Package ‘pbdBASE’

December 8, 2018

Type    Package
Title   Programming with Big Data -- Base Wrappers for Distributed Matrices
Version 0.5-0

Description An interface to and extensions for the 'PBLAS' and 'ScaLAPACK' numerical libraries. This enables R to utilize distributed linear algebra for codes written in the 'SPMD' fashion. This interface is deliberately low-level and mimics the style of the native libraries it wraps. For a much higher level way of managing distributed matrices, see the 'pbdDMAT' package.

License Mozilla Public License 2.0

Depends R (>= 3.0.0), methods
Imports utils, pbdMPI (>= 0.3-8), pbdSLAP(>= 0.2-4)

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

LazyLoad yes
LazyData yes
ByteCompile yes

NeedsCompilation yes

URL http://r-pbd.org/

BugReports http://group.r-pbd.org/

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

Maintainer Drew Schmidt <wraithematics@gmail.com>

RoxygenNote 6.0.1

Author Drew Schmidt [aut, cre], Wei-Chen Chen [aut], Sebastien Lamy de la Chapelle [aut], George Ostrouchov [aut],

1
Pragneshkumar Patel [aut],
Ewan Higgs [ctb]

Repository CRAN
Date/Publication 2018-12-08 15:00:03 UTC

R topics documented:

- pbdBASE-package ........................................... 3
- BASE Global Environment ................................ 5
- base.blacs_gridinit ........................................ 6
- base.crossprod ............................................. 6
- base.dallreduce .......................................... 7
- base.descinit ............................................. 7
- base.dgamx2d ............................................. 8
- base.dgesd2d ............................................. 8
- base.dhilbmk ............................................. 9
- base.dim0 ................................................ 10
- base.free_blacs_system_handle ......................... 10
- base.igamx2d ............................................. 11
- base.igsum2d ............................................. 11
- base.indxg2p ............................................. 12
- base.matexp .............................................. 13
- base.maxdim ............................................. 13
- base.minctxt ............................................ 14
- base.mksubmat .......................................... 14
- base.nbd ................................................ 15
- base.numroc ............................................. 15
- base.ownany ............................................. 16
- base.pdchtri ............................................ 17
- base.pdclvar ............................................ 17
- base.pdhilbmk .......................................... 18
- base.pdmkcpn1 .......................................... 18
- base.pdmvsum .......................................... 19
- base.pdsweep ........................................... 19
- base.procgrid ........................................... 20
- base.p_matexp_pade_wrap ............................... 20
- base.p_matpow_by_squaring_wrap ....................... 21
- base.rcolcpy ........................................... 21
- base.rcolcpy2 ......................................... 22
- base.redist ............................................. 22
- base.rl2blas ............................................ 23
- base.rl2insert .......................................... 23
- base.rpdgecon .......................................... 24
- base.rpdgelqf .......................................... 24
- base.rpdgels ........................................... 25
- base.rpdgemm ........................................... 25
- base.rpdgemr2d .......................................... 26
A package contains the basic methods for dealing with distributed data types, as well as the data types themselves.
Details
This package requires an MPI library (OpenMPI, MPICH2, or LAM/MPI).

**Author(s)**

Drew Schmidt <wrathematics AT gmail.com>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

---

**Description**

The environment for the pbdBASE package where "global" variables are stored.

**Usage**

```
.pbdBASEEnv
```

**Format**

An object of class `environment` of length 0.

**Details**

The `.blacs_gridinfo_` and `.blacs_initialized` objects are stored in this environment.
base.blacs_gridinit  Creating Grid From A System Context

Description

Creates a grid from a System Context obtained from a call to ‘sys2blacs_handle’.

Usage

base.blacs_gridinit(SYSCTX, NPROW, NPCOL, nprocs = pbdMPI::comm.size(comm),
                      comm = .pbd_env$SPMD.CT$comm)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCTX</td>
<td>System context obtained from a call to ‘sys2blacs_handle’</td>
</tr>
<tr>
<td>NPROW</td>
<td>Number of rows in the process grid</td>
</tr>
<tr>
<td>NPCOL</td>
<td>Number of columns in the process grid</td>
</tr>
<tr>
<td>nprocs</td>
<td>Number of processors in the communicator</td>
</tr>
<tr>
<td>comm</td>
<td>An MPI (not BLACS) communicator.</td>
</tr>
</tbody>
</table>

Value

A blacs context number

base.crossprod  crossprod

Description

Crossproduct.

Usage

base.crossprod(uplo, trans, x, descx, descc)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>Triangle whose values to use.</td>
</tr>
<tr>
<td>trans</td>
<td>tcrossprod or crossprod.</td>
</tr>
<tr>
<td>x</td>
<td>Matrix to crossprod.</td>
</tr>
<tr>
<td>descx</td>
<td>ScaLAPACK descriptor array.</td>
</tr>
<tr>
<td>descc</td>
<td>ScaLAPACK descriptor array of output.</td>
</tr>
</tbody>
</table>

Details

For advanced users only.
### base.dallreduce

**dallreduce**

**Description**

Allreduce

**Usage**

```plaintext
base.dallreduce(x, descx, op = "sum", scope = "All")
```

**Arguments**

- `x`: Matrix.
- `descx`: ScaLAPACK descriptor array.
- `op`: Operation.
- `scope`: Rows, columns, or both.

