular values are stored as a global R vector. Sometimes core R (via LAPACK) and ScaLAPACK will disagree as to what the left/right singular vectors are, but the disagreement is always only up to sign.
eigen() computes the eigenvalues, and eigenvectors if requested. As with svd(), eigenvalues are stored in a global R vector.
chol() performs Cholesky factorization.
lu() performs LU factorization.

Methods

signature(x = "ddmatrix")
signature(a = "ddmatrix")
signature(b = "ddmatrix")

See Also

Arithmetic, Reductions, MatMult, MiscMath

Examples

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

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

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

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

finalize()
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, `residuals` - distributed matrix, `effects` - distributed matrix, `rank` - global numeric, `fitted.values` - distributed matrix, `assign` - NULL if `lm.fit()` is called directly, `qr` - list, same as return from `qr()`, `df.residual` - global numeric.

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

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

See Also

QR

Examples

```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)
```

Description

Multiplies two distributed matrices, if they are conformable.

Usage

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

Arguments

- `x, y`: numeric distributed matrices
Details

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

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

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

Value

Returns a distributed matrix.

Methods

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

See Also

Arithmetic, LinAlg, MatMult

Examples

## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r
library(pbdDMAT, quiet = TRUE)
init.grid()

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

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

finalize()

## End(Not run)
Usage

```r
# S4 method for signature 'ddmatrix'
abs(x)
# S4 method for signature 'ddmatrix'
sqrt(x)
# S4 method for signature 'ddmatrix'
exp(x)
# S4 method for signature 'ddmatrix'
log(x, base = exp(1))
# S4 method for signature 'ddmatrix'
log2(x)
# S4 method for signature 'ddmatrix'
log10(x)
# S4 method for signature 'ddmatrix'
log1p(x)
# S4 method for signature 'ddmatrix'
log(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.
NAs

Handle Missing Values in Distributed Matrices

Description

Dealing with NA’s and NaN’s.

Usage

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

Arguments

- **object**: numeric distributed matrix
- **...**: extra arguments
- **ICTXT**: optional BLACS context number for output
Details

Removes rows containing NA’s and NaN’s.

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

Only advanced users should supply an ICTXT value. Most should simply leave this argument blank. The context of the return is dependent on the function arguments. If the ICTXT= argument is missing, then the return will be redistributed across its input context object@CTXT. Otherwise, the return will be redistributed across the supplied ICTXT.

Methods

signature(object = "ddmatrix")

See Also

Type

Examples

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

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

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

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

finalize()

## End(Not run)

---

**Norm**

*Compute the Norm of a Distributed Matrix*

Description

Computes the norm.
Usage

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

Arguments

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

Value

Returns a number.

Methods

```r
signature(x = "ddmatrix")
```

See Also

- `ConditionNumbers`

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)
```

pbdDMAT Control

Some default parameters for pbdDMAT.

Description

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

Objects contain several parameters for communicators and methods.

**Details**

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

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

---

**PCA**

*Principal Components Analysis*

**Description**

Performs the principal components analysis.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

Returns a list.
Methods

signature(x = "ddmatrix")

Author(s)

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

Examples

## Not run:
# Save code in a file "demo.r" and run with 2 processors by
# > mpiexec -np 2 Rscript demo.r
library(pbdDMAT, quiet = TRUE)
init.grid()

comm.set.seed(diff=T)

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

y <- prcomp(x)

comm.print(y)

finalize()

## End(Not run)

Print method for a distributed matrices.

Usage

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

Arguments

x numeric distributed matrix

... additional arguments

all control for whether the entire distributed matrix should be printed to standard output

name character string that will be printed to standard output along with the matrix elements
Details

Print method for class ddmatrix.

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

Value

The function silently returns 0.

