

# pbDDEMO

June 5, 2013

---

pbDDEMO-package

*Demonstrations and Examples for the pbD Project*

---

## Description

Demos

## Details

Package: pbDDMAC  
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.

## References

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

---

DEMO Control

*A set of controls in pbDDDEMO*

---

## Description

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

## Format

Objects contain several default parameters for BLACS.

## Details

The elements of `.DEMO.CT` are default values

Elements	Default	Usage
<code>spmd.major</code>	1L	a default SPMD row-major
<code>ictxt</code>	0L	a default BLACS context
<code>bldim</code>	<code>c(2L,2L)</code>	a default block dimension
<code>divide.method</code>	"block.cyclic"	a default balance method

## Author(s)

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

## References

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

---

timer

*A Timing Function for SPMD Routines*

---

## Description

A timing function for use with parallel codes executed in the batch SPMD style.

## Usage

```
timer(timed)
```

## Arguments

`timed` expression to be timed.

## Details

Finds the min, mean, and max execution time across all independent processes executing the operation `timed`.

**Description**

These functions are examples of simple statistics via MPI calls.

**Usage**

```
mpi.stat(x.spmd)
mpi.bin(x.spmd, breaks = pi/3 * (-3:3))
mpi.quantile(x.spmd, prob = 0.5)
mpi.ols(y.spmd, X.spmd)
```

**Arguments**

x.spmd	a SPMD vector.
breaks	a set to break data in groups.
prob	a desired probability for quantile.
y.spmd	a SPMD vector.
X.spmd	a SPMD matrix.

**Details**

x.spmd and y.spmd are vectors with length N.spmd. X.spmd is a matrix with dimension N.spmd \* p and exists on all processors. N.spmd may be vary across processors.

For demonstration purpose, these objects should not contains weird values such NA.

**Value**

mpi.stat returns sample mean and sample variance.  
mpi.bin returns binning counts for the given breaks.  
mpi.quantile returns a quantile.  
mpi.ols returns ordinary least square estimates (beta\_hat).

**Author(s)**

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

**References**

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

## Examples

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(sample_stat,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(binning,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(quantile,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(ols,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(spmd2dmat,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(balance,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

---

spmd\_dmat

*SPMD Matrix to Distributed Dense Matrix and vice versa*


---

## Description

This function convert a SPMD matrix and a distributed dense matrix.

## Usage

```
spmd2dmat(X.spmd, skip.balance = FALSE, comm = .SPMD.CT$comm,
          spmd.major = .DEMO.CT$spmd.major, bldim = .DEMO.CT$bldim,
          ICTXT = .DEMO.CT$ictxt)
dmat2spmd(X.dmat, bal.info = NULL, comm = .SPMD.CT$comm,
          spmd.major = .DEMO.CT$spmd.major)
```

## Arguments

X.spmd	a SPMD matrix.
skip.balance	if load.balance were skipped.
comm	a communicator number.
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid.
spmd.major	1 for row-major storage, 2 for column-major.
ICTXT	BLACS context number for return.
X.dmat	a ddmatrix matrix.
bal.info	a returned object from balance.info.

**Details**

`X.spmd` is a matrix with dimension `N.spmd * p` and exists on all processors. `N.spmd` may be vary across processors.

If `skip.balance = TRUE`, then `load.balance` will not be called and `X.spmd` is preassumed to be balanced.

For demonstration purpose, these objects should not contains weird values such as NA.

`dmat2spmd` is supposed returned a balanced `spmd` matrix if `bal.info` is not supplied.

**Value**

`spmd2dmat` returns a `ddmatrix` object. `dmat2spmd` returns a (balanced) `spmd` matrix.

**Author(s)**

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

**References**

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

**Examples**

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(spmd_dmat,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

---

load.balance

*Load Balance of Dataset*


---

**Description**

These functions will rearrange data for all processors such that the data amount of each processor is nearly equal.