**Details**

For advanced users only.

---

### base.descinit

**descinit**

**Description**

Creates ScaLAPACK descriptor array.

**Usage**

```plaintext
base.descinit(dim, bldim, ldim, ICTXT = 0)
```

**Arguments**

- `dim`: Global dim.
- `bldim`: Blocking dim.
- `ldim`: Local dim.
- `ICTXT`: BLACS context.

**Details**

For advanced users only.
**base.dgmx2d**  
*BLACS Min*

**Description**

Min value across a process grid.

**Usage**

```
base.dgmx2d(ICTXT, SCOPE, m, n, x, lda, RDEST, CDEST)
```

```
base.igamn2d(ICTXT, SCOPE, m, n, x, lda, RDEST, CDEST)
```

```
base.dgamn2d(ICTXT, SCOPE, m, n, x, lda, RDEST, CDEST)
```

**Arguments**

- **ICTXT**: BLACS ICTXT.
- **SCOPE**: Rows, cols, or both.
- **m, n**: Problem size.
- **x**: Local values.
- **lda**: Leading dimension.
- **RDEST**: Row destination.
- **CDEST**: Col destination.

**Details**

For advanced users only.

---

**base.dgesd2d**  
*BLACS Point to Poin*

**Description**

Sent value across a process grid.

**Usage**

```
base.dgesd2d(ICTXT, SCOPE, m, n, x, lda, RDEST, CDEST)
```

```
base.dgerv2d(ICTXT, SCOPE, m, n, x, lda, RDEST, CDEST)
```
Arguments

ICTXT    BLACS ICTXT.
SCOPE    Rows, cols, or both.
m, n    Problem size.
x    Local values.
lda    Leading dimension.
RDEST    Row destination.
CDEST    Col destination.

Details

For advanced users only.
**Description**

Compute dimensions on process MYROW=MYCOL=0

**Usage**

base.dim0(dim, ICTXT = 0)

**Arguments**

- **dim**
  - Global dim.
- **ICTXT**
  - BLACS context.

**Details**

For advanced users only.

---

**Description**

Free Blacs System Handle

**Usage**

base.free_blacs_system_handle(SHANDLE)

**Arguments**

- **SHANDLE**
  - A system handle. Obtained via a call to ‘sys2blacs.handle’
**Description**

Max value across a process grid.

**Usage**

```
base.igamx2d(ictxt, scope, m, n, x, lda, rdest, cdest)
```

**Arguments**

- **ictxt**: BLACS ICTXT.
- **scope**: Rows, cols, or both.
- **m, n**: Problem size.
- **x**: Local values.
- **lda**: Leading dimension.
- **rdest**: Row destination.
- **cdest**: Col destination.

**Details**

For advanced users only.

---

**Description**

Sum across a process grid.

**Usage**

```
base.igsum2d(ictxt, scope, m, n, x, lda, rdest, cdest)
base.dgsum2d(ictxt, scope, m, n, x, lda, rdest, cdest)
```
Arguments

ICTXT  BLACS ICTXT.
SCOPE  Rows, cols, or both.
m, n  Problem size.
x  Local values.
lda  Leading dimension.
rdest  Row destination.
cdest  Col destination.

Details

For advanced users only.

Description

Computes the process coordinate which contains the entry of a distributed matrix specified by a
global index INDXGLOB. Simplified reimplementation of the ScaLAPACK aux INDXG2P func-
tion.

Usage

base.indxg2p(INDXGLOB, NB, NPROCS)

Arguments

INDXGLOB  Global index.
NB  Block size.
NPROCS  Total number of processors over which matrix is distributed.

Details

For advanced users only.
**base.matexp**

**Description**
Serial matrix exponentiation.

**Usage**
base.matexp(A, p = 6, t = 1)

**Arguments**
- A: Matrix to exponentiate.
- p: Pade’ expansion size.
- t: Scaling factor.

**Details**
For advanced users only.

---

**base.maxdim**

**maxdim**

**Description**
Compute maximum dimension across all nodes

**Usage**
base.maxdim(dim)

**Arguments**
- dim: Global dim.

**Details**
For advanced users only.
**base.minctxt**  
*Get BLACS Context Grid Information*

**Description**

Finds the smallest integers for creating a new BLACS context.

**Usage**

```
base.minctxt(after = 0)
```

**Arguments**

- `after` ignores all values below this integer as possibilities

**Details**

For advanced users only.

Returns the smallest integer which could become a new BLACS context value.

For example, if contexts 0, 1, and 2 are taken, and `after=0`, then the function returns 3. If 0, 1, 2, and 5 are taken, the function returns 3 if `after=0`, but returns 6 if `after=4`.

The function is useful when a transitory grid is needed, such as for reading in data onto a subset of processors before distributing out to the full grid.

**Value**

Returns the minimum value.

---

**base.mksubmat**  
*(Un)Distribute*

**Description**

*(Un)Distribute matrix.*

**Usage**

```
base.mksubmat(x, descx)
base.mkglobmat(x, descx, rsrc, csrc)
```

**Arguments**

- `x` Matrix.
- `descx` ScaLAPACK descriptor array.
- `rsrc`, `csrc` Row/column source.
For advanced users only.