Methods

signature(x = "ddmatrix")

Examples

```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)
```

QR Decomposition

Description

qr() takes the QR decomposition.
qr.Q() recovers Q from the output of qr().
qr.R() recovers R from the output of qr().
qr.qy() multiplies y by Q.
qr.qty() multiplies y by the transpose of Q.
### Usage

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

### Arguments

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

### Details

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

### Value

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

### Methods

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

### See Also

- `lm.fit`
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)
```

### Description

Arithmetic reductions for distributed matrices.

### Usage

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

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

### Arguments

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

### Details

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

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

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

### Value

Returns a global numeric vector.

### Methods

signature(x = "ddmatrix")

### See Also

`arithmetic`

### Examples

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

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

#### Arguments

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

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

#### Value
Returns a distributed matrix.

#### Methods

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

#### See Also

`MiscMath`, `NAS`
Examples

```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)

finalize()

## End(Not run)
```

---

**scale**

<table>
<thead>
<tr>
<th>Scale</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

Returns a distributed matrix.

**Methods**

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

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

**Description**

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

**Usage**

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

---

**Author(s)**

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

**See Also**

`prcomp`

**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 <- scale(x)

print(y)
finalize()

## End(Not run)
```

---
Arguments

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

Details

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

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

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

- `as_colblock()` is the column-wise analogue of `as_rowblock()`. In other words, it block redistributes the data across context 2.

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

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

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

Value

Returns a distributed matrix.

See Also

- `as_ddmatrix`, `Distribute`

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 <- as.rowblock(dx)

x <- as.colblock(dx)

x <- as.rowcyclic(dx)

x <- as.colcyclic(dx)

x <- as.blockcyclic(dx)

finalize()

## End(Not run)

SlotAccessors

Accessor Functions for Distributed Matrix Slots

Description

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

Usage

## S4 method for signature 'ddmatrix'
nrow(x)
## S4 method for signature 'ddmatrix'
ncol(x)
## S4 method for signature 'ddmatrix'
NROW(x)
## S4 method for signature 'ddmatrix'
NCOL(x)
## S4 method for signature 'ddmatrix'
length(x)
## S4 method for signature 'ddmatrix'
dim(x)
## SlotAccessors

```r
## S4 method for signature 'ddmatrix'
submatrix(x)
## S4 method for signature 'ddmatrix'
ldim(x)
## S4 method for signature 'ddmatrix'
bldim(x)
## S4 method for signature 'ddmatrix'
ICTXT(x)
## S4 method for signature 'ddmatrix'
ownany(x, ...)
## S4 method for signature 'missing'
ownany(dim, bldim=.BLDIM, ICTXT=.ICTXT, x)
```

### Arguments

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

### Details

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

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

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

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

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

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

### Value

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

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

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

### Methods

`signature(x = "ddmatrix")`
Examples

```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, 3)
x <- as.ddmatrix(x)
y <- list(dim=dim(x), ldim=ldim(x), bldim=bldim(x))
comm.print(y)
finalise()

## End(Not run)
```

---

**Summary**

**Distributed Matrix Summary**

**Description**

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

**Usage**

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

**Arguments**

- `object` numeric distributed matrix

**Details**

The return is on process 0 only.

**Value**

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

**Methods**

```r
signature(x = "ddmatrix")
```
sweep

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)
```

---

**sweep**

**Sweep**

### Description

Sweep vector or ddmatrix from a distributed matrix.

### Usage

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

### Arguments

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

### Details

Sweep vector or ddmatrix from a distributed matrix.

### Value

Returns a distributed matrix.
Methods

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

Type

<table>
<thead>
<tr>
<th>Type Checks, Including NA, NaN, etc.</th>
</tr>
</thead>
</table>

Description

Functions to check for various types.

Usage

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

Arguments

x     numeric distributed matrix

Details

Performs the appropriate type check.

Value

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

Methods

signature(x = "ddmatrix")

See Also

NAS
Variance/Covariance

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

Returns a distributed matrix.

Methods

signature(x = "ddmatrix")
signature(V = "ddmatrix")

Author(s)

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

See Also

prcomp

Examples