**Usage**

```
balance.info(X.spmd, comm = .SPMD.CT$comm, spmd.major = .DEMO.CT$spmd.major,
             method = .DEMO.CT$divide.method)
load.balance(X.spmd, bal.info = NULL, comm = .SPMD.CT$comm,
             spmd.major = .DEMO.CT$spmd.major)
unload.balance(new.X.spmd, bal.info, comm = .SPMD.CT$comm)
```

**Arguments**

<code>X.spmd</code>	a SPMD data matrix (converted if not).
<code>comm</code>	a communicator number.
<code>bal.info</code>	a returned object from <code>balance.info</code> .
<code>spmd.major</code>	1 for row-major storage, 2 for column-major.
<code>new.X.spmd</code>	a SPMD data matrix or vector
<code>method</code>	"block.cyclic" or "block0".

**Details**

`X.spmd` is the data matrix with dimension  $N.spmd * p$  and exists on all processors where  $N.spmd$  may be vary across processors. If `X.spmd` is a vector, then it is converted to a  $N.spmd * 1$  matrix.

`balance.info` provides the information how to balance data set such that all processors own similar amount of data. This information may be also useful for tracking where the data go or from.

`load.balance` does the job to transfer data from one processor with more data to the other processors with less data based on the balance information `balance.info`.

`unload.balance` is the inversed function of `load.balance`, and it takes the same information `bal.info` to reverse the balanced result back to the original order. `new.X.spmd` is usually the output of `load.balance{X.spmd}` or other results of further computing of it. Again, if `new.X.spmd` is a vector, then it is converted to an one column matrix.

**Value**

`balance.info` returns a list contains two data frames and two vectors.

Two data frames are `send` and `recv` for sending and receiving data. Each data frame has two columns `org` and `belong` for where data original in and new belongs. Number of row of `send` should equal to the  $N.spmd$ , and number of row of `recv` should be nearly equal to  $n = N / COMM.SIZE$  where  $N$  is the total observations of all processors.

Two vectors are `N.allspmd` and `new.N.allspmd` which are all numbers of rows of `X.spmd` on all processes before and after load balance, correspondingly. Both have length equals to `comm.size(comm)`.

`load.balance` returns a matrix for each processor and the matrix has the dimension nearly equal to  $n * p$ .

`unload.balance` returns a matrix with the same length/rows as the original number of row of `X.spmd`.

**Warning(s)**

These function only support total object length is less than  $2^{32} - 1$  for machines using 32-bit integer.

**Author(s)**

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

## References

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

## Examples

```
## Not run:
# Save code in a file "demo.r" and run in 4 processors by
# > mpiexec -np 4 Rscript demo.r

### Setup environment.
library(pbdDEMO, quiet = TRUE)

### Generate an example data.
N.spmd <- 5 * (comm.rank() * 2)
X.spmd <- rnorm(N.spmd * 3)
dim(X.spmd) <- c(N.spmd, 3)
comm.cat("X.spmd[1:5,]\n", quiet = TRUE)
comm.print(X.spmd[1:5,], rank.print = 1, quiet = TRUE)

bal.info <- balance.info(X.spmd)
new.X.spmd <- load.balance(X.spmd, bal.info)
org.X.spmd <- unload.balance(new.X.spmd, bal.info)

comm.cat("org.X.spmd[1:5,]\n", quiet = TRUE)
comm.print(org.X.spmd[1:5,], rank.print = 1, quiet = TRUE)
if(any(org.X.spmd - X.spmd != 0)){
  cat("Unbalance fails in the rank ", comm.rank(), "\n")
}

### Quit.
finalize()

## End(Not run)
```

---

read.sql.ddmatrix	<i>A Simple Parallel SQL Reader</i>
-------------------	-------------------------------------

---

## Description

Read in a table from a SQL database in parallel as a distributed matrix.

## Usage

```
read.sql.ddmatrix(dbname, table, bldim = .BLDIM, num.rdrs = 1, ICTXT = 0)
```

**Arguments**

dbname	database name
table	name of the table from dbname to be read
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid
num.rdrs	number of processes to be used to read in the table
ICTXT	BLACS context number for return

**Details**

The function reads in data from a SQL database using the sqldf package into a distributed matrix.