**base.nbd**

*Next Best Divisor*

**Description**

Given integers \( n \) and \( d \), with \( n > d \), this function finds the "next best divisor" of \( n \) which is greater than or equal to \( d \).

**Usage**

```c
base.nbd(n, d)
```

**Arguments**

- \( n \) The divident (number divided into).
- \( d \) The candidate divisor.

**Details**

Suprisingly useful for thinking about processor grid shapes.

**Examples**

```c
spmd.code = "
  pbdbase::base.nbd(100, 10) # 10 divides 100, so 10 is returned  
  pbdbase::base.nbd(100, 11) # 11 does not, so the 'next best' divisor, 20, is returned  
"

pbdmpi::execmpi(spmd.code = spmd.code, nranks = 1L)
```

**base.numroc**

*NUMber of Rows Or Columns*

**Usage**

```c
base.numroc(dim, bldim, ICTXT = 0, fixme = TRUE)
```
Arguments

- **dim**: Global dim.
- **bldim**: Blocking dim.
- **ICTXT**: BLACS context.
- **fixme**: Should ldims be "rounded" to 0 or not.

Details

For advanced users only.

---

**base.ownany**  
*Determining Local Ownership of a Distributed Matrix*

Description

For advanced users only.

Usage

```r
base.ownany(dim, bldim, ICTXT = 0)
```

Arguments

- **dim**: global dimension
- **bldim**: blocking dimension
- **ICTXT**: BLACS context

Details

A simple wrapper of numroc. The return is the answer to the question 'do I own any of the global matrix?'. Passing a distributed matrix is allowed, but often it is convenient to determine that information without even having a distributed matrix on hand. In this case, explicitly passing the appropriate information to the arguments `dim=`, `bldim=` (and `ICTXT=` as necessary, since it defaults to 0) while leaving `x` missing will produce the desired result. See the examples below for more clarity.

The return for each function is local.

Examples

```r
spmd.code = 
  suppressMessages(library(pbdBASE))
  init.grid()

  iown <- ownany(dim=c(4, 4), bldim=c(2, 2), CTXT=0)
  comm.print(iown, all.rank=T)

  finalize()
```
**base.pdchtri**

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

---

**base.pdchtri  pdchtri**

**Description**
Inverse of cholesky.

**Usage**
```
base.pdchtri(uplo, x, descx, descc)
```

**Arguments**
- **uplo** Triangle whose values to use.
- **x** Matrix to crossprod.
- **descx** ScaLAPACK descriptor array.
- **descc** ScaLAPACK descriptor array of output.

**Details**
For advanced users only.

---

**base.pdclvar  Column Variances**

**Description**
Computes the variances of a ScaLAPCK-like distributed matrix. Significantly faster than using apply(), even in compared to the performance differences you would find comparing these two approaches using just base R.

**Usage**
```
base.pdclvar(x, descx)
```

**Arguments**
- **x** The matrix.
- **descx** ScaLAPACK descriptor array.
Description
Create Hilbert matrix.

Usage
base.pdhillbmk(descx)

Arguments
descx ScaLAPACK descriptor matrix.

Details
For advanced users only.

Description
Create Companion Matrix

Usage
base.pdmkcpn1(coef, descx)

Arguments
coef Coefficients vector.
descx ScaLAPACK descriptor array.

Details
For advanced users only.
**base.pdmvsum**

**R-like Matrix-Vector Sum**

**Description**

For advanced users only.

**Usage**

base.pdmvsum(x, descx, y, descy)

**Arguments**

- **x**: Matrix.
- **descx, descy**: ScaLAPACK descriptor array.
- **y**: Vector.

---

**base.pdsweep**

**pdsweep**

**Description**

Matrix-Vector Sweep

**Usage**

base.pdsweep(x, descx, vec, MARGIN, FUN)

**Arguments**

- **x**: Matrix.
- **descx**: ScaLAPACK descriptor array.
- **vec**: Vector
- **MARGIN**: Rows or columns.
- **FUN**: Function.

**Details**

For advanced users only.
base.procgrid  procgrid

Description
"Optimal" process grid when nprow and npcol are empty

Usage
base.procgrid(nprocs)

Arguments
nprocs  Number of processors.

Details
For advanced users only.

base.p_matexp_pade_wrap  p_matexp_pade_wrap

Description
Pade' expansion.

Usage
base.p_matexp_pade_wrap(A, desca, p = 6)

Arguments
A  Matrix.
desca  ScaLAPACK descriptor array.
p  Order of the Pade' approximation.

Details
For advanced users only.
Matrix power by squaring.

Usage

base.p_matpow_by_squaring_wrap(A, desca, b = 1)

Arguments

A  Matrix.
desca  ScaLAPACK descriptor array.
b  Power.

Details

For advanced users only.

R Column Copy

Description

For advanced users only.

Usage

base.rcolcpy(x, descx, y, descy, xcol, ycol)

Arguments

x, y  Matrix.
descx, descy  ScaLAPACK descriptor array.
xcol, ycol  Columns.
Description

For advanced users only.

Usage

base.rcolcpy2(x, descx, y, descy, xcol, ycol)

Arguments

x, y  Matrix.
descx, descy  ScaLAPACK descriptor array.
xcol, ycol  Columns.

Description

Redistribute a matrix from rank 0 to all ranks in block cyclic fashion.