```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)
ininit.grid()

x <- dddmatrix("rnorm", nrow=3, ncol=3)
cv <- cov(x)
print(cv)

finalize()

## End(Not run)
```
Index

!= (Comparators), 12
!=, ddmatrix, ddmatrix-method (Comparators), 12
!=, ddmatrix, numeric-method (Comparators), 12
!=, numeric, ddmatrix-method (Comparators), 12

*Topic **BLACS**
  Distribute, 16
  SimpleRedistributions, 44
*Topic **Classes**
  ddmatrix-class, 13
*Topic **ConditionNumbers**
  kappa, 25
*Topic **Data Generation**
  Companion, 11
  DistributedMatrixCreation, 18
  Hilbert, 23
*Topic **Distributing Data**
  as.ddmatrix, 6
  Distribute, 16
  SimpleRedistributions, 44
*Topic **Extraction**
  Apply, 3
  Comparators, 12
  Diag, 15
  Extract, 22
  Insert, 24
  lm.fit, 29
  NAs, 33
*Topic **Linear Algebra**
  chol2inv, 10
  eigen2, 20
  Expm, 21
  kappa, 25
  LinAlg, 27
  MatMult, 30
  Norm, 34
  QR Decomposition, 38
*Topic **Methods**
  Apply, 3
  Arithmetic, 5
  as.matrix, 7
  Binders, 9
  chol2inv, 10
  Comparators, 12
  Diag, 15
  eigen2, 20
  Expm, 21
  Extract, 22
  Insert, 24
  kappa, 25
  LinAlg, 27
  lm.fit, 29
  MatMult, 30
  MiscMath, 31
  NAs, 33
  Norm, 34
  PCA, 36
  Print, 37
  QR Decomposition, 38
  Reductions, 40
  Round, 42
  scale, 43
  SlotAccessors, 46
  Summary, 48
  sweep, 49
  Type, 50
  Variance/Covariance, 51
*Topic **Norm**
  Norm, 34
*Topic **Package**
  pbdDMAT-package, 3
*Topic **Type**
  Comparators, 12
  NAs, 33
  Type, 50
* (Arithmetic), 5
*, ddmatrix, ddmatrix-method
  (Arithmetic), 5
*, ddmatrix, numeric-method (Arithmetic), 5
*, numeric, ddmatrix-method (Arithmetic), 5
*+method (Arithmetic), 5
+ (Arithmetic), 5
+ , ddmatrix, ddmatrix-method
  (Arithmetic), 5
+ , ddmatrix, numeric-method (Arithmetic), 5
+, numeric, ddmatrix-method (Arithmetic), 5
+ , method (Arithmetic), 5
- (Arithmetic), 5
-, ddmatrix, ddmatrix-method
  (Arithmetic), 5
-, ddmatrix, missing-method (Arithmetic), 5
-, ddmatrix, numeric-method (Arithmetic), 5
-, numeric, ddmatrix-method (Arithmetic), 5
--method (Arithmetic), 5
.BLDIM (pbdDMAT Control), 35
.ICTXT (pbdDMAT Control), 35
/(Arithmetic), 5
/, ddmatrix, ddmatrix-method
  (Arithmetic), 5
/, ddmatrix, numeric-method (Arithmetic), 5
/, numeric, ddmatrix-method (Arithmetic), 5
/-method (Arithmetic), 5
< (Comparators), 12
< , ddmatrix, ddmatrix-method
  (Comparators), 12
< , ddmatrix, numeric-method
  (Comparators), 12
< , numeric, ddmatrix-method
  (Comparators), 12
<= (Comparators), 12
<= , ddmatrix, ddmatrix-method
  (Comparators), 12
<= , ddmatrix, numeric-method
  (Comparators), 12
<=, numeric, ddmatrix-method
  (Comparators), 12
=> (Comparators), 12
=> , ddmatrix, ddmatrix-method
  (Comparators), 12
=> , ddmatrix, numeric-method
  (Comparators), 12
>= , numeric, ddmatrix-method
  (Comparators), 12
[ (Extract), 22
[, ddmatrix-method (Extract), 22