It operates at a 'bare bones' level, in that it will be assumed that the table desired to be read in 'looks' very much like a matrix. That is, it assumes that the table is basically just a csv-like structure that has been stowed away in a database.

**Value**

Returns a distributed matrix.

---

read.csv.ddmatrix	<i>A Simple Parallel CSV Reader</i>
-------------------	-------------------------------------

---

**Description**

Read in a table from a CSV file in parallel as a distributed matrix.

**Usage**

```
read.csv.ddmatrix(file, sep=",", nrows, ncols, header=FALSE, bldim=4, num.rdrs=1, ICTXT=0)
```

**Arguments**

file	csv file name.
sep	separator character.
nrows, ncols	dimensions of the csv file. Allowed to be missing in function call.
header	logical indicating presence/absence of character header for file.
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid
num.rdrs	number of processes to be used to read in the table
ICTXT	BLACS context number for return



## Details

The function reads in data from a csv file into a distributed matrix. This function sits somewhere between `scan()` and `read.csv()`, but for parallel reads into a distributed matrix.

The arguments `nrow=` and `ncol=` are optional. In the case that they are left blank, they will be determined. However, note that doing so is costly, so knowing the dimensions beforehand can greatly improve performance.

Although frankly, the performance-minded should not be using csv's in the first place. Consider using the `pbdNCDF4` package for managing data.

## Value

Returns a distributed matrix.

---

LinAlgVerify

*At-Scale Verification Routines for Distributed Linear Algebra*


---

## Description

At-scale verification routines for distributed linear algebra.

## Usage

```
verify.svd(nrows=1e3, ncols=1e3, mean=0, sd=1, bldim=8, tol=1e-7,
           ICTXT = .DEMO.CT$ictxt)
verify.chol(nrows=1e3, mean=0, sd=1, bldim=8, tol=1e-7,
            ICTXT = .DEMO.CT$ictxt)
verify.inverse(nrows=1e3, mean=0, sd=1, bldim=8, tol=1e-7,
               ICTXT = .DEMO.CT$ictxt)
verify.solve(nrows=1e3, mean=0, sd=1, const=1, bldim=8, tol=1e-7,
             ICTXT = .DEMO.CT$ictxt)
```

## Arguments

<code>nrows, ncols</code>	global dimension.
<code>bldim</code>	blocking dimension.
<code>mean, sd</code>	mean and standard deviation when sampling from a normal distribution.
<code>const</code>	numerical value for generating a constant <code>ddmatrix</code> .
<code>tol</code>	numeric tolerance for testing equality. Differences smaller than <code>tol</code> are considered equal.
<code>ICTXT</code>	BLACS context

## Details

Finds the min, mean, and max execution time across all independent processes executing the operation timed.

---

## Temperature at Reference Height

### *Surface Air Temperature at Reference Height (TREFHT)*

---

#### Description

This is a practical example in NetCDF4 format and for data reading, writing, and transforming. This dataset is a partial output of the Surface Air Temperature at Reference Height (TREFHT) which is monthly averaged of Jan. 2004 from a CAM5 simulation. This dataset only contains a tiny part of ultra-large simulations conducted by Mr Prabhat and Michael Wehner of Lawrence Berkeley National Laboratory.

#### Format

An R data file contains two lists: `def` for structure definition of “TREFHT” in `ncvar4` class (see **pbdNCDF4** package for details), and data for output values of simulation in a matrix where rows are for 1152 longitudes and columns are for 768 latitudes.

#### Details

Version 5.0 of the Community Atmosphere Model (CAM) is the latest in a series of global atmosphere models developed primarily at the National Center for Atmospheric Research (NCAR).

TREFHT contains two lists: `def` and `data`.