Usage

base.redist(desc, A)

Arguments

desc  ScaLAPACK descriptor array.
A  Matrix.
**base.r2 blas**  
*Level 2 R-like BLAS*

---

**Description**  
For advanced users only.

**Usage**  
```r
base.r2blas(x, descx, vec, FUN)
```

**Arguments**  
- `x`: Matrix.  
- `descx`: ScaLAPACK descriptor array.  
- `vec`: Global vector.  
- `FUN`: Function.

---

**base.r2 insert**  
*R-like Matrix-Vector Insertion*

---

**Description**  
For advanced users only.

**Usage**  
```r
base.r2insert(x, descx, vec, i, j)
```

**Arguments**  
- `x`: Matrix.  
- `descx`: ScaLAPACK descriptor array.  
- `vec`: Global vector.  
- `i`, `j`: Indices.
Description
Inverse condition number of a general matrix.

Usage
base.rpdgecon(norm, m, n, a, desca)

Arguments
- **norm**: Type of norm.
- **m, n**: Problem size
- **a**: Matrix.
- **desca**: ScaLAPACK descriptor array.

Details
For advanced users only.

---

Description
LQ.

Usage
base.rpdgelqf(m, n, x, descx)

Arguments
- **m, n**: Problem size.
- **x**: Matrix.
- **descx**: ScaLAPACK descriptor array.

Details
For advanced users only.
**base.rpdgels**

**Description**

Linear model fitter via rank-revealing QR (with pivoting).

**Usage**

```r
base.rpdgels(tol, m, n, nrhs, a, desca, b, descb)
```

**Arguments**

- **tol**: Numerical tolerance for the QR.
- **m, n**: Problem size.
- **nrhs**: Number of right hand sides.
- **a**: Left hand side.
- **desca**: ScaLAPACK descriptor array.
- **b**: Right hand side.
- **descb**: ScaLAPACK descriptor array.

**Details**

For advanced users only.

---

**base.rpdgemm**

**Description**

Matrix-Matrix Multiply.

**Usage**

```r
base.rpdgemm(transx, transy, x, descx, y, descy, descc)
```

**Arguments**

- **transx, transy**: ’T’ or ’N’ for transpose or not.
- **x, y**: Matrix.
- **descx, descy, descc**: ScaLAPACK descriptor array.

**Details**

For advanced users only.
**Description**

General 2d block cyclic redistribution function.

**Usage**

```
base.rpdgemr2d(x, descx, descy)
```

**Arguments**

- `x` Matrix.
- `descx, descy` ScaLAPACK descriptor array.

**Details**

For advanced users only.

---

**Description**

QR.

**Usage**

```
base.rpdgeqpf(tol, m, n, x, descx, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

- `tol` Numerical tolerance for the QR.
- `m, n` Problem size.
- `x` Matrix.
- `descx` ScaLAPACK descriptor array.
- `comm` An MPI (not BLACS) communicator.

**Details**

For advanced users only.
**Description**

Solving a (square) system of equations.

**Usage**

```r
data.rpdgesv(n, nrhs, a, desca, b, descb)
```

**Arguments**

- `n`: Problem size.
- `nrhs`: Number of right hand sides.
- `a, b`: Matrix.
- `desca, descb`: ScALAPACK descriptor array.

**Details**

For advanced users only.

---

**Description**

SVD.

**Usage**

```r
data.rpdgesvd(jobu, jobvt, m, n, a, desca, descu, descvt, ..., inplace = FALSE, comm = .pbd_env$SPMD.CT$comm)
```

**Arguments**

- `jobu, jobvt`: Control for u/vt return.
- `m, n`: Problem size.
- `a`: Matrix.
- `desca, desc, descvt`: ScALAPACK descriptor array.
- `...`: Ignored
- `inplace`: Should the computation be done in-place or not. For REALLY advanced users only.
- `comm`: An MPI (not BLACS) communicator.
base.rpdgetrf  rpdgetrf

Description
LU factorization.

Usage
base.rpdgetrf(a, desca)

Arguments
a          Matrix.
desca      ScaLAPACK descriptor array.

Details
For advanced users only.

base.rpdgetri  rpdgetri

Description
Matrix inversion.

Usage
base.rpdgetri(n, a, desca)

Arguments
n          Problem size.
a          Matrix.
desca      ScaLAPACK descriptor array.

Details
For advanced users only.
base.rpdlange

Description
Matrix norms.

Usage
base.rpdlange(norm, m, n, a, desca)

Arguments
- norm: Type of norm.
- m, n: Problem size
- a: Matrix.
- desca: ScaLAPACK descriptor array.

Details
For advanced users only.

base.rpdlaprntr

Description
Matrix printer.

Usage
base.rpdlaprnrt(m, n, a, desca)

Arguments
- m, n: Number rows/cols.
- a: Matrix.
- desca: ScaLAPACK descriptor array.

Details
For advanced users only.
Description

Recover Q.

Usage

\texttt{base.rpdorglq}(m, n, k, \texttt{lq}, \texttt{desc}, \texttt{tau})

Arguments

\begin{itemize}
  \item \texttt{m}, \texttt{n} \quad \text{Problem size.}
  \item \texttt{k} \quad \text{Number of elementary reflectors.}
  \item \texttt{lq} \quad \text{QR decomposition.}
  \item \texttt{desc} \quad \text{ScaLAPACK descriptor array.}
  \item \texttt{tau} \quad \text{Elementary reflectors.}
\end{itemize}

Details

For advanced users only.

Description

Recover Q.

Usage

\texttt{base.rpdorgqr}(m, n, k, \texttt{qr}, \texttt{descqr}, \texttt{tau})

Arguments

\begin{itemize}
  \item \texttt{m}, \texttt{n} \quad \text{Problem size.}
  \item \texttt{k} \quad \text{Number of elementary reflectors.}
  \item \texttt{qr} \quad \text{QR decomposition.}
  \item \texttt{descqr} \quad \text{ScaLAPACK descriptor array.}
  \item \texttt{tau} \quad \text{Elementary reflectors.}
\end{itemize}

Details

For advanced users only.
**Description**

\( \text{op}(Q) \cdot y. \)

**Usage**