[ -method (Extract), 22
[<= (Insert), 24
[<= , ddmatrix, ANY, ANY, ANY-method
  (Insert), 24
[<= , ddmatrix, ANY, ANY, ddmatrix-method
  (Insert), 24
[<= -method (Insert), 24
%% (MatMult), 30
%% , ddmatrix, ddmatrix-method (MatMult), 30
%% -method (MatMult), 30
%% (Arithmetic), 5
%% , ddmatrix, ddmatrix-method
  (Arithmetic), 5
%% , ddmatrix, numeric-method
  (Arithmetic), 5
%% , numeric, ddmatrix-method
  (Arithmetic), 5
%% -method (Arithmetic), 5
%% -method (Arithmetic), 5
%% (Arithmetic), 5
%% , ddmatrix, ddmatrix-method
  (Arithmetic), 5
%% , ddmatrix, numeric-method
  (Arithmetic), 5
%% , numeric, ddmatrix-method
  (Arithmetic), 5
%% (Arithmetic), 5
%% , ddmatrix, ddmatrix-method
  (Arithmetic), 5
%% , ddmatrix, numeric-method
  (Arithmetic), 5
%% (Arithmetic), 5
INDEX

%%, numeric, ddmatrix-method
(Arithmetic), 5
%%-method (Arithmetic), 5
& (Comparators), 12
&, ddmatrix, ddmatrix-method
(Comparators), 12
&, ddmatrix, numeric-method
(Comparators), 12
&, numeric, ddmatrix-method
(Comparators), 12
^ (Arithmetic), 5
^, ddmatrix, ddmatrix-method
(Arithmetic), 5
^, numeric, numeric-method (Arithmetic), 5
^, numeric, ddmatrix-method (Arithmetic), 5
^, method (Arithmetic), 5
abs (MiscMath), 31
abs, ddmatrix-method (MiscMath), 31
abs-method (MiscMath), 31
acos (MiscMath), 31
acos, ddmatrix-method (MiscMath), 31
acos-method (MiscMath), 31
all (Comparators), 12
all, ddmatrix-method (Comparators), 12
all-method (Comparators), 12
any (Comparators), 12
any, ddmatrix-method (Comparators), 12
any-method (Comparators), 12
apply, 3
apply (Apply), 3
apply, ddmatrix-method (Apply), 3
apply-method (Apply), 3
Arithmetic, 5, 5, 22, 28, 31, 33, 41
as.block (SimpleRedistributions), 44
as.blockcyclic (SimpleRedistributions), 44
as.colblock (SimpleRedistributions), 44
as.colcyclic (SimpleRedistributions), 44
as.ddmatrix, 6, 17, 20, 24, 45
as.ddmatrix, matrix-method
(as.ddmatrix), 6
as.ddmatrix, NULL-method (as.ddmatrix), 6
as.ddmatrix, vector-method
(as.ddmatrix), 6
as.ddmatrix-method (as.ddmatrix), 6
as.matrix, 7
as.matrix, ddmatrix-method (as.matrix), 7
as.matrix-method (as.matrix), 7
as.rowblock (SimpleRedistributions), 44
as.rowcyclic (SimpleRedistributions), 44
as.vector (as.matrix), 7
as.vector, ANY-method (as.matrix), 7
as.vector, ddmatrix-method (as.matrix), 7
as.vector-method (as.matrix), 7
asin (MiscMath), 31
asin, ddmatrix-method (MiscMath), 31
asin-method (MiscMath), 31
atan (MiscMath), 31
atan, ddmatrix-method (MiscMath), 31
atan-method (MiscMath), 31
Binders, 9
bldim (SlotAccessors), 46
bldim(), 14
bldim, ddmatrix-method (SlotAccessors), 46
bldim-method (SlotAccessors), 46
cbind (Binders), 9
cbind, ...-method (Binders), 9
cbind, ANY-method (Binders), 9
cbind-method (Binders), 9
ceiling (Round), 42
ceiling, ddmatrix-method (Round), 42
ceiling-method (Round), 42