`def` is a list contains usual definitions of NetCDF4. In this case, they define the variable “TREFHT” including 2D dimensions 1152 longitudes and 768 latitudes, 1 time step, the unit in Kelvin, ... etc.

`data` contains values in matrix with dimension  $1152 \times 768$ . Note that this matrix stores data in C format (column major), so it needs a transpose to obtains the R/Fortran format (row major). Also, the longitude order is not the same as the **maps** package. Please see the example below for the adjustment or by calling `demo('trefht', 'pbdDEMO')` inside an R session.

#### Author(s)

Mr Prabhat and Michael Wehner.

#### References

More datasets are available on ESGF (<http://www.earthsystemgrid.org/>) through the C20C project (on the NERSC portal).

CAM5: <http://www.cesm.ucar.edu/models/cesm1.0/cam/>

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

#### See Also

`ncvar_put_2D` and `ncvar_get_2D`.

**Examples**

```

## Not run: library(maps)
library(RColorBrewer)
library(pbdDEMO, quiet = TRUE)

lon <- TREFHT$def$dim[[1]]$vals          # longitude
lat <- TREFHT$def$dim[[2]]$vals          # latitude
da <- TREFHT$data                        # surface temperature

# Define Axes.
x <- c(lon[lon > 180] -360, lon[lon <= 195]) # adjustment for maps
y <- lat
z <- rbind(da[lon > 180,], da[lon <= 195,]) # adjustment for maps
xlim <- range(x)
ylim <- range(y)
zlim <- range(z)
col.z <- c(colorRampPalette(c("#0000FF", "#2BFCD3"))(100),
            colorRampPalette(c("#2BFCD3", "#5300AB"))(100),
            colorRampPalette(c("#5300AB", "#7CFA82"))(100),
            colorRampPalette(c("#7CFA82", "#A90055"))(100),
            colorRampPalette(c("#A90055", "#D6FC28"))(100),
            colorRampPalette(c("#D6FC28", "#FE0001"))(100))

# Plot
layout(matrix(c(1, 2), ncol = 1), heights = c(2, 1))
par(mar = c(4, 4, 4, 0))
plot(NULL, NULL, xlim = xlim, ylim = ylim, type = "n", axes = FALSE,
      xlab = "Longitude", ylab = "Latitude", main = "TREFHT (Jan. 2004)")
image(x, y, z, zlim = zlim, xlim = xlim, ylim = ylim,
      col = col.z, add = TRUE)

# Add Map.
map(add = TRUE)
abline(h = c(-23.5, 0, 23.5), v = 0, lty = 2)
xtickets <- seq(-180, 180, by = 30)
ytickets <- seq(-90, 90, by = 30)
box()
axis(1, at = xtickets, labels = xtickets)
axis(2, at = ytickets, labels = ytickets)

# Add Legend.
z.temp <- matrix(seq(zlim[1], zlim[2], length = 500), ncol = 1)
ztickets <- seq(230, 300, by = 10)
par(mar = c(4, 4, 0, 1))
plot(NULL, NULL, xlim = zlim, ylim = c(0, 1), type = "n", axes = FALSE,
      xlab = "TREFHT (Kelvin)", ylab = "")
image(z.temp, 0, z.temp, zlim = zlim, xlim = zlim, ylim = c(0, 1),
      col = col.z, add = TRUE)
axis(1, at = ztickets, labels = ztickets)

## End(Not run)

```

---

Parallel NetCDF4	<i>Read and Write Parallel NetCDF4 Files in SPMD and ddmatrix Format</i>
------------------	--

---

## Description

These functions write and read NetCDF4 files in SPMD and ddmatrix format.