```r
base.rpdormqr(side, trans, m, n, k, qr, descqr, tau, c, descc)
```

**Arguments**

- `side` 'L' or 'R', for left or right application of Q matrix.
- `trans` Q or Q^T.
- `m, n` Problem size.
- `k` Number of elementary reflectors.
- `qr` QR decomposition.
- `descqr` ScaLAPACK descriptor array.
- `tau` Elementary reflectors.
- `c` Vector.
- `descc` ScaLAPACK descriptor array.

**Details**

For advanced users only.

---

**Description**

Cholesky factorization.

**Usage**

```r
base.rpdpotrf(uplo, n, a, desca)
```

**Arguments**

- `uplo` Triangle where the information is stored (in the symmetric matrix).
- `n` Problem size.
- `a` Matrix.
- `desca` ScaLAPACK descriptor array.
Details
For advanced users only.

Description
Symmetric eigenvalue decomposition.

Usage
base.rpdsyevr(jobz, uplo, n, a, desca, descz)

Arguments
jobz Control for if vectors/values/both are returned.
uplo Triangle where the information is stored (in the symmetric matrix).
n Problem size.
a Matrix.
desca, descz ScaLAPACK descriptor array.

Details
For advanced users only.

Description
Generalized eigenvalue problem.

Usage
base.rpdsyevx(jobz, range, n, a, desca, vl, vu, il, iu, abstol = 1e-08, orfac = 0.001)
**Arguments**

- **jobz**  
  Control for if vectors/values/both are returned.
- **range**  
  Parameter to determine the search criteria for eigenvalues.
- **n**  
  Problem size.
- **a**  
  Matrix.
- **desca**  
  ScaLAPACK descriptor array.
- **vl, vu**  
  Endpoints of the interval subset of the real line in which to search for eigenvalues, if specified by `range`.
- **il, iu**  
  Eigenvalues with indices il, ..., iu will be found, if specified by `range`.
- **abstol**  
  Absolute error tolerance for the eigenvalues.
- **orfac**  
  Eigenvectors with eigenvalues below orfac*norm(a) of each other are reorthogonalized.

**Details**

For advanced users only.

---

**Description**

Transpose.

**Usage**

```
base.rpdtran(a, desca, descc)
```

**Arguments**

- **a**  
  Matrix.
- **desca, descc**  
  ScaLAPACK descriptor array.

**Details**

For advanced users only.
base.rpdtrcon \textit{rpdtroc}\mbox{\textit{on}}

\textbf{Description}

Inverse condition number of a triangular matrix.

\textbf{Usage}

\texttt{base.rpdtrcon(norm, uplo, diag, n, a, desca)}

\textbf{Arguments}

- \texttt{norm} \hspace{1cm} Type of norm.
- \texttt{uplo} \hspace{1cm} Triangle where information is stored.
- \texttt{diag} \hspace{1cm} Specifies if the matrix is unit triangular or not.
- \texttt{n} \hspace{1cm} Problem size
- \texttt{a} \hspace{1cm} Matrix.
- \texttt{desca} \hspace{1cm} ScaLAPACK descriptor array.

\textbf{Details}

For advanced users only.

\textbf{base.rrowcpy \textit{R Row Copy}}

\textbf{Description}

For advanced users only.

\textbf{Usage}

\texttt{base.rrowcpy(x, descx, y, descy, xrow, yrow)}

\textbf{Arguments}

- \texttt{x, y} \hspace{1cm} Matrix.
- \texttt{descx, descy} \hspace{1cm} ScaLAPACK descriptor array.
- \texttt{xrow, yrow} \hspace{1cm} Rows.
**Description**

For advanced users only.

**Usage**

`base.rowcpy2(x, descx, y, descy, xrow, yrow)`

**Arguments**

- `x, y`  Matrix.
- `descx, descy`  ScaLAPACK descriptor array.
- `xrow, yrow`  Rows.

---

**Description**

Zero Triangle

**Usage**

`base.tri2zero(x, descx, uplo = "L", diag = "N")`

**Arguments**

- `x`  Matrix.
- `descx`  ScaLAPACK descriptor array.
- `uplo`  Triangle.
- `diag`  Zero diagonal as well.

**Details**

For advanced users only.
**base.valid_context**  
*BLACS Context Validation*

**Description**

Checks if a supplied ICTXT is valid.

**Usage**