chol (LinAlg), 27
chol, ddmatrix-method (LinAlg), 27
chol-method (LinAlg), 27
chol2inv, 10
chol2inv, ddmatrix-method (chol2inv), 10
chol2inv-method (chol2inv), 10
colMax (Reductions), 40
colMax, ddmatrix-method (Reductions), 40
colMax-method (Reductions), 40
colMeans (Reductions), 40
colMeans, ddmatrix-method (Reductions), 40
colMeans-method (Reductions), 40
colMin (Reductions), 40
colMin, ddmatrix-method (Reductions), 40
colMin-method (Reductions), 40
colSums (Reductions), 40
colSums, ddmatrix-method (Reductions), 40
colSums-method (Reductions), 40
Companion, 11
INDEX

accompany (Companion), 11
Comparators, 12
ConditionNumbers, 35
ConditionNumbers (kappa), 25
cor (Variance/Covariance), 51
cor, ddmatrix-method (Variance/Covariance), 51
cor-method (Variance/Covariance), 51
cor2cov (Variance/Covariance), 51
cov (Variance/Covariance), 51
cov, ddmatrix-method (Variance/Covariance), 51
cov-method (Variance/Covariance), 51
cov2cor (Variance/Covariance), 51
cov2cor-method (Variance/Covariance), 51
crossprod (MatMult), 30
crossprod, ddmatrix, ANY-method (MatMult), 30
crossprod, ddmatrix-method (MatMult), 30
crossprod-method (MatMult), 30

ddmatrix (ddmatrix-class), 13
ddmatrix, character-method (DistributedMatrixCreation), 18
ddmatrix, matrix-method (DistributedMatrixCreation), 18
ddmatrix, missing-method (DistributedMatrixCreation), 18
ddmatrix, vector-method (DistributedMatrixCreation), 18
ddmatrix-class, 13
ddmatrix-method (DistributedMatrixCreation), 18
ddmatrix.local, character-method (DistributedMatrixCreation), 18
ddmatrix.local, matrix-method (DistributedMatrixCreation), 18
ddmatrix.local, missing-method (DistributedMatrixCreation), 18

ddmatrix.local, vector-method (DistributedMatrixCreation), 18
ddmatrix.local-method (DistributedMatrixCreation), 18
Diag, 15
diag (Diag), 15
diag, character-method (Diag), 15
diag, ddmatrix-method (Diag), 15
diag, vector-method (Diag), 15
diag-method (Diag), 15
dim (SlotAccessors), 46
dim(), 14
dim, ddmatrix-method (SlotAccessors), 46
dim-method (SlotAccessors), 46
Distribute, 7, 16, 45
distribute (Distribute), 16
DistributedMatrixCreation, 18
eigen (LinAlg), 27
eigen, ddmatrix-method (LinAlg), 27
eigen-method (LinAlg), 27
eigen2, 20
exp (MiscMath), 31
exp, ddmatrix-method (MiscMath), 31
exp-method (MiscMath), 31
Expm, 21
expm (Expm), 21
expm, ddmatrix-method (Expm), 21
expm-method (Expm), 21
Extract, 16, 22
floor (Round), 42
floor, ddmatrix-method (Round), 42
floor-method (Round), 42
head (Extract), 22
Hilbert, 23
ICTXT (SlotAccessors), 46
ictxt(), 14
ICTXT, ddmatrix-method (SlotAccessors), 46
ICTXT-method (SlotAccessors), 46
Insert, 24
is.ddmatrix (Type), 50
is.infinite (Type), 50
is.infinite, ddmatrix-method (Type), 50
is.infinite-method (Type), 50
INDEX

is.na (Type), 50
is.na, ddmatrix-method (Type), 50
is.na-method (Type), 50
is.nan (Type), 50