## Usage

```
ncvar_get_dmat(nc, varid, verbose = FALSE, signedbyte = TRUE,
  collapse_degen = TRUE, bldim = .DEMO.CT$bldim,
  ICTXT = .DEMO.CT$ictxt, comm = .SPMD.CT$comm)
ncvar_get_spmd(nc, varid, verbose = FALSE, signedbyte = TRUE,
  collapse_degen = TRUE, comm = .SPMD.CT$comm,
  spmd.major = .DEMO.CT$spmd.major)
ncvar_put_dmat(nc, varid, vals, verbose = FALSE,
  comm = .SPMD.CT$comm)
ncvar_put_spmd(nc, varid, vals, verbose = FALSE,
  comm = .SPMD.CT$comm, spmd.major = .DEMO.CT$spmd.major)
```

## Arguments

<code>nc</code>	an object of class <code>ncdf4</code> (as returned by either function <code>nc_open_par</code> or function <code>nc_create_par</code> ), indicating what file to read from.
<code>varid</code>	See <code>ncvar_get</code> for details.
<code>verbose</code>	See <code>ncvar_get</code> for details.
<code>signedbyte</code>	See <code>ncvar_get</code> for details.
<code>collapse_degen</code>	See <code>ncvar_get</code> for details.
<code>vals</code>	See <code>ncvar_put</code> for details.
<code>spmd.major</code>	a SPMD major, either 1 for row-major or 2 for column-major.
<code>bldim</code>	the blocking dimension for block-cyclically distributing the matrix across the process grid.
<code>ICTXT</code>	BLACS context number for return.
<code>comm</code>	a communicator number.

## Details

`ncvar_get_*` are similar to `ncvar_get` of **pbdNCDF4**, but focus on 2D arrays and return a ddmatrix or SPMD matrix.

`ncvar_put_*` are also similar to `ncvar_put` of **pbdNCDF4**, but only dump 2D arrays.

**Value**

`ncvar_get_dmat` returns a `ddmatrix`, and `ncvar_get_spmc` returns a SPMD matrix in either row- or column major specified by `spmd.major`.

**Author(s)**

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

**References**

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

**See Also**

`.DEMO.CT`.

**Examples**

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(nc4_serial,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_parallel,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_dmat,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_spmc,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

# Index

## \*Topic **Distributing Data**

read.csv.ddmatrix, [8](#)

read.sql.ddmatrix, [7](#)

## \*Topic **Package**

pbddemo-package, [1](#)

## \*Topic **Timing**

LinAlgVerify, [9](#)

timer, [2](#)

## \*Topic **datasets**

Temperature at Reference Height,  
[10](#)

## \*Topic **global variables**

DEMO Control, [1](#)

## \*Topic **programming**

load.balance, [5](#)

mpi.demo, [3](#)

Parallel NetCDF4, [12](#)

spmd\_dmat, [4](#)

.DEMO.CT (DEMO Control), [1](#)

balance.info (load.balance), [5](#)

DEMO Control, [1](#)

dmat2spmd (spmd\_dmat), [4](#)

LinAlgVerify, [9](#)

load.balance, [5](#)

mpi.bin (mpi.demo), [3](#)

mpi.demo, [3](#)

mpi.ols (mpi.demo), [3](#)

mpi.quantile (mpi.demo), [3](#)

mpi.stat (mpi.demo), [3](#)

ncvar\_get\_dmat (Parallel NetCDF4), [12](#)

ncvar\_get\_spmd (Parallel NetCDF4), [12](#)

ncvar\_put\_dmat (Parallel NetCDF4), [12](#)

ncvar\_put\_spmd (Parallel NetCDF4), [12](#)

Parallel NetCDF4, [12](#)

pbddemo-package, [1](#)

Practical Example (Temperature at  
Reference Height), [10](#)

read.csv.ddmatrix, [8](#)

read.sql.ddmatrix, [7](#)

spmd2dmat (spmd\_dmat), [4](#)

spmd\_dmat, [4](#)

Temperature at Reference Height, [10](#)  
timer, [2](#)

TREFHT (Temperature at Reference  
Height), [10](#)

unload.balance (load.balance), [5](#)

verify.chol (LinAlgVerify), [9](#)

verify.inverse (LinAlgVerify), [9](#)

verify.solve (LinAlgVerify), [9](#)

verify.svd (LinAlgVerify), [9](#)