```plaintext
base.valid_context(ICTXT, ..., override = FALSE)
```

**Arguments**

- **ICTXT**: BLACS context number.
- **...**: Not used.
- **override**: If override=FALSE, the context number will produce an error if it is any of the reserved contexts (0, 1, or 2).

**blacsexit**  
*BLACS Exit*

**Description**

Shuts down all BLACS communicators.

**Usage**

```plaintext
base.blacsexit(CONT = TRUE)
blacsexit(CONT = TRUE)
```

**Arguments**

- **CONT**: logical; determines whether or not to shut down all MPI communicators

**Details**

If the user wishes to shut down BLACS communicators but still have access to MPI, then call this function with CONT=TRUE. Calling blacsexit(CONT=FALSE) will shut down all MPI communicators, equivalent to calling

```plaintext
> blacsexit(CONT=TRUE) > finalize(mpi.finalize=TRUE)
```

This function is automatically invoked if BLACS communicators are running and finalize() is called.
blacs_apts

Value

Has an invisible return of 0 when successful.

Examples

```r
spmdNcode = "
suppressMessages(library(pbdBASE))
init.grid()

blacsexit()

# finalize()  # This should be off since blacexit().
"

pbdMPI::execmpi(spmdNcode = spmdNcode, nranks = 2L)
```

---

**blacs_apts**  
*Functions to set and get BLACS_APTS*

Description

To set and get BLACS array/object/whatever pointers needed in and from R. Because other packages has it's own memory stack vision that may not be visible by this package or vice versa.

Usage

```r
set.blacs.apts()

get.blacs.apts()
```

Details

The `set.blacs.apts()` is for advanced users. This one is needed to be called within R from 'pbdBASE' package to set the pointers to the memory where BLACS had initialized so that the pointers are set to the right address of the memory stack.

The `get.blacs.apts()` is for debugging only. The advanced user mainly calls the C version `get_BLACS_APTS_from_R()` in 'src/export_blacs/pkgools.c'.

I am lazy to use `.C()`, but should not hurt performance here. Eventually, .pbdBASEEnv should pass to `.C()` and set/get pointers from it instead of .GlobalEnv.
coords

Local to Global/Global to Local Indexing

Description
Get the local index given global information.

Usage
indxg2l(INDXGLOB, NB, IPROC, ISRCPROC, NPROCS)
indx12g(INDXLOC, NB, IPROC, ISRCPROC, NPROCS)

Arguments
- INDEXGLOB: Global index.
- NB: Block size.
- IPROC: Coordinate of the process whose local info is to be determined.
- ISRCPROC: The coordinate of the process that possesses the first row/column of the distributed matrix. That's always 0 pbdDMAT.
- NPROCS: Total number of processors over which matrix is distributed.
- INDXLOC: Local index.

Details
For advanced users only.

coordspair

Global to Local/Local to Global Pair Indexing

Description
Get the local index-pair given global information.

Usage
g2lpair(gi, gj, bldim, ICTXT)
l2gpair(i, j, bldim, ICTXT)
**Arguments**

- \(gi, gj\)  
  Global indices.
- \(bldim\)  
  Blocking dimension
- \(ICTXT\)  
  BLACS context.
- \(i, j\)  
  Local indices.

**Details**

For advanced users only.

---

**Description**

Grab diagonal or create distributed diagonal matrix.

**Usage**

```plaintext
base.ddiagtk(x, descx, proc.dest = "all")

base.ddiagmk(diag, descx)
```

**Arguments**

- \(x\)  
  Matrix.
- \(descx\)  
  ScaLAPACK descriptor array.
- \(proc.dest\)  
  Who owns the result.
- \(diag\)  
  Diagonal.

**Details**

For advanced users only.
finalizer  

*Finalizer*

**Description**

A replacement for `pbdMPI::finalize()` that automatically shuts BLACS communicators down.

**Usage**

```r
base.finalize(mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize)

finalize(mpi.finalize = .pbd_env$SPMD.CT$mpi.finalize)
```

**Arguments**

- `mpi.finalize`  If MPI should be shut down.

---

g2lcoord  

*g2lcoord*

**Description**

Global to local coordinates with explicit ownership given.

**Usage**

```r
g2lcoord(dim, bldim, gi, gj, gridinfo)
```

**Arguments**

- `dim`  Global dimension.
- `bldim`  Blocking dimension.
- `gi, gj`  Global row and column indices, respectively.
- `gridinfo`  The return of `base.blacs(ICTXT(x))`. See the Details section for more information.

**Value**

For the process that owns the desired local data at global indices `(gi, gj)`, the return is the local index. Otherwise, NA is returned.
**g2l_coord**

**Description**

Global to local coords.

**Usage**

```c
base.g2l Coord(ind, bldim, ICTXT = 0, dim = NULL)
g2l Coord(ind, bldim, ICTXT = 0, dim = NULL)
```

**Arguments**

- `ind` Matrix indices.
- `bldim` Blocking dimension.
- `ICTXT` BLACS context.
- `dim` Ignored; will be removed in a future version.

**Details**

For advanced users only.

---

**get.comm.from.ICTXT  Getting Communicator From BLACS Context**

**Description**

Blacs context are associated with a certain communicator. It can be useful to retrieve this communicator to manipulate the matrix accordingly.

**Usage**