is.nan, ddmatrix-method (Type), 50
is.nan-method (Type), 50
is.numeric (Type), 50
is.numeric, ddmatrix-method (Type), 50
is.numeric-method (Type), 50
isSymmetric (LinAlg), 27
isSymmetric, ddmatrix-method (LinAlg), 27
isSymmetric-method (LinAlg), 27

class (Type), 7

isSymmetric (LinAlg), 27

isnumeric (Type), 50

isnumeric, ddmatrix-method (Type), 50

isnumeric-method (Type), 50

issymmetric (Type), 50

issymmetric, ddmatrix-method (Type), 50

issymmetric-method (Type), 50

laSVD (LinAlg), 27

L. fit, 10, 29, 39

length, ddmatrix-method (SlotAccessors), 46

length-method (SlotAccessors), 46

LinAlg, 5, 22, 27, 31, 36

lm.fit, 10, 29, 39

lm.fit, ddmatrix-method (lm.fit), 29

lm.fit-method (lm.fit), 29

log (MiscMath), 31

log, ddmatrix-method (MiscMath), 31

log-method (MiscMath), 31

log10 (MiscMath), 31

log10, ddmatrix-method (MiscMath), 31

log10-method (MiscMath), 31

log1p (MiscMath), 31

log1p, ddmatrix-method (MiscMath), 31

log1p-method (MiscMath), 31

log2 (MiscMath), 31

log2, ddmatrix-method (MiscMath), 31

log2-method (MiscMath), 31

lu (LinAlg), 27

lu, ddmatrix-method (LinAlg), 27

lu-method (LinAlg), 27

max, ddmatrix-method (Regressions), 40

max-method (Regressions), 40

mean (Regressions), 40

mean, ddmatrix-method (Regressions), 40

median (Regressions), 40

median, ddmatrix-method (Regressions), 40

min (Regressions), 40

min, ddmatrix-method (Regressions), 40

min-method (Regressions), 40

MiscMath, 28, 31, 42

na.exclude (NAs), 33

na.exclude, ddmatrix-method (NAs), 33

na.exclude-method (NAs), 33

NAs, 33, 42, 50

NCOL (SlotAccessors), 46

col (SlotAccessors), 46

NCOL, ddmatrix-method (SlotAccessors), 46

col, ddmatrix-method (SlotAccessors), 46

NCOL-method (SlotAccessors), 46

col-method (SlotAccessors), 46

Norm, 26, 34

norm (Norm), 34

norm, ddmatrix-method (Norm), 34

norm-method (Norm), 34

NROW (SlotAccessors), 46

nrow (SlotAccessors), 46

NROW, ddmatrix-method (SlotAccessors), 46

nrow, ddmatrix-method (SlotAccessors), 46

NROW-method (SlotAccessors), 46

nrow-method (SlotAccessors), 46

ownany (SlotAccessors), 46

ownany (), 14

ownany, ddmatrix-method (SlotAccessors), 46

ownany, missing-method (SlotAccessors), 46

ownany-method (SlotAccessors), 46

pbddMAT Control, 35

pbddMAT-package, 3

PCA, 36

prcomp, 4, 44, 52

prcomp (PCA), 36

prcomp, ddmatrix-method (PCA), 36

prcomp-method (PCA), 36
Print, 37
print(Print), 37
print,ddmatrix-method(Print), 37
print-method(Print), 37
prod (Reductions), 40
prod,ddmatrix-method (Reductions), 40
prod-method (Reductions), 40
QR, 30
QR (QR Decomposition), 38
qr (QR Decomposition), 38
QR Decomposition, 38
qr,ddmatrix-method (QR Decomposition), 38
qr-method (QR Decomposition), 38
qr.Q (QR Decomposition), 38
qr.Q,ANY-method (QR Decomposition), 38
qr.Q-method (QR Decomposition), 38
qr.qty (QR Decomposition), 38
qr.qty,ANY-method (QR Decomposition), 38
qr.qty-method (QR Decomposition), 38
qr.qy (QR Decomposition), 38
qr.qy,ANY-method (QR Decomposition), 38
qr.qy-method (QR Decomposition), 38
qr.R (QR Decomposition), 38
qr.R,ANY-method (QR Decomposition), 38
qr.R-method (QR Decomposition), 38
rbind (Binders), 9
rbind,...-method (Binders), 9
rbind,ANY-method (Binders), 9
rbind-method (Binders), 9
rcond (kappa), 25
rcond,ddmatrix-method (kappa), 25
rcond-method (kappa), 25
redistribute (Distribute), 16
Reductions, 22, 28, 33, 40
Round, 42
round (Round), 42
round,ddmatrix-method (Round), 42
round-method (Round), 42
rowMax (Reductions), 40
rowMax,ddmatrix-method (Reductions), 40
rowMax-method (Reductions), 40
rowMeans (Reductions), 40
rowMeans,ddmatrix-method (Reductions), 40
rowMeans-method (Reductions), 40
rowMin (Reductions), 40
rowMin,ddmatrix-method (Reductions), 40
rowSums (Reductions), 40
rowSums,ddmatrix-method (Reductions), 40
rowSums-method (Reductions), 40
scale, 43
scale,ddmatrix,ANY,ANY-method (scale), 43
scale,ddmatrix-method (scale), 43
scale-method (scale), 43
sd (Variance/Covariance), 51
sd,ANY-method (Variance/Covariance), 51
sd,ddmatrix-method (Variance/Covariance), 51
SimpleRedistributions, 17, 44
sin (MiscMath), 31
sin,ddmatrix-method (MiscMath), 31
sin-method (MiscMath), 31
sinh (MiscMath), 31
sinh,ddmatrix-method (MiscMath), 31
sinh-method (MiscMath), 31
SlotAccessors, 14, 46
solve (LinAlg), 27
solve,ddmatrix,ANY-method (LinAlg), 27
solve,ddmatrix,ddmatrix-method (LinAlg), 27
solve-method (LinAlg), 27
sqrt (MiscMath), 31
sqrt,ddmatrix-method (MiscMath), 31
sqrt-method (MiscMath), 31
submatrix (SlotAccessors), 46
submatrix(), 14
submatrix,ddmatrix-method (SlotAccessors), 46
submatrix-method (SlotAccessors), 46
submatrix<-(Insert), 24
submatrix<-,ddmatrix-method (Insert), 24
submatrix<--method (Insert), 24
sum (Reductions), 40
sum,ddmatrix-method (Reductions), 40
sum-method (Reductions), 40
Summary, 48
summary (Summary), 48
summary,ddmatrix-method (Summary), 48
summary-method (Summary), 48
svd (LinAlg), 27
svd,ddmatrix-method (LinAlg), 27
INDEX

svd-method (LinAlg), 27
sweep, 49
sweep, ddmatrix, ANY, ddmatrix-method
   (sweep), 49
sweep, ddmatrix, ANY, vector-method
   (sweep), 49
sweep-method (sweep), 49

t (LinAlg), 27
t, ddmatrix-method (LinAlg), 27
t-method (LinAlg), 27
tail (Extract), 22
tan (MiscMath), 31
tan, ddmatrix-method (MiscMath), 31
tan-method (MiscMath), 31
tanh (MiscMath), 31
tanh, ddmatrix-method (MiscMath), 31
tanh-method (MiscMath), 31
tcrossprod (MatMult), 30
tcrossprod, ddmatrix, ANY-method
   (MatMult), 30
tcrossprod, ddmatrix-method (MatMult), 30
tcrossprod-method (MatMult), 30
Type, 13, 34, 50

var (Variance/Covariance), 51
var, ddmatrix-method
   (Variance/Covariance), 51
var-method (Variance/Covariance), 51
Variance/Covariance, 51