```c
get.comm.from.ICTXT(ICTXT)
```

**Arguments**

- `ICTXT` a BLACS context
gridexit

Description
Frees a BLACS context.

Usage
base.gridexit(ICTXT, override = FALSE)
gridexit(ICTXT, override = FALSE)

Arguments
ICTXT  BLACS context number.
override  logical; if TRUE, ignores normal check preventing the closing of ICTXT values of 0, 1, and 2. This could cause things to go crazy and I do not recommend it.

Details
For advanced users only.
The function frees the requested BLACS context. It is a trivial wrapper for the BLACS routine BLACS_GRIDEXIT. Also removes the object __blacs_gridinfo_ICTXT.
Contexts 0, 1, and 2 can not be freed in this way unless the argument override=FALSE. This will probably break something and I do not recommend it.

Value
Silently returns 0 when successful. Silently returns 1 when requested ICTXT does not exist.

gridinfo

Get BLACS Context Grid Information

Description
Grabs the existing BLACS context grid information.

Usage
base.blacs(ICTXT = 0)
blacs(ICTXT = 0)
Arguments
ICTXT       BLACS context number.

Details
BLACS contexts have important internal use, and advanced users familiar with ScaLAPACK might find some advantage in directly manipulating these process grids. Most users should not need to directly manage BLACS contexts, in this function or elsewhere.
The function effectively serves as a shorthand for
eval(parse(text=paste("._blacs_gridinfo_", ICTXT, sep="")))

Value
Returns a list with 5 elements: NPROW and NPCOL, the number of process rows and columns respectively; ICTXT, the associated BLACS context number; MYROW and MYCOL, the current process' row and column position in the process grid.

Examples
```
spmd_code = ""
suppressMessages(library(pbdBASE))
init.grid()

mygrid <- blacs(0)
pbdMPI::comm.print(mygrid)

finalize()
```
```
pbdMPI::execmpi(spmd_code = spmd_code, nranks = 2L)
```

Description
BLACS grid initialization.

Usage
```
base.blacs_init(ICTXT, NPROW, NPCOL, ..., quiet = FALSE)
blacs_init(ICTXT, NPROW, NPCOL, ..., quiet = FALSE)
blacs_gridinit(ICTXT, NPROW, NPCOL, ..., quiet = FALSE)
```
Arguments

ICTXT    BLACS context.
NPROW, NPCOL Number of process rows/cols.
...        Additional arguments.
quiet     Verbose initialization or not.

Details

For advanced users only.

InitGrid  Initialize Process Grid

Description

Manages the creation of BLACS context grids.

Usage

init.grid(NPROW, NPCOL, ICTXT, quiet = FALSE)

Arguments

NPROW number of process rows. Can be missing; see details.
NPCOL number of process columns. Can be missing; see details.
ICTXT  BLACS context number.
quiet logical; controls whether or not information about grid size should be printed.

Details

blacs_init() is for experienced users only. It is a shallow wrapper of the BLACS routine BLACS_INIT, with the addition of creating the __blacs_gridinfo_ICTXT objects, as described below.

The remainder of this section applies only to init.grid().

If ICTXT is missing, three variables will be created in the .pbdBASEEnv environment:

__blacs_gridinfo_0
__blacs_gridinfo_1
__blacs_gridinfo_2

These variables store the BLACS process grid information for the BLACS context corresponding to the trailing digit of the variable. Most users should invoke init.grid() in this fashion, namely with ICTXT missing, and only do so once.

Contexts 0, 1, and 2 are reserved. Additional custom contexts are possible to create, but they must be integers >= 3.
Context 0 is the “full” process grid of NPROW by NPCOL processes; contexts 1 is the process grid consisting of 1 process row and NPROW*NPCOL processes columns; context 2 is the process grid consisting of NPROW*NPCOL processes rows and 1 process column. These contexts can be redundant depending on the number of processes available.

BLACS contexts have important internal use, and advanced users familiar with ScaLAPACK might find some advantage in directly manipulating these process grids. Most users should not need to directly manage BLACS contexts, in this function or elsewhere.

If the NPROW and NPCOL values are missing, then a best process grid will be chosen for the user based on the total available number of processes. Here “best” means as close to a square grid as possible.

The variables __blacs_gridinfo_ICTX are just storage mechanisms to avoid needing to directly invoke the BLACS routine BLACS_GRIDINFO.

Additionally, another variable is created in the .pbdbASEEnv environment, namely __blacs_initialized. Its existence is to alert finalize() to shut down BLACS communicators, if necessary, to prevent memory leaks.

Value

Silently returns 0 when successful. Additionally, several variables are created in the .pbdbASEEnv environment. See Details section.

Examples

```cpp
spmdNcode = "
suppressMessages(library(pbdBASE))
init.grid()

   finalize()
"

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

---

**Description**

Local to global coords.

**Usage**

```cpp
base.12g_coord(ind, bldim, ICTXT = 0, dim = NULL)
12g_coord(ind, bldim, ICTXT = 0, dim = NULL)
```
Arguments

ind       Matrix indices.
bldim     Blocking dimension.
ICTXT     BLACS context.
dim       Ignored; will be removed in a future version.

Details

For advanced users only.

Description

A better version of NUMROC (NUMber Rows Or Columns). Returns the local dimension given
global matrix + distribution parameters.

Usage

numroc2(N, NB, IPROC, NPROCS)

Arguments

N         Global number of rows/cols.
NB        Block size.
IPROC     Coordinate of the process whose local info is to be determined.
NPROCS    Total number of processors over which matrix is distributed.

Details

For advanced users only.
Description
Grabs the existing BLACS context grid information.

Usage
base.pnum(ICTXT, PROW, PCOL)
base.pcoord(ICTXT, PNUM)

Arguments
ICTXT BLACS context number.
PROW, PCOL BLACS grid location row/column
PNUM process rank

Details
For advanced users only. These functions are simple recreations of the BLACS routines BLACS_PNUM and BLACS_PCOORD. The former gets the process number associated with the BLACS process grid location c(MYPROW, MYPCOL), while the latter does the reverse.

Value
pnum returns an integer; pcoord returns a list containing elements PROW and PCOL.

Examples
spmd.code = "
suppressMessages(library(pbdBASE))
init.grid()

blacs_ <- blacs(ICTXT = 0)
# get the ICTXT = 0 BLACS coordimates for process 0
myCoords <- base.pcoord(ICTXT = 0, PNUM = 0)

comm.print(myCoords)

finalize()
"

pbdMPI::execmpi(spmd.code = spmd.code, nranks = 2L)
sys2blacs.handle  Context Within a Given Communicator

Description

Creates a context that will be valid for a given communicator

Usage

sys2blacs.handle(comm)

Arguments

comm  Communicator for which you want to set the BLACS context

Value

A system handle, i.e. the system context number. System contexts can be used to have ScalaPACK methods run in different communicators.

See Also

base.free_blacs_system_handle, base.blacs_gridinit
Index

*Topic BLACS
  base.minctxt, 14
  base.ownany, 16
  blacsexit, 36
  gridexit, 42
  gridinfo, 42
  InitGrid, 44
  pcoords, 47

*Topic Data
  base.ownany, 16

*Topic Distributing
  base.ownany, 16

*Topic Package
  pbdbaseMpackage, 3

*Topic datasets
  BASE Global Environment, 5
  .pbdbASEEnv (BASE Global Environment), 5
  BASE Global Environment, 5
  base.blacs (gridinfo), 42
  base.blacs_gridinit, 6
  base.blacs_init (gridinit), 43
  base.blacsexit (blacsexit), 36
  base.crossprod, 6
  base.dallreduce, 7
  base.ddiagm (diag), 39
  base.ddiagtk (diag), 39
  base.descinit, 7
  base.dgam2d (base.dgamx2d), 8
  base.dgamx2d, 8
  base.dger2d (base.dgesd2d), 8
  base.dgesd2d, 8
  base.dgsum2d (base.igsum2d), 11
  base.dhilbmk, 9
  base.dim0, 10
  base.finalize (finalizer), 40
  base.free_blacs_system_handle, 10
  base.g21_coord (g21_coord), 41
  base.gridexit (gridexit), 42
  base.igamn2d (base.dgamx2d), 8
  base.igamx2d, 11
  base.igsum2d, 11
  base.indxg2p, 12
  base.12g_coord (12g_coord), 45
  base.matexp, 13
  base.maxdim, 13
  base.minctxt, 14
  base.mkgb1mat (base.mksubmat), 14
  base.mksubmat, 14
  base.nbd, 15
  base.numroc, 15
  base.ownany, 16
  base.p_matexp_pade_wrap, 20
  base.p_matpow_by_squaring_wrap, 21
  base.pcoord (pcoords), 47
  base.pdcchtri, 17
  base.pdc1var, 17
  base.pdhi1bmk, 18
  base.pdmkcpn1, 18
  base.pdmvsum, 19
  base.pdsweep, 19
  base.pnum (pcoords), 47
  base.procgrid, 20
  base.rcolcpy, 21
  base.rcolcpy2, 22
  base.redist, 22
  base.r12blas, 23
  base.r12insert, 23
  base.rpdgecon, 24
  base.rpdgelqf, 24
  base.rpdgels, 25
  base.rpdgemm, 25
  base.rpdgmr2d, 26
  base.rpdgeqpf, 26
  base.rpdgesv, 27
  base.rpdgesvd, 27
  base.rpdgetrf, 28
  base.rpdgetri, 28
  base.rpdilange, 29
base.rpdlaprt, 29
base.rpdorglq, 30
base.rpdorgqr, 30
base.rpdormqr, 31
base.rpdpotrf, 31
base.rpdsevq, 32
base.rpdsevx, 32
base.rpdttran, 33
base.rpdtcon, 34
base.rrowcpy, 34
base.rrowcpy2, 35
base.tri2zero, 35
base.valid_context, 36
blacs(gridinfo), 42
blacs_aps, 37
blacs_gridinit(gridinit), 43
blacs_init(gridinit), 43
blacsexit, 36
cords, 38
cordspair, 38
diag, 39
finalize(finalizer), 40
finalizer, 40
g2l_coord, 41
g2lcoord, 40
g2lpair(cordspair), 38
get.blacs_aps(blacs_aps), 37
get.comm.from.ICTXT, 41
gridexit, 42
gridinfo, 42
gridinit, 43
indxg2l(coords), 38
indxl2g(coords), 38
init.grid(InitGrid), 44
InitGrid, 44
l2g_coord, 45
l2gpair(cordspair), 38
numroc2, 46
pbdBASE-package, 3
pcoords, 47
set.blacs_aps(blacs_aps), 37
sys2blacs.handle, 48